O'Mega: Optimal Monte-Carlo Event Generation Amplitudes

Thorsten Ohl*

Institut für Theoretische Physik und Astrophysik Julius-Maximilians-Universität Würzburg Emil-Hilb-Weg 22, 97074 Würzburg, Germany

Jürgen Reuter[†]

DESY Theory Group, Notkestr. 85, 22603 Hamburg, Germany

Wolfgang Kilian c,‡

Theoretische Physik 1 Universität Siegen Walter-Flex-Str. 3, 57068 Siegen, Germany

with contributions from Christian Speckner^{d,§} as well as Christian Schwinn et al.

unpublished draft, printed 12/13/2022, 11:43

Abstract

^{*}ohl@physik.uni-wuerzburg.de, http://physik.uni-wuerzburg.de/ohl

[†]juergen.reuter@desy.de

[‡]kilian@physik.uni-siegen.de

[§]cnspeckn@googlemail.com

Copyright © 1999-2017 by

- Wolfgang Kilian <kilian@hep.physik.uni-siegen.de>
- Thorsten Ohl <ohl@physik.uni-wuerzburg.de>
- Jürgen Reuter <juergen.reuter@desy.de>

WHIZARD is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

WHIZARD is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

CONTENTS

1	INTE	RODUCTION	1
	1.1	Complexity	 1
	1.2	Ancestors	 1
	1.3	Architecture	 2
		1.3.1 General purpose libraries	 2
		1.3.2 O'Mega	2
		1.3.3 Abstract interfaces	2
		1.3.4 Models	2
		1.3.5 Targets	2
		1.3.6 Applications	4
	1.4	The Big To Do Lists	4
	1.4	1.4.1 Required	4
		1.4.2 Useful	4
		1.4.3 Future Features	4
		1.4.4 Science Fiction	4
		1.4.4 Detente Pietton	 4
2	Тпр	PLES AND POLYTUPLES	5
-	2.1	Interface of Tuple	5
	2.2	Implementation of Tuple	7
	2.2	2.2.1 Typesafe Combinatorics	8
		2.2.2 Pairs	8
		2.2.3 Triples	10
		2.2.4 Pairs and Triples	11
		2.2.5 and All The Rest	13
		2.2.0 that The Hest	 10
3	Тор	POLOGIES	16
Ĭ	3.1	Interface of Topology	 16
		3.1.1 Diagnostics: Counting Diagrams and Factorizations for $\sum_{n} \lambda_{n} \phi^{n}$	17
		3.1.2 Emulating HELAC	 18
	3.2	Implementation of Topology	18
	٥.٠	3.2.1 Factorizing Diagrams for ϕ^3	18
		3.2.2 Factorizing Diagrams for $\sum_{n} \lambda_n \phi^n$	 20
		3.2.3 Factorizing Diagrams for ϕ^4	 22
		3.2.4 Factorizing Diagrams for $\phi^3 + \phi^4$	 $\frac{22}{22}$
		3.2.5 Diagnostics: Counting Diagrams and Factorizations for $\sum_{n} \lambda_{n} \phi^{n}$	$\frac{22}{22}$
		3.2.6 Emulating HELAC	$\frac{22}{27}$
		5.2.0 Emailing HEBAC	 41
4	Diri	ECTED ACYCLICAL GRAPHS	29
	4.1	Interface of DAG	29
	4.1	4.1.1 Forests	29
		4.1.2 DAGs	$\frac{25}{30}$
		4.1.2 DAGs	$\frac{30}{32}$
	4.2	Implementation of DAG	$\frac{32}{33}$
	4.2	4.2.1 The Forest Functor	34
		4.2.2 Gradings	$\frac{34}{35}$
		4.2.3 The DAG Functor	 $\frac{36}{36}$
		4. W. U 1100 1211 U 1 U 160 U 1	 -00

5	Mon	MENTA		41
	5.1	Interfa	e of $Momentum$	41
		5.1.1	Scattering Kinematics	42
		5.1.2	Decay Kinematics	43
	5.2		entation of Momentum	44
	0.2	5.2.1	$Lists \ of \ Integers \ \ldots \ \ldots \ \ldots \ \ldots \ \ldots \ \ldots \ \ldots$	44
		5.2.2	Bit Fiddlings	49
		5.2.3	Whizard	53
		5.2.4	Suggesting a Default Implementation	54
6	CAS	CADES		55
	6.1	Interfa	e of Cascade_syntax	55
	6.2	Implen	entation of Cascade_syntax	55
	6.3	Lexer		57
	6.4	Parser		57
	$6.\overline{5}$		e of Cascade	59
	6.6		entation of Cascade	60
	0.0	impien	contained of Cascade	00
7	Col	OR		68
	7.1		e of Color	68
	7.1	7.1.1	$Quantum\ Numbers$	68
		7.1.2	Color Flows	68
	m (0	7.1.3	Vertex Color Flows	69
	7.2		entation of Color	73
		7.2.1	$Quantum\ Numbers$	73
		7.2.2	Color Flows	75
		7.2.3	Vertex Color Flows	79
		7.2.4	$\mathrm{SU}(N_C)$	99
		7.2.5	$\mathrm{U}(\grave{N}_C)^{'}$	118
8	Fusi	IONS		122
	8.1	Interfa	e of Fusion	122
		8.1.1	Multiple Amplitudes	125
		8.1.2	Tags	126
	8.2		$entation \ of \ Fusion \ . \ . \ . \ . \ . \ . \ . \ . \ . \ $	126
	0.2			
		8.2.1	Fermi Statistics	127
		8.2.2	Dirac Fermions	128
		8.2.3	Tags	133
		8.2.4	Tagged, the Fusion. Make Functor	134
		8.2.5	Fusions with Majorana Fermions	164
		8.2.6	Multiple Amplitudes	171
9	Lor	ENTZ R	epresentations, Couplings, Models and Targets	178
	9.1	Interfa	e of Coupling	178
		9.1.1	Propagators	178
		9.1.2	Vertices	180
		9.1.3	$Gauge\ Couplings$	184
		9.1.4	SU(2) Gauge Bosons	193
		9.1.5		193
			Quartic Couplings and Auxiliary Fields	
		9.1.6	Gravitinos and supersymmetric currents	194
		9.1.7	Perturbative Quantum Gravity and Kaluza-Klein Interactions	202
		9.1.8	Dependent Parameters	203
		9.1.9	More Exotic Couplings	208
	9.2	Interfa	e of Model	208
		9.2.1	General Quantum Field Theories	208
		9.2.2	Mutable Quantum Field Theories	211
		9.2.3	Gauge Field Theories	212
		9.2.4	Gauge Field Theories with Broken Gauge Symmetries	212
	9.3	•	e of Dirac	213
	0.0	9.3.1	$Dirac \ \gamma$ -matrices	213
	0 /		'	$\frac{213}{214}$
	9.4	тириет	$entation of Dirac \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots $	214

$9.4.1$ Dirac γ -matrices 5 Interface of Vertex 6 Implementation of Vertex $9.6.1$ New Implementation: Next Version $9.6.2$ New Implementation: Obsolete Version 1	. 221. 221. 222. 233
7 Interface of Target	. 246
ONSERVED QUANTUM NUMBERS 0.1 Interface of Charges	. 248. 248. 248
OLORIZATION 1.1 Interface of Colorize 11.1.1 1.2 Implementation of Colorize 11.2.1 Auxiliary functions 11.2.2 Flavors Adorned with Colorflows 11.2.3 The Legacy Implementation 11.2.4 Colorizing a Monochrome Model 11.2.5 Colorizing a Monochrome Gauge Model	. 250. 250. 250. 255. 255. 268
ROCESSES 2.1 Interface of Process	. 278. 278. 278
IODEL FILES 3.1 Interface of Vertex_syntax 13.1.1 Abstract Syntax 3.2 Implementation of Vertex_syntax 13.2.1 Abstract Syntax 3.3 Lexer 3.4 Parser 3.5 Interface of Vertex 3.6 Implementation of Vertex 13.6.1 New Implementation: Next Version 13.6.2 New Implementation: Obsolete Version 1	. 284. 288. 288. 296. 298. 303. 303. 303
FO Models 4.1 Interface of UFOx_syntax 14.1.1 Abstract Syntax 4.2 Implementation of UFOx_syntax 14.2.1 Abstract Syntax 4.3 Expression Lexer 4.4 Expression Parser 4.5 Interface of UFOx 4.6 Implementation of UFOx 4.7 Interface of UFO_syntax 14.7.1 Abstract Syntax 4.8 Implementation of UFO_syntax 14.8.1 Abstract Syntax 4.9 Lexer 4.10Parser	. 329 . 329 . 330 . 331 . 332 . 335 . 358 . 359 . 360
4.11Interface of UFO_Lorentz 14.11.1Processed UFO Lorentz Structures 4.12Implementation of UFO_Lorentz	. 364

	•	365
	· · · · · · · · · · · · · · · · · · ·	368
	· ·	369
		$\frac{377}{276}$
		$\frac{379}{422}$
		422
		422
		423
		423
15		444
		444
		444
		448
		448 448
	v	446
		440
		474
		572
		572
		572
		572
		572
16		583
		583 583
		ออล 583
		584
	10.2.2 Thuse space I arameterization frees	00-
17		589
		589
		589
		590
		590
		591
	· · · · · · · · · · · · · · · · · · ·	593
	17.2.5 Makefile	594
18	Applications	596
		596
	18.2 Interface of Omega	596
	18.3 Implementation of Omega	596
		601
		606
		607
	18.6 Implementation of Omega_SYM	607
٨	AUTOTOOLS	614
A		614
	11.1 Investigate of Config	OT-
В	TEXTUAL OPTIONS	615
	B.1 Interface of Options	615
		615
\sim	Dr. corres Dr. corres	01.
U		616 616
		616 616
	C. S. Inspections and the Contract of the Cont	σT

D	More on Filenames D.1 Interface of ThoFilename	619 619 619
E	Cache Files E.1 Interface of Cache	620 620 620
F		623 623 625
G	More On Arrays G.1 Interface of ThoArray	635 635 635
Н	More On Strings H.1 Interface of ThoString	640 640 640
I	POLYMORPHIC MAPS I.1 Interface of Pmap	644 645 652 653 654
J	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	656 656 656 656 657 657 658 660 660
K	Tensor Product K.1 Interface of Product K.1.1 Lists K.1.2 Sets K.2 Implementation of Product K.2.1 Lists K.2.2 Sets	663 663 663 663 663 665
L	(FIBER) BUNDLES L.1 Interface of Bundle	666 666 667
M	POWER SETS M.1 Interface of PowSet	670 670 670
N	COMBINATORICS N.1 Interface of Combinatorics	673 673 673 674 674

	N.2	mplementation of Combinatorics	
		<u> 1.2.2 Partitions</u>	76
		7.2.3 <i>Choices</i>	78
		7. 2.4 <i>Permutations</i>	' 9
	N.3	nterface of Permutation	3
	N.4	$nplementation of Permutation \dots 68$	34
O	PART		0
		nterface of Partition)()
	O.2	$mplementation of Partition \dots \dots$	0
Р	You	G DIAGRAMS AND TABLEAUX 69	12
	P.1	nterface of Young)2
		P.1.1 Young Diagrams Young Diagrams 69)2
		1.1.2 Young Tableaux Young Tableaux 69)3
		<i>P.1.3 Unit Tests </i>)3
	P.2	$mplementation of Young \dots \dots$)4
Q	TREI	69	18
	Q.1	$nterface \ of \ Tree \ \ldots \$	9
		0.1.1 Abstract Data Type	9
		0.1.2 Homomorphisms	9
		0.1.3 Output	0(
	Q.2	$mplementation of Tree \dots $	
		0.2.1 Abstract Data Type	
		0.2.2 Homomorphisms	
		0.2.3 Output	
		0.2.4 Least Squares Layout)7
R		DENCY TREES 71	
		nterface of Tree2	
	R.2	mplementation of Tree2	.1
\mathbf{S}	Con	STENCY CHECKS 71	.3
Т	Сом	LEX NUMBERS 71	.4
U		71	.5
	U.1	$aterface \ of \ Algebra \ \ldots $	_
		7.1.1 Coefficients $\dots \dots \dots$	
		7.1.2 Naive Rational Arithmetic	
		7.1.3 Rational Complex Numbers	
		7.1.4 Laurent Polynomials	
	T T 0	7.1.5 Expressions: Terms, Rings and Linear Combinations	
	U.2	$mple mentation of Algebra \dots \dots$	
		7.2.1 Coefficients	
		V.2.2 Naive Rational Arithmetic	
		V.2.3 Rational Complex Numbers	
		7.2.4 Laurent Polynomials	
	C		
V		E LINEAR ALGEBRA 73	
		nterface of Linalg	
	v.2	$mplementation of Linalg \dots 73$	
		$7.2.1 \;\; LU \; Decomposition \ldots \qquad 73$) Z
W		AL MAPS	
		nterface of Partial	
	W.2	nplementation of Partial	
		$V.2.1 \ Unit \ Tests$	ıń

TAL	To The WHiZard	
	AN LIBRARIES	
Y.1	Trivia	
	Y.1.1 Inner $Product$	
	Y.1.2 Spinor Vector Space	
	Y.1.3 Norm	
Y.2	$Spinors \; Revisited \;\; \ldots \; \ldots \;$	
	Y.2.1 Spinor Vector Space	
	Y.2.2 Norm	
\mathbf{V} o		
Y.3	Vectorspinors	
	Y.3.1 Vectorspinor Vector Space	
	Y.3.2 Norm	
Y.4	Vectors and Tensors	
	Y.4.1 Constructors	
	Y.4.2 Inner Products	
	Y.4.3 Not Entirely Inner Products	
	Y.4.4 Outer Products	
	Y.4.5 Vector Space	
	Y.4.6 Norm	
	Y.4.7 Conjugation	
	$Y.4.8$ $\epsilon ext{-}Tensors$	
	Y.4.9 Utilities	
Y.5	Polarization vectors	
Y.6	Polarization vectors revisited	
Y.7	Symmetric Tensors	
1.1	Y.7.1 Vector Space	
T Z 0		
Y.8	Symmetric Polarization Tensors	
Y.9	Couplings	
	Y.9.1 Triple Gauge Couplings	
	Y.9.2 Quadruple Gauge Couplings	
	Y.9.3 Scalar Current	
	Y.9.4 Transversal Scalar-Vector Coupling	
	Y.9.5 Transversal TensorScalar-Vector Coupling	
3 7.4.	\mathcal{F}	
	Tensorvector - Scalar coupling	
	$Scalar ext{-}Vector\ Dim ext{-}5\ Couplings$	
	$Dim ext{-}6$ Anoumalous Couplings with Higgs \dots	
Y.13	Dim-6 Anoumalous Couplings without $Higgs$	
	$ extit{Dim-6 Anoumalous Couplings with } AWW$	
	$Dim ext{-}6$ Quartic Couplings	
	$Scalar 3 \ Dim - 5 \ Couplings$	
	$Tensors calar - Scalar \ Couplings \ \ldots \ $	
	Scalar2-Vector2 Dim-8 Couplings	
	Scalar4 Dim-8 Couplings	
Y.20	$Mixed\ Scalar 2 ext{-}Vector 2\ Dim ext{-}8\ Couplings$	
Y.2	$Transversal\ Gauge 4\ Dim - 8\ Couplings$	
	Mixed Gauge4 Dim-8 Couplings	
	Graviton Couplings	
	Tensor Couplings	
Y.28	Spinor Couplings	
	Y.25.1 Fermionic Vector and Axial Couplings	
	Y.25.2 Fermionic Scalar and Pseudo Scalar Couplings	
	Y.25.3 On Shell Wave Functions	
	Y.25.4 Off Shell Wave Functions	
	Y.25.5 Propagators	
V a		
r.20	Spinor Couplings Revisited	
	Y.26.1 Fermionic Vector and Axial Couplings	
	Y.26.2 Fermionic Scalar and Pseudo Scalar Couplings	
	Y.26.3 Couplings for BRST Transformations	
	Y.26.4 Gravitino Couplings	

Y.26.5 Gravitino 4-	Couplings	 	 	 	 	 		
Y.26.6 On Shell W								
Y.26.7 Off Shell W								
Y.26.8 Propagators								
Y.27 Polarization vectors								
Y.28 Color								
Y.28.1 Color Sum								
Y.29.1 Helicity Sele								
Y.29.2 Diagnostics								
Y.29.3 Obsolete Sur								
Y.30 omega 95								
Y.31 omega95 Revisited								
$Y.32 Testing \dots$								
Y.33 O'Mega Virtual Ma								
Y.33.1 Memory Lag								
Y.33.2 Controlling								
Y.33.3 Reading the								
Y.33.4 Main Decod								
Y.33.5 Helper funct								
Y.33.6 O'Mega Inte								
Z Index								

—1— Introduction

1.1 Complexity

There are

$$P(n) = \frac{2^{n} - 2}{2} - n = 2^{n-1} - n - 1 \tag{1.1}$$

independent internal momenta in a n-particle scattering amplitude [1]. This grows much slower than the number

$$F(n) = (2n-5)!! = (2n-5) \cdot (2n-7) \cdot \dots \cdot 3 \cdot 1 \tag{1.2}$$

of tree Feynman diagrams in vanilla ϕ^3 (see table 1.1). There are no known corresponding expressions for theories with more than one particle type. However, empirical evidence from numerical studies [1, 2] as well as explicit counting results from O'Mega suggest

$$P^*(n) \propto 10^{n/2} \tag{1.3}$$

while he factorial growth of the number of Feynman diagrams remains unchecked, of course.

The number of independent momenta in an amplitude is a better measure for the complexity of the amplitude than the number of Feynman diagrams, since there can be substantial cancellations among the latter. Therefore it should be possible to express the scattering amplitude more compactly than by a sum over Feynman diagrams.

1.2 Ancestors

Some of the ideas that O'Mega is based on can be traced back to HELAS [5]. HELAS builts Feynman amplitudes by recursively forming off-shell 'wave functions' from joining external lines with other external lines or off-shell 'wave functions'.

The program Madgraph [6] automatically generates Feynman diagrams and writes a Fortran program corresponding to their sum. The amplitudes are calculated by calls to HELAS [5]. Madgraph uses one straightforward optimization: no statement is written more than once. Since each statement corresponds to a collection of trees, this optimization is very effective for up to four particles in the final state. However, since the amplitudes are

n	P(n)	F(n)
4	3	3
5	10	15
6	25	105
7	56	945
8	119	10395
9	246	135135
10	501	2027025
11	1012	34459425
12	2035	654729075
13	4082	13749310575
14	8177	316234143225
15	16368	7905853580625
16	32751	213458046676875

Table 1.1: The number of ϕ^3 Feynman diagrams F(n) and independent poles P(n).

given as a sum of Feynman diagrams, this optimization can, by design, not remove the factorial growth and is substantially weaker than the algorithms of [1, 2] and the algorithm of O'Mega for more particles in the final state.

Then ALPHA [1] (see also the slightly modified variant [2]) provided a numerical algorithm for calculating scattering amplitudes and it could be shown empirically, that the calculational costs are rising with a power instead of factorially.

1.3 Architecture

1.3.1 General purpose libraries

Functions that are not specific to O'Mega and could be part of the O'Caml standard library

ThoList: (mostly) simple convenience functions for lists that are missing from the standard library module List (section F, p. 623)

Product: efficient tensor products for lists and sets (section K, p. 663)

Combinatorics: combinatorical formulae, sets of subsets, etc. (section N, p. 673)

1.3.2 O'Mega

The non-trivial algorithms that constitute O'Mega:

DAG: Directed Acyclical Graphs (section 4, p. 29)

Topology: unusual enumerations of unflavored tree diagrams (section 3, p. 16)

Momentum: finite sums of external momenta (section 5, p. 41)

Fusion: off shell wave functions (section 8, p. 122)

Omega: functor constructing an application from a model and a target (section 18, p. 596)

1.3.3 Abstract interfaces

The domains and co-domains of functors (section 9, p. 178)

Coupling: all possible couplings (not comprensive yet)

Model: physical models

Target: target programming languages

1.3.4 Models

(section ??, p. ??)

 $Modellib_S M.QED$: Quantum Electrodynamics

Modellib_S M. QCD: Quantum Chromodynamics (not complete yet)

Modellib_S M.SM: Minimal Standard Model (not complete yet)

etc.

1.3.5 Targets

Any programming language that supports arithmetic and a textual representation of programs can be targeted by O'Caml. The implementations translate the abstract expressions derived by *Fusion* to expressions in the target (section 15, p. 444).

 ${\it Targets. Fortran:} \ {\it Fortran95 \ language \ implementation, calling \ subroutines}$

Other targets could come in the future: C, C++, O'Caml itself, symbolic manipulation languages, etc.

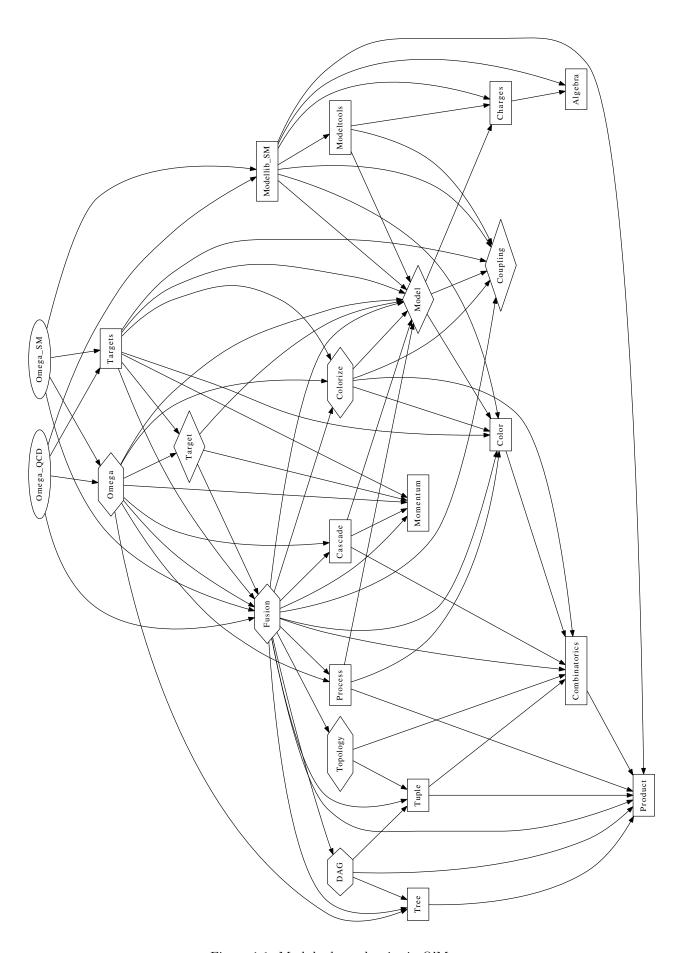


Figure 1.1: Module dependencies in O'Mega.

1.3.6 Applications

(section 18, p. 596)

1.4 The Big To Do Lists

1.4.1 Required

All features required for leading order physics applications are in place.

1.4.2 Useful

- 1. select allowed helicity combinations for massless fermions
- 2. Weyl-Van der Waerden spinors
- 3. speed up helicity sums by using discrete symmetries
- 4. general triple and quartic vector couplings
- 5. diagnostics: count corresponding Feynman diagrams more efficiently for more than ten external lines
- 6. recognize potential cascade decays $(\tau, b, \text{ etc.})$
 - warn the user to add additional
 - kill fusions (at runtime), that contribute to a cascade
- 7. complete standard model in R_{ε} -gauge
- 8. groves (the simple method of cloned generations works)

1.4.3 Future Features

- 1. investigate if unpolarized squared matrix elements can be calculated faster as traces of densitiy matrices. Unfortunately, the answer apears to be no for fermions and up to a constant factor for massive vectors. Since the number of fusions in the amplitude grows like $10^{n/2}$, the number of fusions in the squared matrix element grows like 10^n . On the other hand, there are $2^{\#\text{fermions}+\#\text{massless vectors}} \cdot 3^{\#\text{massive vectors}}$ terms in the helicity sum, which grows slower than $10^{n/2}$. The constant factor is probably also not favorable. However, there will certainly be asymptotic gains for sums over gauge (and other) multiplets, like color sums.
- 2. compile Feynman rules from Lagrangians
- 3. evaluate amplitues in O'Caml by compiling it to three address code for a virtual machine

```
\begin{array}{lll} \text{type } mem &=& scalar \ array \times spinor \ array \times spinor \ array \times vector \ array \\ \text{type } instr &=& \\ &-& VSS \ \text{of } int \times int \times int \\ &-& SVS \ \text{of } int \times int \times int \\ &-& AVA \ \text{of } int \times int \times int \\ &\cdots \end{array}
```

this could be as fast as [1] or [2].

- 4. a virtual machine will be useful for for other target as well, because native code appears to become to large for most compilers for more than ten external particles. Bytecode might even be faster due to improved cache locality.
- 5. use the virtual machine in O'Giga

1.4.4 Science Fiction

1. numerical and symbolical loop calculations with O'TERA: O'MEGA TOOL FOR EVALUATING RENOR-MALIZED AMPLITUDES

<u>---2</u>---

TUPLES AND POLYTUPLES

2.1 Interface of Tuple

The *Tuple.Poly* interface abstracts the notion of tuples with variable arity. Simple cases are binary polytuples, which are simply pairs and indefinite polytuples, which are nothing but lists. Another example is the union of pairs and triples. The interface is very similar to *List* from the O'Caml standard library, but the *Tuple.Poly* signature allows a more fine grained control of arities. The latter provides typesafe linking of models, targets and topologies.

```
\begin{array}{ll} \text{module type } \mathit{Mono} &= \\ \text{sig} \\ \text{type } \alpha \ t \end{array}
```

The size of the tuple, i.e. arity(a1, a2, a3) = 3.

```
val arity : \alpha t \rightarrow int
```

The maximum size of tuples supported by the module. A negative value means that there is no limit. In this case the functions *power_fold* may raise the exception *No_termination*.

```
\begin{array}{l} \text{val } max\_arity \ : \ unit \to \ int \\ \\ \text{val } compare \ : \ (\alpha \to \alpha \to int) \to \alpha \ t \to \alpha \ t \to int \\ \\ \text{val } for\_all \ : \ (\alpha \to bool) \to \alpha \ t \to bool \\ \\ \text{val } map \ : \ (\alpha \to \beta) \to \alpha \ t \to \beta \ t \\ \\ \text{val } iter \ : \ (\alpha \to unit) \to \alpha \ t \to unit \\ \\ \text{val } fold\_left \ : \ (\alpha \to \beta \to \alpha) \to \alpha \to \beta \ t \to \alpha \\ \\ \text{val } fold\_right \ : \ (\alpha \to \beta \to \beta) \to \alpha \ t \to \beta \to \beta \\ \end{array}
```

We have applications, where no sensible intial value can be defined:

```
\begin{array}{l} \text{val } fold\_left\_internal \ : \ (\alpha \ \rightarrow \ \alpha \ \rightarrow \ \alpha) \ \rightarrow \ \alpha \ t \ \rightarrow \ \alpha \\ \text{val } fold\_right\_internal \ : \ (\alpha \ \rightarrow \ \alpha \ \rightarrow \ \alpha) \ \rightarrow \ \alpha \ t \ \rightarrow \ \alpha \\ \text{val } map2 \ : \ (\alpha \ \rightarrow \ \beta \ \rightarrow \ \gamma) \ \rightarrow \ \alpha \ t \ \rightarrow \ \beta \ t \ \rightarrow \ \gamma \ t \\ \text{val } split \ : \ (\alpha \ \times \ \beta) \ t \ \rightarrow \ \alpha \ t \ \times \ \beta \ t \end{array}
```

The distributive tensor product expands a tuple of lists into list of tuples, e.g. for binary tuples:

$$product([x_1; x_2], [y_1; y_2]) = [(x_1, y_1); (x_1, y_2); (x_2, y_1); (x_2, y_2)]$$
(2.1)

NB: product_fold is usually much more memory efficient than the combination of product and List.fold_right for large sets.

```
\begin{array}{lll} \mathsf{val} \ \mathit{product} \ : \ \alpha \ \mathit{list} \ t \ \to \ \alpha \ t \ \mathit{list} \\ \mathsf{val} \ \mathit{product\_fold} \ : \ (\alpha \ t \ \to \ \beta \ \to \ \beta) \ \to \ \alpha \ \mathit{list} \ t \ \to \ \beta \ \to \ \beta \end{array}
```

For homogeneous tuples the *power* function could trivially be built from *product*, e.g.:

$$power [x_1; x_2] = product ([x_1; x_2], [x_1; x_2]) = [(x_1, x_1); (x_1, x_2); (x_2, x_1); (x_2, x_2)]$$

$$(2.2)$$

but it is also well defined for polytuples, e.g. for pairs and triples

$$power[x_1; x_2] = product([x_1; x_2], [x_1; x_2]) \cup product([x_1; x_2], [x_1; x_2], [x_1; x_2])$$
(2.3)

For tuples and polytuples with bounded arity, the *power* and *power_fold* functions terminate. In polytuples with unbounded arity, the the *power* function raises *No_termination* unless a limit is given by ?truncate. power_fold also raises *No_termination*, but could be changed to run until the argument function raises an exception. However, if we need this behaviour, we should probably implement power_iter instead.

```
val power: ?truncate:int \rightarrow \alpha \ list \rightarrow \alpha \ t \ list val power\_fold: ?truncate:int \rightarrow (\alpha \ t \rightarrow \beta \rightarrow \beta) \rightarrow \alpha \ list \rightarrow \beta \rightarrow \beta
```

We can also identify all (poly)tuples with permuted elements and return only one representative, e.g.:

$$sym_{-}power[x_1; x_2] = [(x_1, x_1); (x_1, x_2); (x_2, x_2)]$$
(2.4)

NB: this function has not yet been implemented, because O'Mega only needs the more efficient special case $graded_sym_power$.

If a set X is graded (i.e. there is a map $\phi: X \to \mathbb{N}$, called rank below), the results of power or sym_power can canonically be filtered by requiring that the sum of the ranks in each (poly)tuple has one chosen value. Implementing such a function directly is much more efficient than constructing and subsequently disregarding many (poly)tuples. The elements of rank n are at offset (n-1) in the array. The array is assumed to be immutable, even if O'Caml doesn't support immutable arrays. NB: $graded_power$ has not yet been implemented, because O'Mega only needs $graded_sym_power$.

We hope to be able to avoid the next one in the long run, because it mildly breaks typesafety for arities.

The next one is only used for Fermi statistics in the obsolescent Fusion_vintage module below, but can not

```
type \alpha graded = \alpha list array val graded\_sym\_power : int \rightarrow \alpha graded \rightarrow \alpha t list val graded\_sym\_power\_fold : int \rightarrow (\alpha \ t \rightarrow \beta \rightarrow \beta) \rightarrow \alpha graded \rightarrow \beta \rightarrow \beta
```

Y Unfortunately, we're still working on it ...

```
val to\_list : \alpha t \rightarrow \alpha list
```

 $\stackrel{\searrow}{\perp}$ be implemented if there are no binary tuples. It must be retired as soon as possible.

```
val of 2 _ kludge : \alpha \rightarrow \alpha \rightarrow \alpha t
   end
module type Poly =
   sig
      include Mono
      exception Mismatched_arity
      exception No\_termination
   end
module type Binary =
         include Poly (* should become Mono! *)
         val of 2: \alpha \rightarrow \alpha \rightarrow \alpha t
      end
module Binary : Binary
module type Ternary =
      sig
         include Mono
         \mathsf{val}\ \mathit{of3}\ :\ \alpha\ \to\ \alpha\ \to\ \alpha\ \to\ \alpha\ t
module Ternary : Ternary
type \alpha pair\_or\_triple = T2 of \alpha \times \alpha \mid T3 of \alpha \times \alpha \times \alpha
module type Mixed23 =
      sig
         include Poly
         \mathsf{val}\ \mathit{of2}\ :\ \alpha\ \to\ \alpha\ \to\ \alpha\ t
         val of 3: \alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha t
```

```
end
module Mixed23 : Mixed23
module type Nary =
         include Poly
         val of 2: \alpha \rightarrow \alpha \rightarrow \alpha t
         val of 3: \alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha t
         val of\_list : \alpha list \rightarrow \alpha t
module Unbounded\_Nary : Nary
```



 \triangleright It seemed like a good idea, but hardcoding max_arity here prevents optimizations for processes with fewer external particles than max_arity . For $max_arity \ge 8$ things become bad! Need to implement a truncating version of power and power_fold.

```
module type Bound = \text{sig val } max\_arity : unit \rightarrow int \text{ end}
Module Nary (B: Bound) : Nary
```



```
For compleneteness sake, we could add most of the List signature
      ullet val length: \alpha t \rightarrow int
     \bullet val hd : \alpha t \rightarrow \alpha
     • val nth : \alpha t \rightarrow int \rightarrow \alpha
     ullet val rev : \alpha \ t \ 
ightarrow \ \alpha \ t
     • val rev\_map : (\alpha \rightarrow \beta) \rightarrow \alpha t \rightarrow \beta t
     • val iter2: (\alpha \rightarrow \beta \rightarrow unit) \rightarrow \alpha t \rightarrow \beta t \rightarrow unit
     • val rev\_map2 : (\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow \alpha t \rightarrow \beta t \rightarrow \gamma t
     • val fold\_left2 : (\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \alpha) \rightarrow \alpha \rightarrow \beta \ t \rightarrow \gamma \ t \rightarrow \alpha
     • val fold\_right2 : (\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \gamma) \rightarrow \alpha t \rightarrow \beta t \rightarrow \gamma \rightarrow \gamma
     ullet val exists: (lpha 
ightarrow bool) 
ightarrow lpha t 
ightarrow bool
     ullet val for\_all2 : (lpha 
ightarrow eta 
ightarrow bool) 
ightarrow lpha t 
ightarrow eta t 
ightarrow bool
     ullet val exists2 : (\alpha 
ightarrow eta 
ightarrow bool) 
ightarrow lpha t 
ightarrow eta t 
ightarrow bool
     ullet val mem : \alpha \rightarrow \alpha \ t \rightarrow bool
     • val memq: \alpha \rightarrow \alpha \ t \rightarrow bool
     • val find : (\alpha \rightarrow bool) \rightarrow \alpha t \rightarrow \alpha
     • val find\_all : (\alpha \rightarrow bool) \rightarrow \alpha t \rightarrow \alpha list
     • val assoc : \alpha \rightarrow (\alpha \times \beta) t \rightarrow \beta
     ullet val assq: \alpha \rightarrow (\alpha \times \beta) \ t \rightarrow \beta
     • val mem\_assoc : \alpha \rightarrow (\alpha \times \beta) t \rightarrow bool
     • val mem\_assq: \alpha \rightarrow (\alpha \times \beta) t \rightarrow bool
```

but only if we ever have too much time on our hand ...

• val $stable_sort$: $(\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha t \rightarrow \alpha t$

• val $combine : \alpha t \rightarrow \beta t \rightarrow (\alpha \times \beta) t$ • val sort : $(\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha t \rightarrow \alpha t$

2.2 Implementation of Tuple

```
\mathsf{module}\ \mathsf{type}\ \mathit{Mono}\ =
    sig
        {\rm type} \,\, \alpha \,\, t
        val arity : \alpha t \rightarrow int
        val max\_arity : unit \rightarrow int
```

```
val compare : (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha t \rightarrow \alpha t \rightarrow int
         val\ for\_all\ :\ (\alpha \rightarrow bool)\ \rightarrow\ \alpha\ t\ \rightarrow\ bool
         \mathsf{val}\ map\ :\ (\alpha\ \to\ \beta)\ \to\ \alpha\ t\ \to\ \beta\ t
         \mathsf{val}\ iter\ :\ (\alpha\ \to\ unit)\ \to\ \alpha\ t\ \to\ unit
         \mathsf{val}\; \mathit{fold\_left}\; :\; (\alpha\;\rightarrow\;\beta\;\rightarrow\;\alpha)\;\rightarrow\;\alpha\;\rightarrow\;\beta\;t\;\rightarrow\;\alpha
         \mathsf{val}\; fold\_right\;:\; (\alpha\;\rightarrow\;\beta\;\rightarrow\;\beta)\;\rightarrow\;\alpha\;t\;\rightarrow\;\beta\;\rightarrow\;\beta
         \mathsf{val}\; fold\_left\_internal\;:\; (\alpha\;\rightarrow\;\alpha\;\rightarrow\;\alpha)\;\rightarrow\;\alpha\;t\;\rightarrow\;\alpha
         \mathsf{val}\; fold\_right\_internal\;:\; (\alpha\;\rightarrow\;\alpha\;\rightarrow\;\alpha)\;\rightarrow\;\alpha\;t\;\rightarrow\;\alpha
         val map2 : (\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow \alpha t \rightarrow \beta t \rightarrow \gamma t
         \mathsf{val}\ \mathit{split}\ :\ (\alpha\ \times\ \beta)\ t\ \to\ \alpha\ t\ \times\ \beta\ t
         val product : \alpha \ list \ t \rightarrow \alpha \ t \ list
         val product\_fold : (\alpha\ t\ \to\ \beta\ \to\ \beta)\ \to\ \alpha\ list\ t\ \to\ \beta\ \to\ \beta
         \mathsf{val}\ power\ :\ ?truncate: int \rightarrow\ \alpha\ list \rightarrow\ \alpha\ t\ list
         \textit{val power\_fold} \; : \; ?truncate : int \rightarrow \; (\alpha \; t \; \rightarrow \; \beta \; \rightarrow \; \beta) \; \rightarrow \; \alpha \; list \rightarrow \; \beta \; \rightarrow \; \beta
         type \alpha graded = \alpha list array
         val\ graded\_sym\_power: int \rightarrow \ \alpha\ graded \rightarrow \ \alpha\ t\ list
         val graded\_sym\_power\_fold: int \rightarrow (\alpha \ t \rightarrow \beta \rightarrow \beta) \rightarrow \alpha \ graded \rightarrow
             \beta \rightarrow \beta
         \mathsf{val}\ to\_\mathit{list}\ :\ \alpha\ t\ \to\ \alpha\ \mathit{list}
         val of2_kludge : \alpha \rightarrow \alpha \rightarrow \alpha t
module type Poly =
    sig
         include Mono
         exception Mismatched_arity
         exception No\_termination
    end
```

2.2.1 Typesafe Combinatorics

Wrap the combinatorical functions with varying arities into typesafe functions with fixed arities. We could provide specialized implementations, but since we *know* that *Impossible* is *never* raised, the present approach is just as good (except for a tiny inefficiency).

```
exception Impossible of string
let impossible name = raise (Impossible name)
let choose2 set =
  List.map (function [x;\ y] \rightarrow (x,\ y) \mid \_ \rightarrow impossible "choose2")
       (Combinatorics.choose 2 set)
let choose3 set =
  List.map (function [x; y; z] \rightarrow (x, y, z) \mid \_ \rightarrow impossible "choose3")
     (Combinatorics.choose 3 set)
                                                       2.2.2 Pairs
module type Binary =
     sig
       include Poly (* should become Mono! *)
       \mathsf{val}\ \mathit{of2}\ :\ \alpha\ \to\ \alpha\ \to\ \alpha\ t
     end
module Binary =
  struct
     type \alpha t = \alpha \times \alpha
     let arity _{-} = 2
     let max\_arity() = 2
     let of 2 x y = (x, y)
```

```
let compare cmp (x1, y1) (x2, y2) =
       let cx = cmp x1 x2 in
       if cx \neq 0 then
          cx
       else
          cmp \ y1 \ y2
     let for\_all p (x, y) = p x \land p y
     let map f (x, y) = (f x, f y)
     let iter f(x, y) = f x; f y
     let fold\_left\ f\ init\ (x,\ y)\ =\ f\ (f\ init\ x)\ y
     let fold_right\ f\ (x,\ y)\ init\ =\ f\ x\ (f\ y\ init)
     \mathsf{let}\ \mathit{fold\_left\_internal}\ f\ (x,\ y)\ =\ f\ x\ y
     let fold\_right\_internal\ f\ (x,\ y)\ =\ f\ x\ y
     exception Mismatched_arity
     let map2 f(x1, y1) (x2, y2) = (f x1 x2, f y1 y2)
     let split((x1, x2), (y1, y2)) = ((x1, y1), (x2, y2))
     let product (lx, ly) =
        Product.list2 (fun x y \rightarrow (x, y)) lx ly
     let product\_fold\ f\ (lx,\ ly)\ init\ =
        Product.fold2 (fun x y \rightarrow f(x, y)) lx ly init
     let power?truncate l =
       match truncate with
          None \rightarrow product (l, l)
        \mid Some \ n \rightarrow
            if n \geq 2 then
               product (l, l)
            else
               invalid_arg "Tuple.Binary.power: utruncate < 2"
     let power\_fold?truncate f l =
       match truncate with
         None \rightarrow product\_fold f(l, l)
        \mid Some \ n \rightarrow
            if n > 2 then
               product\_fold\ f\ (l,\ l)
            else
               invalid\_arg "Tuple.Binary.power_fold: _{\sqcup}truncate _{\sqcup}<_{\sqcup}2"
In the special case of binary fusions, the implementation is very concise.
     type \alpha graded = \alpha list array
     let fuse2 \ f \ set \ (i, \ j) \ acc =
       if i = j then
          List.fold\_right (fun (x, y) \rightarrow f x y) (choose2 set.(pred i)) acc
       else
          Product.fold2\ f\ set.(pred\ i)\ set.(pred\ j)\ acc
     {\tt let} \ graded\_sym\_power\_fold \ rank \ f \ set \ acc \ =
       let max\_rank = Array.length set in
       List.fold\_right (fuse2 (fun x y \rightarrow f (of2 x y)) set)
          (Partition.pairs rank 1 max_rank) acc
     let graded_sym_power rank set =
       graded\_sym\_power\_fold\ rank\ (fun\ pair\ acc\ 	o \ pair\ ::\ acc)\ set\ [\ ]
     let to\_list (x, y) = [x; y]
     let of2_kludge = of2
     exception No\_termination
  end
```

2.2.3 Triples

```
module type Ternary =
     sig
         include Mono
         val of 3: \alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha t
     end
module Ternary =
  struct
     \mathsf{type}\;\alpha\;t\;=\;\alpha\;\times\;\alpha\;\times\;\alpha
     \mathsf{let}\ \mathit{arity}\ {\tt \_}\ =\ 3
     let max_arity() = 3
     let of 3 x y z = (x, y, z)
     let compare cmp (x1, y1, z1) (x2, y2, z2) =
         let cx = cmp x1 x2 in
         if cx \neq 0 then
            cx
         else
            let cy = cmp y1 y2 in
            if cy \neq 0 then
               cy
            else
               cmp z1 z2
     \mathsf{let}\; for\_all\; p\; (x,\; y,\; z) \; = \; p\; x\; \wedge\; p\; y\; \wedge\; p\; z
     let map f (x, y, z) = (f x, f y, f z)
     let iter f(x, y, z) = f x; f y; f z
     let fold\_left\ f\ init\ (x,\ y,\ z)\ =\ f\ (f\ (f\ init\ x)\ y)\ z
     let fold\_right f (x, y, z) init = f x (f y (f z init))
     let fold\_left\_internal\ f\ (x,\ y,\ z)\ =\ f\ (f\ x\ y)\ z
     let fold\_right\_internal f (x, y, z) = f x (f y z)
      exception Mismatched\_arity
     \mathsf{let}\ \mathit{map2}\ f\ (\mathit{x1},\ \mathit{y1},\ \mathit{z1})\ (\mathit{x2},\ \mathit{y2},\ \mathit{z2})\ =\ (\mathit{f}\ \mathit{x1}\ \mathit{x2},\ \mathit{f}\ \mathit{y1}\ \mathit{y2},\ \mathit{f}\ \mathit{z1}\ \mathit{z2})
     let split((x1, x2), (y1, y2), (z1, z2)) = ((x1, y1, z1), (x2, y2, z2))
     let product (lx, ly, lz) =
         Product.list3 (fun x y z \rightarrow (x, y, z)) lx ly lz
     let product\_fold\ f\ (lx,\ ly,\ lz)\ init\ =
         Product.fold3 (fun x y z \rightarrow f(x, y, z)) lx ly lz init
     let power?truncate l =
         match truncate with
           None \rightarrow product (l, l, l)
         \mid Some \ n \rightarrow
               if n \geq 3 then
                  product (l, l, l)
                  invalid\_arg "Tuple.Ternary.power: _{\square}truncate _{\square}<_{\square}3"
     let power\_fold ?truncate f l =
         match truncate with
           None \rightarrow product\_fold f (l, l, l)
           Some \ n \rightarrow
               if n \geq 3 then
                  product\_fold\ f\ (l,\ l,\ l)
               else
                  invalid\_arg "Tuple.Ternary.power_fold:_{\sqcup}truncate_{\sqcup}<_{\sqcup}3"
     type \alpha graded = \alpha list array
```

```
let fuse3 \ f \ set \ (i, j, k) \ acc =
        if i = j then begin
           if j = k then
              List.fold\_right (fun (x, y, z) \rightarrow f x y z) (choose3 set.(pred i)) acc
           else
              Product.fold2 \text{ (fun } (x, y) z \rightarrow f x y z)
                 (choose2\ set.(pred\ i))\ set.(pred\ k)\ acc
        end else begin
           if j = k then
              Product.fold2 \text{ (fun } x \text{ } (y, z) \rightarrow f \text{ } x \text{ } y \text{ } z)
                 set.(pred\ i)\ (choose2\ set.(pred\ j))\ acc
           else
              Product.fold3 \text{ (fun } x \text{ } y \text{ } z \text{ } \rightarrow \text{ } f \text{ } x \text{ } y \text{ } z)
                 set.(pred\ i)\ set.(pred\ j)\ set.(pred\ k)\ acc
        end
     let graded\_sym\_power\_fold rank f set acc =
        let max\_rank = Array.length set in
        List.fold\_right (fuse3 (fun x y z \rightarrow f (of3 x y z)) set)
           (Partition.triples rank 1 max_rank) acc
     let graded_sym_power rank set =
         graded\_sym\_power\_fold\ rank\ (fun\ pair\ acc\ 	o \ pair\ ::\ acc)\ set\ [\ ]
     let to\_list(x, y, z) = [x; y; z]
     let of2_kludge _ = failwith "Tuple.Ternary.of2_kludge"
  end
                                                   2.2.4 Pairs and Triples
type \alpha pair\_or\_triple = T2 of \alpha \times \alpha \mid T3 of \alpha \times \alpha \times \alpha
module type Mixed23 =
     sig
        include Poly
        \mathsf{val}\ \mathit{of2}\ :\ \alpha\ \to\ \alpha\ \to\ \alpha\ t
        val of 3: \alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha t
module Mixed23 =
  struct
     type \alpha t = \alpha pair\_or\_triple
     let arity = function
        | \quad T2 \ \_ \ \rightarrow \ 2
         |T3\_\rightarrow 3
     let max\_arity() = 3
     let of2 x y = T2 (x, y)
     let of3 x y z = T3 (x, y, z)
     let compare \ cmp \ m1 \ m2 =
        match m1, m2 with
           T2 _, T3 _ \rightarrow -1
           T3 _, T2 _ \rightarrow 1
         T2 (x1, y1), T2 (x2, y2) \rightarrow
              let cx = cmp x1 x2 in
              if cx \neq 0 then
                 cx
              else
                 cmp \ y1 \ y2
        \mid T3 (x1, y1, z1), T3 (x2, y2, z2) \rightarrow
              let cx = cmp x1 x2 in
```

```
if cx \neq 0 then
           cx
        else
           let cy = cmp y1 y2 in
          if cy \neq 0 then
             cy
           else
              cmp \ z1 \ z2
let for\_all p = function
     T2(x, y) \rightarrow p x \wedge p y
     T3 \ (x, \ y, \ z) \ \rightarrow \ p \ x \ \wedge \ p \ y \ \wedge \ p \ z
let map f = function
    T2 (x, y) \rightarrow T2 (f x, f y)
   | T3 (x, y, z) \rightarrow T3 (f x, f y, f z)|
let iter f = function
   T2(x, y) \rightarrow f x; f y
   T3(x, y, z) \rightarrow f x; f y; f z
let fold\_left f init = function
  \mid T2(x, y) \rightarrow f(f init x) y
   T3 (x, y, z) \rightarrow f (f (f init x) y) z
let fold_right f m init =
  \mathsf{match}\ m with
     T2(x, y) \rightarrow f x (f y init)
     T3(x, y, z) \rightarrow f x (f y (f z init))
let fold\_left\_internal f m =
  match \ m \ with
     T2(x, y) \rightarrow f x y
   T3 (x, y, z) \rightarrow f (f x y) z
let fold\_right\_internal f m =
  match m with
     T2(x, y) \rightarrow f x y
   | T3 (x, y, z) \rightarrow f x (f y z)
exception Mismatched_arity
let map2 f m1 m2 =
  \mathsf{match}\ m1,\ m2\ \mathsf{with}
     T2 (x1, y1), T2 (x2, y2) \rightarrow T2 (f x1 x2, f y1 y2)
     T3\ (x1,\ y1,\ z1),\ T3\ (x2,\ y2,\ z2)\ \to\ T3\ (f\ x1\ x2,\ f\ y1\ y2,\ f\ z1\ z2)
   \mid T2 -, T3 - \mid T3 -, T2 - \rightarrow raise\ Mismatched - arity
let split = function
     T2\ ((x1,\ x2),\ (y1,\ y2))\ \to\ (T2\ (x1,\ y1),\ T2\ (x2,\ y2))
    T3\ ((x1,\ x2),\ (y1,\ y2),\ (z1,\ z2))\ \to\ (T3\ (x1,\ y1,\ z1),\ T3\ (x2,\ y2,\ z2))
let product = function
     T2\ (lx,\ ly)\ 	o\ Product.list2\ (fun\ x\ y\ 	o\ T2\ (x,\ y))\ lx\ ly
   \mid T3 \ (lx, \ ly, \ lz) \rightarrow Product.list3 \ (fun \ x \ y \ z \rightarrow T3 \ (x, \ y, \ z)) \ lx \ ly \ lz
let product\_fold\ f\ m\ init\ =
  \mathsf{match}\ m\ \mathsf{with}
     T2\ (lx,\ ly)\ 	o\ Product.fold2\ (fun\ x\ y\ 	o\ f\ (T2\ (x,\ y)))\ lx\ ly\ init
     T3 (lx, ly, lz) \rightarrow
        Product.fold3 (fun x y z \rightarrow f (T3(x, y, z))) lx ly lz init
exception No\_termination
let power\_fold23 \ f \ l \ init =
  product\_fold\ f\ (T2\ (l,\ l))\ (product\_fold\ f\ (T3\ (l,\ l,\ l))\ init)
let power_fold2 f l init =
  product\_fold\ f\ (T2\ (l,\ l))\ init
```

```
let power_fold ?truncate f l init =
        match truncate with
          None \rightarrow power\_fold23 \ f \ l \ init
          Some \ n \rightarrow
             if n \geq 3 then
                power_fold23 f l init
             else if n = 2 then
                power_fold2 f l init
                invalid\_arg "Tuple.Mixed23.power_fold: _\truncate_\<\_2"
     let power ?truncate l =
        power\_fold\ ?truncate\ (fun\ m\ acc\ 	o\ m\ ::\ acc)\ l\ []
     type \alpha graded = \alpha list array
     let graded_sym_power_fold rank f set acc =
        let max\_rank = Array.length set in
        List.fold\_right (Binary.fuse2 (fun x y \rightarrow f (of2 x y)) set)
           (Partition.pairs rank 1 max_rank)
           (List.fold_right (Ternary.fuse3 (fun x y z \rightarrow f (of3 x y z)) set)
               (Partition.triples rank 1 max_rank) acc)
     let graded_sym_power rank set =
        graded\_sym\_power\_fold\ rank\ (fun\ pair\ acc\ 	o \ pair\ ::\ acc)\ set\ [\ ]
     let to\_list = function
          T2(x, y) \rightarrow [x; y]
          T3 (x, y, z) \rightarrow [x; y; z]
     let of2\_kludge = of2
  end
                                             2.2.5 ... and All The Rest
module type Nary =
     sig
        include Poly
        \mathsf{val}\ \mathit{of2}\ :\ \alpha\ \to\ \alpha\ \to\ \alpha\ t
        val of 3: \alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha t
        \mathsf{val}\ of\_list\ :\ \alpha\ list\ \to\ \alpha\ t
module Nary\ (A : sig\ val\ max\_arity : unit \rightarrow int\ end) =
  struct
     type \alpha t = \alpha \times \alpha \textit{ list}
     let arity(_-, y) = succ(List.length y)
     let max\_arity() =
       try A.max\_arity () with \_ \rightarrow -1
     let of 2 x y = (x, [y])
     let of 3 x y z = (x, [y; z])
     let of_list = function
        | x :: y \rightarrow (x, y)
        [] \rightarrow invalid\_arg "Tuple.Nary.of_list:\( \text{\left} = mpty \)"
     let compare cmp (x1, y1) (x2, y2) =
        \mathsf{let}\ c\ =\ cmp\ x1\ x2\ \mathsf{in}
        if c \neq 0 then
          c
        else
           ThoList.compare ~cmp y1 y2
```

```
let for\_all\ p\ (x,\ y)\ =\ p\ x\ \land\ List.for\_all\ p\ y
let map f (x, y) = (f x, List.map f y)
let iter f (x, y) = f x; List.iter f y
let fold\_left\ f\ init\ (x,\ y)\ =\ List.fold\_left\ f\ (f\ init\ x)\ y
let fold\_right\ f\ (x,\ y)\ init\ =\ f\ x\ (List.fold\_right\ f\ y\ init)
let fold\_left\_internal f (x, y) = List.fold\_left f x y
let fold\_right\_internal\ f\ (x,\ y) =
  match List.rev y with
   [] \rightarrow x
   | y\theta :: y\_sans\_y\theta \rightarrow
       f \ x \ (List.fold\_right \ f \ (List.rev \ y\_sans\_y\theta) \ y\theta)
exception Mismatched_arity
let map2 \ f \ (x1, \ y1) \ (x2, \ y2) =
  try (f \ x1 \ x2, \ List.map2 \ f \ y1 \ y2) with
  | Invalid\_argument \_ \rightarrow raise Mismatched\_arity
let split((x1, x2), y12) =
  let y1, y2 = List.split y12 in
  ((x1, y1), (x2, y2))
let product(xl, yl) =
   Product.list (function
     | x :: y \rightarrow (x, y)
     [] \rightarrow failwith "Tuple.Nary.product") (xl :: yl)
let product\_fold\ f\ (xl,\ yl)\ init\ =
   Product.fold (function
     | x :: y \rightarrow f(x, y)
     [] 
ightarrow failwith "Tuple.Nary.product_fold") (xl :: yl) init
exception No\_termination
let truncated\_arity ?truncate() =
  let ma = max\_arity () in
  match truncate with
     None \rightarrow ma
   \mid Some \ n \rightarrow
       \quad \text{if } n \ < \ 2 \ \text{then} \\
          invalid\_arg "Tuple.Nary.power: utruncate \le 2"
       else if ma \geq 2 then
          min n ma
       else
let power_fold ?truncate f l init =
  let ma = truncated\_arity ?truncate () in
  if ma > 0 then
     List.fold_right
        (\text{fun } n \rightarrow product\_fold f (l, ThoList.clone l (pred n)))
        (ThoList.range\ 2\ ma)\ init
  else
     raise No_termination
let power?truncate l =
  power\_fold\ ?truncate\ (fun\ t\ acc\ 	o\ t\ ::\ acc)\ l\ []
type \alpha graded = \alpha list array
let fuse\_n \ f \ set \ partition \ acc =
  let choose(n, r) =
     n \ r \ (List.length \ set.(pred \ r));
     Combinatorics.choose \ n \ set.(pred \ r) in
   Product.fold (fun \ wfs \rightarrow f \ (List.concat \ wfs))
     (List.map choose (ThoList.classify partition)) acc
```

```
\begin{array}{lll} \text{let } \mathit{fuse\_n} \ f \ \mathit{set partition} \ \mathit{acc} &= \\ & \text{let } \mathit{choose} \ (n, \ r) \ = \ \mathit{Combinatorics.choose} \ n \ \mathit{set.(pred} \ r) \ \mathsf{in} \\ & \mathit{Product.fold} \ (\mathsf{fun} \ \mathit{wfs} \ \rightarrow \ f \ (\mathit{List.concat} \ \mathit{wfs})) \\ & (\mathit{List.map} \ \mathit{choose} \ (\mathit{ThoList.classify} \ \mathit{partition})) \ \mathit{acc} \end{array}
```



graded_sym_power_fold is well defined for unbounded arities as well: derive a reasonable replacement from set. The length of the flattened set is an upper limit, of course, but too pessimistic in most cases.

```
let graded\_sym\_power\_fold\ rank\ f\ set\ acc\ =
let max\_rank\ =\ Array.length\ set in
let degrees\ =\ ThoList.range\ 2\ (max\_arity\ ()) in
let partitions\ =
ThoList.flatmap
(fun\ deg\ \to\ Partition.tuples\ deg\ rank\ 1\ max\_rank)\ degrees\ in
List.fold\_right\ (fuse\_n\ (fun\ wfs\ \to\ f\ (of\_list\ wfs))\ set)\ partitions\ acc
let graded\_sym\_power\ rank\ set\ =
graded\_sym\_power\_fold\ rank\ (fun\ pair\ acc\ \to\ pair\ ::\ acc)\ set\ []
let to\_list\ (x,\ y)\ =\ x\ ::\ y
let of2\_kludge\ =\ of2
end
module type Bound\ =\ sig\ val\ max\_arity\ :\ unit\ \to\ int\ end
module Unbounded\_Nary\ =\ Nary\ (struct\ let\ max\_arity\ ()\ =\ -1\ end)
```

__3__

TOPOLOGIES

3.1 Interface of Topology

 $\begin{array}{ccc} \mathsf{module} \ \mathsf{type} \ T \ = \\ \mathsf{sig} \end{array}$

partition is a collection of integers, with arity one larger than the arity of α children below. These arities can one fixed number corresponding to homogeneous tuples or a collection of tupes or lists.

type partition

partitions n returns the union of all $[n_1; n_2; \ldots; n_d]$ with $1 \le n_1 \le n_2 \le \ldots \le n_d \le \lfloor n/2 \rfloor$ and

$$\sum_{i=1}^{d} n_i = n \tag{3.1}$$

for d from 3 to d_{\max} , where d_{\max} is a fixed number for each module implementating T. In particular, if type $partition = int \times int \times int$, then partitions n returns all (n_1, n_2, n_3) with $n_1 \le n_2 \le n_3$ and $n_1 + n_2 + n_3 = n_3$.

 $val partitions : int \rightarrow partition list$

A (poly)tuple as implemented by the modules in *Tuple*:

type α children

 $keystones\ externals\ returns\ all\ keystones\ for\ the\ amplitude\ with\ external\ states\ externals\ in\ the\ vanilla\ scalar\ theory\ with\ a$

$$\sum_{3 \le k \le d_{\text{max}}} \lambda_k \phi^k \tag{3.2}$$

interaction. One factor of the products is factorized. In particular, if

type α children = α Tuple.Binary.t = $\alpha \times \alpha$,

then keystones externals returns all keystones for the amplitude with external states externals in the vanilla scalar $\lambda \phi^3$ -theory.

val keystones : α list \rightarrow (α list \times α list children list) list

The maximal depth of subtrees for a given number of external lines.

 $\mathsf{val}\ max_subtree\ :\ int \to\ int$

Only for diagnostics:

 $\mbox{ val } inspect_partition \ : \ partition \ \rightarrow \ int \ list \ \mbox{end} \ \label{eq:partition}$

module Binary: T with type α $children = \alpha$ Tuple.Binary.t module Ternary: T with type α $children = \alpha$ Tuple.Ternary.t module Mixed23: T with type α $children = \alpha$ Tuple.Mixed23.t module Nary: functor $(B: Tuple.Bound) \rightarrow (T \text{ with type } \alpha \text{ } children = \alpha \text{ } Tuple.Nary(B).t)$

3.1.1 Diagnostics: Counting Diagrams and Factorizations for $\sum_{n} \lambda_n \phi^n$

The number of diagrams for many particles can easily exceed the range of native integers. Even if we can not calculate the corresponding amplitudes, we want to check combinatorical factors. Therefore we code a functor that can use arbitray implementations of integers.

```
module type Integer =
   sig
       type t
       val zero : t
       val one : t
       \mathsf{val}\;(\;+\;)\;:\;t\;\to\;t\;\to\;t
       \mathsf{val}\;(\;-\;)\;:\;t\;\to\;t\;\to\;t
       \mathsf{val}\;(\;\times\;)\;:\;t\;\to\;t\;\to\;t
       \mathsf{val}(\ /\ ) : t \to t \to t
       \mathsf{val}\ pred\ :\ t\ 	o\ t
       \mathsf{val}\ succ\ :\ t\ 	o\ t
       val (=): t \rightarrow t \rightarrow bool
       \mathsf{val}\;(\neq)\;:\;t\;\rightarrow\;t\;\rightarrow\;bool
       \mathsf{val} \; ( \; < \; ) \; : \; t \; \rightarrow \; t \; \rightarrow \; bool
       \mathsf{val}\;(\;\leq\;)\;:\;t\;\to\;t\;\to\;
       \mathsf{val} \; (\ >\ ) \; : \; t \; 	o \; bool
       \mathsf{val}\;(\;\geq\;)\;:\;t\;\to\;t\;\to\;
       val of_int : int \rightarrow t
       val to_int : t \rightarrow int
       val to\_string : t \rightarrow string
       val\ compare\ :\ t\ 
ightarrow\ t\ 
ightarrow\ int
       \mathsf{val}\ factorial\ :\ t\ 	o\ t
   end
```

Of course, native integers will provide the fastest implementation:

```
\begin{array}{lll} \text{module } Int \ : \ Integer \\ \\ \text{module type } Count \ = \\ \\ \text{sig} \\ \\ \text{type } integer \end{array}
```

 $diagrams\ f\ d\ n$ returns the number of tree diagrams contributing to the n-point amplitude in vanilla scalar theory with

$$\sum_{3 \le k \le d \land f(k)} \lambda_k \phi^k \tag{3.3}$$

interaction. The default value of f returns true for all arguments.

```
val diagrams: ?f: (integer \rightarrow bool) \rightarrow integer \rightarrow integer \rightarrow integer val diagrams\_via\_keystones: integer \rightarrow integer \rightarrow integer
```

$$\frac{1}{S(n_k, n - n_k)} \frac{1}{S(n_1, n_2, \dots, n_k)} \binom{n_1 + n_2 + \dots + n_k}{n_1, n_2, \dots, n_k}$$
(3.4)

 $\mathsf{val}\ keystones\ :\ integer\ list \to\ integer$

 $diagrams_via_keystones\ d\ n$ must produce the same results as $diagrams\ d\ n$. This is shown explicitly in tables 3.2, 3.3 and 3.4 for small values of d and n. The test program in appendix S can be used to verify this relation for larger values.

```
\mbox{val } diagrams\_per\_keystone \ : \ integer \ \to \ integer \ list \to \ integer end \mbox{module } Count \ : \ \mbox{functor} \ (I \ : \ Integer) \ \to \ Count \ \mbox{with type } integer \ = \ I.t
```

```
partitions n
 4
     (1,1,2)
     (1,2,2)
 5
6
     (1,2,3), (2,2,2)
7
     (1,3,3), (2,2,3)
     (1,3,4), (2,2,4), (2,3,3)
8
9
     (1,4,4), (2,3,4), (3,3,3)
10
     (1,4,5), (2,3,5), (2,4,4), (3,3,4)
11
     (1,5,5), (2,4,5), (3,3,5), (3,4,4)
12
     (1,5,6), (2,4,6), (2,5,5), (3,3,6), (3,4,5), (4,4,4)
     (1,6,6), (2,5,6), (3,4,6), (3,5,5), (4,4,5)
     (1,6,7), (2,5,7), (2,6,6), (3,4,7), (3,5,6), (4,4,6), (4,5,5)
15
     (1,7,7), (2,6,7), (3,5,7), (3,6,6), (4,4,7), (4,5,6), (5,5,5)
16
    (1,7,8), (2,6,8), (2,7,7), (3,5,8), (3,6,7), (4,4,8), (4,5,7), (4,6,6), (5,5,6)
```

Table 3.1: partitions n for moderate values of n.

3.1.2 Emulating HELAC

We can also proceed á la [2].

```
module Helac: functor (B: Tuple.Bound) \rightarrow (T \text{ with type } \alpha \text{ } children = \alpha \text{ } Tuple.Nary(B).t)
```



The following has never been tested, but it is no rocket science and should work anyway \dots

module $Helac_Binary$: T with type α $children = \alpha$ Tuple.Binary.t

3.2 Implementation of Topology

```
module type T=\sup_{\mathbf{sig}} type partition val partitions: int \to partition \ list type \alpha children val keystones: \alpha list \to (\alpha \ list \times \alpha \ list \ children \ list) list val max\_subtree: int \to int val inspect\_partition: partition \to int \ list end
```

3.2.1 Factorizing Diagrams for ϕ^3

```
\begin{array}{lll} \text{module } Binary &= \\ \text{struct} & \\ \text{type } partition &= int \times int \times int \\ \text{let } inspect\_partition \ (n1, \ n2, \ n3) \ = \ [n1; \ n2; \ n3] \end{array}
```

One way [1] to lift the degeneracy is to select the vertex that is closest to the center (see table 3.1):

```
partitions: n \to \{(n_1, n_2, n_3) \mid n_1 + n_2 + n_3 = n \land n_1 \le n_2 \le n_3 \le \lfloor n/2 \rfloor \} 
(3.5)
```

Other, less symmetric, approaches are possible. The simplest of these is: choose the vertex adjacent to a fixed external line [2]. They will be made available for comparison in the future.

An obvious consequence of $n_1 + n_2 + n_3 = n$ and $n_1 \le n_2 \le n_3$ is $n_1 \le \lfloor n/3 \rfloor$:

```
\begin{array}{ll} \text{let rec } partitions' \ n \ n1 \ = \\ & \text{if } n1 \ > \ n \ / \ 3 \ \text{then} \\ & \quad \  \  \, [\,\,] \\ & \text{else} \end{array}
```

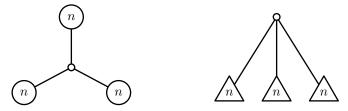


Figure 3.1: Topologies with a blatant three-fold permutation symmetry, if the number of external lines is a multiple of three

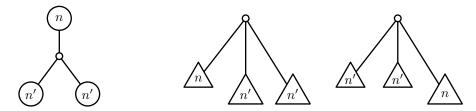


Figure 3.2: Topologies with a blatant two-fold symmetry.

$$List.map \ (\text{fun} \ (n2, \ n3) \ \rightarrow \ (n1, \ n2, \ n3)) \\ (Partition.pairs \ (n \ - \ n1) \ n1 \ (n \ / \ 2)) \ @ \ partitions' \ n \ (succ \ n1)$$
 let $partitions \ n \ = \ partitions' \ n \ 1$

type α children = α Tuple.Binary.t

There remains one peculiar case, when the number of external lines is even and $n_3 = n_1 + n_2$ (cf. figure 3.3). Unfortunately, this reflection symmetry is not respected by the equivalence classes. E. g.

$$\{1\}\{2,3\}\{4,5,6\} \mapsto \{\{4\}\{5,6\}\{1,2,3\};\{5\}\{4,6\}\{1,2,3\};\{6\}\{4,5\}\{1,2,3\}\}$$
 (3.6)

However, these reflections will always exchange the two halves and a representative can be chosen by requiring that one fixed momentum remains in one half. We choose to filter out the half of the partitions where the element p appears in the second half, i.e. the list of length n3.

Finally, a closed expression for the number of Feynman diagrams in the equivalence class (n_1, n_2, n_3) is

$$N(n_1, n_2, n_3) = \frac{(n_1 + n_2 + n_3)!}{S(n_1, n_2, n_3)} \prod_{i=1}^{3} \frac{(2n_i - 3)!!}{n_i!}$$
(3.7)

where the symmetry factor from the above arguments is

$$S(n_1, n_2, n_3) = \begin{cases} 3! & \text{for } n_1 = n_2 = n_3 \\ 2 \cdot 2 & \text{for } n_3 = 2n_1 = 2n_2 \\ 2 & \text{for } n_1 = n_2 \lor n_2 = n_3 \\ 2 & \text{for } n_1 + n_2 = n_3 \end{cases}$$

$$(3.8)$$

Indeed, the sum of all Feynman diagrams

$$\sum_{\substack{n_1+n_2+n_3=n\\1\leq n_1\leq n_2\leq n_3\leq \lfloor n/2\rfloor}} N(n_1, n_2, n_3) = (2n-5)!!$$
(3.9)

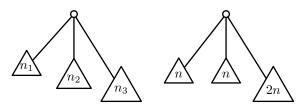


Figure 3.3: If $n_3 = n_1 + n_2$, the apparently asymmetric topologies on the left hand side have a non obvious two-fold symmetry, that exchanges the two halves. Therefore, the topologies on the right hand side have a four fold symmetry.

n	(2n-5)!!	$\sum N(n_1, n_2, n_3)$
4	3	$3\cdot (1,1,2)$
5	15	$15 \cdot (1,2,2)$
6	105	$90 \cdot (1,2,3) + 15 \cdot (2,2,2)$
7	945	$630 \cdot (1,3,3) + 315 \cdot (2,2,3)$
8	10395	$6300 \cdot (1,3,4) + 1575 \cdot (2,2,4) + 2520 \cdot (2,3,3)$
9	135135	$70875 \cdot (1,4,4) + 56700 \cdot (2,3,4) + 7560 \cdot (3,3,3)$
10	2027025	$992250 \cdot (1,4,5) + 396900 \cdot (2,3,5)$
		$+354375 \cdot (2,4,4) + 283500 \cdot (3,3,4)$
11	34459425	$15280650 \cdot (1,5,5) + 10914750 \cdot (2,4,5)$
		$+4365900 \cdot (3,3,5) + 3898125 \cdot (3,4,4)$
12	654729075	$275051700 \cdot (1,5,6) + 98232750 \cdot (2,4,6)$
		$+91683900 \cdot (2,5,5) + 39293100 \cdot (3,3,6)$
		$+ 130977000 \cdot (3, 4, 5) + 19490625 \cdot (4, 4, 4)$

Table 3.2: Equation (3.9) for small values of n.

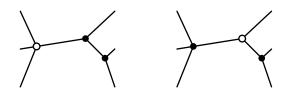


Figure 3.4: Degenerate (1, 1, 1, 3) and (1, 2, 3).

can be checked numerically for large values of $n = n_1 + n_2 + n_3$, verifying the symmetry factor (see table 3.2).



P. M. claims to have seen similar formulae in the context of Young tableaux. That's a good occasion to read the new edition of Howard's book the new edition of Howard's book . . .

Return a list of all inequivalent partitions of the list l in three lists of length n1, n2 and n3, respectively. Common first lists are factored. This is nothing more than a typedafe wrapper around Combinatorics.factorized_keystones.

```
exception Impossible of string
  let \ tuple\_of\_list2 \ = \ function
     |[x1; x2] \rightarrow Tuple.Binary.of2 x1 x2
     | \ \_ \ \rightarrow \ raise \ (Impossible \ \texttt{"Topology.tuple\_of\_list"})
  let keystone (n1, n2, n3) l =
      List.map (fun (p1, p23) \rightarrow (p1, List.rev\_map tuple\_of\_list2 p23))
        (Combinatorics.factorized_keystones [n1; n2; n3] l)
  \mathsf{let}\ \mathit{keystones}\ l\ =
      ThoList.flatmap 	ext{ (fun } n123 	o keystone n123 	ext{ l) } (partitions (List.length 	ext{ l)})
  \mathsf{let}\ max\_subtree\ n\ =\ n\ /\ 2
end
```

3.2.2 Factorizing Diagrams for $\sum_{n} \lambda_n \phi^n$

Mixed ϕ^n adds new degeneracies, as in figure 3.4. They appear if and only if one part takes exactly half of the external lines and can relate central vertices of different arity.

```
module Nary (B : Tuple.Bound) =
  struct
    type partition = int list
    let inspect\_partition p = p
    \mathsf{let}\ partition\ d\ sum\ =
       Partition.tuples d sum 1 (sum / 2)
```

n	\sum	\sum
4	4	$1 \cdot (1, 1, 1, 1) + 3 \cdot (1, 1, 2)$
5	25	$10 \cdot (1, 1, 1, 2) + 15 \cdot (1, 2, 2)$
6	220	$40 \cdot (1,1,1,3) + 45 \cdot (1,1,2,2) + 120 \cdot (1,2,3) + 15 \cdot (2,2,2)$
7	2485	$840 \cdot (1,1,2,3) + 105 \cdot (1,2,2,2) + 1120 \cdot (1,3,3) + 420 \cdot (2,2,3)$
8	34300	$5250 \cdot (1, 1, 2, 4) + 4480 \cdot (1, 1, 3, 3) + 3360 \cdot (1, 2, 2, 3)$
		$+105 \cdot (2,2,2,2) + 14000 \cdot (1,3,4)$
		$+2625 \cdot (2,2,4) + 4480 \cdot (2,3,3)$
9	559405	$126000 \cdot (1, 1, 3, 4) + 47250 \cdot (1, 2, 2, 4) + 40320 \cdot (1, 2, 3, 3)$
		$+5040 \cdot (2,2,2,3) + 196875 \cdot (1,4,4)$
		$+126000 \cdot (2,3,4) + 17920 \cdot (3,3,3)$
10	10525900	$1108800 \cdot (1, 1, 3, 5) + 984375 \cdot (1, 1, 4, 4) + 415800 \cdot (1, 2, 2, 5)$
		$+1260000 \cdot (1,2,3,4) + 179200 \cdot (1,3,3,3) + 78750 \cdot (2,2,2,4)$
		$+100800 \cdot (2,2,3,3) + 3465000 \cdot (1,4,5) + 1108800 \cdot (2,3,5)$
		$+984375 \cdot (2,4,4) + 840000 \cdot (3,3,4)$

Table 3.3: $\mathcal{L} = \lambda_3 \phi^3 + \lambda_4 \phi^4$

```
1 \cdot (1, 1, 1, 1) + 3 \cdot (1, 1, 2)
4
5
                1 \cdot (1, 1, 1, 1, 1) + 10 \cdot (1, 1, 1, 2) + 15 \cdot (1, 2, 2)
6
        236
                1 \cdot (1, 1, 1, 1, 1, 1) + 15 \cdot (1, 1, 1, 1, 2) + 40 \cdot (1, 1, 1, 3)
                    +45 \cdot (1,1,2,2) + 120 \cdot (1,2,3) + 15 \cdot (2,2,2)
7
       2751
                 21 \cdot (1, 1, 1, 1, 1, 2) + 140 \cdot (1, 1, 1, 1, 3) + 105 \cdot (1, 1, 1, 2, 2)
                    +840 \cdot (1,1,2,3) + 105 \cdot (1,2,2,2) + 1120 \cdot (1,3,3) + 420 \cdot (2,2,3)
                 224 \cdot (1,1,1,1,1,3) + 210 \cdot (1,1,1,1,2,2) + 910 \cdot (1,1,1,1,4)
8
     39179
                    +2240 \cdot (1, 1, 1, 2, 3) + 420 \cdot (1, 1, 2, 2, 2) + 5460 \cdot (1, 1, 2, 4)
                    +4480 \cdot (1,1,3,3) + 3360 \cdot (1,2,2,3) + 105 \cdot (2,2,2,2)
                    +14560 \cdot (1,3,4) + 2730 \cdot (2,2,4) + 4480 \cdot (2,3,3)
```

Table 3.4: $\mathcal{L} = \lambda_3 \phi^3 + \lambda_4 \phi^4 + \lambda_5 \phi^5 + \lambda_6 \phi^6$

```
let rec partitions' d sum =
        if d < 3 then
        else
           partition d sum @ partitions' (pred d) sum
     let \ partitions \ sum \ = \ partitions' \ (succ \ (B.max\_arity \ ())) \ sum
     module Tuple = Tuple.Nary(B)
     \mathsf{type} \,\, \alpha \,\, \mathit{children} \,\, = \,\, \alpha \,\, \mathit{Tuple.t}
     let keystones' l =
        \mathsf{let}\ n\ =\ \mathit{List.length}\ \mathit{l}\ \mathsf{in}
        ThoList.flatmap (fun p \rightarrow Combinatorics.factorized\_keystones p l)
           (partitions n)
     let keystones l =
        List.map (fun (bra, kets) \rightarrow (bra, List.map Tuple.of\_list kets))
           (keystones' l)
     let max\_subtree \ n = n / 2
module Nary 4 = Nary (struct let max\_arity () = 3 end)
```

3.2.3 Factorizing Diagrams for ϕ^4

```
module Ternary =
  struct
     type partition = int \times int \times int \times int
     let inspect\_partition\ (n1,\ n2,\ n3,\ n4)\ =\ [n1;\ n2;\ n3;\ n4]
     type \alpha children = \alpha Tuple. Ternary.t
     \mathsf{let}\ \mathit{collect4}\ \mathit{acc}\ =\ \mathsf{function}
        [x; y; z; u] \rightarrow (x, y, z, u) :: acc
        - \rightarrow acc
     let partitions n =
        List.fold\_left\ collect4\ [\ ]\ (Nary4.partitions\ n)
     let collect3 acc = function
        [x; y; z] \rightarrow Tuple.Ternary.of3 x y z :: acc
        - \rightarrow acc
     let keystones l =
        List.map (fun (bra, kets) \rightarrow (bra, List.fold_left collect3 [] kets))
           (Nary4.keystones' l)
     \mathsf{let}\ max\_subtree\ =\ Nary4.max\_subtree
  end
                                      3.2.4 Factorizing Diagrams for \phi^3 + \phi^4
module Mixed23 =
  struct
     type partition =
        | P3 of int \times int \times int
        | P4 of int \times int \times int \times int
     let inspect\_partition = function
        | P3 (n1, n2, n3) \rightarrow [n1; n2; n3]
        P4 (n1, n2, n3, n4) \rightarrow [n1; n2; n3; n4]
     type \alpha children = \alpha Tuple.Mixed23.t
     let collect34 acc = function
         [x; y; z] \rightarrow P3(x, y, z) :: acc
        [x; y; z; u] \rightarrow P4(x, y, z, u) :: acc
        - \rightarrow acc
     let partitions n =
        List.fold\_left\ collect34\ []\ (Nary4.partitions\ n)
     let collect23 acc = function
        [x; y] \rightarrow Tuple.Mixed23.of2 x y :: acc
        [x; y; z] \rightarrow Tuple.Mixed23.of3 x y z :: acc
        |  \rightarrow  acc
     l = l = l = l
        List.map (fun (bra, kets) \rightarrow (bra, List.fold_left collect23 [] kets))
           (Nary4.keystones' l)
     let max\_subtree = Nary4.max\_subtree
  end
                         Diagnostics: Counting Diagrams and Factorizations for \sum_n \lambda_n \phi^n
{\sf module\ type}\ {\it Integer}\ =
  sig
     type t
     val zero : t
     val one : t
     \mathsf{val} \; (\; +\; ) \; : \; t \; \rightarrow \; t \; \rightarrow \; t
     \mathsf{val}\;(\;-\;)\;:\;t\;\to\;t\;\to\;t
     \mathsf{val}\;(\;\times\;)\;:\;t\;\to\;t\;\to\;t
     \mathsf{val}\;(\;/\;)\;:\;t\;\to\;t\;\to\;t
```

```
\begin{array}{l} \text{val } pred \ : \ t \ \rightarrow \ t \\ \text{val } succ \ : \ t \ \rightarrow \ t \\ \text{val } (\ = \ ) \ : \ t \ \rightarrow \ t \ \rightarrow \ bool \\ \text{val } (\ \neq \ ) \ : \ t \ \rightarrow \ t \ \rightarrow \ bool \\ \text{val } (\ < \ ) \ : \ t \ \rightarrow \ t \ \rightarrow \ bool \\ \text{val } (\ \leq \ ) \ : \ t \ \rightarrow \ t \ \rightarrow \ bool \\ \text{val } (\ \geq \ ) \ : \ t \ \rightarrow \ t \ \rightarrow \ bool \\ \text{val } (\ \geq \ ) \ : \ t \ \rightarrow \ t \ \rightarrow \ bool \\ \text{val } of\_int \ : \ int \ \rightarrow \ t \\ \text{val } to\_int \ : \ t \ \rightarrow \ int \\ \text{val } to\_string \ : \ t \ \rightarrow \ string \\ \text{val } compare \ : \ t \ \rightarrow \ t \ \rightarrow \ int \\ \text{val } factorial \ : \ t \ \rightarrow \ t \\ \text{end} \end{array}
```

O'Caml's native integers suffice for all applications, but in appendix S, we want to use big integers for numeric checks in high orders:

```
\mathsf{module}\ Int\ :\ Integer\ =
  struct
     \mathsf{type}\ t\ =\ int
     let zero = 0
     let one = 1
     let ( + ) = ( + )
     let ( - ) = ( - )
     let ( \times ) = ( \times )
     let ( / ) = ( / )
     let pred = pred
     let succ = succ
     let ( = ) = ( = )
     \mathsf{let} \; (\; \neq \;) \; = \; (\; \neq \;)
     \mathsf{let} \; (\; <\; )\; =\; (\; <\; )
     let ( \leq ) = ( \leq )
     let ( > ) = ( > )
     let ( \geq ) = ( \geq )
     let of_int n = n
     let to_{-}int n = n
     \mathsf{let}\ to\_string\ =\ string\_of\_int
     let compare = compare
     let\ factorial\ =\ Combinatorics.factorial
  end
module type Count =
  sig
     type integer
     val\ diagrams: ?f: (integer \rightarrow bool) \rightarrow integer \rightarrow integer \rightarrow integer
     val\ diagrams\_via\_keystones: integer 
ightarrow integer 
ightarrow integer
     val\ keystones: integer\ list 
ightarrow\ integer
     \verb|val| diagrams_per_keystone|: integer| \rightarrow integer| list \rightarrow integer|
  end
module \ Count \ (I : Integer) =
  struct
     let description = ["(still_inoperational)_phi^n_topology"]
     \mathsf{type} \ integer \ = \ I.t
     open {\it I}
     let two = of_int 2
     let three = of_int 3
```

If I.t is an abstract datatype, the polymorphic *Pervasives.min* can fail. Provide our own version using the specific comparison " (\leq) ".

```
\begin{array}{c} x \\ \text{else} \\ y \end{array}
```

Counting Diagrams for
$$\sum_{n} \lambda_n \phi^n$$

Classes of diagrams are defined by the number of vertices and their degrees. We could use fixed size arrays, but we will use a map instead. For efficiency, we also maintain the number of external lines and the total number of propagators.

The numbers of external lines, propagators and vertices are determined by the degrees and multiplicities of vertices:

$$E({n_3, n_4, \ldots}) = 2 + \sum_{d=3}^{\infty} (d-2)n_d$$
(3.10a)

$$P(\{n_3, n_4, \ldots\}) = \sum_{d=3}^{\infty} n_d - 1 = V(\{n_3, n_4, \ldots\}) - 1$$
(3.10b)

$$V(\{n_3, n_4, \ldots\}) = \sum_{d=2}^{\infty} n_d$$
 (3.10c)

```
let num\_ext\ v = List.fold\_left\ (fun\ sum\ (d,\ n) \rightarrow sum\ + (d\ -\ two)\ \times\ n)\ two\ v
let num\_prop\ v = List.fold\_left\ (fun\ sum\ (\_,\ n) \rightarrow sum\ +\ n)\ (zero\ -\ one)\ v
```

The sum of all vertex degrees must be equal to the number of propagator end points. This can be verified easily:

$$2P(\{n_3, n_4, \ldots\}) + E(\{n_3, n_4, \ldots\}) = \sum_{d=3}^{\infty} dn_d$$
(3.11)

```
let add\_degree \ map \ (d, \ n) =
  if d < three then
     invalid\_arg "add_degree: _{\sqcup}d_{\sqcup}<_{\sqcup}3"
  else if n < zero then
     invalid\_arg "add_degree:_{\square}n_{\square}<=_{\square}0"
  else if n = zero then
      map
  else
     IMap.add d n map
let create\_class \ v =
   \{ ext = num_ext \ v; \}
     prop = num\_prop v;
     v = List.fold\_left \ add\_degree \ IMap.empty \ v 
let multiplicity \ cl \ d =
  if d \geq three then
     try
         IMap.find \ d \ cl.v
     | Not\_found \rightarrow zero
  else
      invalid\_arg "multiplicity:_{\sqcup}d_{\sqcup}<_{\sqcup}3"
```

Remove one vertex of degree d, maintaining the invariants. Raises Zero if all vertices of degree d are exhausted.

24

```
exception Zero
```

let $remove \ cl \ d =$

```
\begin{array}{lll} \text{let } n &=& pred \; (multiplicity \; cl \; d) \; \text{in} \\ \text{if } n &<& zero \; \text{then} \\ &raise \; Zero \\ \text{else} \\ & \{ \; ext \; = \; cl.ext \; - \; (d \; - \; two); \\ &prop \; = \; pred \; cl.prop; \\ &v \; = \; \text{if} \; n \; = \; zero \; \text{then} \\ &IMap.remove \; d \; cl.v \\ &\text{else} \\ &IMap.add \; d \; n \; cl.v \; \} \end{array}
```

Add one vertex of degree d, maintaining the invariants.

Count the number of diagrams. Any diagram can be obtained recursively either from a diagram with one ternary vertex less by insertion if a ternary vertex in an internal or external propagator or from a diagram with a higher order vertex that has its degree reduced by one:

```
D(\{n_3, n_4, \ldots\}) = (P(\{n_3 - 1, n_4, \ldots\}) + E(\{n_3 - 1, n_4, \ldots\})) D(\{n_3 - 1, n_4, \ldots\}) + \sum_{d=4}^{\infty} (n_{d-1} + 1) D(\{n_3, n_4, \ldots, n_{d-1} + 1, n_d - 1, \ldots\}) (3.12)
```

```
let rec class\_size \ cl =
if cl.ext = two \lor cl.prop = zero then
one
else
IMap.fold \ (fun \ d \_s \rightarrow class\_size\_n \ cl \ d \ + \ s) \ cl.v \ (class\_size\_3 \ cl)
```

Purely ternary vertices recurse among themselves:

```
and class\_size\_3 cl = try let d' = remove cl three in (d'.ext + d'.prop) \times class\_size d' with | Zero \rightarrow zero
```

Vertices of higher degree recurse one step towards lower degrees:

```
and class\_size\_n cl d =
    if d > three then begin
    try
    let d' = pred d in
    let cl' = add (remove cl d) d' in
    multiplicity cl' d' × class\_size cl'
    with
    | Zero → zero
end else
    zero
```

Find all $\{n_3, n_4, \ldots, n_d\}$ with

$$E(\{n_3, n_4, \dots, n_d\}) - 2 = \sum_{i=3}^{c} l(i-2)n_i = sum$$
(3.13)

The implementation is a variant of tuples above.

```
let rec distribute\_degrees' \ d \ sum =  if d < three \ then
```

```
invalid\_arg \; "distribute\_degrees" else \; if \; d \; = \; three \; then \\ \; [[(d,\; sum)]] \\ else \\ \; distribute\_degrees" \; d \; sum \; (sum \; / \; (d \; - \; two)) and \; distribute\_degrees" \; d \; sum \; n \; = \\ \; if \; n \; < \; zero \; then \\ \; [] \\ else \\ \; List.fold\_left \; (fun \; ll \; l \; \rightarrow \; ((d,\; n) \; :: \; l) \; :: \; ll) \\ \; (distribute\_degrees" \; d \; sum \; (pred \; n)) \\ \; (distribute\_degrees' \; (pred \; d) \; (sum \; - \; (d \; - \; two) \; \times \; n))
```

Actually, we need to find all $\{n_3, n_4, \dots, n_d\}$ with

$$E(\{n_3, n_4, \dots, n_d\}) = sum \tag{3.14}$$

let distribute_degrees d sum = distribute_degrees' d (sum - two)

Finally we can count all diagrams by adding all possible ways of splitting the degrees of vertices. We can also count diagrams where all degrees satisfy a predicate f:

```
let diagrams\ ?(f = \text{fun} \_ \to \text{true})\ deg\ n = List.fold\_left\ (\text{fun}\ s\ d \to \text{if}\ List.for\_all\ (\text{fun}\ (d',\ n') \to f\ d'\ \lor\ n' = zero)\ d\ \text{then}\ s + class\_size\ (create\_class\ d) else s) zero\ (distribute\_degrees\ deg\ n)
```

The next two are duplicated from *ThoList* and *Combinatorics*, in order to use the specific comparison functions.

```
let classify l =
  let rec add\_to\_class a = function
     | [] \rightarrow [of\_int 1, a]
     \mid (n, a') :: rest \rightarrow
          if a = a' then
             (succ\ n,\ a)\ ::\ rest
           else
             (n, a') :: add\_to\_class \ a \ rest
  in
  let rec classify' cl = function
     | [] \rightarrow cl
     \mid a :: rest \rightarrow classify' (add\_to\_class \ a \ cl) \ rest
  in
   classify' [] l
let permutation\_symmetry l =
   List.fold\_left (fun s (n, \_) \rightarrow factorial n \times s) one (classify l)
let symmetry l =
  let sum = List.fold\_left (+) zero l in
  if List.exists (fun x \rightarrow two \times x = sum) l then
     two \times permutation\_symmetry l
  else
     permutation\_symmetry l
```

The number of Feynman diagrams built of vertices with maximum degree d_{max} in a partition $N_{d,n} = \{n_1, n_2, \dots, n_d\}$ with $n = n_1 + n_2 + \dots + n_d$ and

$$\tilde{F}(d_{\max}, N_{d,n}) = \frac{n!}{|\mathcal{S}(N_{d,n})|\sigma(n_d, n)} \prod_{i=1}^{d} \frac{F(d_{\max}, n_i + 1)}{n_i!}$$
(3.15)

with |S(N)| the size of the symmetric group of N, $\sigma(n,2n)=2$ and $\sigma(n,m)=1$ otherwise.

```
\begin{array}{lll} \text{let } keystones \ p &= \\ & \text{let } sum \ = \ List.fold\_left \ (+) \ zero \ p \ \text{in} \\ & List.fold\_left \ (\text{fun } acc \ n \ \rightarrow \ acc \ / \ (factorial \ n)) \ (factorial \ sum) \ p \\ & / \ symmetry \ p \\ & \text{let } diagrams\_per\_keystone \ deg \ p \ = \\ & List.fold\_left \ (\text{fun } acc \ n \ \rightarrow \ acc \ \times \ diagrams \ deg \ (succ \ n)) \ one \ p \end{array}
```

We must find

$$F(d_{\max}, n) = \sum_{d=3}^{d_{\max}} \sum_{\substack{N = \{n_1, n_2, \dots, n_d\} \\ n_1 + n_2 + \dots + n_d = n \\ 1 \le n_1 \le n_2 \le \dots \le n_d \le |n/2|}} \tilde{F}(d_{\max}, N)$$
(3.16)

```
let diagrams\_via\_keystones\ deg\ n= let module N=Nary (struct let max\_arity () = to\_int (pred\ deg) end) in List.fold\_left (fun acc\ p\to acc\ +\ diagrams\_per\_keystone\ deg\ p\ \times\ keystones\ p) zero\ (List.map\ (List.map\ of\_int)\ (N.partitions\ (to\_int\ n))) end
```

3.2.6 Emulating HELAC

```
In [2], one leg is singled out:
module\ Helac\ (B:\ Tuple.Bound) =
  struct
     module Tuple = Tuple.Nary(B)
     type partition = int list
     let inspect\_partition p = p
     let partition d sum =
        Partition.tuples \ d \ sum \ 1 \ (sum - d + 1)
     let rec partitions' d sum =
       let d' = pred d in
       if d' < 2 then
       else
          List.map (fun p \rightarrow 1 :: p) (partition d' (pred sum)) @ partitions' d' sum
     let partitions \ sum = partitions' (succ (B.max\_arity ())) \ sum
     type \alpha children = \alpha Tuple.t
     let keystones' l =
       \mathsf{match}\ \mathit{l}\ \mathsf{with}
        | [] \rightarrow []
        \mid head :: tail \rightarrow
            [([head],
               ThoList.flatmap (fun p \rightarrow Combinatorics.partitions (List.tl p) tail)
                 (partitions (List.length l)))
     l = l = l = l
        List.map (fun (bra, kets) \rightarrow (bra, List.map Tuple.of\_list kets))
          (keystones' l)
     let max\_subtree n = pred n
```

Ś

The following is not tested, but it is no rocket science either ...

```
\begin{array}{lll} \text{module } \textit{Helac\_Binary} &= \\ \text{struct} & \\ \text{type } \textit{partition} &= \textit{int} \times \textit{int} \times \textit{int} \end{array}
```

```
let inspect\_partition\ (n1,\ n2,\ n3)\ =\ [n1;\ n2;\ n3]
   let \ partitions \ sum \ =
       List.map (fun (n2, n3) \rightarrow (1, n2, n3))
          (Partition.pairs (sum - 1) 1 (sum - 2))
   type \alpha children = \alpha Tuple.Binary.t
   \mathsf{let}\ \mathit{keystones'}\ l\ =
       \mathsf{match}\ \mathit{l}\ \mathsf{with}
       |\hspace{.1cm}[\hspace{.1cm}]\hspace{.1cm}\rightarrow\hspace{.1cm}[\hspace{.1cm}]
       \mid head :: tail \rightarrow
             [([head],
                  ThoList.flatmap \ (fun \ (\_, \ p2, \ \_) \ 	o \ Combinatorics.split \ p2 \ tail)
                     (partitions (List.length l)))]
   \mathsf{let}\ \mathit{keystones}\ l\ =
       List.map \ (fun \ (bra, \ kets) \ 
ightarrow
          (\mathit{bra},\ \mathit{List.map}\ (\mathsf{fun}\ (x,\ y)\ \rightarrow\ \mathit{Tuple.Binary.of2}\ x\ y)\ \mathit{kets}))
          (keystones' l)
   \mathsf{let}\ max\_subtree\ n\ =\ pred\ n
end
```

__4__

DIRECTED ACYCLICAL GRAPHS

4.1 Interface of DAG

This data structure describes large collections of trees with many shared nodes. The sharing of nodes is semantically irrelevant, but can turn a factorial complexity to exponential complexity. Note that DAG implements only a very specialized subset of Directed Acyclical Graphs (DAGs).

If T(n, D) denotes the set of all binary trees with root n encoded in D, while

$$O(n, D) = \{(e_1, n_1, n'_1), \dots, (e_k, n_k, n'_k)\}$$
(4.1)

denotes the set of all offspring of n in D, and tree(e, t, t') denotes the binary tree formed by joining the binary trees t and t' with the label e, then

$$T(n,D) = \left\{ \text{tree}(e_i, t_i, t_i') \mid (e_i, t_i, t_i') \in \{e_1\} \times T(n_1, D) \times T(n_1', D) \cup \dots \right.$$
$$\dots \cup \left\{ e_k \right\} \times T(n_k, D) \times T(n_k', D) \right\} \quad (4.2)$$

is the recursive definition of the binary trees encoded in D. It is obvious how this definitions translates to n-ary trees (including trees with mixed arity).

4.1.1 Forests

We require edges and nodes to be members of ordered sets. The sematics of *compare* are compatible with *Pervasives.compare*:

$$compare(x,y) = \begin{cases} -1 & \text{for } x < y \\ 0 & \text{for } x = y \\ 1 & \text{for } x > y \end{cases}$$

$$(4.3)$$

Note that this requirement does *not* exclude any trees. Even if we consider only topological equivalence classes with anonymous nodes, we can always construct a canonical labeling and order from the children of the nodes. However, if practical applications, we will often have more efficient labelings and orders at our disposal.

```
\begin{array}{lll} \text{module type } \mathit{Ord} &= \\ & \text{sig} \\ & \text{type } t \\ & \text{val } \mathit{compare} \ : \ t \ \rightarrow \ t \ \rightarrow \ \mathit{int} \\ & \text{end} \end{array}
```

A forest F over a set of nodes and a set of edges is a map from the set of nodes N, to the direct product of the set of edges E and the power set 2^N of N augmented by a special element \bot ("bottom").

$$F: N \to (E \times 2^N) \cup \{\bot\}$$

$$n \mapsto \begin{cases} (e, \{n'_1, n'_2, \ldots\}) \\ \bot \end{cases}$$

$$(4.4)$$

The nodes are ordered so that cycles can be detected

$$\forall n \in N : F(n) = (e, x) \Rightarrow \forall n' \in x : n > n'$$

$$\tag{4.5}$$

A suitable function that exists for *all* forests is the depth of the tree beneath a node.

Nodes that are mapped to \perp are called *leaf* nodes and nodes that do not appear in any F(n) are called *root* nodes. There are as many trees in the forest as there are root nodes.

```
\begin{array}{ll} \text{module type } Forest &= \\ \text{sig} \\ & \text{module } Nodes \ : \ Ord \\ & \text{type } node \ = \ Nodes.t \\ & \text{type } edge \end{array}
```

A subset $X \subset 2^N$ of the powerset of the set of nodes. The members of X can be be characterized by a fixed number of members (e. g. two for binary trees, as in QED). We can also have mixed arities (e. g. two and three for QCD) or even arbitrary arities. However, in most cases, the members of X will have at least two members.

```
type children
```

This type abbreviation and order allow to apply the Set. Make functor to $E \times X$.

```
\begin{array}{lll} \text{type} \ t \ = \ edge \ \times \ children \\ \text{val} \ compare \ : \ t \ \rightarrow \ t \ \rightarrow \ int \end{array}
```

Test a predicate for *all* children.

```
val\ for\_all\ :\ (node\ 	o\ bool)\ 	o\ t\ 	o\ bool
```

fold f (_, children) acc will calculate

$$f(x_1, f(x_2, \cdots f(x_n, acc))) \tag{4.6}$$

where the *children* are $\{x_1, x_2, \dots, x_n\}$. There are slightly more efficient alternatives for fixed arity (in particular binary), but we want to be general.

```
\begin{array}{l} \text{val } fold \ : \ (node \ \rightarrow \ \alpha \ \rightarrow \ \alpha) \ \rightarrow \ t \ \rightarrow \ \alpha \ \rightarrow \ \alpha \\ \text{end} \\ \\ \text{module } Forest \ : \ \text{functor} \ (PT \ : \ Tuple.Poly) \ \rightarrow \\ \\ \text{functor} \ (N \ : \ Ord) \ \rightarrow \ \text{functor} \ (E \ : \ Ord) \ \rightarrow \\ \\ Forest \ \text{with module} \ Nodes \ = \ N \ \text{and type} \ edge \ = \ E.t \\ \\ \text{and type} \ node \ = \ N.t \ \text{and type} \ children \ = \ N.t \ PT.t \end{array}
```

4.1.2 DAGs

```
\begin{array}{ll} \text{module type } T & = \\ \text{sig} & \\ \text{type } node & \\ \text{type } edge & \end{array}
```

In the description of the function we assume for definiteness DAGs of binary trees with type $children = node \times node$. However, we will also have implementations with type children = node list below.

Other possibilities include type children = V3 of $node \times node \mid V4$ of $node \times node \times node$. There's probable never a need to use sets with logarithmic access, but it is easy to add.

```
\begin{array}{c} {\rm type} \ children \\ {\rm type} \ t \end{array}
```

The empty DAG.

```
val\ empty : t
```

 $add_node\ n\ dag$ returns the DAG dag with the node n. If the node n already exists in dag, it is returned unchanged. Otherwise n is added without offspring.

```
val\ add\_node:\ node \rightarrow t \rightarrow t
```

 $add_offspring\ n\ (e,\ (n1,\ n2))\ dag$ returns the DAG dag with the node n and its offspring n1 and n2 with edge label e. Each node can have an arbitrary number of offspring, but identical offspring are added only once. In order to prevent cycles, $add_offspring$ requires both n>n1 and n>n2 in the given ordering. The nodes n1 and n2 are added as by add_node . NB: Adding all nodes n1 and n2, even if they are sterile, is not strictly necessary for our applications. It even slows down the code by a few percent. But it is desirable for consistency and allows much more efficient $iter_nodes$ and $fold_nodes$ below.

```
val\ add\_offspring: node 
ightarrow edge 	imes children 
ightarrow t 
ightarrow t
```

exception Cycle

Just like add_offspring, but does not check for potential cycles.

```
val add\_offspring\_unsafe : node \rightarrow edge \times children \rightarrow t \rightarrow t
```

 $is_node \ n \ dag \ returns \ true \ iff \ n \ is a node in \ dag.$

```
val is\_node : node \rightarrow t \rightarrow bool
```

is_sterile n dag returns true iff n is a node in dag and boasts no offspring.

```
val\ is\_sterile\ :\ node\ 	o\ t\ 	o\ bool
```

is_offspring n (e, (n1, n2)) dag returns true iff n1 and n2 are offspring of n with label e in dag.

```
val\ is\_offspring: node \rightarrow edge \times children \rightarrow t \rightarrow bool
```

Note that the following functions can run into infinite recursion if the DAG given as argument contains cycles. The usual functionals for processing all nodes (including sterile) . . .

```
val\ iter\_nodes: (node \rightarrow unit) \rightarrow t \rightarrow unit
val\ map\_nodes: (node \rightarrow node) \rightarrow t \rightarrow t
\mathsf{val}\; fold\_nodes\;:\; (node\;\rightarrow\;\alpha\;\rightarrow\;\alpha)\;\rightarrow\;t\;\rightarrow\;\alpha\;\rightarrow\;\alpha
```

... and all parent/offspring relations. Note that map requires two functions: one for the nodes and one for the edges and children. This is so because a change in the definition of node is not propagated automatically to where it is used as a child.

```
val\ iter\ :\ (node\ 
ightarrow\ edge\ 	imes\ children\ 
ightarrow\ unit)\ 
ightarrow\ t\ 
ightarrow\ unit
val\ map\ :\ (node\ 	o\ node)\ 	o
    (node \rightarrow edge \times children \rightarrow edge \times children) \rightarrow t \rightarrow t
\mathsf{val}\; fold\;:\; (node\;\rightarrow\; edge\;\times\; children\;\rightarrow\;\alpha\;\rightarrow\;\alpha)\;\rightarrow\;t\;\rightarrow\;\alpha\;\rightarrow\;\alpha
```



Note that in it's current incarnation, fold add_offspring dag empty copies only the fertile nodes, while fold add_offspring dag (fold_nodes add_node dag empty) includes sterile ones, as does map (fun $n \rightarrow$ n) (fun n $ec \rightarrow ec)$ dag.

Return the DAG as a list of lists.

```
val lists: t \rightarrow (node \times (edge \times children) \ list) \ list
```

dependencies dag node returns a canonically sorted Tree2.t of all nodes reachable from node.

```
val dependencies : t \rightarrow node \rightarrow (node, edge) Tree2.t
```

harvest dag n roots returns the DAG roots enlarged by all nodes in dag reachable from n.

```
\mathsf{val}\ \mathit{harvest}\ :\ t\ \to\ \mathit{node}\ \to\ t\ \to\ t
```

harvest_list dag nlist returns the part of the DAG dag that is reachable from the nodes in nlist.

```
\mathsf{val}\ \mathit{harvest\_list}\ :\ t\ \to\ \mathit{node}\ \mathit{list}\to\ t
```

size dag returns the number of nodes in the DAG dag.

```
\mathsf{val}\ size\ :\ t\ \to\ int
```

eval f mul_edge mul_nodes add null unit root dag interprets the part of dag beneath root as an algebraic expression:

- each node is evaluated by $f: node \rightarrow \alpha$
- each set of children is evaluated by iterating the binary $mul_nodes: \alpha \rightarrow \gamma \rightarrow \gamma$ on the values of the nodes, starting from unit: γ
- each offspring relation (node, (edge, children)) is evaluated by applying $mul_edge: node \rightarrow edge \rightarrow$ $\gamma \rightarrow \delta$ to node, edge and the evaluation of children.
- all offspring relations of a node are combined by iterating the binary $add: \delta \rightarrow \alpha \rightarrow \alpha$ starting from $null: \alpha$

In our applications, we will always have $\alpha = \gamma = \delta$, but the more general type is useful for documenting the relationships. The memoizing variant eval_memoized f mul_edge mul_nodes add null unit root dag requires some overhead, but can be more efficient for complex operations.

```
val\ eval\ :\ (node\ 
ightarrow\ lpha)\ 
ightarrow\ (node\ 
ightarrow\ edge\ 
ightarrow\ \gamma\ 
ightarrow\ \delta)\ 
ightarrow
     (\alpha \rightarrow \gamma \rightarrow \gamma) \rightarrow (\delta \rightarrow \alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \gamma \rightarrow node \rightarrow t \rightarrow \alpha
\textit{val} \ eval\_memoized \ : \ (node \ \rightarrow \ \alpha) \ \rightarrow \ (node \ \rightarrow \ edge \ \rightarrow \ \gamma \ \rightarrow \ \delta) \ \rightarrow
     (\alpha \rightarrow \gamma \rightarrow \gamma) \rightarrow (\delta \rightarrow \alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \gamma \rightarrow node \rightarrow t \rightarrow \alpha
```

forest root dag expands the dag beneath root into the equivalent list of trees Tree.t. children are represented as list of nodes.



 \diamondsuit A sterile node n is represented as Tree.Leaf ((n, None), n), cf. page 699. There might be a better way, but we need to change the interface and semantics of *Tree* for this.

```
\mathsf{val}\ forest\ :\ node\ 	o\ t\ 	o\ (node\ 	imes\ edge\ option,\ node)\ \mathit{Tree.t}\ \mathit{list}
val forest\_memoized: node \rightarrow t \rightarrow (node \times edge\ option,\ node)\ Tree.t\ list
```

count_trees n dag returns the number of trees with root n encoded in the DAG dag, i.e. |T(n,D)|. NB: the current implementation is very naive and can take a very long time for moderately sized DAGs that encode a large set of trees.

```
val\ count\_trees\ :\ node\ 	o\ t\ 	o\ int
   end
module\ Make\ (F:Forest):
    T with type node = F.node and type edge = F.edge
    and type children = F.children
```

4.1.3 Graded Sets, Forests & DAGs

A graded ordered set is an ordered set with a map into another ordered set (often the non-negative integers). The grading does not necessarily respect the ordering.

```
module type Graded\_Ord =
  sig
     include Ord
      \mathsf{module}\ G\ :\ \mathit{Ord}
      val \ rank : t \rightarrow G.t
  end
```

For all ordered sets, there are two canonical gradings: a Chaotic grading that assigns the same rank (e.g. unit) to all elements and the *Discrete* grading that uses the identity map as grading.

```
module type Grader = functor (O : Ord) \rightarrow Graded\_Ord with type t = O.t
module Chaotic: Grader
module Discrete: Grader
```

A graded forest is just a forest in which the nodes form a graded ordered set.



There doesn't appear to be a nice syntax for avoiding the repetition here. Fortunately, the signature is short

```
module type Graded\_Forest =
      module Nodes: Graded_Ord
     \mathsf{type} \ node \ = \ Nodes.t
      type edge
      type children
     type t = edge \times children
     \mathsf{val}\ compare\ :\ t\ \to\ t\ \to\ int
     val\ for\_all\ :\ (node\ 	o\ bool)\ 	o\ t\ 	o\ bool
      val fold: (node \rightarrow \alpha \rightarrow \alpha) \rightarrow t \rightarrow \alpha \rightarrow \alpha
   end
```

¹We don't appear to have use for graded unordered sets.

```
module type Forest\_Grader = functor(G:Grader) \rightarrow functor(F:Forest) \rightarrow
  Graded\_Forest with type Nodes.t = F.node
  and type node = F.node
  and type edge = F.edge
  and type children = F.children
  and type t = F.t
module\ Grade\_Forest\ :\ Forest\_Grader
Finally, a graded DAG is a DAG in which the nodes form a graded ordered set and the subsets with a given
rank can be accessed cheaply.
module type Graded =
  sig
     include T
     type rank
     \mathsf{val}\ \mathit{rank}\ :\ \mathit{node}\ \to\ \mathit{rank}
     val \ ranks : t \rightarrow rank \ list
     val min_max_rank : t \rightarrow rank \times rank
     val\ ranked\ :\ rank\ 	o\ t\ 	o\ node\ list
  end
module \ Graded \ (F : Graded\_Forest) :
     Graded with type node = F.node and type edge = F.edge
     and type children = F.children and type rank = F.Nodes.G.t
                                      4.2 Implementation of DAG
module type Ord =
  sig
     type t
     val compare: t \rightarrow t \rightarrow int
module type Forest =
     module Nodes: Ord
     type node = Nodes.t
     type edge
     type children
     type t = edge \times children
     \mathsf{val}\ compare\ :\ t\ \to\ t\ \to\ int
     val\ for\_all\ :\ (node\ 	o\ bool)\ 	o\ t\ 	o\ bool
     \mathsf{val}\ fold\ :\ (node\ \to\ \alpha\ \to\ \alpha)\ \to\ t\ \to\ \alpha\ \to\ \alpha
module type T =
  sig
     type node
     type edge
     type children
     type t
     val\ empty : t
     \mathsf{val}\ add\_node\ :\ node\ \to\ t\ \to\ t
     val\ add\_offspring: node 
ightarrow edge 	imes children 
ightarrow t 
ightarrow t
     exception Cycle
     \mathsf{val}\ add\_offspring\_unsafe\ :\ node\ \to\ edge\ \times\ children\ \to\ t\ \to\ t
     \mathsf{val}\ is\_node\ :\ node\ \to\ t\ \to\ bool
```

 $val\ is_sterile\ :\ node\ o\ t\ o\ bool$

 $\mathsf{val}\ is_offspring\ :\ node\ \to\ edge\ \times\ children\ \to\ t\ \to\ bool$

 $val\ fold_nodes: (node \rightarrow \alpha \rightarrow \alpha) \rightarrow t \rightarrow \alpha \rightarrow \alpha$

 $\begin{array}{lll} \mathsf{val}\ iter_nodes\ :\ (node\ \to\ unit)\ \to\ t\ \to\ unit\\ \mathsf{val}\ map_nodes\ :\ (node\ \to\ node)\ \to\ t\ \to\ t \end{array}$

```
\mathsf{val}\ iter\ :\ (node\ 	o\ edge\ 	imes\ children\ 	o\ unit)\ 	o\ t\ 	o\ unit
      val\ map\ :\ (node\ 	o\ node)\ 	o
         (node \rightarrow edge \times children \rightarrow edge \times children) \rightarrow t \rightarrow t
      val fold: (node \rightarrow edge \times children \rightarrow \alpha \rightarrow \alpha) \rightarrow t \rightarrow \alpha \rightarrow \alpha
      \mathsf{val}\ lists\ :\ t\ 	o\ (node\ 	imes\ (edge\ 	imes\ children)\ list)\ list
      val dependencies : t \rightarrow node \rightarrow (node, edge) Tree2.t
      \mathsf{val}\ harvest\ :\ t\ \to\ node\ \to\ t\ \to\ t
      \mathsf{val}\ \mathit{harvest\_list}\ :\ t\ \to\ \mathit{node}\ \mathit{list}\ \to\ \mathit{t}
      \mathsf{val}\ size\ :\ t\ \to\ int
      val\ eval\ : (node\ 
ightarrow\ \alpha)\ 
ightarrow\ (node\ 
ightarrow\ edge\ 
ightarrow\ \gamma\ 
ightarrow\ \delta)\ 
ightarrow
         (\alpha \rightarrow \gamma \rightarrow \gamma) \rightarrow (\delta \rightarrow \alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \gamma \rightarrow node \rightarrow t \rightarrow \alpha
      val eval\_memoized : (node \rightarrow \alpha) \rightarrow (node \rightarrow edge \rightarrow \gamma \rightarrow \delta) \rightarrow
         (\alpha \rightarrow \gamma \rightarrow \gamma) \rightarrow (\delta \rightarrow \alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \gamma \rightarrow node \rightarrow t \rightarrow \alpha
      \mathsf{val}\ forest\ :\ node\ 	o\ t\ 	o\ (node\ 	imes\ edge\ option,\ node)\ \mathit{Tree.t}\ \mathit{list}
      val\ forest\_memoized: node \rightarrow t \rightarrow (node \times edge\ option,\ node)\ Tree.t\ list
      val\ count\_trees\ :\ node\ 	o\ t\ 	o\ int
    end
module type Graded\_Ord =
   sig
      include Ord
      \mathsf{module}\ G\ :\ \mathit{Ord}
      val \ rank : t \rightarrow G.t
module type Grader = functor (O : Ord) \rightarrow Graded\_Ord with type t = O.t
module type Graded\_Forest =
   sig
      module\ Nodes\ :\ Graded\_Ord
      type node = Nodes.t
      type edge
      type children
      type t = edge \times children
      val compare: t \rightarrow t \rightarrow int
      val\ for\_all\ :\ (node\ 	o\ bool)\ 	o\ t\ 	o\ bool
      \mathsf{val}\ fold\ :\ (node\ \to\ \alpha\ \to\ \alpha)\ \to\ t\ \to\ \alpha\ \to\ \alpha
   end
module type Forest\_Grader = functor (G : Grader) \rightarrow functor (F : Forest) \rightarrow
   Graded\_Forest with type Nodes.t = F.node
   and type node = F.node
   and type edge = F.edge
   and type children = F.children
   and type t = F.t
                                                      4.2.1
                                                                  The Forest Functor
module\ Forest\ (PT\ :\ Tuple.Poly)\ (N\ :\ Ord)\ (E\ :\ Ord)\ :
      Forest with module Nodes = N and type edge = E.t
      and type node = N.t and type children = N.t PT.t =
   struct
      module Nodes = N
      type edge = E.t
      type node = N.t
      type children = node PT.t
      type t = edge \times children
      let compare (e1, n1) (e2, n2) =
         let c = PT.compare \ N.compare \ n1 \ n2 in
         if c \neq 0 then
            c
         else
```

```
E.compare\ e1\ e2 let for\_all\ f\ (\_,\ nodes)\ =\ PT.for\_all\ f\ nodes let fold\ f\ (\_,\ nodes)\ acc\ =\ PT.fold\_right\ f\ nodes\ acc end
```

4.2.2 Gradings

```
module \ Chaotic \ (O : Ord) =
  struct
    include O
    module G =
       struct
         type t = unit
         let compare _ = 0
    let rank _{-} = ()
  end
module \ Discrete \ (O : Ord) =
  struct
    include O
    \mathsf{module}\ G = O
    \mathsf{let} \ \mathit{rank} \ x \ = \ x
module Fake\_Grading (O : Ord) =
  struct
    include O
    exception Impossible of string
    \mathsf{module}\ G =
       struct
         type t = unit
         let compare _ _ = raise (Impossible "G.compare")
    let rank _ = raise (Impossible "G.compare")
  end
module \ Grade\_Forest \ (G : Grader) \ (F : Forest) =
  struct
    module\ Nodes\ =\ G(F.Nodes)
    \mathsf{type}\ node\ =\ Nodes.t
    type edge = F.edge
    type children = F.children
    type t = F.t
    let compare = F.compare
    let for\_all = F.for\_all
    let fold = F.fold
  end
```

Ś

The following can easily be extended to $\mathit{Map.S}$ in its full glory, if we ever need it.

```
\begin{array}{l} \text{module type } \textit{Graded\_Map} \ = \\ \text{sig} \\ \text{type } \textit{key} \\ \text{type } \textit{rank} \\ \text{type } \alpha \ t \\ \text{val } \textit{empty} \ : \ \alpha \ t \\ \text{val } \textit{add} \ : \ \textit{key} \ \rightarrow \ \alpha \ t \ \rightarrow \ \alpha \ t \\ \text{val } \textit{find} \ : \ \textit{key} \ \rightarrow \ \alpha \ t \ \rightarrow \ \alpha \\ \text{val } \textit{mem} \ : \ \textit{key} \ \rightarrow \ \alpha \ t \ \rightarrow \ \textit{bool} \end{array}
```

```
val\ iter: (key \rightarrow \alpha \rightarrow unit) \rightarrow \alpha t \rightarrow unit
    \mathsf{val}\ fold\ :\ (key\ \to\ \alpha\ \to\ \beta\ \to\ \beta)\ \to\ \alpha\ t\ \to\ \beta\ \to\ \beta
    val \ ranks : \alpha \ t \rightarrow rank \ list
    val min\_max\_rank : \alpha t \rightarrow rank \times rank
    val \ ranked : rank \rightarrow \alpha \ t \rightarrow key \ list
module type Graded\_Map\_Maker = functor (O : Graded\_Ord) \rightarrow
  Graded\_Map with type key = O.t and type rank = O.G.t
module Graded\_Map (O : Graded\_Ord) :
     Graded\_Map with type key = O.t and type rank = O.G.t =
  struct
    module M1 = Map.Make(O.G)
    module M2 = Map.Make(O)
    type key = O.t
    type rank = O.G.t
    type (+\alpha) t = \alpha M2.t M1.t
    let empty = M1.empty
    let add key data map1 =
       let rank = O.rank key in
       let map2 = try M1.find \ rank \ map1 \ with \ Not_found \rightarrow M2.empty \ in
       M1.add rank (M2.add key data map2) map1
    let find key map = M2.find key (M1.find (O.rank key) map)
    let mem \ key \ map =
       M2.mem\ key\ (try\ M1.find\ (O.rank\ key)\ map\ with\ Not\_found\ 	o\ M2.empty)
    let iter\ f\ map1 = M1.iter\ (fun\ rank \rightarrow M2.iter\ f)\ map1
    let fold\ f\ map1\ acc1\ =\ M1.fold\ (fun\ rank\ 	o\ M2.fold\ f)\ map1\ acc1
   The set of ranks and its minimum and maximum should be maintained explicitely!
    module S1 = Set.Make(O.G)
    let ranks \ map = M1.fold \ (fun \ key \ data \ acc \rightarrow key :: acc) \ map \ []
    let rank\_set\ map\ =\ M1.fold\ (fun\ key\ data\ \to\ S1.add\ key)\ map\ S1.empty
    let min_max_rank map =
       let s = rank\_set map in
       (S1.min\_elt\ s,\ S1.max\_elt\ s)
    module S2 = Set.Make(O)
    let keys map = M2.fold (fun key data acc \rightarrow key :: acc) map []
    let sorted\_keys map =
       S2.elements (M2.fold (fun key data \rightarrow S2.add key) map S2.empty)
    let ranked rank map =
       keys (try M1.find rank map with Not\_found \rightarrow M2.empty)
  end
                                           4.2.3
                                                   The DAG Functor
module\ Maybe\_Graded\ (GMM\ :\ Graded\_Map\_Maker)\ (F\ :\ Graded\_Forest)\ =
  struct
    \mathsf{module}\ G\ =\ F.Nodes.G
    type node = F.node
    type rank = G.t
    type edge = F.edge
    type children = F.children
```

If we get tired of graded DAGs, we just have to replace $Graded_Map$ by Map here and remove ranked below and gain a tiny amount of simplicity and efficiency.

```
module \ Parents = GMM(F.Nodes)
```

```
module Offspring = Set.Make(F)
type t = Offspring.t Parents.t
let rank = F.Nodes.rank
let ranks = Parents.ranks
let \ min\_max\_rank \ = \ Parents.min\_max\_rank
let ranked = Parents.ranked
let empty = Parents.empty
let \ add\_node \ node \ dag =
  if Parents.mem node dag then
  else
     Parents.add node Offspring.empty dag
let add_offspring_unsafe node offspring dag =
  let offsprings =
    try Parents.find\ node\ dag\ with\ Not\_found\ 	o\ Offspring.empty in
  Parents.add node (Offspring.add offspring offsprings)
     (F.fold\ add\_node\ offspring\ dag)
exception Cycle
let add\_offspring \ node \ offspring \ dag =
  if F.for\_all (fun n \rightarrow F.Nodes.compare n node < 0) offspring then
     add_offspring_unsafe node offspring dag
  else
     raise Cycle
let is\_node \ node \ dag =
  Parents.mem node dag
let is\_sterile node dag =
     Offspring.is_empty (Parents.find node dag)
  with
  \mid Not\_found \rightarrow \mathsf{false}
let is\_offspring \ node \ offspring \ dag =
     Offspring.mem offspring (Parents.find node dag)
  with
  \mid Not\_found \rightarrow \mathsf{false}
let iter\_nodes\ f\ dag\ =
  Parents.iter (fun n \rightarrow f n) dag
let iter f daq =
  Parents.iter (fun node \rightarrow Offspring.iter (f node)) dag
let map\_nodes f dag =
  Parents.fold (fun n \rightarrow Parents.add (f n)) dag Parents.empty
let map fn fo dag =
  Parents.fold (fun node offspring \rightarrow
     Parents.add (fn node)
       (Offspring.fold (fun o \rightarrow Offspring.add (fo node o))
           offspring Offspring.empty)) dag Parents.empty
let fold\_nodes f dag acc =
  Parents.fold (fun n \rightarrow f n) dag acc
let fold f dag acc =
  Parents.fold (fun node \rightarrow Offspring.fold (f node)) dag acc
```



Note that in it's current incarnation, fold $add_offspring\ dag\ empty$ copies only the fertile nodes, while fold $add_offspring\ dag\ (fold_nodes\ add_node\ dag\ empty)$ includes sterile ones, as does $map\ (fun\ n\ \to\ n)\ (fun\ n\ ec\ \to\ ec)\ dag.$

```
let dependencies dag node =
       let rec dependencies' node' =
          let offspring = Parents.find node' dag in
          if Offspring.is_empty offspring then
             Tree2.leaf node'
          else
             Tree 2.cons
               (Offspring.fold
                   (fun o \ acc \rightarrow
                      (fst \ o,
                       node',
                       F.fold~(\mathsf{fun}~wf~acc'~\rightarrow~dependencies'~wf~::~acc')~o~[\,])~::~acc)
        dependencies' node
     let lists dag =
       List.sort (fun (n1, \_) (n2, \_) \rightarrow F.Nodes.compare n1 n2)
          (Parents.fold (fun node offspring l \rightarrow
             (node, Offspring.elements offspring) :: l) dag [])
     let size \ dag =
        Parents.fold (fun \_ \_ n \rightarrow succ n) dag 0
     let rec harvest dag node roots =
        O\!f\!f\!spring.fold
          (fun offspring\ roots' \rightarrow
             if is_offspring node offspring roots' then
               roots'
             else
               F.fold\ (harvest\ dag)
                  offspring (add_offspring_unsafe node offspring roots'))
          (Parents.find node dag) (add_node node roots)
     let harvest_list dag nodes =
        List.fold\_left (fun roots node \rightarrow harvest dag node roots) empty nodes
Build a closure once, so that we can recurse faster:
     let eval f mule muln add null unit node dag =
       let rec eval' n =
          if is\_sterile \ n \ dag then
            f n
          else
             Offspring.fold
               (fun (e, \_ as \mathit{offspring}) \ v\theta \ 	o
                  add (mule n \in (F.fold \ muln' \ offspring \ unit)) \ v\theta)
               (Parents.find n dag) null
       and muln' \ n \ = \ muln \ (eval' \ n) in
        eval' node
     let count\_trees node dag =
        eval (fun \_ \rightarrow 1) (fun \_ \_ p \rightarrow p) ( \times ) (+) 0 1 node dag
     let \ build\_forest \ evaluator \ node \ dag =
        evaluator (fun n \rightarrow [Tree.leaf (n, None) n])
          (fun n \ e \ p \rightarrow List.map (fun p' \rightarrow Tree.cons (n, Some \ e) \ p') \ p)
          (\mathsf{fun}\ p1\ p2\ \to\ Product.fold2\ (\mathsf{fun}\ n\ nl\ pl\ \to\ (n\ ::\ nl)\ ::\ pl)\ p1\ p2\ [])
          (@) [] [[]] node dag
     let forest = build\_forest eval
At least for count_trees, the memoizing variant eval_memoized is considerably slower than direct recursive
evaluation with eval.
     let eval_offspring f mule muln add null unit dag values (node, offspring) =
       let muln' n = muln (Parents.find n values) in
```

```
let v =
         if is_sterile node dag then
           f node
         else
            Offspring.fold
              (fun (e, \_ as offspring) v\theta \rightarrow
                 add (mule node e (F.fold muln' offspring unit)) v\theta)
              offspring null
       in
       (v, Parents.add node v values)
    let eval\_memoized' f mule muln add null unit dag =
       let result, _{-} =
         List.fold\_left
            (\text{fun } (v, values) \rightarrow eval\_offspring f mule muln add null unit dag values)
            (null, Parents.empty)
            (List.sort (fun (n1, \_) (n2, \_) \rightarrow F.Nodes.compare n1 n2)
               (Parents.fold
                   (fun node offspring l \rightarrow (node, offspring) :: l) dag [])) in
       result
    let eval_memoized f mule muln add null unit node dag =
       eval_memoized' f mule muln add null unit
         (harvest dag node empty)
    let\ forest\_memoized\ =\ build\_forest\ eval\_memoized
  end
module type Graded =
  sig
    include T
    type rank
    val \ rank : node \rightarrow rank
    val \ ranks : t \rightarrow rank \ list
    val min_max_rank : t \rightarrow rank \times rank
    \mathsf{val}\ \mathit{ranked}\ :\ \mathit{rank}\ \to\ t\ \to\ \mathit{node}\ \mathit{list}
  end
module\ Graded\ (F:\ Graded\_Forest)\ =\ Maybe\_Graded\ (Graded\_Map)(F)
The following is not a graded map, obviously. But it can pass as one by the typechecker for constructing
non-graded DAGs.
module Fake\_Graded\_Map (O : Graded\_Ord) :
     Graded\_Map with type key = O.t and type rank = O.G.t =
  struct
     module M = Map.Make(O)
    type key = O.t
    \mathsf{type}\ (+\alpha)\ t\ =\ \alpha\ M.t
    let empty = M.empty
    let add = M.add
    let find = M.find
    let mem = M.mem
    let iter = M.iter
    let fold = M.fold
We make sure that the remaining three are never called inside DAG and are not visible outside.
    type rank = O.G.t
    exception Impossible of string
    let ranks _ = raise (Impossible "ranks")
    let min_max_rank _ = raise (Impossible "min_max_rank")
    let ranked _ _ = raise (Impossible "ranked")
  end
```

We could also have used signature projection with a chaotic or discrete grading, but the Graded_Map can cost some efficiency. This is probably not the case for the current simple implementation, but future embellishment can change this. Therefore, the ungraded DAG uses Map directly, without overhead.

```
\mathsf{module}\ \mathit{Make}\ (F\ :\ \mathit{Forest})\ =
   Maybe\_Graded(Fake\_Graded\_Map)(Grade\_Forest(Fake\_Grading)(F))
```



If O'Caml had polymorphic recursion, we could think of even more elegant implementations unifying nodes and offspring (cf. the generalized tries in [4]) and offspring (cf. the generalized tries in [4]).

—5— Momenta

5.1 Interface of Momentum

Model the finite combinations

$$p = \sum_{n=1}^{k} c_k \bar{p}_n, \quad \text{(with } c_k \in \{0, 1\})$$
 (5.1)

of $n_{\rm in}$ incoming and $k-n_{\rm in}$ outgoing momenta p_n

$$\bar{p}_n = \begin{cases} -p_n & \text{for } 1 \le n \le n_{\text{in}} \\ p_n & \text{for } n_{\text{in}} + 1 \le n \le k \end{cases}$$
 (5.2)

where momentum is conserved

$$\sum_{n=1}^{k} \bar{p}_n = 0 \tag{5.3}$$

below, we need the notion of 'rank' and 'dimension':

$$dim(p) = k (5.4a)$$

$$rank(p) = \sum_{n=1}^{k} c_k \tag{5.4b}$$

where 'dimension' is not the dimension of the underlying space-time, of course.

module type T =

sig

 $\mathsf{type}\ t$

Constructor: $(k, N) \to p = \sum_{n \in N} \bar{p}_n$ and k = dim(p) is the *overall* number of independent momenta, while rank(p) = |N| is the number of momenta in p. It would be possible to fix dim as a functor argument instead. This might be slightly faster and allow a few more compile time checks, but would be much more tedious to use, since the number of particles will be chosen at runtime.

$$val\ of_ints : int \rightarrow int\ list \rightarrow t$$

No two indices may be the same. Implementions of of_ints can either raise the exception Duplicate or ignore the duplicate, but implementations of add are required to raise Duplicate.

exception Duplicate of int

Raise Range iff n > k:

exception Range of int

Binary oparations require that both momenta have the same dimension. *Mismatch* is raised if this condition is violated.

exception $\mathit{Mismatch}$ of $\mathit{string} \times t \times t$

Negative is raised if the result of sub is undefined.

exception Negative

The inverses of the constructor (we have $rank \ p = List.length \ (to_ints \ p)$, but rank might be more efficient):

val to_ints : $t \rightarrow int \ list$

Shortcuts: $singleton d p = of_ints d [p]$ and $zero d = of_ints d []$:

 $\begin{array}{lll} \mathsf{val} \ singleton \ : \ int \rightarrow \ int \rightarrow \ t \\ \mathsf{val} \ zero \ : \ int \rightarrow \ t \end{array}$

An arbitrary total order, with the condition $rank(p_1) < rank(p_2) \Rightarrow p_1 < p_2$.

 $val\ compare\ :\ t\
ightarrow\ t\
ightarrow\ int$

Use momentum conservation to construct the negative momentum with positive coefficients:

 $\mathsf{val}\ neg\ :\ t\ \to\ t$

Return the momentum or its negative, whichever has the lower rank. NB: the present implementation does not guarantee that

$$absp = absq \iff p = p \lor p = -q \tag{5.5}$$

for momenta with rank = $\dim/2$.

 $val \ abs : t \rightarrow t$

Add and subtract momenta. This can fail, since the coefficients c_k must me either 0 or 1.

Once more, but not raising exceptions this time:

Not the total order provided by compare, but set inclusion of non-zero coefficients instead:

 $\begin{array}{lll} \text{val } less \ : \ t \ \rightarrow \ t \ \rightarrow \ bool \\ \text{val } lesseq \ : \ t \ \rightarrow \ t \ \rightarrow \ bool \end{array}$

 $p_1 + (\pm p_2) + (\pm p_3) = 0$

 $val try_fusion : t \rightarrow t \rightarrow t \rightarrow (bool \times bool) option$

A textual representation for debugging:

val $to_string : t \rightarrow string$

split i n p splits \bar{p}_i into n momenta $\bar{p}_i \to \bar{p}_i + \bar{p}_{i+1} + \ldots + \bar{p}_{i+n-1}$ and makes room via $\bar{p}_{j>i} \to \bar{p}_{j+n-1}$. This is used for implementating cascade decays, like combining

$$e^{+}(p_1)e^{-}(p_2) \to W^{-}(p_3)\nu_e(p_4)e^{+}(p_5)$$
 (5.6a)

$$W^{-}(p_3) \to d(p_3')\bar{u}(p_4')$$
 (5.6b)

to

$$e^{+}(p_1)e^{-}(p_2) \to d(p_3)\bar{u}(p_4)\nu_e(p_5)e^{+}(p_6)$$
 (5.7)

in narrow width approximation for the W⁻.

 $\mathsf{val}\ split\ :\ int \to\ int \to\ t\ \to\ t$

5.1.1 Scattering Kinematics

From here on, we assume scattering kinematics $\{1,2\} \rightarrow \{3,4,\ldots\}$, i.e. $n_{\rm in}=2$.

\$

Since functions like *timelike* can be used for decays as well (in which case they must *always* return true, the representation—and consequently the constructors—should be extended by a flag discriminating between the two cases!

 $\mathsf{module}\ \mathit{Scattering}\ :$

sig

Test if the momentum is an incoming one: $p = \bar{p}_1 \vee p = \bar{p}_2$

```
\mathsf{val}\ incoming\ :\ t\ \to\ bool
p = \bar{p}_3 \vee p = \bar{p}_4 \vee \dots
              val\ outgoing\ :\ t\ 	o\ bool
p^2 \ge 0. NB: par abus de language, we report the incoming individual momenta as spacelike, instead as timelike.
This will be useful for phasespace constructions below.
              \mathsf{val}\ timelike\ :\ t\ \to\ bool
p^2 \le 0. NB: the simple algebraic criterion can be violated for heavy initial state particles.
              val\ spacelike: t \rightarrow bool
p = \bar{p}_1 + \bar{p}_2
              val s\_channel\_in : t \rightarrow bool
p = \bar{p}_3 + \bar{p}_4 + \ldots + \bar{p}_n
              val s\_channel\_out : t \rightarrow bool
p = \bar{p}_1 + \bar{p}_2 \lor p = \bar{p}_3 + \bar{p}_4 + \ldots + \bar{p}_n
              val s\_channel : t \rightarrow bool
\bar{p}_1 + \bar{p}_2 \rightarrow \bar{p}_3 + \bar{p}_4 + \ldots + \bar{p}_n
              val flip_s_channel_in : t \rightarrow t
         end
                                                     5.1.2 Decay Kinematics
     module Decay :
           sig
Test if the momentum is an incoming one: p = \bar{p}_1
              val\ incoming\ :\ t\ 	o\ bool
p = \bar{p}_2 \vee p = \bar{p}_3 \vee \dots
              val\ outgoing\ :\ t\ 	o\ bool
p^2 \ge 0. NB: here, we report the incoming individual momenta as timelike.
              val\ timelike\ :\ t\ 	o\ bool
p^2 \le 0.
              val\ spacelike\ :\ t\ 	o\ bool
           end
  end
module Lists : T
\mathsf{module}\ Bits\ :\ T
module\ Default\ :\ T
Wolfgang's funny tree codes:
                                                    (2^n, 2^{n-1}) \to (1, 2, 4, \dots, 2^{n-2})
                                                                                                                                           (5.8)
module type Whizard =
  sig
     type t
     val\ of\_momentum : t \rightarrow int
     val\ to\_momentum : int \rightarrow int \rightarrow t
   end
\mathsf{module}\ \mathit{Lists} W\ :\ \mathit{Whizard}\ \mathsf{with}\ \mathsf{type}\ t\ =\ \mathit{Lists}.t
\mathsf{module}\ BitsW\ :\ Whizard\ \mathsf{with}\ \mathsf{type}\ t\ =\ Bits.t
module\ DefaultW\ :\ Whizard\ with\ type\ t\ =\ Default.t
```

5.2 Implementation of Momentum

```
module type T =
   sig
      type t
      \mathsf{val}\ of\_ints\ :\ int\ \rightarrow\ int\ list\ \rightarrow\ t
      exception Duplicate of int
      exception Range of int
      exception Mismatch of string \times t \times t
      exception Negative
      val to\_ints : t \rightarrow int \ list
      val \ dim : t \rightarrow int
      val rank : t \rightarrow int
      \mathsf{val}\ singleton\ :\ int \to\ int \to\ t
      \mathsf{val}\ \mathit{zero}\ :\ \mathit{int}\ \rightarrow\ \mathit{t}
      \mathsf{val}\ compare\ :\ t\ \to\ t\ \to\ int
      \mathsf{val}\ neg\ :\ t\ \to\ t
      \mathsf{val}\ abs\ :\ t\ \to\ t
      \mathsf{val}\ add\ :\ t\ \to\ t\ \to\ t
      \mathsf{val}\ sub\ :\ t\ \to\ t\ \to\ t
      val try\_add: t \rightarrow t \rightarrow t \ option
      val try\_sub : t \rightarrow t \rightarrow t option
      \mathsf{val}\ less\ :\ t\ \to\ t\ \to\ bool
      \mathsf{val}\ lesseq\ :\ t\ \to\ t\ \to\ bool
      \mathsf{val}\ try\_fusion\ :\ t\ \to\ t\ \to\ t\ \to\ (bool\times bool)\ option
      val to\_string : t \rightarrow string
      \mathsf{val}\ split\ :\ int \to\ int \to\ t \ \to\ t
      module Scattering :
             sig
                val\ incoming\ :\ t\ 	o\ bool
                val\ outgoing: t \rightarrow bool
                \mathsf{val}\ timelike\ :\ t\ \to\ bool
                \mathsf{val}\ spacelike\ :\ t\ \to\ bool
                \mathsf{val}\ s\_channel\_in\ :\ t\ \to\ bool
                val s\_channel\_out : t \rightarrow bool
                \mathsf{val}\ s\_channel\ :\ t\ \to\ bool
                val\ flip\_s\_channel\_in : t \rightarrow t
             end
      module Decay :
             sig
                val\ incoming\ :\ t\ 	o\ bool
                val\ outgoing\ :\ t\ 	o\ bool
                val\ timelike: t \rightarrow bool
                val\ spacelike\ :\ t\ 	o\ bool
             end
   end
```

5.2.1 Lists of Integers

The first implementation (as part of *Fusion*) was based on sorted lists, because I did not want to preclude the use of more general indices that integers. However, there's probably not much use for this generality (the indices are typically generated automatically and integer are the most natural choice) and it is no longer supported. by the current signature. Thus one can also use the more efficient implementation based on bitvectors below.

```
\begin{array}{ll} \mathsf{module}\ Lists \ = \\ \mathsf{struct} \\ \\ \mathsf{type}\ t \ = \ \{\ d\ :\ int;\ r\ :\ int;\ p\ :\ int\ list\ \} \\ \\ \mathsf{exception}\ Range\ \mathsf{of}\ int \\ \mathsf{exception}\ Duplicate\ \mathsf{of}\ int \end{array}
```

```
\mathsf{let}\ \mathsf{rec}\ \mathit{check}\ d\ =\ \mathsf{function}
   p1 :: p2 :: \_ when p2 \le p1 \rightarrow raise (Duplicate p1)
     p1 \ :: \ (p2 \ :: \ \_ \ \mathsf{as} \ \mathit{rest}) \ \to \ \mathit{check} \ d \ \mathit{rest}
     [p] when p < 1 \lor p > d \rightarrow raise (Range p)
     [p] \rightarrow ()
   | [] \rightarrow ()
let of_ints d p =
   let p' = List.sort compare p in
   check d p';
   \{d = d; r = List.length p; p = p'\}
let to_ints p = p.p
let dim p = p.d
\mathsf{let}\ \mathit{rank}\ p\ =\ p.r
let zero d = \{ d = d; r = 0; p = [] \}
let singleton \ d \ p = \{ \ d = d; \ r = 1; \ p = [p] \}
let to\_string p =
   "[" \hat{S}tring.concat "," (List.map string\_of\_int p.p) \hat{}
   "/" ^{\circ} string\_of\_int p.r <math>^{\circ} "/" ^{\circ} string\_of\_int p.d <math>^{\circ} "]"
exception Mismatch of string \times t \times t
let mismatch \ s \ p1 \ p2 = raise \ (Mismatch \ (s, \ p1, \ p2))
\mathsf{let} \ \mathit{matching} \ f \ s \ \mathit{p1} \ \mathit{p2} \ = \\
   if p1.d = p2.d then
     f p1 p2
   else
      mismatch s p1 p2
let compare p1 p2 =
   if p1.d = p2.d then begin
     \mathsf{let}\ c\ =\ compare\ p1.r\ p2.r\ \mathsf{in}
     if c \neq 0 then
        c
     else
         compare p1.p p2.p
   end else
      mismatch "compare" p1 p2
let rec neg' d i = function
  | [] \rightarrow
        if i \leq d then
          i :: neg' \ d \ (succ \ i) \ []
        else
           \mid i' :: rest \text{ as } p \rightarrow
        if i' > d then
           failwith "Integer_List.neg:uinternaluerror"
         else if i' = i then
           neg' \ d \ (succ \ i) \ rest
           i :: neg' \ d \ (succ \ i) \ p
let neg \ p \ = \ \{ \ d \ = \ p.d; \ r \ = \ p.d \ - \ p.r; \ p \ = \ neg' \ p.d \ 1 \ p.p \ \}
let abs p =
   if 2 \times p.r > p.d then
     neg p
   else
let rec add' p1 p2 =
   match p1, p2 with
   | [], p \rightarrow p
```

```
p, [] \rightarrow p
  | x1 :: p1', x2 :: p2' \rightarrow
      if x1 < x2 then
        x1 :: add' p1' p2
       else if x2 < x1 then
         x2 :: add' p1 p2'
         raise (Duplicate x1)
let add p1 p2 =
  if p1.d = p2.d then
    \{d = p1.d; r = p1.r + p2.r; p = add' p1.p p2.p\}
  else
     mismatch "add" p1 p2
let rec try_add' d r acc p1 p2 =
  match p1, p2 with
   [\ ],\ p \rightarrow Some\ (\{\ d=d;\ r=r;\ p=List.rev\_append\ acc\ p\ \})
  [p, [] \rightarrow Some (\{d = d; r = r; p = List.rev\_append acc p\})]
  | x1 :: p1', x2 :: p2' \rightarrow
      if x1 < x2 then
         try\_add' d r (x1 :: acc) p1' p2
       else if x2 < x1 then
         try\_add' d r (x2 :: acc) p1 p2'
       else
         None
let try_add p1 p2 =
  if p1.d = p2.d then
     try_add' \ p1.d \ (p1.r + p2.r) \ [] \ p1.p \ p2.p
  else
     mismatch "try_add" p1 p2
exception Negative
let rec sub' p1 p2 =
  match p1, p2 with
  p, [] \rightarrow p
  | [], \_ \rightarrow raise\ Negative
  | x1 :: p1', x2 :: p2' \rightarrow
       if x1 < x2 then
         x1 :: sub' p1' p2
       else if x1 = x2 then
         sub'\ p1'\ p2'
       else
         raise Negative
let rec sub p1 p2 =
  if p1.d = p2.d then begin
    if p1.r \ge p2.r then
       \{d = p1.d; r = p1.r - p2.r; p = sub' p1.p p2.p\}
    else
       neg (sub p2 p1)
   end else
    mismatch "sub" p1 p2
let rec try\_sub' d r acc p1 p2 =
  match p1, p2 with
  [\ p,\ [\ ]\ \rightarrow\ Some\ (\{\ d\ =\ d;\ r\ =\ r;\ p\ =\ List.rev\_append\ acc\ p\ \})
  | [], \_ \rightarrow None
  | x1 :: p1', x2 :: p2' \rightarrow
       if x1 < x2 then
         try\_sub' d r (x1 :: acc) p1' p2
       else if x1 = x2 then
         try\_sub' d r acc p1' p2'
```

```
else
             None
let try\_sub p1 p2 =
   if p1.d = p2.d then begin
      if p1.r \ge p2.r then
          try\_sub' p1.d (p1.r - p2.r) [] p1.p p2.p
      else
          match try\_sub' p1.d (p2.r - p1.r) [] p2.p p1.p with
          | None \rightarrow None
          \mid Some p \rightarrow Some (neg p)
   end else
      mismatch "try_sub" p1 p2
let rec less' equal p1 p2 =
   match p1, p2 with
   [], [] \rightarrow \neg equal
   |~[],~\_ \to {\sf true}
   | x1 :: \_, [] \rightarrow \mathsf{false}
   | \hspace{.1cm} x1 \hspace{.1cm} :: \hspace{.1cm} p1', \hspace{.1cm} x2 \hspace{.1cm} :: \hspace{.1cm} p2' \hspace{.1cm} \text{ when } \hspace{.1cm} x1 \hspace{.1cm} = \hspace{.1cm} x2 \hspace{.1cm} \rightarrow \hspace{.1cm} less' \hspace{.1cm} equal \hspace{.1cm} p1' \hspace{.1cm} p2'
   x1 :: p1', x2 :: p2' \rightarrow less'  false p1 p2'
let less p1 p2 =
   if p1.d = p2.d then
      less' \ {\rm true} \ p1.p \ p2.p
   else
      mismatch "sub" p1 p2
let rec lesseq' p1 p2 =
   match p1, p2 with
   | [], \_ \rightarrow \mathsf{true}
     x1 :: \_, [] \rightarrow \mathsf{false}
   \mid x1 :: p1', x2 :: p2' \text{ when } x1 = x2 \rightarrow lesseq' p1' p2'
   | x1 :: p1', x2 :: p2' \rightarrow lesseq' p1 p2'
let lesseq p1 p2 =
   if p1.d = p2.d then
      lesseq' p1.p p2.p
      mismatch "lesseq" p1 p2
module Scattering =
   struct
      \mathsf{let} \ incoming \ p \ = \\
         if p.r = 1 then
            match p.p with
             \mid \ [1] \ \mid \ [2] \ 	o \ \mathsf{true}
              _{-} 
ightarrow false
         else
            false
      \mathsf{let} \ \mathit{outgoing} \ p \ = \\
         if p.r = 1 then
            match p.p with
             | [1] | [2] \rightarrow \mathsf{false}
               _{-} 
ightarrow true
         else
            false
      let s\_channel\_in p =
         match p.p with
          | [1; 2] \rightarrow \mathsf{true}
          \mid \ \_ \rightarrow \mathsf{false}
      let rec s\_channel\_out' d i = function
```

```
| \ [] \rightarrow i = succ d
            i' :: p when i' = i \rightarrow s\_channel\_out' d (succ i) <math>p
         \mid \ \_ \rightarrow \mathsf{false}
      let s\_channel\_out p =
         match p.p with
         \mid 3 :: p' \rightarrow s\_channel\_out' p.d 4 p'
         \mid \ \_ \rightarrow \mathsf{false}
      \mathsf{let}\ s\_channel\ p\ =\ s\_channel\_in\ p\ \lor\ s\_channel\_out\ p
      let timelike p =
         match p.p with
         | \ p1 \ :: \ p2 \ :: \ \_ \ \to \ p1 \ > \ 2 \ \lor \ (p1 \ = \ 1 \ \land \ p2 \ = \ 2)
         | p1 :: \_ \rightarrow p1 > 2
         | [] \rightarrow \mathsf{false}
      let spacelike p = \neg (timelike p)
      let flip_s_channel_in p =
         if s\_channel\_in\ p then
            neg\ (of\_ints\ p.d\ [1;2])
         else
            p
   end
module Decay =
   struct
      \mathsf{let} \ incoming \ p \ = \\
         if p.r = 1 then
            match p.p with
            | [1] \rightarrow \mathsf{true}
            \mid \ \_ \ 	o \ \mathsf{false}
         else
            false
      \mathsf{let}\ \mathit{outgoing}\ p\ =
         \text{if } p.r \ = \ 1 \text{ then} \\
            match p.p with
             | [1] \rightarrow \mathsf{false}
            \mid \_ \rightarrow true
         else
            false
      let timelike p =
         match p.p with
         | [1] \rightarrow \mathsf{true}
         | p1 :: \_ \rightarrow p1 > 1
         | [] \rightarrow \mathsf{false}
      \mathsf{let} \; \mathit{spacelike} \; p \; = \; \neg \; (\mathit{timelike} \; p)
   end
let test\_sum \ p \ inv1 \ p1 \ inv2 \ p2 =
   if p.d = p1.d then begin
      if p.d = p2.d then begin
         match (if inv1 then try\_add else try\_sub) p p1 with
         | None \rightarrow false
         \mid Some p' \rightarrow
               begin match (if inv2 then try\_add else try\_sub) p' p2 with
                | None \rightarrow false
                | Some p'' \rightarrow p''.r = 0 \lor p''.r = p.d
                end
      end else
         mismatch "test_sum" p p2
```

```
end else
       mismatch "test_sum" p p1
  let try\_fusion p p1 p2 =
     if test\_sum p false p1 false p2 then
        Some (false, false)
     else if test\_sum p true p1 false p2 then
        Some (true, false)
     else if test\_sum p false p1 true p2 then
       Some (false, true)
     else if test\_sum p true p1 true p2 then
       Some (true, true)
     else
        None
  \mathsf{let}\ split\ i\ n\ p\ =
     let n' = n - 1 in
     let rec split' head = function
       [] \rightarrow (p.r, List.rev head)
       | i1 :: ilist \rightarrow
            if i1 < i then
               split' (i1 :: head) ilist
            else if i1 > i then
               (p.r, List.rev\_append head (List.map ((+) n') (i1 :: ilist)))
            else
               (p.r + n',
                List.rev_append head
                   ((\mathit{ThoList.range}\ i1\ (i1\ +\ n'))\ @\ (\mathit{List.map}\ ((+)\ n')\ ilist)))\ \mathsf{in}
     let r', p' = split' [] p.p in
     \{ d = p.d + n'; r = r'; p = p' \}
end
```

5.2.2 Bit Fiddlings

Bit vectors are popular in Fortran based implementations [1, 2, 11] and can be more efficient. In particular, when all infomation is packed into a single integer, much of the memory overhead is reduced.

```
\begin{array}{rcl} \text{module } Bits & = \\ \text{struct} & \\ & \text{type } t = int \end{array}
```

Bits 1...21 are used as a bitvector, indicating whether a particular momentum is included. Bits 22...26 represent the numbers of bits set in bits 1...21 and bits 27...31 denote the maximum number of momenta.

```
let mask\ n=(1\ lsl\ n)-1
let mask2=mask\ 2
let mask5=mask\ 5
let mask21=mask\ 21
let mask4=mask5\ lsl\ 26
let maskr=mask5\ lsl\ 21
let maskb=mask21
let dim0\ p=p\ land\ maskd
let rank0\ p=p\ land\ maskr
let bits0\ p=p\ land\ maskb
let dim\ p=(dim0\ p)\ lsr\ 26
let rank\ p=(rank0\ p)\ lsr\ 21
let bits\ p=bits0\ p
let drb0\ d\ r\ b=d\ lor\ r\ lor\ b
let drb0\ d\ r\ b=d\ lsl\ 26\ lor\ r\ lsl\ 21\ lor\ b
```

For a 64-bit architecture, the corresponding sizes could be increased to $1 \dots 51, 52 \dots 57$, and $58 \dots 63$. However, the combinatorical complexity will have killed us long before we can reach these values.

```
exception Range of int
exception Duplicate of int
exception Mismatch of string \times t \times t
let mismatch \ s \ p1 \ p2 = raise \ (Mismatch \ (s, \ p1, \ p2))
let of_ints d p =
  let r = List.length p in
  if d \leq 21 \wedge r \leq 21 then begin
     List.fold\_left (fun b p' \rightarrow
       if p' \leq d then
          b \text{ lor } (1 \text{ lsl } (pred p'))
          raise (Range p')) (drb d r 0) p
  end else
     raise (Range r)
let zero d = drb d 0 0
let singleton d p = drb d 1 (1 lsl (pred p))
let rec to\_ints' acc p b =
  if b = 0 then
     List.rev acc
  else if (b \text{ land } 1) = 1 \text{ then}
     to\_ints' (p :: acc) (succ p) (b | sr 1)
     to\_ints' \ acc \ (succ \ p) \ (b \ lsr \ 1)
let to\_ints p = to\_ints' [] 1 (bits p)
let to\_string p =
   "[" ^ String.concat "," (List.map string_of_int (to_ints p)) ^
   "/" \hat{string\_of\_int} (rank p) \hat{"}" \hat{string\_of\_int} (dim p) \hat{"}"]"
let compare p1 p2 =
  if dim0 \ p1 = dim0 \ p2 then begin
     let c = compare (rank0 p1) (rank0 p2) in
     if c \neq 0 then
        c
     else
        compare (bits p1) (bits p2)
  end else
     mismatch "compare" p1 p2
  let d = dim p and r = rank p in
  drb \ d \ (d - r) \ ((mask \ d) \ land \ (lnot \ p))
let abs p =
  if 2 \times (rank \ p) > dim \ p then
     neg p
  else
     p
let add p1 p2 =
  let d1 = dim0 \ p1 and d2 = dim0 \ p2 in
  if d1 = d2 then begin
     let b1 = bits p1 and b2 = bits p2 in
     if b1 land b2 = 0 then
        drb0 \ d1 \ (rank0 \ p1 + rank0 \ p2) \ (b1 \ lor \ b2)
        raise (Duplicate 0)
  end else
```

```
mismatch "add" p1 p2
exception Negative
let rec sub p1 p2 =
  let d1 = dim0 \ p1 and d2 = dim0 \ p2 in
  if d1 = d2 then begin
    let r1 = rank0 \ p1 and r2 = rank0 \ p2 in
    if r1 \geq r2 then begin
       let b1 = bits p1 and b2 = bits p2 in
       if b1 \text{ lor } b2 = b1 \text{ then}
         drb0 \ d1 \ (r1 - r2) \ (b1 \ \text{lxor} \ b2)
         raise\ Negative
    end else
       neg (sub p2 p1)
  end else
     mismatch "sub" p1 p2
let try_add p1 p2 =
  let d1 = dim0 \ p1 and d2 = dim0 \ p2 in
  if d1 = d2 then begin
    let b1 = bits \ p1 and b2 = bits \ p2 in
    if b1 land b2 = 0 then
       Some (drb0 \ d1 \ (rank0 \ p1 + rank0 \ p2) \ (b1 \ lor \ b2))
    else
       None
  end else
    mismatch "try_add" p1 p2
let rec try\_sub p1 p2 =
  let d1 = dim0 \ p1 and d2 = dim0 \ p2 in
  if d1 = d2 then begin
    let r1 = rank0 \ p1 and r2 = rank0 \ p2 in
    if r1 \geq r2 then begin
       let b1 = bits p1 and b2 = bits p2 in
       if b1 \text{ lor } b2 = b1 \text{ then}
         Some (drb0 \ d1 \ (r1 - r2) \ (b1 \ \mathsf{lxor} \ b2))
       else
         None
     end else
       begin match try\_sub p2 p1 with
       | Some p \rightarrow Some (neg p) |
       |\ None\ 	o\ None
       end
  end else
     mismatch "sub" p1 p2
let lesseq p1 p2 =
  let d1 = dim0 \ p1 and d2 = dim0 \ p2 in
  if d1 = d2 then begin
    let r1 = rank0 \ p1 and r2 = rank0 \ p2 in
    if r1 \le r2 then begin
       let b1 = bits p1 and b2 = bits p2 in
       b1 \text{ lor } b2 = b2
    end else
       false
  end else
     mismatch "less" p1 p2
let less p1 p2 = p1 \neq p2 \land lesseq p1 p2
let mask_in1 = 1
\mathsf{let}\ mask\_in2\ =\ 2
let <math>mask\_in = mask\_in1 lor mask\_in2
```

```
module Scattering =
  struct
     \mathsf{let} \ incoming \ p \ = \\
       rank \ p = 1 \ \land \ (mask\_in \ land \ p \neq 0)
     let outgoing p =
       rank p = 1 \land (mask\_in \text{ land } p = 0)
     let timelike p =
       (rank \ p > 0 \land (mask\_in \ land \ p = 0)) \lor (bits \ p = mask\_in)
     let spacelike p =
       (rank \ p > 0) \land \neg (timelike \ p)
     let s\_channel\_in p =
        bits p = mask\_in
     let s_-channel_-out p =
       rank p > 0 \land (mask\_in | kor p = 0)
     let s\_channel p =
       s\_channel\_in p \lor s\_channel\_out p
     let flip_s_channel_in p =
       if s\_channel\_in p then
          neg p
       else
          p
  end
module Decay =
  struct
     let incoming p =
       rank \ p = 1 \ \land \ (mask\_in1 \ land \ p = mask\_in1)
     let outgoing p =
       rank p = 1 \land (mask\_in1 \text{ land } p = 0)
     let timelike p =
       incoming \ p \ \lor \ (rank \ p \ > \ 0 \ \land \ mask\_in1 \ land \ p \ = \ 0)
     let spacelike p =
        \neg (timelike p)
  end
let test\_sum p inv1 p1 inv2 p2 =
  let d = dim p in
  if d = dim \ p1 then begin
     if d = dim \ p2 then begin
       match (if inv1 then try\_add else try\_sub) p p1 with
        | None \rightarrow false
        \mid Some p' \rightarrow
            begin match (if inv2 then try\_add else try\_sub) p' p2 with
             | None \rightarrow false
             \mid Some p'' \rightarrow
                 let r = rank p'' in
                  r = 0 \lor r = d
            end
     end else
       mismatch "test_sum" p p2
  end else
     mismatch "test_sum" p p1
let try\_fusion p p1 p2 =
  if test\_sum\ p false p1 false p2 then
     Some (false, false)
```

```
else if test\_sum p true p1 false p2 then
          Some (true, false)
       else if test\_sum p false p1 true p2 then
          Some (false, true)
       else if test\_sum p true p1 true p2 then
          Some (true, true)
       else
          None
First create a gap of size n-1 and subsequently fill it if and only if the bit i was set.
     let split i n p =
       let \ delta_{-}d = n - 1
       and b = bits p in
       let mask\_low = mask (pred i)
       and mask_i = 1 Isl (pred i)
       and mask\_high = lnot (mask i) in
       let b\_low = mask\_low land b
       and b_med, delta_r =
          if mask_i land b \neq 0 then
             ((mask \ n) \ lsl \ (pred \ i), \ delta\_d)
          else
             (0, 0)
       and b_high =
          if delta_{-}d > 0 then
             (mask\_high \ land \ b) \ lsl \ delta\_d
          else if delta_{-}d = 0 then
             mask\_high land b
          else
             (mask\_high \ land \ b) \ lsr \ (-delta\_d) \ in
        drb \ (dim \ p + delta\_d) \ (rank \ p + delta\_r) \ (b\_low \ lor \ b\_med \ lor \ b\_high)
  end
                                                    5.2.3
                                                              Whizard
module type Whizard =
  sig
     type \ t
     val\ of\_momentum: t \rightarrow int
     val to\_momentum : int \rightarrow int \rightarrow t
  end
module BitsW =
  struct
     type t = Bits.t
     open Bits (* NB: this includes the internal functions not in T! *)
     let of\_momentum p =
       \mathsf{let}\ d\ =\ dim\ p\ \mathsf{in}
       let bit_in1 = 1 land p
       and bit_in2 = 1 land (p | sr 1)
       and bits\_out = ((mask \ d) \ land \ p) \ lsr \ 2 in
       bits\_out \text{ for } (bit\_in1 \text{ Isl } (d-1)) \text{ for } (bit\_in2 \text{ Isl } (d-2))
     let rec count\_non\_zero' acc i last b =
       if i > last then
       else if (1 \text{ Isl } (pred i)) \text{ land } b = 0 \text{ then }
          count_non_zero' acc (succ i) last b
          count_non_zero' (succ acc) (succ i) last b
     let count\_non\_zero first last b =
```

```
\begin{array}{lll} count\_non\_zero' \ 0 \ first \ last \ b \\ & | \mathsf{let} \ to\_momentum \ d \ w \ = \\ & | \mathsf{let} \ bit\_in1 \ = \ 1 \ \mathsf{land} \ (w \ \mathsf{lsr} \ (d \ - \ 1)) \\ & | \mathsf{and} \ bit\_in2 \ = \ 1 \ \mathsf{land} \ (w \ \mathsf{lsr} \ (d \ - \ 2)) \\ & | \mathsf{and} \ bits\_out \ = \ (mask \ (d \ - \ 2)) \ \mathsf{land} \ w \ \mathsf{in} \\ & | \mathsf{let} \ b \ = \ (bits\_out \ \mathsf{lsl} \ 2) \ \mathsf{lor} \ bit\_in1 \ \mathsf{lor} \ (bit\_in2 \ \mathsf{lsl} \ 1) \ \mathsf{in} \\ & | drb \ d \ (count\_non\_zero \ 1 \ d \ b) \ b \\ & \mathsf{end} \end{array}
```

The following would be a tad more efficient, if coded directly, but there's no point in wasting effort on this.

```
\begin{array}{lll} \operatorname{module}\ ListsW &= \\ & \operatorname{struct} \\ & \operatorname{type}\ t &= Lists.t \\ & \operatorname{let}\ of\_momentum\ p &= \\ & BitsW.of\_momentum\ (Bits.of\_ints\ p.Lists.d\ p.Lists.p) \\ & \operatorname{let}\ to\_momentum\ d\ w &= \\ & Lists.of\_ints\ d\ (Bits.to\_ints\ (BitsW.to\_momentum\ d\ w)) \\ & \operatorname{end} \end{array}
```

5.2.4 Suggesting a Default Implementation

Lists is better tested, but the more recent Bits appears to work as well and is much more efficient, resulting in a relative factor of better than 2. This performance ratio is larger than I had expected and we are not likely to reach its limit of 21 independent vectors anyway.

```
\begin{array}{lll} \text{module } Default &=& Bits \\ \text{module } DefaultW &=& BitsW \end{array}
```

—6— Cascades

6.1 Interface of Cascade_syntax

```
type ('flavor, 'p, 'constant) t =
       True
       False
       On\_shell of 'flavor list \times 'p
       On\_shell\_not of 'flavor list \times 'p
       Off\_shell of 'flavor list \times 'p
       Off\_shell\_not of 'flavor list \times 'p
       Gauss of 'flavor list \times 'p
       Gauss\_not of 'flavor list \times 'p
       Any_flavor of 'p
       And of ('flavor, 'p, 'constant) t list
       X_Flavor of 'flavor list
       X\_Vertex of 'constant list \times 'flavor list list
val mk\_true : unit \rightarrow ('flavor, 'p, 'constant) t
val mk\_false : unit \rightarrow ('flavor, 'p, 'constant) t
\mathsf{val}\ \mathit{mk\_on\_shell}\ :\ \mathit{`flavor}\ \mathit{list}\ \to\ \mathit{`p}\ \to\ (\mathit{`flavor},\ \mathit{`p},\ \mathit{'constant})\ t
\mathsf{val}\ \mathit{mk\_on\_shell\_not}\ :\ \mathit{`flavor}\ \mathit{list}\ \to\ \mathit{`p}\ \to\ (\mathit{`flavor},\ \mathit{`p},\ \mathit{`constant})\ \mathit{t}
\mathsf{val}\ \mathit{mk\_off\_shell}\ :\ \mathit{`flavor}\ \mathit{list}\ \to\ \mathit{`p}\ \to\ (\mathit{`flavor},\ \mathit{'p},\ \mathit{'constant})\ \mathit{t}
\mathsf{val}\ \mathit{mk\_off\_shell\_not}\ :\ \mathit{`flavor}\ \mathit{list}\ \to\ \mathit{`p}\ \to\ (\mathit{`flavor},\ \mathit{`p},\ \mathit{`constant})\ \mathit{t}
val mk\_gauss : 'flavor list \rightarrow 'p \rightarrow ('flavor, 'p, 'constant) t
\mathsf{val}\ \mathit{mk\_gauss\_not}\ :\ \mathit{`flavor}\ \mathit{list}\ \to\ \mathit{`p}\ \to\ (\mathit{`flavor},\ \mathit{`p},\ \mathit{'constant})\ \mathit{t}
val mk\_any\_flavor : 'p \rightarrow ('flavor, 'p, 'constant) t
val mk\_and : ('flavor, 'p, 'constant) t \rightarrow
   (\textit{'flavor}, \textit{'p}, \textit{'constant}) \ t \ \rightarrow \ (\textit{'flavor}, \textit{'p}, \textit{'constant}) \ t
\mathsf{val}\ \mathit{mk\_x\_flavor}\ :\ \mathit{`flavor}\ \mathit{list}\ \to\ (\mathit{`flavor},\ \mathit{'p},\ \mathit{'constant})\ \mathit{t}
val mk\_x\_vertex : 'constant list \rightarrow 'flavor list list \rightarrow
    ('flavor, 'p, 'constant) t
\mathsf{val}\ to\_string\ :\ (\mathit{`flavor}\ \to\ string)\ \to\ (\mathit{`p}\ \to\ string)\ \to
    ('constant \rightarrow string) \rightarrow ('flavor, 'p, 'constant) t \rightarrow string
exception Syntax\_Error of string \times int \times int
```

6.2 Implementation of Cascade_syntax

Concerning the Gaussian propagators, we admit the following: In principle, they would allow for flavor sums like the off-shell lines, but for all practical purposes they are used only for determining the significance of a specified intermediate state. So we select them in the same manner as on-shell states. False is probably redundant.

```
 \begin{tabular}{ll} type & ('flavor, 'p, 'constant) t & = \\ & | True \\ & | False \\ & | On\_shell \ of 'flavor \ list \times 'p \\ & | On\_shell\_not \ of 'flavor \ list \times 'p \\ & | Off\_shell \ of 'flavor \ list \times 'p \\ \end{tabular}
```

```
Off\_shell\_not of 'flavor list \times 'p
     Gauss of 'flavor list \times 'p
     Gauss\_not of 'flavor list \times 'p
     Any_-flavor of 'p
     And of ('flavor, 'p, 'constant) t list
     X_Flavor of 'flavor list
    X\_Vertex of 'constant list \times 'flavor list list
let mk\_true() = True
let mk\_false () = False
let mk\_on\_shell f p = On\_shell (f, p)
let mk\_on\_shell\_not f p = On\_shell\_not (f, p)
let mk\_off\_shell f p = Off\_shell (f, p)
let mk\_off\_shell\_not f p = Off\_shell\_not (f, p)
let mk\_gauss f p = Gauss (f, p)
let mk\_gauss\_not f p = Gauss\_not (f, p)
let mk\_any\_flavor p = Any\_flavor p
let mk\_and c1 c2 =
  match c1, c2 with
    c, True | True, c \rightarrow c
    c, False \mid False, c \rightarrow False
    And cs, And cs' \rightarrow And (cs @ cs')
    And cs, c \mid c, And cs \rightarrow And (c :: cs)
   | c, c' \rightarrow And [c; c']
let mk_x_flavor f = X_Flavor f
let mk\_x\_vertex \ c \ fs = X\_Vertex \ (c, \ fs)
let to_string flavor_to_string momentum_to_string coupling_to_string cascades =
  let flavors\_to\_string fs =
     String.concat ":" (List.map flavor_to_string fs)
  and couplings\_to\_string \ cs =
     String.concat \ ":" \ (List.map \ coupling\_to\_string \ cs) \ {\rm in}
  let rec to\_string' = function
       True \rightarrow "true"
       False 
ightarrow "false"
       On\_shell (fs, p) \rightarrow
          momentum\_to\_string\ p\ ^ " _ \sqcup = _ \sqcup "\ ^ flavors\_to\_string\ fs
     | On\_shell\_not (fs, p) \rightarrow
          momentum\_to\_string\ p\ ^{"} = !"\ ^{flavors\_to\_string\ fs}
     | Off\_shell (fs, p) \rightarrow
          momentum\_to\_string\ p\ ^ " \_ "\ ^ flavors\_to\_string\ fs
       Off\_shell\_not\ (fs,\ p)\ \rightarrow
          momentum\_to\_string\ p\ ^ " \_ " \_ ! "\ ^ flavors\_to\_string\ fs
     | Gauss (fs, p) \rightarrow
          momentum\_to\_string\ p\ ^ " \_ \# \_ "\ ^ flavors\_to\_string\ fs
       Gauss\_not (fs, p) \rightarrow
          momentum\_to\_string\ p\ ^ " \_ \# \_ ! "\ ^ flavors\_to\_string\ fs
       Any_flavor p \rightarrow
          momentum\_to\_string\ p\ ^ " _ " _ " ? "
       And cs \rightarrow
          String.concat "_\&&\\\ (List.map (fun c \rightarrow "(" ^ to_string' c ^ ")") cs)
     \mid X_{-}Flavor\ fs \rightarrow
          "!" \hat{\ } String.concat ":" (List.map \ flavor\_to\_string \ fs)
     \mid X\_Vertex\ (cs,\ fss) \rightarrow
          "^" \hat{} couplings_to_string cs \hat{}
          "[" ^ (String.concat "," (List.map flavors_to_string fss)) ^ "]"
  in
  to_string' cascades
let int\_list\_to\_string p =
  String.concat "+" (List.map string_of_int (List.sort compare p))
exception Syntax\_Error of string \times int \times int
```

6.3 Lexer

```
open Cascade\_parser
let \ unquote \ s =
  String.sub \ s \ 1 \ (String.length \ s \ - \ 2)
let digit = [,0,-,9,]
let upper = ['A'-'Z']
let \ lower = ['a'-'z']
let char = upper \mid lower
let white = [', ', '\t', '\n']
We use a very liberal definition of strings for flavor names.
rule token = parse
    white { token lexbuf } (* skip blanks *)
   '%' [^'\n']* '\n'
               { token lexbuf } (* skip comments *)
    digit^+ { INT (int\_of\_string (Lexing.lexeme \ lexbuf))} }
    '+' { PLUS }
    ':' { COLON }
        { OFFSHELL }
    '=' { ONSHELL }
    '#' { GAUSS }
    '!' \{ NOT \}
    '&' '&'? { AND }
    '(' { LPAREN }
    ')' { RPAREN }
    ',' { COMMA }
    '['] \{ LBRACKET \}
    char [^ ', ', '\t', '\n', '&', '(', '), ', [', '], ', ', ', ']*
    \{ \ STRING \ (Lexing.lexeme \ lexbuf) \ \} 
                \{ STRING (unquote (Lexing.lexeme lexbuf)) \}
   eof \{END\}
                                          6.4 Parser
                                              Header
open Cascade\_syntax
let parse\_error msg =
  raise (Syntax_Error (msg, symbol_start (), symbol_end ()))
                                        Token declarations
\%token < string > STRING
\%token < int > INT
\%token LPAREN\ RPAREN\ LBRACKET\ RBRACKET
%token AND PLUS COLON COMMA NOT HAT
%token ONSHELL OFFSHELL GAUSS
\%token END
```

%left AND

```
%left PLUS COLON COMMA %left NOT HAT %start main %type < (string, int list, string) Cascade\_syntax.t > main
```

Grammar rules

```
main ::=
    END \{ mk\_true () \}
  | cascades END { $1 }
cascades ::=
    exclusion { $1 }
   vertex { $1 }
   cascade \{ \$1 \}
   LPAREN cascades RPAREN { $2 }
  | cascades \ AND \ cascades \{ mk\_and \$1 \$3 \} |
exclusion ::=
    NOT\ string\_list\ \{\ mk\_x\_flavor\ \$2\ \}
vertex ::=
    HAT \ string\_list \{ mk\_x\_vertex \$2 [] \}
  | HAT string_list LBRACKET RBRACKET
                                        \{ mk\_x\_vertex \$2 [] \}
  | HAT LBRACKET string_lists RBRACKET
                                        \{mk\_x\_vertex[] \$3 \}
  \mid HAT string_list LBRACKET string_lists RBRACKET
                                        \{mk\_x\_vertex \$2 \$4 \}
cascade ::=
    momentum_list { mk_any_flavor $1 }
  | momentum_list ONSHELL string_list
                                        \{ mk\_on\_shell \$3 \$1 \}
  | momentum_list ONSHELL NOT string_list
                                        \{ mk\_on\_shell\_not \$4 \$1 \}
  | momentum_list OFFSHELL string_list
                                        { mk\_off\_shell \$3 \$1 }
  |\ momentum\_list\ OFFSHELL\ NOT\ string\_list
                                        { mk\_off\_shell\_not \$4 \$1 }
   momentum_list GAUSS string_list { mk_gauss $3 $1 }
   momentum_list GAUSS NOT string_list
                                        \{ mk\_gauss\_not \$4 \$1 \}
momentum\_list ::=
   momentum { [$1] }
  | momentum_list PLUS momentum { $3 :: $1 }
momentum ::=
    INT { $1 }
string\_list ::=
    STRING { [$1] }
  | string_list COLON STRING { $3 :: $1 }
```

```
 \begin{array}{l} string\_lists ::= \\ string\_list \ \{ \ [\$1] \ \} \\ | \ string\_lists \ COMMA \ string\_list \ \{ \ \$3 :: \$1 \ \} \end{array}
```

6.5 Interface of Cascade

```
\begin{array}{l} \text{module type } T &= \\ \text{sig} \\ \\ \text{type } constant \\ \text{type } flavor \\ \\ \text{type } p \\ \\ \text{type } t \\ \\ \text{val } of\_string\_list \ : \ int \rightarrow \ string \ list \rightarrow \ t \\ \\ \text{val } to\_string \ : \ t \rightarrow \ string \end{array}
```

An opaque type that describes the set of all constraints on an amplitude and how to construct it from a cascade description.

```
 \begin{tabular}{ll} {\it type } selectors \\ {\it val } to\_selectors : t \rightarrow selectors \\ \end{tabular}
```

Don't throw anything away:

```
val\ no\_cascades : selectors
```

 $select_wf \ s \ is_timelike \ f \ p \ ps$ returns true iff either

- \bullet the flavor f and momentum p match the selection s or
- all combinations of the momenta in ps are compatible, i.e. $\pm \sum p_i \leq q$.

The latter test is only required in theories with quartic or higher vertices, where ps will be the list of all incoming momenta in a fusion. $is_timelike$ is required to determine, whether particles and anti-particles should be distinct.

```
val select\_wf: selectors \to (p \to bool) \to flavor \to p \to p \ list \to bool
select\_p \ s \ p \ ps same as select\_wf \ s \ f \ p \ ps, but ignores the flavor f
val select\_p: selectors \to p \to p \ list \to bool
on\_shell \ s \ p
val on\_shell: selectors \to flavor \to p \to bool
is\_gauss \ s \ p
val is\_gauss: selectors \to flavor \to p \to bool
val select\_vtx: selectors \to constant \ Coupling.t \to flavor \to flavor \ list \to bool
```

 $partition\ s$ returns a partition of the external particles that can not be reordered without violating the cascade constraints.

```
val partition : selectors \rightarrow int list list

Diagnostics:

val description : selectors \rightarrow string option

end

module Make~(M~:~Model.T)~(P~:~Momentum.T)~:

T~ with type flavor~=~M.flavor

and type constant~=~M.constant

and type p~=~P.t
```

6.6 Implementation of Cascade

```
module type T =
  sig
     type constant
     type flavor
     type p
     type t
     val of\_string\_list : int \rightarrow string\ list \rightarrow t
     val to\_string : t \rightarrow string
     type selectors
     val\ to\_selectors : t \rightarrow selectors
     val\ no\_cascades : selectors
     \mathsf{val}\ select\_wf\ :\ selectors\ \to\ (p\ \to\ bool)\ \to\ flavor\ \to\ p\ \to\ p\ list\ \to\ bool
     \mathsf{val}\ select\_p\ :\ selectors\ \to\ p\ \to\ p\ list\ \to\ bool
     \verb|val|| on\_shell : selectors \rightarrow flavor \rightarrow p \rightarrow bool|
     val is\_gauss: selectors \rightarrow flavor \rightarrow p \rightarrow bool
     val\ select\_vtx\ :\ selectors\ 	o\ constant\ Coupling.t\ 	o
        flavor \rightarrow flavor \ list \rightarrow \ bool
     \mathsf{val}\ partition\ :\ selectors\ \to\ int\ list\ list
     val\ description: selectors \rightarrow string\ option
  end
Model.T (P:Momentum.T):
     (T with type flavor = M.flavor and type constant = M.constant and type p = P.t) =
  struct
     module CS = Cascade\_syntax
     type constant = M.constant
     type flavor = M.flavor
     type p = P.t
Since we have
                                                   p \le q \iff (-q) \le (-p)
                                                                                                                            (6.1)
```

also for \leq as set inclusion lesseq, only four of the eight combinations are independent

$$p \leq q \qquad \iff (-q) \leq (-p)$$

$$q \leq p \qquad \iff (-p) \leq (-q)$$

$$p \leq (-q) \qquad \iff q \leq (-p)$$

$$(-q) \leq p \qquad \iff (-p) \leq q$$

$$(6.2)$$

```
let one\_compatible\ p\ q = let neg\_q = P.neg\ q in P.lesseq\ p\ q\ \lor P.lesseq\ p\ neg\_q\ \lor P.lesseq\ p\ neg\_q\ p
```

'tis wasteful \dots (at least by a factor of two, because every momentum combination is generated, including the negative ones.

```
\begin{array}{l} \text{let } all\_compatible \ p \ p\_list \ q = \\ \\ \text{let } l = List.length \ p\_list \ \text{in} \\ \\ \text{if } l \leq 2 \ \text{then} \\ \\ one\_compatible \ p \ q \\ \\ \text{else} \\ \\ \text{let } tuple\_lengths = ThoList.range \ 2 \ (succ \ l \ / \ 2) \ \text{in} \\ \\ \text{let } tuples = ThoList.flatmap \ (\text{fun } n \rightarrow Combinatorics.choose \ n \ p\_list) \ tuple\_lengths \ \text{in} \\ \end{array}
```

```
let momenta = List.map \ (List.fold\_left \ P.add \ (P.zero \ (P.dim \ q))) \ tuples in List.for\_all \ (one\_compatible \ q) \ momenta
```

The following assumes that the *flavor list* is always very short. Otherwise one should use an efficient set implementation.

```
type wf =
    True
    False
    On\_shell of flavor\ list \times P.t
    On\_shell\_not of flavor\ list \times P.t
    Off_shell of flavor list \times P.t
    Off\_shell\_not of flavor list \times P.t
    Gauss of flavor list \times P.t
    Gauss\_not of flavor\ list \times P.t
    Any_flavor of P.t
    And of wf list
module Constant = Modeltools.Constant (M)
\mathsf{type}\ vtx\ =
     { couplings : M.constant list;
       fields: flavor list }
type t =
     \{ wf : wf;
       (* TODO: The following lists should be sets for efficiency. *)
       flavors : flavor list;
       vertices : vtx list }
let default =
  \{ wf = True; 
    flavors = [];
     vertices = [] 
let of string s =
  Cascade_parser.main Cascade_lexer.token (Lexing.from_string s)
```

\$

If we knew that we're dealing with a scattering, we could apply $P.flip_s_channel_in$ to all momenta, so that 1+2 accepts the particle and not the antiparticle. Right now, we don't have this information.

```
let only\_wf \ wf = \{ \ default \ with \ wf = wf \}
let cons\_and\_wf c wfs =
  match c.wf, wfs with
     True, wfs \rightarrow wfs
     False, \_ \rightarrow [False]
     wf, [] \rightarrow [wf]
     wf, wfs \rightarrow wf :: wfs
let \ and\_cascades\_wf \ c =
  match List.fold\_right\ cons\_and\_wf\ c\ [\ ] with
   | [] \rightarrow True
     [wf] \rightarrow wf
   | wfs \rightarrow And wfs
let uniq l =
   ThoList.uniq\ (List.sort\ compare\ l)
let import dim cascades =
  let rec import' = function
     \mid CS.True \rightarrow
           only_wf True
        CS.False \rightarrow
           only_wf False
     \mid CS.On\_shell (f, p) \rightarrow
```

```
only\_wf
             (On\_shell\ (List.map\ M.flavor\_of\_string\ f,\ P.of\_ints\ dim\ p))
     | CS.On\_shell\_not(f, p) \rightarrow
          only\_wf
             (On\_shell\_not\ (List.map\ M.flavor\_of\_string\ f,\ P.of\_ints\ dim\ p))
     | CS.Off\_shell (fs, p) \rightarrow
          only\_wf
             (Off\_shell\ (List.map\ M.flavor\_of\_string\ fs,\ P.of\_ints\ dim\ p))
     |CS.Off\_shell\_not(fs, p)| \rightarrow
           only\_wf
             (Off\_shell\_not\ (List.map\ M.flavor\_of\_string\ fs,\ P.of\_ints\ dim\ p))
     \mid CS.Gauss (f, p) \rightarrow
           only\_wf
             (Gauss\ (List.map\ M.flavor\_of\_string\ f,\ P.of\_ints\ dim\ p))
     | CS.Gauss\_not(f, p) \rightarrow
          only\_wf
             (Gauss\ (List.map\ M.flavor\_of\_string\ f,\ P.of\_ints\ dim\ p))
     \mid CS.Any\_flavor p \rightarrow
           only\_wf (Any\_flavor (P.of\_ints \ dim \ p))
     \mid CS.And \ cs \rightarrow
          let cs = List.map import' cs in
          \{ wf = and\_cascades\_wf cs; \}
             flavors = uniq (List.concat
                                     (List.map (fun c \rightarrow c.flavors) cs));
             vertices = uniq (List.concat
                                      (List.map (fun c \rightarrow c.vertices) cs)) 
     \mid CS.X\_Flavor\ fs \rightarrow
          let fs = List.map \ M.flavor\_of\_string \ fs in
          \{ default with flavors = uniq (fs @ List.map M.conjugate fs) \}
     \mid CS.X\_Vertex\ (cs,\ fss) \rightarrow
          let cs = List.map\ Constant.of\_string\ cs
          and fss = List.map (List.map M.flavor\_of\_string) fss in
          let expanded =
             List.map
                (\text{fun } fs \rightarrow \{ couplings = cs; fields = fs \})
                (match fss with
                 [] \rightarrow [[]] (* Subtle: not an empty list! *)
                 fss \rightarrow Product.list (fun fs \rightarrow fs) fss) in
          \{ default with vertices = expanded \}
  in
  import'\ cascades
let of\_string\_list \ dim \ strings =
  match List.map of string strings with
    [] \rightarrow default
   \mid first :: next \rightarrow
        import dim (List.fold_right CS.mk_and next first)
let flavors\_to\_string fs =
   (String.concat ":" (List.map M.flavor\_to\_string fs))
let momentum\_to\_string p =
   String.concat "+" (List.map string\_of\_int (P.to_ints p))
let rec wf_to_string = function
    True \rightarrow
        "true"
    False \rightarrow
        "false"
  \mid On\_shell (fs, p) \rightarrow
        momentum\_to\_string\ p\ ^ "_{\sqcup}=_{\sqcup}"\ ^ flavors\_to\_string\ fs
  | On\_shell\_not (fs, p) \rightarrow
        momentum\_to\_string\ p\ ^ " \_ = \_ ! "\ ^ flavors\_to\_string\ fs
```

```
| Off\_shell (fs, p) \rightarrow
         momentum\_to\_string \ p \ ^ \ " \_ ~ \_ " \ ^ flavors\_to\_string \ fs
   Off\_shell\_not\ (fs,\ p) \rightarrow
        momentum\_to\_string\ p\ ^ " \_ " \_ " ! "\ ^ flavors\_to\_string\ fs
     Gauss (fs, p) \rightarrow
         momentum\_to\_string\ p\ ^ " \_ \# \_ "\ ^ flavors\_to\_string\ fs
   | Gauss\_not (fs, p) \rightarrow
         momentum\_to\_string\ p\ ^ " \_ \# \_ ! "\ ^ flavors\_to\_string\ fs
   \mid Any\_flavor p \rightarrow
         momentum\_to\_string\ p\ ^ "_{\sqcup}"_{\sqcup}"?"
   \mid And cs \rightarrow
        String.concat "_\&&\_" (List.map (fun c \rightarrow "(" ^ wf_to_string c ^ ")") cs)
let vertex\_to\_string v =
   "^" ^ String.concat ":" (List.map M.constant_symbol v.couplings)
   "[" ^ String.concat "," (List.map M.flavor_to_string v.fields) ^ "]"
let \ vertices\_to\_string \ vs =
   (String.concat `` \bot \&\& \bot `` (List.map vertex\_to\_string vs))
let to\_string = function
   | \{ \textit{wf} = \textit{True}; \textit{flavors} = []; \textit{vertices} = [] \} \rightarrow
   | \{ wf = True; flavors = fs; vertices = [] \} \rightarrow
        "!" \hat{f}lavors\_to\_string\ fs
   | \{ wf = True; flavors = []; vertices = vs \} \rightarrow
        vertices\_to\_string\ vs
   | \{ wf = True; flavors = fs; vertices = vs \} \rightarrow
         "!" \hat{\ } flavors\_to\_string \ fs \hat{\ } " \  \  \  \&\& \sqcup " \hat{\ } vertices\_to\_string \ vs
   | \{ wf = wf; flavors = []; vertices = [] \} \rightarrow
         wf\_to\_string \ wf
   | \{ wf = wf; flavors = []; vertices = vs \} \rightarrow
         vertices\_to\_string \ vs \ ^ " \_ \& \& \_ " \ ^ wf\_to\_string \ wf
   \{ wf = wf; flavors = fs; vertices = [] \} \rightarrow
        | \ \{ \ \textit{wf} \ = \ \textit{wf} \, ; \, \textit{flavors} \ = \ \textit{fs} \, ; \, \, \textit{vertices} \ = \ \textit{vs} \ \} \ \rightarrow
        "!" ^ flavors_to_string fs ^
         "__&&__" ^ vertices_to_string vs ^
         "_{\square}&&_{\square}" ^ wf\_to\_string \ wf
type selectors =
      \{ select\_p : p \rightarrow p \ list \rightarrow bool; \}
        select\_wf: (p \rightarrow bool) \rightarrow flavor \rightarrow p \rightarrow p \ list \rightarrow bool;
         on\_shell : flavor \rightarrow p \rightarrow bool;
         is\_gauss : flavor \rightarrow p \rightarrow bool;
         select\_vtx : constant \ Coupling.t \rightarrow flavor \rightarrow flavor \ list \rightarrow bool;
         partition : int list list;
        description : string option }
let no\_cascades =
   \{ select_p = (fun_{-} \rightarrow true); \}
      select_wf = (fun_u - v - v + true);
      on\_shell = (fun \_ \_ \rightarrow false);
      is\_gauss = (fun\_\_ \rightarrow false);
      select\_vtx = (fun \_ \_ \_ \rightarrow true);
      partition = [];
      description = None
let select_p s = s.select_p
let select_wf s = s.select_wf
let on\_shell s = s.on\_shell
let is\_gauss \ s = s.is\_gauss
let select\_vtx \ s = s.select\_vtx
{\tt let} \ partition \ s \ = \ s.partition
```

```
let description s = s.description
let to\_select\_p cascades p p\_in =
  let rec to\_select\_p' = function
       \mathit{True} \ 	o \ \mathsf{true}
       False \rightarrow false
       On\_shell\ (\_,\ momentum)\ |\ On\_shell\_not\ (\_,\ momentum)
       Off_shell (_, momentum) | Off_shell_not (_, momentum)
       Gauss (_, momentum) | Gauss_not (_, momentum)
       Any\_flavor\ momentum \rightarrow all\_compatible\ p\ p\_in\ momentum
       And [] \rightarrow \mathsf{false}
       And \ cs \rightarrow List.for\_all \ to\_select\_p' \ cs \ in
   to_select_p' cascades
let to_select_wf cascades is_timelike f p p_in =
  let f' = M.conjugate f in
  let rec to\_select\_wf' = function
       \mathit{True} \ 	o \ \mathsf{true}
       False \rightarrow false
       Off\_shell\ (flavors,\ momentum) \rightarrow
          if p = momentum then
             List.mem\ f'\ flavors\ \lor\ (if\ is\_timelike\ p\ then\ false\ else\ List.mem\ f\ flavors)
          else if p = P.neq momentum then
             List.mem\ f\ flavors\ \lor\ (if\ is\_timelike\ p\ then\ false\ else\ List.mem\ f'\ flavors)
          else
             one\_compatible\ p\ momentum\ \land\ all\_compatible\ p\ p\_in\ momentum
     | On\_shell (flavors, momentum) | Gauss (flavors, momentum) \rightarrow
            if is\_timelike p then begin
              if p = momentum then
                 List.mem\ f'\ flavors
              else if p = P.neg momentum then
                 List.mem\ f\ flavors
              else
                 one\_compatible\ p\ momentum\ \land\ all\_compatible\ p\ p\_in\ momentum
            end else
              false
     Off\_shell\_not\ (flavors,\ momentum) \rightarrow
          if p = momentum then
             \neg (List.mem \ f' \ flavors \ \lor \ (if \ is\_timelike \ p \ then \ false \ else \ List.mem \ f \ flavors))
          else if p = P.neg momentum then
             \neg (List.mem \ f \ flavors \ \lor \ (if \ is\_timelike \ p \ then \ false \ else \ List.mem \ f' \ flavors))
          else
             one\_compatible\ p\ momentum\ \land\ all\_compatible\ p\ p\_in\ momentum
     | On\_shell\_not (flavors, momentum) | Gauss\_not (flavors, momentum) \rightarrow
          if is\_timelike p then begin
             if p = momentum then
                \neg (List.mem f' flavors)
             else if p = P.neg momentum then
                \neg (List.mem \ f \ flavors)
             else
                one\_compatible\ p\ momentum\ \land\ all\_compatible\ p\ p\_in\ momentum
          end else
             false
      Any_{-}flavor\ momentum \rightarrow
           one\_compatible\ p\ momentum\ \land\ all\_compatible\ p\ p\_in\ momentum
       And [] \rightarrow \mathsf{false}
      And \ cs \rightarrow List.for\_all \ to\_select\_wf' \ cs \ in
   \neg (List.mem \ f \ cascades.flavors) \land to\_select\_wf' \ cascades.wf
```

In case you're wondering: $to_on_shell\ f\ p$ and $is_gauss\ f\ p$ only search for on shell conditions and are to be used in a target, not in Fusion!

```
let to\_on\_shell cascades f p =
```

```
let f' = M.conjugate f in
  let rec to\_on\_shell' = function
       True | False | Any_flavor _
       Off\_shell(\_, \_) \mid Off\_shell\_not(\_, \_)
       Gauss (_, _) | Gauss\_not (_, _) \rightarrow false
     | On\_shell (flavors, momentum) \rightarrow |
          (p = momentum \lor p = P.neg\ momentum) \land (List.mem\ f\ flavors \lor List.mem\ f'\ flavors)
     | On\_shell\_not (flavors, momentum) \rightarrow
          (p = momentum \lor p = P.neq momentum) \land \neg (List.mem f flavors \lor List.mem f' flavors)
      And [] \rightarrow \mathsf{false}
     \mid And \ cs \rightarrow List.for\_all \ to\_on\_shell' \ cs \ in
  to_on_shell' cascades
let to\_gauss cascades f p =
  let f' = M.conjugate f in
  let rec to\_gauss' = function
       True | False | Any_flavor _
       Off\_shell(\_, \_) \mid Off\_shell\_not(\_, \_)
       On\_shell (\_, \_) \mid On\_shell\_not (\_, \_) \rightarrow false
     \mid Gauss (flavors, momentum) \rightarrow
          (p = momentum \lor p = P.neg\ momentum) \land
          (List.mem f flavors \vee List.mem f' flavors)
     \mid Gauss\_not (flavors, momentum) \rightarrow
          (p = momentum \lor p = P.neg\ momentum) \land
          \neg (List.mem \ f \ flavors \lor List.mem \ f' \ flavors)
      And [] \rightarrow \mathsf{false}
     \mid And cs \rightarrow List.for\_all \ to\_gauss' \ cs \ in
  to_gauss' cascades
module \ Fields =
  struct
     type f = M.flavor
     type c = M.constant list
     let compare = compare
     let conjugate = M.conjugate
  end
module Fusions = Modeltools.Fusions (Fields)
let dummy3 = Coupling.Scalar\_Scalar\_Scalar 1
let dummy4 = Coupling.Scalar4 1
let dummyn = Coupling.UFO (Algebra.QC.unit, "dummy", [], [], Color.Vertex.one)
```

Translate the vertices in a pair of lists: the first is the list of always rejected couplings and the second the remaining vertices suitable as input to $Fusions.of_vertices$.

```
let translate\_vertices vertices =
   List.fold\_left
      (fun (cs, (v3, v4, vn) \text{ as } acc) v \rightarrow
        match v.fields with
         [] \rightarrow (v.couplings @ cs, (v3, v4, vn))
          [\_] \mid [\_;\_] \rightarrow acc
        | [f1; f2; f3] \rightarrow
              (cs,\;(((\mathit{f1},\;\mathit{f2},\;\mathit{f3}),\;\mathit{dummy3},\;v.\mathit{couplings}) :: v3,\;v4,\;vn))
        | [f1; f2; f3; f4] \rightarrow
              (cs, (v3, ((f1, f2, f3, f4), dummy4, v.couplings) :: v4, vn))
        |fs \rightarrow (cs, (v3, v4, (fs, dummyn, v.couplings) :: vn)))|
      ([], ([], [])) vertices
let \ unpack\_constant = function
     Coupling. V3 (_, _, cs) \rightarrow cs
     Coupling. V4 (_, _, cs) \rightarrow cs
     Coupling. Vn(-, -, cs) \rightarrow cs
```

Sometimes, the empty list is a wildcard and matches any coupling:

```
let match\_coupling\ c\ cs\ =
        List.mem c cs
    let match\_coupling\_wildcard\ c\ =\ {\sf function}
       | \ | \ \rightarrow  true
       | cs \rightarrow match\_coupling \ c \ cs
    let to\_select\_vtx \ cascades =
       match cascades.vertices with
       | \ | \ | \rightarrow
             (* No vertex constraints means that we always accept. *)
             (\text{fun } c f fs \rightarrow \text{true})
       |vertices \rightarrow
             match translate_vertices vertices with
             [], ([], [], []) \rightarrow
                  (* If cascades.vertices is not empty, we mustn't get here ...*)
                 failwith "Cascade.to_select_vtx:unexpected"
            \mid couplings, ([],[],[]) \rightarrow
                  (* No constraints on the fields. Just make sure that the coupling c doesn't appear in the vetoed
couplings. *)
                  (fun c f fs \rightarrow
                    let c = unpack\_constant c in
                    \neg (match\_coupling \ c \ couplings))
             | couplings, vertices \rightarrow
                  (* Make sure that Fusions. of _vertices is only evaluated once for efficiency. *)
                 let fusions = Fusions.of_vertices vertices in
                  (fun c f fs \rightarrow
                    \mathsf{let}\ c\ =\ unpack\_constant\ c\ \mathsf{in}
                    (* Make sure that none of the vetoed couplings matches. Here an empty couplings list is not
a wildcard. *)
                    if match\_coupling \ c \ couplings then
                       false
                    else
                       (* Also make sure that none of the vetoed vertices matches. Here an empty couplings list
is a wildcard. *)
                       \neg (List.exists)
                                (fun (f', cs') \rightarrow
                                  let cs' = unpack\_constant \ cs' in
                                  f = f' \wedge match\_coupling\_wildcard \ c \ cs')
                                (Fusions.fuse\ fusions\ fs)))
   Not a working implementation yet, but it isn't used either ...
     module IPowSet =
        PowSet.Make (struct type t = int let compare = compare let to\_string = string\_of\_int end)
     let rec coarsest\_partition' = function
            True \mid False \rightarrow IPowSet.empty
            On\_shell\ (\_,\ momentum)\ |\ On\_shell\_not\ (\_,\ momentum)
            Off_shell (_, momentum) | Off_shell_not (_, momentum)
            Gauss (_, momentum) | Gauss_not (_, momentum)
            Any\_flavor\ momentum \rightarrow IPowSet.of\_lists\ [P.to\_ints\ momentum]
            And [] \rightarrow IPowSet.empty
            And \ cs \rightarrow IPowSet.basis \ (IPowSet.union \ (List.map \ coarsest\_partition' \ cs))
     let coarsest\_partition cascades =
       let p = coarsest\_partition' cascades in
       if IPowSet.is\_empty p then
          else
          IPowSet.to_lists p
    let part_to_string part =
```

```
"\{" \ ^{\circ} \ String.concat \ "," \ (List.map \ string\_of\_int \ part) \ ^{\circ} \ "\}"
  \mathsf{let}\ partition\_to\_string\ =\ \mathsf{function}
     | [] \rightarrow ""
     | parts \rightarrow
          \verb"``List.map part_to_string parts") ^ "}" \\
  \mathsf{let}\ to\_selectors\ =\ \mathsf{function}
     | \{ wf = True; flavors = []; vertices = [] \} \rightarrow no\_cascades
          let \ partition = coarsest\_partition \ c.wf \ in
          \{ select\_p = to\_select\_p \ c.wf; 
             select\_wf = to\_select\_wf c;
             on\_shell = to\_on\_shell c.wf;
             is\_gauss = to\_gauss \ c.wf;
             select\_vtx = to\_select\_vtx \ c;
             partition = partition;
             description = Some (to\_string c \hat{\ } partition\_to\_string partition) \}
end
```

--7Color

7.1 Interface of Color

```
\begin{array}{lll} \text{module type } Test & = \\ & \text{sig} \\ & \text{val } suite \ : \ OUnit.test \\ & \text{val } suite\_long \ : \ OUnit.test \\ & \text{end} \end{array}
```

7.1.1 Quantum Numbers

Color is not necessarily the SU(3) of QCD. Conceptually, it can be any unbroken symmetry (broken symmetries correspond to Model.flavor). In order to keep the group theory simple, we confine ourselves to the fundamental and adjoint representation of a single SU(N_C) for the moment. Therefore, particles are either color singlets or live in the defining representation of SU(N_C): $SUN(|N_C|)$, its conjugate $SUN(-|N_C|)$ or in the adjoint representation of SU(N_C): $AdjSUN(N_C)$.

```
type t = Singlet \mid SUN of int \mid AdjSUN of int val conjugate: t \rightarrow t val compare: t \rightarrow t \rightarrow int
```

7.1.2 Color Flows

This computes the color flow as used by WHIZARD:

```
\begin{array}{l} \text{module type } Flow = \\ \text{sig} \\ \\ \text{type } color \\ \text{type } t = color \ list \times color \ list \\ \text{val } rank : t \rightarrow int \\ \\ \text{val } of\_list : int \ list \rightarrow color \\ \text{val } ghost : unit \rightarrow color \\ \\ \text{val } in\_to\_lists : t \rightarrow int \ list \ list \\ \\ \text{val } in\_to\_lists : t \rightarrow int \ list \ list \\ \\ \text{val } out\_to\_lists : t \rightarrow int \ list \ list \\ \\ \text{val } ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \text{val } out\_ghost\_flags : t \rightarrow bool \ list \\ \\ \end{array}
```

A factor is a list of powers

$$\sum_{i} \left(\frac{num_i}{den_i}\right)^{power_i} \tag{7.1}$$

```
\begin{array}{lll} \mbox{type } power &= \{ \ num \ : \ int; \ den \ : \ int; \ power \ : \ int \ \} \\ \mbox{type } factor &= power \ list \\ \mbox{val } factor \ : \ t \ \rightarrow \ t \ \rightarrow \ factor \\ \mbox{val } zero \ : \ factor \end{array}
```

module Test: Test

 $module\ Flow\ :\ Flow$

Vertex Color Flows 7.1.3



The following is (still work-in-progress) in frastructure for translating UFO style color factors into color flows.



It might be beneficial, to use the color flow representation here. This will simplify the colorizer at the price of some complexity in *UFO* or here.

The datatypes Arrow.free and Arrow.factor will be used as building blocks for Birdtracks.t below.

```
module type Arrow =
  sig
```

For fundamental and adjoint representations, the endpoints of arrows are uniquely specified by a vertex (which will be represented by a number). For representations with more than one outgoing or incoming arrow, we need an additional index. This is abreated in the *endpoint* type.

```
type endpoint
```

Endpoints can be the tip or tail of an arrow or a ghost. Currently, we use the types for illustration only, but we might eventually try to make them abstract for additional safety...

```
\mathsf{type}\ \mathit{tip}\ =\ \mathit{endpoint}
type \ tail = endpoint
type ghost = endpoint
```

The position of the endpoint is encoded as an integer, which can be mapped, if necessary.

```
\mathsf{val}\ position\ :\ endpoint\ 	o\ int
val\ relocate\ :\ (int 
ightarrow\ int)\ 
ightarrow\ endpoint\ 
ightarrow\ endpoint
```

An Arrow.t is either a genuine arrow or a ghost . . .

```
type ('tail, 'tip, 'ghost) t =
    Arrow of 'tail \times 'tip
    Ghost of 'ghost
    Epsilon of 'tip list
  | Epsilon_bar of 'tail list
```

... and we distuish free arrows that must not contain summation indices from factors that may. Indices are opaque. ('tail, 'tip, 'qhost) t is polymorphic so that we can use richer 'tail, 'tip and 'qhost in factor.

```
type free = (tail, tip, ghost) t
type factor
```

For debugging, logging, etc.

```
val\ free\_to\_string : free \rightarrow string
val\ factor\_to\_string : factor \rightarrow string
```

Change the *endpoints* in a *free* arrow.

```
val\ map\ :\ (endpoint\ 	o\ endpoint)\ 	o\ free\ 	o\ free
```

Turn the endpoints satisfying the predicate into a left or right hand side summation index. Left and right refer to the two factors in a product and we must only match arrows with endpoints in both factors, not double lines on either side. Typically, the predicate will be set up to select only the summation indices that appear on both sides.

```
val\ to\_left\_factor\ :\ (endpoint\ 	o\ bool)\ 	o\ free\ 	o\ factor
val\ to\_right\_factor\ :\ (endpoint\ 	o\ bool)\ 	o\ free\ 	o\ factor
```

The incomplete inverse of factor raises an exception if there are remaining summation indices. is free can be used to check first.

```
val\ of\_factor\ :\ factor\ 	o\ free
```

```
val is\_free : factor \rightarrow bool
```

Return all the endpoints of the arrow that have a *position* encoded as a negative integer. These are treated as summation indices in our applications.

```
val\ negatives\ :\ free\ 	o\ endpoint\ list
```

We will need to test whether an arrow represents a ghost.

```
val is\_ghost : free \rightarrow bool
```

An arrow looping back to itself.

$$val\ is_tadpole\ :\ factor\ o\ bool$$

An ϵ or an $\bar{\epsilon}$

$$val\ is_epsilon\ :\ factor\ o\ bool$$

If arrow is an ϵ (or $\bar{\epsilon}$) and arrows contains an $\bar{\epsilon}$ (or ϵ), use

$$\forall n, N \in \mathbf{N}, 2 \le n \le N : \epsilon_{i_1 i_2 \cdots i_n} \bar{\epsilon}^{j_1 j_2 \cdots j_n} = \sum_{\sigma \in S_n} (-1)^{\varepsilon(\sigma)} \delta_{i_1}^{\sigma(j_1)} \delta_{i_2}^{\sigma(j_2)} \cdots \delta_{i_n}^{\sigma(j_n)}, \tag{7.2}$$

where $N = \delta_i^i$ is the dimension, to expand the pair into two lists of list of arrows: the first corresponding to the even permutations, the second to the odd ones. In addition, return the remaining arrows.

```
val\ match\_epsilon: factor\ 	o\ factor\ list 	o\ (factor\ list\ list\ 	imes\ factor\ list)\ option
```

Merging two arrows can give a variety of results. NB: $\epsilon - \bar{\epsilon}$ pairs are assumed to have been already expanded by $match_epsilon$.

```
type merge = | Match \text{ of } factor \text{ (* a tip fits the other's tail: make one arrow out of two *)} | Ghost_Match \text{ (* two matching ghosts *)} | Loop_Match \text{ (* both tips fit both tails: drop the arrows *)} | Mismatch \text{ (* ghost meets arrow: error *)} | No_Match \text{ (* nothing to be done *)} | val <math>merge : factor \rightarrow factor \rightarrow merge
```

Break up an arrow tee a (i => j) \rightarrow [i => a; a => j], i. e. insert a gluon. Returns an empty list for a ghost and raises an exception for ϵ and $\bar{\epsilon}$.

```
val\ tee\ :\ int 
ightarrow\ free\ 
ightarrow\ free\ list
```

dir i j arrow returns the direction of the arrow relative to j=>i. Returns 0 for a ghost and raises an exception for ϵ and $\bar{\epsilon}$.

```
val\ dir\ :\ int 
ightarrow\ int 
ightarrow\ free 
ightarrow\ int
```

It's intuitive to use infix operators to construct the lines.

```
val single: endpoint \rightarrow endpoint \rightarrow free val double: endpoint \rightarrow endpoint \rightarrow free list val ghost: endpoint \rightarrow free module Infix: sig
```

single i j or i = j creates a single line from i to j and i = j is a shorthard for [i = j].

$$\begin{array}{lll} \mathsf{val} \ (=>) \ : \ int \rightarrow \ int \rightarrow \ free \\ \mathsf{val} \ (==>) \ : \ int \rightarrow \ int \rightarrow \ free \ list \end{array}$$

double i j or $i \ll j$ creates a double line from i to j and back.

$$val (<=>) : int \rightarrow int \rightarrow free list$$

Single lines with subindices at the tip and/or tail

```
\begin{array}{lll} \mathsf{val} \; (>=>) \; : \; int \; \times \; int \; \to \; int \; \to \; free \\ \mathsf{val} \; (=>>) \; : \; int \; \to \; int \; \times \; int \; \to \; free \\ \mathsf{val} \; (>=>>) \; : \; int \; \times \; int \; \to \; int \; \times \; int \; \to \; free \end{array}
```

?? i creates a ghost at i.

```
\mathsf{val}\ (??)\ :\ \mathit{int}\ \rightarrow\ \mathit{free}
```

NB: I wanted to use ~~ instead of ??, but ocamlweb can't handle operators starting with ~ in the index properly.

end

```
\begin{array}{ll} \mathsf{val}\ epsilon\ :\ int\ list \to\ free \\ \mathsf{val}\ epsilon\_bar\ :\ int\ list \to\ free \end{array}
```

chain [1; 2; 3] is a shorthand for [1 => 2; 2 => 3] and cycle [1; 2; 3] for [1 => 2; 2 => 3; 3 => 1]. Other lists and edge cases are handled in the natural way.

```
val chain: int\ list \to free\ list val cycle: int\ list \to free\ list module Test: Test
```

Pretty printer for the toplevel.

```
val pp\_free: Format.formatter \to free \to unit val pp\_factor: Format.formatter \to factor \to unit end
```

 $\mathsf{module}\ \mathit{Arrow}\ :\ \mathit{Arrow}$

Possible color flows for a single propagator, as currently supported by WHIZARD.

module Propagator : Propagator

Implement birdtracks operations as generally as possible. Below, the signature will be extended with group specific generators for $SU(N_C)$ and $U(N_C)$ and even $N_C = 3$.

```
\label{eq:module_type} \begin{array}{ll} \text{module type } Birdtracks &= \\ \text{sig} \\ \text{type } t \end{array}
```

Strip out redundancies.

```
\mathsf{val}\ canonicalize\ :\ t\ \to\ t
```

Debugging, logging, etc.

```
val to\_string : t \rightarrow string
```

Test for trivial color flows that are just a number.

```
\mathsf{val}\ trivial\ :\ t\ \to\ bool
```

Test for vanishing coefficients.

```
val is\_null : t \rightarrow bool
```

Purely numeric factors, implemented as Laurent polynomials (cf. Algebra. Laurent in N_C with complex rational coefficients.

```
\begin{array}{lll} \mathrm{val}\ const\ :\ Algebra.Laurent.t\ \rightarrow\ t\\ \mathrm{val}\ null\ :\ t\ (*\ 0\ *)\\ \mathrm{val}\ one\ :\ t\ (*\ 1\ *)\\ \mathrm{val}\ two\ :\ t\ (*\ 1/2\ *)\\ \mathrm{val}\ half\ :\ t\ (*\ 1/2\ *)\\ \mathrm{val}\ third\ :\ t\ (*\ 1/3\ *)\\ \mathrm{val}\ minus\ :\ t\ (*\ -1\ *)\\ \mathrm{val}\ int\ :\ int\ \rightarrow\ t\ (*\ n\ *)\\ \mathrm{val}\ fraction\ :\ int\ \rightarrow\ t\ (*\ 1/n\ *)\\ \mathrm{val}\ nc\ :\ t\ (*\ N_C\ *) \end{array}
```

```
\begin{array}{l} \text{val } over\_nc \ : \ t \ (*\ 1/N_C\ *) \\ \text{val } imag \ : \ t \ (*\ i\ *) \\ \\ \text{Shorthand: } \{(c_i,p_i)\}_i \rightarrow \sum_i c_i (N_C)^{p_i} \\ \text{val } ints \ : \ (int \times int) \ list \rightarrow \ t \\ \text{val } scale \ : \ Algebra.QC.t \ \rightarrow \ t \ \rightarrow \ t \\ \text{val } sum \ : \ t \ list \rightarrow \ t \\ \text{val } diff \ : \ t \rightarrow \ t \ \rightarrow \ t \\ \text{val } times \ : \ t \rightarrow \ t \ \rightarrow \ t \\ \text{val } multiply \ : \ t \ list \rightarrow \ t \\ \end{array}
```

For convenience, here are infix versions of the above operations.

```
\begin{array}{l} \text{module } \mathit{Infix} \ : \ \mathsf{sig} \\ \text{val} \ ( \ +++ \ ) \ : \ t \ \rightarrow \ t \ \rightarrow \ t \\ \text{val} \ ( \ --- \ ) \ : \ t \ \rightarrow \ t \ \rightarrow \ t \\ \text{val} \ ( \ *** \ ) \ : \ t \ \rightarrow \ t \ \rightarrow \ t \end{array}
```

We can compute the f_{abc} and d_{abc} invariant tensors from the generators of an arbitrary representation:

$$f_{a_1 a_2 a_3} = -i \operatorname{tr} \left(T_{a_1} \left[T_{a_2}, T_{a_3} \right]_{-} \right) = -i \operatorname{tr} \left(T_{a_1} T_{a_2} T_{a_3} \right) + i \operatorname{tr} \left(T_{a_1} T_{a_3} T_{a_2} \right)$$

$$(7.3a)$$

$$d_{a_1 a_2 a_3} = \operatorname{tr} \left(T_{a_1} \left[T_{a_2}, T_{a_3} \right]_+ \right) = \operatorname{tr} \left(T_{a_1} T_{a_2} T_{a_3} \right) + \operatorname{tr} \left(T_{a_1} T_{a_3} T_{a_2} \right)$$

$$(7.3b)$$

assuming the normalization $tr(T_aT_b) = \delta_{ab}$.

NB: this uses the summation indices -1, -2 and -3. Therefore it must not appear unevaluated more than once in a product!

```
val f\_of\_rep: (int \rightarrow int \rightarrow int \rightarrow t) \rightarrow int \rightarrow int \rightarrow int \rightarrow t
val d\_of\_rep: (int \rightarrow int \rightarrow int \rightarrow t) \rightarrow int \rightarrow int \rightarrow t
```

Rename the indices of endpoints in a birdtrack.

```
val\ relocate\ :\ (int 
ightarrow\ int)\ 
ightarrow\ t\ 
ightarrow\ t
```

fuse nc vertex children use the color flows in the vertex to combine the color flows in the incoming children and return the color flows for outgoing particle together with their weights.

```
 \text{val } \textit{fuse} \; : \; \textit{int} \rightarrow \; t \; \rightarrow \; \textit{Propagator.t } \textit{list} \rightarrow \; (\textit{Algebra.QC.t} \; \times \; \textit{Propagator.t}) \; \textit{list} \\ \text{module } \textit{Test} \; : \; \textit{Test}
```

Pretty printer for the toplevel.

```
\mbox{ val } pp \ : \ Format.formatter \ \rightarrow \ t \ \rightarrow \ unit \ \mbox{end}
```

module Birdtracks : Birdtracks

```
\begin{array}{ll} \text{module type } SU3 & = \\ \text{sig} \end{array}
```

include Birdtracks val $delta3: int \rightarrow int \rightarrow t$

 $\begin{array}{lll} \text{val } delta8 \ : \ int \rightarrow \ int \rightarrow \ t \\ \text{val } delta8_loop \ : \ int \rightarrow \ int \rightarrow \ t \end{array}$

 $\begin{array}{lll} \mathsf{val} \ gluon \ : \ int \rightarrow \ int \rightarrow \ t \\ \mathsf{val} \ delta6 \ : \ int \rightarrow \ int \rightarrow \ t \end{array}$

val $delta10 : int \rightarrow int \rightarrow t$

 $val\ epsilon_bar\ :\ int\ list\
ightarrow\ t$

 $\begin{array}{lll} \mathsf{val}\ t10\ :\ int \to\ int \to\ int \to\ t \\ \mathsf{val}\ k6\ :\ int \to\ int \to\ int \to\ t \end{array}$

```
\begin{array}{c} \text{val } k6bar : int \rightarrow int \rightarrow int \rightarrow t \\ \text{val } delta\_of\_tableau : int \ Young.tableau \rightarrow int \rightarrow int \rightarrow t \\ \text{val } t\_of\_tableau : int \ Young.tableau \rightarrow int \rightarrow int \rightarrow int \rightarrow t \\ \text{end} \\ \\ \text{module } SU3 : SU3 \\ \\ \text{module } Vertex : SU3 \\ \end{array}
```



This must not be used, because it has not yet been updated to the correctly symmetrized version!

module U3:SU3

7.2 Implementation of Color

Avoid referring to *Pervasives.compare*, because *Pervasives* will become *Stdlib.Pervasives* in O'Caml 4.07 and *Stdlib* in O'Caml 4.08.

```
\begin{array}{lll} \text{let } pcompare &=& compare \\ \\ \text{module type } Test &=& \\ \text{sig} \\ & \text{val } suite : OUnit.test \\ & \text{val } suite\_long : OUnit.test \\ & \text{end} \end{array}
```

7.2.1 Quantum Numbers

```
type t =
     Singlet
     SUN of int
     AdjSUN of int
let conjugate = function
     Singlet \rightarrow Singlet
     SUN \ n \rightarrow SUN \ (-n)
     AdjSUN \ n \rightarrow AdjSUN \ n
let compare \ c1 \ c2 =
   match c1, c2 with
     Singlet, Singlet \rightarrow 0
     Singlet, - \rightarrow -1
     \_, \ \mathit{Singlet} \ \to \ 1
     SUN \ n, \ SUN \ n' \rightarrow compare \ n \ n'
     SUN _, AdjSUN _ \rightarrow -1
     AdjSUN _, SUN _ \rightarrow 1
     AdjSUN \ n, \ AdjSUN \ n' \rightarrow compare \ n \ n'
module type Line =
  sig
     \mathsf{type}\ t
     val\ conj\ :\ t\ 	o\ t
     val\ equal\ :\ t\ 	o\ t\ 	o\ bool
     val to\_string : t \rightarrow string
module type Cycles =
  sig
     \mathsf{type}\ \mathit{line}
     type t = (line \times line) list
```

Contract the graph by connecting lines and return the number of cycles together with the contracted graph.



The semantics of the contracted graph is not yet 100% ly fixed.

```
\mathsf{val}\ contract\ :\ t\ \to\ int\ \times\ t
```

The same as contract, but returns only the number of cycles and raises Open_line when not all lines are closed.

```
val count: t \to int exception Open\_line

Mainly for debugging ...

val to\_string: t \to string

end

module Cycles\;(L:Line): Cycles\; \text{with type } line\; =\; L.t\; =\; \text{struct}

type line\; =\; L.t\;

type t\; =\; (line\; \times\; line)\; list\;

exception Open\_line
```

NB: The following algorithm for counting the cycles is quadratic since it performs nested scans of the lists. If this was a serious problem one could replace the lists of pairs by a Map and replace one power by a logarithm.

```
let rec find\_fst c\_final c1 disc seen = function
   [] \rightarrow ((L.conj\ c\_final,\ c1)\ ::\ disc,\ List.rev\ seen)
   | (c1', c2') \text{ as } c12' :: rest \rightarrow
        if L.equal \ c1 \ c1' then
           find\_snd\ c\_final\ (L.conj\ c2')\ disc\ []\ (List.rev\_append\ seen\ rest)
        else
           find\_fst\ c\_final\ c1\ disc\ (c12'\ ::\ seen)\ rest
and find\_snd c\_final c2 disc seen = function
   [] \rightarrow ((L.conj \ c\_final, \ L.conj \ c2) :: disc, \ List.rev \ seen)
   (c1', c2') as c12' :: rest \rightarrow
        if L.equal \ c2' \ c2 then begin
           if L.equal\ c1'\ c\_final\ then
              (disc, List.rev_append seen rest)
              find\_fst\ c\_final\ (L.conj\ c1')\ disc\ [\ ]\ (List.rev\_append\ seen\ rest)
        end else
           find\_snd\ c\_final\ c2\ disc\ (c12'\ ::\ seen)\ rest
let consume = function
   | [] \rightarrow ([], [])
   [(c1, c2) :: rest \rightarrow find\_snd(L.conj c1)(L.conj c2)[][] rest
let contract lines =
   \mathsf{let} \ \mathsf{rec} \ \mathit{contract'} \ \mathit{acc} \ \mathit{disc} \ = \ \mathsf{function}
      [] \rightarrow (acc, List.rev \ disc)
       rest \rightarrow
           begin match consume rest with
           [\ ], rest' \rightarrow contract' (succ acc) disc rest'
           | disc', rest' \rightarrow contract' acc (List.rev\_append disc' disc) rest'
           end in
   contract' 0 [] lines
let \ count \ lines =
   match contract lines with
   \mid n, \mid \rightarrow n
   | n, \_ \rightarrow raise\ Open\_line
let to\_string\ lines =
   String.concat ""
          (\text{fun } (c1, c2) \rightarrow \text{"["^} L.to\_string c1 ^", "^L.to\_string c2 ^"]")
          lines)
```

end

7.2.2 Color Flows

```
module type Flow =
  sig
     type color
     type t = color \ list \times color \ list
     \mathsf{val}\ \mathit{rank}\ :\ t\ \to\ \mathit{int}
     val of\_list : int list \rightarrow color
     val\ ghost: unit \rightarrow color
     val to\_lists : t \rightarrow int \ list \ list
     val in\_to\_lists : t \rightarrow int list list
     \mathsf{val}\ out\_to\_lists\ :\ t\ \to\ int\ list\ list
     val\ ghost\_flags\ :\ t\ 	o\ bool\ list
     \mathsf{val}\ in\_ghost\_flags\ :\ t\ \to\ bool\ list
     \mathsf{val}\ out\_ghost\_flags\ :\ t\ \to\ bool\ list
     \mathsf{type}\ power\ =\ \{\ num\ :\ int;\ den\ :\ int;\ power\ :\ int\ \}
     type factor = power list
     \mathsf{val}\; factor\;:\; t\;\to\; t\;\to\; factor
     \mathsf{val}\ \mathit{zero}\ :\ \mathit{factor}
     module Test: Test
module\ Flow\ :\ Flow\ =
  struct
All ints are non-zero!
     type color =
           N of int
           N\_bar of int
           SUN of int \times int
           Singlet
           Ghost
Incoming and outgoing, since we need to cross the incoming states.
      type t = color \ list \times color \ list
     let rank cflow =
        2
                                                                Constructors
     let ghost() =
         Ghost
     let of_list = function
        |~[0;~0]~\rightarrow~\mathit{Singlet}
           [c; 0] \rightarrow N c
           [0; c] \rightarrow N_{-}bar c
           [c1; c2] \rightarrow SUN(c1, c2)
           \_ \rightarrow invalid\_arg "Color.Flow.of_list:\_num\_lines\_!=\_2"
     let to\_list = function
           N \ c \rightarrow [c; \ 0]
           N\_bar\ c\ 	o\ [0;\ c]
           SUN (c1, c2) \rightarrow [c1; c2]
           Singlet \rightarrow [0; 0]
         | Ghost \rightarrow [0; 0]
     \mathsf{let}\ \mathit{to\_lists}\ (\mathit{cfin},\ \mathit{cfout})\ =
         (List.map to_list cfin) @ (List.map to_list cfout)
     let in\_to\_lists (cfin, \_) =
         List.map to_list cfin
```

```
let out\_to\_lists (_, cfout) =
        List.map to_list cfout
     let ghost\_flag = function
        \mid N \_ \mid N\_bar \_ \mid SUN (\_, \_) \mid Singlet \rightarrow false
        Ghost \rightarrow true
     let ghost\_flags (cfin, cfout) =
       (List.map ghost_flag cfin) @ (List.map ghost_flag cfout)
     let in\_ghost\_flags\ (cfin, \_) =
        List.map ghost_flag cfin
     let out\_ghost\_flags (_, cfout) =
        List.map ghost_flag cfout
                                                        Evaluation
     type power = \{ num : int; den : int; power : int \}
     type factor = power list
     let zero = []
     let count_ghosts1 colors =
       List.fold\_left
          (fun acc \rightarrow function \ Ghost \rightarrow succ \ acc \mid \_ \rightarrow \ acc)
          0 colors
     let count\_ghosts (fin, fout) =
        count\_ghosts1 \ fin + count\_ghosts1 \ fout
     type \alpha square =
          Square of \alpha
         Mismatch
     let conjugate = function
         N c \rightarrow N_{-}bar (-c)
          N\_bar\ c\ 	o\ N\ (-c)
          SUN (c1, c2) \rightarrow SUN (-c2, -c1)
         Singlet \rightarrow Singlet
        | Ghost \rightarrow Ghost
     let \ cross_in \ (cin, \ cout) =
        cin @ (List.map conjugate cout)
     let \ cross\_out \ (cin, \ cout) =
       (List.map conjugate cin) @ cout
     \mathsf{module}\ C\ =\ \mathit{Cycles}\ (\mathsf{struct}
       \mathsf{type}\ t\ =\ int
       let \ conj = (-)
       let equal = (=)
       let to\_string = string\_of\_int
     end)
Match lines in the color flows f1 and f2 after crossing the incoming states. This will be used to compute squared
diagrams in square and square 2 below.
     let match\_lines match1 match2 f1 f2 =
       let rec match\_lines' acc f1' f2' =
          match f1', f2' with
If we encounter an empty list, we're done — unless the lengths don't match (which should never happen!):
          [], [] \rightarrow Square (List.rev acc)
          | \ \_ \ :: \ \_, \ [] \ | \ [], \ \_ \ :: \ \_ \ \rightarrow \ \mathit{Mismatch}
Handle matching ...
          | Ghost :: rest1, Ghost :: rest2
```

```
| Singlet :: rest1, Singlet :: rest2 \rightarrow match_lines' acc rest1 rest2
```

... and mismatched ghosts and singlet gluons:

Ghosts and singlet gluons can't match anything else

```
 \mid (Ghost \mid Singlet) :: \_, (N \_ \mid N\_bar \_ \mid SUN (\_, \_)) :: \_ \\ \mid (N \_ \mid N\_bar \_ \mid SUN (\_, \_)) :: \_, (Ghost \mid Singlet) :: \_ \rightarrow \\ Mismatch
```

Handle matching ...

```
| N\_bar\ c1 :: rest1, N\_bar\ c2 :: rest2
| N\ c1 :: rest1, N\ c2 :: rest2 \rightarrow match\_lines' (match1\ c1\ c2\ acc) rest1\ rest2
```

... and mismatched N or \bar{N} states:

```
\mid N \_ :: \_, N\_bar \_ :: \_
\mid N\_bar \_ :: \_, N \_ :: \_ \rightarrow
Mismatch
```

The N and \bar{N} don't match non-singlet gluons:

Now we're down to non-singlet gluons:

```
\mid SUN\ (c1,\ c1')\ ::\ rest1,\ SUN\ (c2,\ c2')\ ::\ rest2 \rightarrow \\ match\_lines'\ (match2\ c1\ c1'\ c2\ c2'\ acc)\ rest1\ rest2\ in \\ match\_lines'\ [\ ]\ (cross\_out\ f1)\ (cross\_out\ f2)
```

NB: in WHIZARD versions before 3.0, the code for $match_lines$ contained a bug in the pattern matching of Singlet, N, N_bar and SUN states, because they all were represented as SUN (c1, c2), only distinguished by the numeric conditions c1 = 0 and/or c2 = 0. This prevented the use of exhaustiveness checking and introduced a subtle dependence on the pattern order.

```
let square\ f1\ f2= match\_lines \\ (fun\ c1\ c2\ pairs\ \rightarrow\ (c1,\ c2)\ ::\ pairs) \\ (fun\ c1\ c1'\ c2\ c2'\ pairs\ \rightarrow\ (c1',\ c2')\ ::\ (c1,\ c2)\ ::\ pairs) \\ f1\ f2
```

In addition to counting closed color loops, we also need to count closed gluon loops. Fortunately, we can use the same algorithm on a different data type, provided it doesn't require all lines to be closed.

```
\begin{array}{lll} \text{let } int\_power \ n \ p &= \\ & \text{let rec } int\_power' \ acc \ i &= \\ \end{array}
```

```
\begin{array}{l} \text{if } i < 0 \text{ then} \\ invalid\_arg \text{ "int\_power"} \\ \text{else if } i = 0 \text{ then} \\ acc \\ \text{else} \\ int\_power' \left( n \times acc \right) \left( pred \ i \right) \text{ in} \\ int\_power' \ 1 \ p \end{array}
```

Instead of implementing a full fledged algebraic evaluator, let's simply expand the binomial by hand:

$$\left(\frac{N_C^2 - 2}{N_C^2}\right)^n = \sum_{i=0}^n \binom{n}{i} (-2)^i N_C^{-2i} \tag{7.4}$$

NB: Any result of square other than Mismatch guarantees $count_ghosts\ f1 = count_ghosts\ f2$.

```
let factor f1 f2 =
  match square f1 f2, square2 f1 f2 with
    Mismatch, \_ | \_, Mismatch \rightarrow []
    Square f12, Square f12' \rightarrow
      let num\_cycles = C.count f12
       and num\_cycles2, disc = C2.contract f12'
       and num\_ghosts = count\_ghosts f1 in
       List.map
         (fun i \rightarrow
           let parity = if num\_ghosts \mod 2 = 0 then 1 else -1
           and power = num\_cycles - num\_qhosts in
           let coeff = int\_power(-2) i \times Combinatorics.binomial num\_cycles2 i
           and power2 = -2 \times i in
           \{ num = parity \times coeff; \}
              den = 1;
              power = power + power2 \})
         (ThoList.range 0 num_cycles2)
module Test : Test =
  struct
    open OUnit
```

Here and elsewhere, we have to resist the temptation to define these tests as functions with an additional argument () in the hope to avoid having to package them into an explicit thunk fun () $\rightarrow eq \ v1 \ v2$ in order to delay evaluation. It turns out that the runtime would then sometimes evaluate the argument v1 or v2 even before the test is run. For pure functions, there is no difference, but the compiler appears to treat explicit thunks specially.



I haven't yet managed to construct a small demonstrator to find out in which circumstances the premature evaluation happens.

```
 \begin{split} & || \textbf{let } \textit{suite} \_\textit{square} = \\ & || \textbf{"square} || > ::: \\ & || [ || \textbf{"square} || ([], || []) || ([], || []) || > :: \\ & || (\textbf{fun } () \rightarrow \\ & \textit{assert} \_\textit{equal } (\textit{Square } []) (\textit{square } ([], []) ([], []))); \\ & || \textbf{"square} || ([3], || [3; || 0]) || ([3], || [3; || 0]) || > :: \\ & || (\textbf{fun } () \rightarrow \\ & \textit{assert} \_\textit{equal} \\ & || (\textit{Square } [(-1, -1); (1, 1)]) \\ & || (\textit{square } ([N \ 1], [N \ 1; \ \textit{Singlet}]) \\ & || ([N \ 1], [N \ 1; \ \textit{Singlet}])); \\ & || \textbf{"square} || ([0], || [3; || -3]) || ([0], || [3; || -3]) || > :: \\ & || (\textbf{fun } () \rightarrow \\ & \textit{assert} \_\textit{equal} | \end{aligned}
```

```
(Square [(1, 1); (-1, -1)])
                          (square
                             ([Singlet], [N 1; N\_bar (-1)])
                             ([Singlet], [N 1; N\_bar (-1)]));
                  "square_([3],_[3;_0])_([0],_[3;_-3])" >::
                    (\mathsf{fun}\ ()\ \to
                       assert\_equal
                          Mismatch
                          (square
                             ([N \ 1], [N \ 1; Singlet])
                             ([Singlet], [N 1; N\_bar (-1)]));
                  "square_\([3;\\]8],\\[3])\\([3;\\]8],\\[3])">::
                    (fun () \rightarrow
                       assert\_equal
                          (Square [-1, -1; 1, 1; -2, -2; 2, 2])
                             ([N \ 1; \ SUN \ (2, \ -1)], \ [N \ 2])
                             ([N \ 1; \ SUN \ (2, \ -1)], \ [N \ 2])))]
          let suite =
             "Color.Flow" >:::
               [suite\_square]
          let suite\_long =
             "Color.Flow\sqcuplong" >:::
       end
  end
later:
module General\_Flow =
  struct
     \mathsf{type} \,\, color \,\, = \,\,
         Lines of int list
          Ghost of int
     type t = color \ list \times color \ list
     let rank\_default = 2 (* Standard model *)
     let rank cflow =
       try
          begin match List.hd cflow with
          | Lines lines \rightarrow List.length lines
          | Ghost n\_lines \rightarrow n\_lines
          end
       with
       \downarrow \rightarrow rank\_default
  end
```

7.2.3 Vertex Color Flows

```
\begin{array}{lll} \operatorname{module} \ Q &= \ Algebra. Q \\ \operatorname{module} \ QC &= \ Algebra. QC \\ \operatorname{module} \ \operatorname{type} \ Arrow &= \\ \operatorname{sig} & \operatorname{type} \ endpoint \\ \operatorname{type} \ tip &= \ endpoint \\ \operatorname{type} \ tail &= \ endpoint \\ \end{array}
```

```
type ghost = endpoint
     val\ position\ :\ endpoint\ 	o\ int
     val\ relocate\ :\ (int 
ightarrow\ int)\ 
ightarrow\ endpoint\ 
ightarrow\ endpoint
     type ('tail, 'tip, 'ghost) t =
           Arrow of 'tail \times 'tip
           Ghost of 'ghost
           Epsilon of 'tip list
          Epsilon_bar of 'tail list
     type free = (tail, tip, ghost) t
     type factor
     val\ free\_to\_string : free \rightarrow string
     val\ factor\_to\_string : factor \rightarrow string
     val\ map\ :\ (endpoint\ 	o\ endpoint)\ 	o\ free\ 	o\ free
     val \ to\_left\_factor : (endpoint \rightarrow bool) \rightarrow free \rightarrow factor
     val\ to\_right\_factor\ :\ (endpoint\ 	o\ bool)\ 	o\ free\ 	o\ factor
     \mathsf{val}\ of\_factor\ :\ factor\ \to\ free
     val is\_free : factor \rightarrow bool
     val\ negatives: free \rightarrow endpoint\ list
     \mathsf{val}\ is\_ghost\ :\ free\ \to\ bool
     \mathsf{val}\ is\_tadpole\ :\ factor\ \to\ bool
     val\ is\_epsilon\ :\ factor\ 	o\ bool
     val\ match\_epsilon: factor\ 	o factor\ list\ 	o (factor\ list\ list\ 	imes\ factor\ list)\ option
     type merge =
           {\it Match} of {\it factor}
           Ghost\_Match
           Loop\_Match
           Mismatch
           No\_Match
     val\ merge\ :\ factor\ 	o\ factor\ 	o\ merge
     \mathsf{val}\ tee\ :\ int\rightarrow\ free\ \rightarrow\ free\ list
     \mathsf{val}\ dir\ :\ int\rightarrow\ int\rightarrow\ free\ \rightarrow\ int
     val\ single\ :\ endpoint\ 	o\ endpoint\ 	o\ free
     val\ double\ :\ endpoint\ 	o\ endpoint\ 	o\ free\ list
     val\ ghost\ :\ endpoint\ 	o\ free
     module Infix: sig
        val (=>) : int \rightarrow int \rightarrow free
        val (==>) : int \rightarrow int \rightarrow free \ list
        \mathsf{val}\ (<=>)\ :\ int \to\ int \to\ free\ list
        \mathsf{val}\ (>=>)\ :\ int\ \times\ int\ \to\ int\ \to\ free
        val (=>>) : int \rightarrow int \times int \rightarrow free
        val (>=>>) : int \times int \rightarrow int \times int \rightarrow free
        val (??) : int \rightarrow free
     end
     val\ epsilon\ :\ int\ list\ 
ightarrow\ free
     val\ epsilon\_bar\ :\ int\ list\ 
ightarrow\ free
     val\ chain\ :\ int\ list\ 
ightarrow\ free\ list
     val\ cycle\ :\ int\ list\ 
ightarrow\ free\ list
     module Test: Test
     val\ pp\_free\ :\ Format.formatter\ 	o\ free\ 	o\ unit
     val\ pp\_factor\ :\ Format.formatter\ 	o\ factor\ 	o\ unit
module Arrow : Arrow =
  struct
     type endpoint =
        I of int
        M of int \times int
     let position = function
        \mid I i \rightarrow i
        M(i, \_) \rightarrow i
```

```
\begin{array}{ll} \text{let } relocate \ f &=& \text{function} \\ \mid \ I \ i \ \rightarrow \ I \ (f \ i) \\ \mid \ M \ (i, \ n) \ \rightarrow \ M \ (f \ i, \ n) \end{array} \text{type } tip \ = \ endpoint} \text{type } tail \ = \ endpoint} \text{type } ghost \ = \ endpoint}
```

Note that in the case of double lines for the adjoint representation the *same endpoint* appears twice: once as a *tip* and once as a *tail*. If we want to multiply two factors by merging arrows with matching *tip* and *tail*, we must make sure that the *tip* is from one factor and the *tail* from the other factor.

The Free variant contains positive indices as well as negative indices that don't appear on both sides and will be summed in a later product. SumL and SumR indices appear on both sides.

```
type \alpha index =
     Free of \alpha
     SumL of \alpha
     SumR of \alpha
let is\_free\_index = function
     \mathit{Free} \ \_ \ \to \ \mathsf{true}
     SumL \ \_ \ | \ SumR \ \_ \ 	o \ \mathsf{false}
type ('tail, 'tip, 'ghost) t =
     Arrow of 'tail \times 'tip
     Ghost of 'ghost
     Epsilon of 'tip list
    Epsilon_bar of 'tail list
type free = (tail, tip, ghost) t
type \ factor = (tail \ index, \ tip \ index, \ ghost \ index) \ t
let \ endpoint\_to\_string \ = \ function
   | I i \rightarrow string\_of\_int i |
    M(i, n) \rightarrow Printf.sprintf "%d.%d" i n
let index\_to\_string = function
     Free i \rightarrow endpoint\_to\_string i
     SumL~i~\rightarrow~endpoint\_to\_string~i~``L"
     SumR \ i \rightarrow endpoint\_to\_string \ i \ "R"
let to\_string i2s = function
     Arrow\ (tail,\ tip)\ 	o\ Printf.sprintf "%s>%s" (i2s tail) (i2s tip)
     Ghost ghost \rightarrow Printf.sprintf "{\%s}" (i2s ghost)
     Epsilon tips → Printf.sprintf ">>>%s" (ThoList.to_string i2s tips)
    Epsilon_bar tails → Printf.sprintf "<<<%s" (ThoList.to_string i2s tails)
let free_to_string = to_string endpoint_to_string
let factor_to_string = to_string index_to_string
let index\_matches i1 i2 =
  match i1, i2 with
     SumL i1, SumR i2 \mid SumR i1, SumL i2 \rightarrow i1 = i2
   \mid \ \_ \rightarrow \mathsf{false}
let map f = function
     Arrow\ (tail,\ tip)\ 	o\ Arrow\ (f\ tail,\ f\ tip)
     Ghost\ ghost\ 	o\ Ghost\ (f\ ghost)
     Epsilon \ tips \rightarrow Epsilon \ (List.map \ f \ tips)
    Epsilon\_bar\ tails \rightarrow Epsilon\_bar\ (List.map\ f\ tails)
\mathsf{let} \; \mathit{free\_index} \; = \; \mathsf{function}
     Free i \rightarrow i
     SumL \ i \rightarrow invalid\_arg "Color.Arrow.free_index:,leftover,LHS,summation"
    SumR \ i \rightarrow invalid\_arg "Color.Arrow.free_index:\sqcupleftover\sqcupRHS\sqcupsummation"
let to\_left\_index is\_sum i =
  if is\_sum i then
```

```
SumL i
   else
      Free i
\mathsf{let}\ to\_right\_index\ is\_sum\ i\ =
   if is\_sum i then
      SumRi
   else
      Free i
let to\_left\_factor is\_sum = map (to\_left\_index is\_sum)
let to\_right\_factor is\_sum = map (to\_right\_index is\_sum)
let of\_factor = map free\_index
let negatives = function
   \mid Arrow (tail, tip) \rightarrow
       if position \ tail \ < \ 0 then
          if position \ tip < 0 then
             [tail; tip]
          else
             [tail]
       else if position tip < 0 then
          [tip]
       else
         \mid Ghost \ ghost \rightarrow
       if position \ ghost < 0 then
          [ghost]
       else
     Epsilon tips \rightarrow List.filter (fun tip \rightarrow position tip < 0) tips
   \mid Epsilon\_bar\ tails \rightarrow List.filter\ (fun\ tail \rightarrow position\ tail < 0)\ tails
let is\_free = function
     Arrow (Free \_, Free \_) \mid Ghost (Free \_) \rightarrow true
     Arrow\ (\_,\ \_)\ |\ Ghost\ \_\ \to\ \mathsf{false}
     Epsilon\ tips\ 	o\ List.for\_all\ is\_free\_index\ tips
     Epsilon\_bar\ tails \rightarrow List.for\_all\ is\_free\_index\ tails
let is\_ghost = function
     Ghost \, \_ \, \rightarrow \, {\sf true}
     Arrow _{-} \rightarrow \mathsf{false}
     Epsilon \_ \mid Epsilon \_bar \_ \rightarrow false
let is\_epsilon = function
     Epsilon \_ \mid Epsilon\_bar \_ \rightarrow true
   \mid \ Ghost \ \_ \ \mid \ Arrow \ \_ \ 	o \ \mathsf{false}
\mathsf{let} \ \mathit{single} \ \mathit{tail} \ \mathit{tip} \ = \\
   Arrow (tail, tip)
let double \ a \ b =
   if a = b then
     [single \ a \ b]
   else
      [single a b; single b a]
let ghost g =
   Ghost g
module Infix =
   struct
     let ( => ) i j = single (I i) (I j)
     \mathsf{let} \ ( \ ==> \ ) \ i \ j \ = \ [i \ => \ j]
     let ( <=> ) i j = double (I i) (I j)
     let (>=>) (i, n) j = single (M (i, n)) (I j)
```

```
\begin{array}{lll} & \text{let ( } =>> \text{ ) } i\text{ } (j\text{, } m) \text{ } = \text{ } single\text{ } (I\text{ } i)\text{ } (M\text{ } (j\text{, } m)) \\ & \text{let ( } >=>> \text{ ) } (i\text{, } n)\text{ } (j\text{, } m) \text{ } = \text{ } single\text{ } (M\text{ } (i\text{, } n))\text{ } (M\text{ } (j\text{, } m)) \\ & \text{let ( } ??\text{ ) } i\text{ } = \text{ } ghost\text{ } (I\text{ } i) \\ & \text{end} \\ & \text{open } Infix \end{array}
```

Split a_list at the first element equal to a according to eq. Return the reversed first part and the rest as a pair and wrap it in Some. Return None if there is no match.

```
 \begin{array}{l} \text{let } take\_first\_match\_opt \ ?(eq = (=)) \ a \ a\_list = \\ \\ \text{let } \text{rec } take\_first\_match\_opt' \ rev\_head = \ \text{function} \\ \\ | \ [] \ \rightarrow \ None \\ \\ | \ elt \ :: \ tail \ \rightarrow \\ \\ \text{if } \ eq \ elt \ a \ \text{then} \\ \\ \ Some \ (rev\_head, \ tail) \\ \\ \text{else} \\ \ take\_first\_match\_opt' \ (elt \ :: \ rev\_head) \ tail \ \text{in} \\ \\ take\_first\_match\_opt' \ [] \ a\_list \\ \end{array}
```

Split a_list and b_list at the first element equal according to eq. Return the reversed first part and the rest of each as a pair of pairs wrap it in Some. Return None if there is no match.

```
let take\_first\_matching\_pair\_opt ? (eq = (=)) \ a\_list \ b\_list =  let rec \ take\_first\_matching\_pair\_opt' \ rev\_a\_head = function | [] \rightarrow None | a :: a\_tail \rightarrow | begin match take\_first\_match\_opt \ ^eq \ a \ b\_list with | Some \ (rev\_b\_head, \ b\_tail) \rightarrow | Some \ ((rev\_a\_head, \ a\_tail), \ (rev\_b\_head, \ b\_tail)) | None \rightarrow | take\_first\_matching\_pair\_opt' \ (a :: rev\_a\_head) \ a\_tail end in | take\_first\_matching\_pair\_opt' \ [] \ a\_list
```

Replace the first occurrence of an element equal to a according to eq in a_list by a' and wrap the new list in Some. Return None if there is no match.

```
let replace\_first\_opt ? (eq = (=)) a a' a\_list =
  match take\_first\_match\_opt \tilde{\ }eq a a\_list with
     Some\ (rev\_head,\ tail) \rightarrow Some\ (List.rev\_append\ rev\_head\ (a'::tail))
   | None \rightarrow None |
let tee \ a = function
     Arrow (tail, tip) \rightarrow [Arrow (tail, I a); Arrow (I a, tip)]
     Ghost \ \_ as g \rightarrow [g]
     Epsilon \_ \rightarrow invalid\_arg "Arrow.teeunotudefineduforuEpsilon"
   Epsilon\_bar\_ 
ightarrow invalid\_arg "Arrow.tee\sqcupnot\sqcupdefined\sqcupfor\sqcupEpsilon\_bar"
let dir i j = function
  \mid Arrow (tail, tip) \rightarrow
      let tail = position tail
      and tip = position tip in
      if tip = i \wedge tail = j then
      else if tip = j \wedge tail = i then
         -1
      else
         invalid_arg "Arrow.dir"
  | Ghost \_ | Epsilon \_ | Epsilon \_bar \_ \rightarrow 0
type merge =
     Match of factor
     Ghost\_Match
     Loop_Match
     Mismatch
```

| No_Match

When computing

$$\epsilon_{ki_1i_2\cdots i_n}\bar{\epsilon}^{kj_1j_2\cdots j_n} = \sum_{\sigma\in S_n} (-1)^{\epsilon(\sigma)} \delta_{i_1}^{\sigma(j_1)} \delta_{i_2}^{\sigma(j_2)} \cdots \delta_{i_n}^{\sigma(j_n)}, \qquad (7.5)$$

we must keep track of the position of summation indices. We can use the fact that cyclic permutations are even for ϵ -tensors with an odd number of indices, corresponding to n even and odd otherwise.

```
let fuse_epsilons1 tails tips =
  match take_first_matching_pair_opt ~eq: index_matches tails tips with
    None \rightarrow None
    Some\ ((rev\_tails\_head,\ tails\_tail),\ (rev\_tips\_head,\ tips\_tail))\ 	o
      \mathsf{let}\ \mathit{tails}\ =\ \mathit{tails\_tail}\ @\ \mathit{List.rev}\ \mathit{rev\_tails\_head}
      and tips = tips\_tail @ List.rev rev\_tips\_head in
      let num\_tails = List.length tails
      and num\_tips = List.length \ tips in
      if num\_tails \neq num\_tips then
        invalid\_arg
           (Printf.sprintf
               "Color.Arrow.fuse\_epsilons1: \_length\_mismatch\_\%d\_<>_\bot\%d"
               (succ\ num\_tails)\ (succ\ num\_tips))
      else
        let is\_odd \ n = n \mod 2 \neq 0 in
        let flip =
           is\_odd\ num\_tips\ \land
             is_odd (List.length rev_tails_head - List.length rev_tips_head) in
        let even_tips = Combinatorics.permute_even tips
        and odd\_tips = Combinatorics.permute\_odd\ tips in
        let even = List.rev_map (List.rev_map2 single tails) even_tips
        and odd = List.rev\_map (List.rev\_map2 single tails) odd\_tips in
        if flip then
           Some (odd, even)
        else
           Some (even, odd)
```

We can also use the following (slightly less efficient) version that does not need to keep track of signs by itself and is more general, since it does not depend on N = n and works for $N \ge n$.

Starting with the case of matching dimension N and rank of ϵ and $\bar{\epsilon}$, there is the well known formula

$$\forall k, n = N \in \mathbf{N}, 0 \le k \le n \ge 2: \ \epsilon_{i_1 \cdots i_n} \bar{\epsilon}^{i_1 \cdots i_k j_{k+1} \cdots j_n} = k! \sum_{\sigma \in S_{n-k}} (-1)^{\varepsilon(\sigma)} \delta_{i_{k+1}}^{\sigma(j_{k+1})} \delta_{i_{k+2}}^{\sigma(j_{k+2})} \cdots \delta_{i_n}^{\sigma(j_n)}.$$
 (7.6)

In the general case, we have from anti-symmetry alone

$$\forall n, N \in \mathbf{N}, 2 \le n \le N : \epsilon_{i_1 i_2 \cdots i_n} \bar{\epsilon}^{j_1 j_2 \cdots j_n} = \sum_{\sigma \in S_n} (-1)^{\varepsilon(\sigma)} \delta_{i_1}^{\sigma(j_1)} \delta_{i_2}^{\sigma(j_2)} \cdots \delta_{i_n}^{\sigma(j_n)}, \tag{7.7}$$

where $N = \delta_i^i$ is the dimension.

```
let fuse\_epsilons\ tails\ tips =

if List.length\ tails = List.length\ tips then

List.fold\_left

(fun (even,\ odd)\ (eps,\ tips) \to

if eps > 0 then

(List.rev\_map2\ single\ tails\ tips\ ::\ even,\ odd)

else if eps < 0 then

(even,\ List.rev\_map2\ single\ tails\ tips\ ::\ odd)

else

failwith\ "Color.Arrow.fuse\_epsilons:\_Combinatorics.permute\_signed\_returned\_garbage")

([], []) (Combinatorics.permute\_signed\ tips)

else

invalid\_arg\ "Color.Arrow.fuse\_epsilons:\_|tails|_{\square} <>_{\square}|tips|"
```

From this, we derive

$$\forall n, N \in \mathbf{N}, 2 \leq n \leq N : \epsilon_{ki_2 \cdots i_n} \bar{\epsilon}^{kj_2 \cdots j_n} = \sum_{\sigma \in S_n} (-1)^{\varepsilon(\sigma)} \delta_k^{\sigma(k)} \delta_{i_2}^{\sigma(j_2)} \cdots \delta_{i_n}^{\sigma(j_2)} \cdots \delta_{i_n}^{\sigma(j_n)}$$

$$= (N - n + 1) \sum_{\sigma \in S_{n-1}} (-1)^{\varepsilon(\sigma)} \delta_{i_2}^{\sigma(j_2)} \cdots \delta_{i_n}^{\sigma(j_n)}, \quad (7.8)$$

where the $N = \delta_k^k$ comes from the permutations with $\sigma(k) = k$ that correspond to a loop in the color flow and the n-1 from the permutations with $\sigma(k) \in \{i_2, \ldots, i_n\}$ that do not lead to a loop. Note that N-n+1=1 in the special case N=n when rank and dimension match.

By induction

$$\forall k, n, N \in \mathbf{N}, 2 \leq n \leq N \land 1 \leq k \leq n : \epsilon_{i_1 \cdots i_n} \bar{\epsilon}^{i_1 \cdots i_k j_{k+1} \cdots j_n}$$

$$= \frac{(N-n+k)!}{(N-n)!} \sum_{\sigma \in S_{n-k}} (-1)^{\varepsilon(\sigma)} \delta_{i_{k+1}}^{\sigma(j_{k+1})} \delta_{i_{k+2}}^{\sigma(j_{k+2})} \cdots \delta_{i_n}^{\sigma(j_n)}, \quad (7.9)$$

where

$$\frac{(N-n+k)!}{(N-n)!} = (N-n+1)(N-n+2)\cdots(N-n+k)$$
(7.10)

and in the special case N=n

$$\frac{(N-n+k)!}{(N-n)!} = k!. (7.11)$$

In the case k=1 we get (7.8), which reduces to (7.5) for N=n, of course.



We also need to handle disconnected pairs of ϵ and $\bar{\epsilon}$. These never appear in merge' below, because merge skips all cases without matching summation indices. Handling them in merge doesn't work yet.

```
let merge' arrow1 arrow2 =
  match arrow1, arrow2 with
  | Ghost g1, Ghost g2 \rightarrow
      if index\_matches q1 q2 then
         Ghost_Match
      else
         No\_Match
     Arrow\ (tail,\ tip),\ Ghost\ g
     Ghost q, Arrow (tail, tip) \rightarrow
      if index\_matches\ g\ tail\ \lor\ index\_matches\ g\ tip\ then
         Mismatch
      else
         No\_Match
  | Arrow (tail, tip), Arrow (tail', tip') \rightarrow
      if index\_matches\ tip\ tail' then
         if index\_matches\ tip'\ tail\ then
           Loop_Match
         else
           Match (Arrow (tail, tip'))
      else if index\_matches\ tip'\ tail\ then
         Match (Arrow (tail', tip))
      else
         No\_Match
  \mid Arrow (tail, tip), Epsilon tips \mid Epsilon tips, Arrow (tail, tip) \rightarrow
      begin match replace_first_opt ~eq: index_matches tail tip tips with
        None \rightarrow No\_Match
      | Some \ tips \rightarrow Match \ (Epsilon \ tips) |
  \mid Arrow (tail, tip), Epsilon_bar tails \mid Epsilon_bar tails, Arrow (tail, tip) \rightarrow
      begin match replace_first_opt ~eq: index_matches tip tail tails with
        None \rightarrow No\_Match
        Some \ tails \rightarrow Match \ (Epsilon\_bar \ tails)
  \mid Epsilon tips, Ghost g \mid Ghost g, Epsilon tips \rightarrow
      if List.exists (index\_matches g) tips then
```

```
Mismatch
           else
              No\_Match
        \mid Epsilon_bar tails, Ghost g \mid Ghost g, Epsilon_bar tails \rightarrow
           if List.exists (index_matches g) tails then
              Mismatch
           else
              No\_Match
        \mid Epsilon \_, Epsilon \_ \mid Epsilon \_bar \_, Epsilon \_bar \_ \rightarrow
           No\_Match
       \mid Epsilon tips, Epsilon_bar tails \mid Epsilon_bar tails, Epsilon tips \rightarrow
           failwith "Color.Arrow.merge':_{\sqcup}impossible"
As an optimization, don't attempt to merge if neither of the arrows contains a summation index and return
immediately.
    let merge \ arrow1 \ arrow2 =
       if is\_free \ arrow1 \ \lor \ is\_free \ arrow2 then
          No\_Match
       else
          merge' arrow1 arrow2
     let merge\_to\_string = function
        \mid Match factor \rightarrow
           Printf.sprintf "Match (%s)" (factor_to_string factor)
          Ghost\_Match \rightarrow "Ghost"
          Loop\_Match \rightarrow "Loop"
         Mismatch \rightarrow "Mismatch"
         No\_Match \rightarrow "No\_Match"
     let logging_merge arrow1 arrow2 =
       let \ result = merge \ arrow1 \ arrow2 \ in
        Printf.eprintf
          "merge_\s_with_\s_==>_\s\n"
          (factor_to_string arrow1)
          (factor_to_string arrow2)
          (merge\_to\_string\ result);
        result
    let is\_tadpole = function
       \mid Arrow (tail, tip) \rightarrow
           index_matches tail tip
       \perp \rightarrow false
    let merge\_epsilon\_pair\ arrow1\ arrow2\ =
       match arrow1, arrow2 with
        \mid Epsilon tips, Epsilon_bar tails \mid Epsilon_bar tails, Epsilon tips \rightarrow
           Some (fuse_epsilons tails tips)
       \mid _ \rightarrow None
    let match\_epsilon arrow arrows =
       let rec match\_epsilon' seen = function
          | [] \rightarrow None
           arrow' :: arrows' \rightarrow
             begin match merge_epsilon_pair arrow arrow' with
              \mid Some (even, odd) \rightarrow
                  Some (even, odd, List.rev_append seen arrows')
              | None \rightarrow match\_epsilon' (arrow' :: seen) arrows'
             end in
       match_epsilon' [] arrows
     let epsilon = function
       [] \rightarrow invalid\_arg "Color.Arrow.epsilon_{\sqcup}[]"
         [\_] \rightarrow invalid\_arg "Color.Arrow.epsilon\_lone\_index"
       | tips \rightarrow
```

```
Epsilon (List.map (fun tip \rightarrow I tip) tips)
     let epsilon_bar = function
       [] \rightarrow invalid\_arg "Color.Arrow.epsilon[]"
       [\_] \rightarrow invalid\_arg "Color.Arrow.epsilon_lone_index"
       \mid tails \rightarrow
            Epsilon\_bar (List.map (fun tail \rightarrow I tail) tails)
Composite Arrows.
     \mathsf{let} \ \mathsf{rec} \ \mathit{chain} \ = \ \mathsf{function}
       | [] \rightarrow []
         [a] \rightarrow [a => a]
       [a; b] \rightarrow [a => b]
       | a :: (b :: \_ as rest) \rightarrow (a => b) :: chain rest
     let rec cycle' a = function
       | [] \rightarrow [a => a]
         [b] \rightarrow [b => a]
       b :: (c :: \_as rest) \rightarrow (b => c) :: cycle' a rest
     let \ cycle = function
       | [] \rightarrow []
       | a :: \_ as a\_list \rightarrow cycle' \ a \ a\_list
     module Test : Test =
       struct
          open OUnit
          \mathsf{let} \ \mathit{suite\_chain} \ = \\
             "chain" >:::
               ["[]">:: (fun () \rightarrow assert\_equal [] (chain []));
                  "[1]" >:: (fun () \rightarrow assert_equal [1 => 1] (chain [1]));
                  "[1;2]" >:: (fun () \rightarrow assert_equal [1 => 2] (chain [1; 2]));
                  "[1;2;3]" >:: (fun () \rightarrow assert\_equal [1 => 2; 2 => 3] (chain [1; 2; 3]));
                  "[1;2;3;4]" >:: (fun () \rightarrow assert\_equal [1 => 2; 2 => 3; 3 => 4] (chain [1; 2; 3; 4]))]
          let suite\_cycle =
             "cycle" >:::
                ["[]">:: (fun () \rightarrow assert\_equal [] (cycle []));
                  "[1]" >:: (fun () \rightarrow assert_equal [1 => 1] (cycle [1]));
                  "[1;2]" >:: (fun () \rightarrow assert_equal [1 => 2; 2 => 1] (cycle [1; 2]));
                  "[1;2;3]" >:: (fun () \rightarrow assert_equal [1 => 2; 2 => 3; 3 => 1] (cycle [1; 2; 3]));
                  "[1;2;3;4]">:: (fun () \rightarrow assert_equal [1 => 2; 2 => 3; 3 => 4; 4 => 1] (cycle [1; 2; 3; 4]))]
          let suite_take =
             "take" >:::
                ["1_{\sqcup}[]">:: (fun () \rightarrow assert\_equal None (take\_first\_match\_opt 1 []));
                  "1_{\sqcup}[1]" > :: (fun () \rightarrow assert\_equal (Some ([], [])) (take\_first\_match\_opt 1 [1]));
                  "1_{\sqcup}[2;3;4]">:: (fun () \rightarrow assert\_equal None (take\_first\_match\_opt 1 [2;3;4]));
                  "1_{\sqcup}[1;2;3]">::(fun() \rightarrow assert\_equal(Some([],[2;3]))(take\_first\_match\_opt 1[1;2;3]));
                  "2 [1;2;3]" > :: (fun () \rightarrow assert\_equal (Some ([1], [3])) (take\_first\_match\_opt 2 [1;2;3]));
                  "3_{\sqcup}[1;2;3]">::(fun() \rightarrow assert\_equal(Some([2;1], []))(take\_first\_match\_opt 3[1;2;3]))]
          let suite\_take2 =
             "take2" >:::
               ["[]">::
                     (fun () \rightarrow assert\_equal None (take\_first\_matching\_pair\_opt [] []));
                  "[]<sub>\(\|</sub>[1;2;3]">::
                     (fun () \rightarrow assert\_equal None (take\_first\_matching\_pair\_opt [] [1; 2; 3]));
                  "[1],,[2;3;4]">::
                     (fun () \rightarrow assert\_equal None (take\_first\_matching\_pair\_opt [1] [2; 3; 4]));
                  "[2;3;4]<sub>\(\)</sub>[1]">::
                     (fun () \rightarrow assert\_equal None (take\_first\_matching\_pair\_opt [2; 3; 4] [1]));
```

```
"[1;2;3]_{11}[4;5;6;7]">::
                      (\text{fun } () \rightarrow assert\_equal\ None\ (take\_first\_matchinq\_pair\_opt\ [1; 2; 3]\ [4; 5; 6; 7]));
                   "[1]<sub>\(\\\\\</sub>[1;2;3]">::
                      (fun () \rightarrow
                         assert\_equal
                            (Some\ (([],[]),\ ([],[2;3])))
                            (take\_first\_matching\_pair\_opt [1] [1; 2; 3]));
                   "[1;2;3]<sub>\(\subseteq\)</sub>[1;20;30]">::
                      (fun () \rightarrow
                         assert\_equal
                            (Some\ (([],[2;3]),\ ([],[20;30])))
                            (take\_first\_matching\_pair\_opt [1; 2; 3] [1; 20; 30]));
                   "[1;2;3;4;5;6]_[10;20;4;30;40]">::
                      (fun () \rightarrow
                         assert\_equal
                            (Some\ (([3;2;1],[5;6]),\ ([20;10],[30;40])))
                            (take_first_matching_pair_opt [1; 2; 3; 4; 5; 6] [10; 20; 4; 30; 40])) ]
           let suite\_replace =
              "replace" >:::
                 [ \ "1 \sqcup 10 \sqcup [] \ "> :: (fun \ () \ \rightarrow \ assert\_equal \ None \ (replace\_first\_opt \ 1 \ 2 \ []));
                   "1 \sqcup 10 \sqcup [1]" > :: (fun () \rightarrow assert\_equal (Some [10]) (replace\_first\_opt 1 10 [1]));
                   "1_{\sqcup}[2;3;4]">:: (fun () \rightarrow assert\_equal None (replace\_first\_opt \ 1 \ 10 \ [2;3;4]));
                   "1_{\sqcup}[1;2;3]">:: (fun () \rightarrow assert\_equal (Some [10;2;3]) (replace\_first\_opt 1 10 [1;2;3]));
                   "2_{\sqcup}[1;2;3]">:: (fun () \rightarrow assert\_equal (Some [1;10;3]) (replace\_first\_opt 2 10 [1;2;3]));
                   "3_{\square}[1;2;3]" >:: (fun () \rightarrow assert_equal (Some [1;2;10]) (replace_first_opt 3 10 [1;2;3]))]
           let suite =
              "Color.Arrow" >:::
                [suite\_chain;
                  suite\_cycle;
                  suite\_take;
                  suite\_take2;
                  suite\_replace
           let suite\_long =
              "Color.Arrow_long" >:::
        end
     let pp\_free fmt f =
        Format.fprintf fmt "%s" (free\_to\_string \ f)
     let pp\_factor\ fmt\ f =
        Format.fprintf fmt "%s" (factor\_to\_string f)
  end
module type Propagator =
  sig
     \mathsf{type}\ cf\_in\ =\ int
     type cf_out = int
     type t = W \mid I of cf\_in \mid O of cf\_out \mid IO of cf\_in \times cf\_out \mid G
     val to\_string : t \rightarrow string
module Propagator : Propagator =
  struct
     \mathsf{type}\ cf\_in\ =\ int
     type cf\_out = int
     type t = W \mid I \text{ of } cf\_in \mid O \text{ of } cf\_out \mid IO \text{ of } cf\_in \times cf\_out \mid G
     let to\_string = function
        \mid W \rightarrow \text{"W"}
```

```
I \ cf \rightarrow Printf.sprintf "I(%d)" \ cf
          O \ cf' \rightarrow Printf.sprintf "O(%d)" \ cf'
          IO(cf, cf') \rightarrow Printf.sprintf "IO(%d,%d)" cf cf'
          G \ \to \ \text{"G"}
  end
module type LP =
  sig
     val\ rationals\ :\ (Algebra.Q.t\ 	imes\ int)\ list 
ightarrow\ Algebra.Laurent.t
     val ints: (int \times int) \ list \rightarrow Algebra.Laurent.t
     val\ rational\ :\ Algebra.Q.t\ 	o\ Algebra.Laurent.t
     \mathsf{val}\ int: int 	o \ Algebra. Laurent.t
     \mathsf{val}\ fraction\ :\ int \to\ Algebra.Laurent.t
     val\ imag\ :\ int \rightarrow\ Algebra.Laurent.t
     val\ nc: int \rightarrow Algebra.Laurent.t
     val\ over\_nc: int \rightarrow Algebra.Laurent.t
  end
\mathsf{module}\ \mathit{LP}\ :\ \mathit{LP}\ =
  struct
     module L = Algebra.Laurent
Rationals from integers.
     let q_int n = Q.make n 1
     let q-fraction n = Q.make 1 n
Complex rationals:
     let qc\_rational \ q = QC.make \ q \ Q.null
     let qc\_int \ n = qc\_rational \ (q\_int \ n)
     \mathsf{let}\ \mathit{qc\_fraction}\ n\ =\ \mathit{qc\_rational}\ (\mathit{q\_fraction}\ n)
     let qc\_imag\ n\ =\ QC.make\ Q.null\ (q\_int\ n)
Laurent polynomials:
     let of_pairs f pairs =
        L.sum (List.map (fun (coeff, power) \rightarrow L.atom (f coeff) power) pairs)
     let rationals = of\_pairs \ qc\_rational
     let ints = of\_pairs qc\_int
     let rational \ q = rationals \ [(q, \ 0)]
     let int \ n = ints \ [(n, \ 0)]
     let fraction \ n = L.const \ (qc\_fraction \ n)
     let imag \ n = L.const \ (qc\_imag \ n)
     let nc n = ints [(n, 1)]
     let over\_nc \ n = ints \ [(n, -1)]
  end
module type Birdtracks =
  sig
     type t
     \mathsf{val}\ canonicalize\ :\ t\ \to\ t
     val to\_string : t \rightarrow string
     \mathsf{val}\ trivial\ :\ t\ \to\ bool
     val is\_null : t \rightarrow bool
     val\ const\ :\ Algebra.Laurent.t\ 	o\ t
     val null : t
     val one : t
     val two : t
     \mathsf{val}\ \mathit{half}\ :\ \mathit{t}
     val third : t
     val minus : t
     \mathsf{val}\ int: int \to\ t
     \mathsf{val}\ fraction : int \to t
```

```
\mathsf{val}\ nc\ :\ t
          val\ over\_nc: t
          val imag : t
          val\ ints\ :\ (int \times int)\ list 
ightarrow\ t
          \mathsf{val}\ scale\ :\ QC.t\ \to\ t\ \to\ t
          val sum : t list \rightarrow t
          \mathsf{val}\ \mathit{diff}\ :\ t\ \to\ t\ \to\ t
          \mathsf{val}\ times\ :\ t\ \to\ t\ \to\ t
          val multiply: t \ list \rightarrow t
           module Infix: sig
                \mathsf{val}\;(\;+++\;)\;:\;t\;\to\;t\;\to\;t
                \mathsf{val}\;(\;---\;)\;:\;t\;\to\;t\;\to\;t
                \mathsf{val}\;(\;***\;)\;:\;t\;\to\;t\;\to\;t
          val\ d\_of\_rep\ :\ (int 
ightarrow\ int 
ight
          val relocate: (int \rightarrow int) \rightarrow t \rightarrow t
          val fuse : int \rightarrow t \rightarrow Propagator.t \ list \rightarrow (QC.t \times Propagator.t) \ list
          \mathsf{module}\ \mathit{Test}\ :\ \mathit{Test}
          \mathsf{val}\ pp\ :\ Format.formatter\ 	o\ t\ 	o\ unit
module Birdtracks =
     struct
           \mathsf{module}\ A\ =\ \mathit{Arrow}
          open A.Infix
          module P = Propagator
          module L = Algebra.Laurent
          type connection = L.t \times A.free \ list
           \mathsf{type}\ t\ =\ connection\ list
           let 	ext{ } trivial = let 	ext{ } function
                | \ | \ | \rightarrow \text{true}
                    [(coeff, [])] \rightarrow coeff = L.unit
                 \mid \ \_ \ 	o \ \mathsf{false}
Rationals from integers.
           \mathsf{let}\ q\_int\ n\ =\ Q.make\ n\ 1
          \mathsf{let}\ q\_fraction\ n\ =\ Q.make\ 1\ n
Complex rationals:
           \mathsf{let}\ qc\_rational\ q\ =\ QC.make\ q\ Q.null
          let qc\_int \ n = qc\_rational \ (q\_int \ n)
          let qc\_fraction \ n = qc\_rational \ (q\_fraction \ n)
          let \ qc\_imag \ n \ = \ QC.make \ Q.null \ (q\_int \ n)
Laurent polynomials:
          let laurent\_of\_pairs\ f\ pairs\ =
                 L.sum (List.map (fun (coeff, power) \rightarrow L.atom (f coeff) power) pairs)
          let l\_rationals = laurent\_of\_pairs qc\_rational
          let l\_ints = laurent\_of\_pairs qc\_int
          let l_rational q = l_rationals [(q, 0)]
          let l_int n = l_ints [(n, 0)]
          let l_fraction n = L.const (qc\_fraction n)
          let l\_imag \ n = L.const \ (qc\_imag \ n)
          let l_nc n = l_ints [(n, 1)]
          let l\_over\_nc n = l\_ints [(n, -1)]
Expressions
          let const \ c = [c, []]
```

```
let ints pairs = const (LP.ints pairs)
let null = const L.null
let half = const (LP.fraction 2)
let third = const (LP.fraction 3)
let fraction n = const (LP.fraction n)
let one = const (LP.int 1)
let two = const (LP.int 2)
let minus = const (LP.int (-1))
let int n = const (LP.int n)
let nc = const (LP.nc 1)
let over\_nc = const (LP.ints [(1, -1)])
let imag = const (LP.imag 1)
module AMap = Pmap.Tree
let find\_arrows\_opt arrows map =
  try Some~(AMap.find~pcompare~arrows~map) with Not\_found~\rightarrow~None
let canonicalize1 (coeff, io_list) =
  (coeff, List.sort pcompare io_list)
let canonicalize terms =
  let map =
    List.fold\_left
       (fun acc \ term \rightarrow
         let coeff, arrows = canonicalize1 term in
         if L.is\_null coeff then
            acc
         else
            match find_arrows_opt arrows acc with
              None \rightarrow AMap.add prompare arrows coeff acc
            | Some coeff' \rightarrow
               let coeff'' = L.add coeff coeff' in
               if L.is\_null\ coeff'' then
                  AMap.remove pcompare arrows acc
               else
                  AMap.add pcompare arrows coeff" acc)
       AMap.empty terms in
  if AMap.is\_empty map then
    null
  else
     AMap.fold (fun arrows coeff acc \rightarrow (coeff, arrows) :: acc) map []
let arrows_to_string_aux f arrows =
  ThoList.to_string f arrows
let to\_string1\_aux\ f\ (coeff,\ arrows) =
  Printf.sprintf
     "(%s)<sub>\\\</sub>*\\\\s"
    (L.to\_string "N" coeff) (arrows\_to\_string\_aux f arrows)
let to\_string\_raw\_aux f v =
  ThoList.to\_string\ (to\_string1\_aux\ f)\ v
let to\_string\_aux\ f\ v\ =
  to\_string\_raw\_aux \ f \ (canonicalize \ v)
let \ factor\_arrows\_to\_string \ = \ arrows\_to\_string\_aux \ A.factor\_to\_string
let factor_to_string1 = to_string1_aux A.factor_to_string
let factor_to_string_raw = to_string_raw_aux A.factor_to_string
let \ factor\_to\_string \ = \ to\_string\_aux \ A.factor\_to\_string
let arrows_to_string = arrows_to_string_aux A.free_to_string
let to\_string1 = to\_string1\_aux A.free\_to\_string
let to_string_raw = to_string_raw_aux A.free_to_string
let \ to\_string \ = \ to\_string\_aux \ A.free\_to\_string
```

```
\begin{array}{lll} \text{let }pp \; \textit{fmt} \; v \; = \\ & Format. \textit{fprint} f \; \textit{fmt} \; \text{"%s"} \; (\textit{to\_string} \; v) \\ \\ \text{let } is\_\textit{null} \; v \; = \\ & List. \textit{for\_all} \; (\textit{fun} \; (c, \; \_) \; \rightarrow \; L.is\_\textit{null} \; c) \; (\textit{canonicalize} \; v) \\ \\ \text{let } is\_\textit{white} \; = \; \textit{function} \\ & \mid P.W \; \rightarrow \; \textit{true} \\ & \mid \_ \; \rightarrow \; \textit{false} \\ \\ \text{let } \textit{relocate1} \; f \; (c, \; v) \; = \\ & (c, \; List.map \; (A.map \; (A.relocate \; f)) \; v) \\ \\ \text{let } \textit{relocate} \; f \; = \; List.map \; (\textit{relocate1} \; f) \\ \end{array}
```

Only for documentiation: a term is a list of arrows with a coefficient.

```
type term = L.t \times A.factor\ list
```



New version: there can be ϵ or $\bar{\epsilon}$, but not both at the same time.

```
 \begin{tabular}{lll} type & term\_new & = \\ & | & Only\_Deltas \ \mbox{of} \ L.t \ \times \ A.factor \ list \\ & | & Epsilon\_Bars \ \mbox{of} \ L.t \ \times \ A.factor \ list \ \times \ A.endpoint \ list \ list \\ & | & Epsilon\_Bars \ \mbox{of} \ L.t \ \times \ A.factor \ list \ \times \ A.endpoint \ list \ list \\ \end{tabular}
```

Avoid the recursion, if there is no summation index in arrow. If arrow loops back to itself, replace it by a factor of N_C .

```
let rec add\_arrow: A.factor \rightarrow term \rightarrow term \ option = fun \ arrow \ (coeff, \ arrows) \rightarrow if \ A.is\_free \ arrow \ then Some \ (coeff, \ arrow \ :: \ arrows) else if A.is\_tadpole \ arrow \ then Some \ (L.mul \ (LP.nc \ 1) \ coeff, \ arrows) else add\_arrow' \ coeff \ [] \ arrow \ arrows
```

Add one arrow to a list of arrows, updating coeff if necessary. Accumulate already processed arrows in seen. Returns an empty list if there is a mismatch (a gluon meeting a ghost) and a list of pairs consisting of a coefficient and a list of arrows otherwise. There can be more than one pair, because matching ϵ and $\bar{\epsilon}$ results in a sum over permutations.

```
and add\_arrow': L.t \rightarrow A.factor\ list \rightarrow A.factor\ \rightarrow A.factor\ list \rightarrow term\ option =
  fun coeff seen arrow \rightarrow function
   [] \rightarrow (* \text{ visited all } arrows: \text{ no opportunities for further matches } *)
      Some\ (coeff,\ arrow\ ::\ seen)
    arrow' :: arrows' \rightarrow
      begin match A.merge\ arrow\ arrow' with
      A.Mismatch \rightarrow
          None
        A.Ghost\_Match \rightarrow (* replace matching ghosts by <math>-1/N_C *)
          Some (L.mul\ (LP.over\_nc\ (-1))\ coeff,\ List.rev\_append\ seen\ arrows')
        A.Loop\_Match \rightarrow (* replace a loop by N_C *)
          Some (L.mul (LP.nc 1) coeff, List.rev_append seen arrows')
      A.Match \ arrow'' \rightarrow (* \text{ two arrows have been merged into one } *)
          if A.is\_free\ arrow'' then (* no opportunities for further matches *)
             Some (coeff, arrow" :: List.rev_append seen arrows')
          else (* the new arrow" ist not yet saturated, try again: *)
             add_arrow' coeff seen arrow" arrows'
      A.No\_Match \rightarrow (* recurse to the remaining arrows *)
           add\_arrow' \ coeff \ (arrow' :: seen) \ arrow \ arrows'
and add\_determinant: A.factor\ list 
ightarrow\ A.factor\ list\ list 
ightarrow\ A.factor\ list\ list 
ightarrow\ term\ list =
  fun seen even odd (coeff, arrows as term) \rightarrow
```

```
distribute seen even term (distribute seen odd (L.neg coeff, arrows) []) and distribute: A.factor list \rightarrow A.factor list list \rightarrow term \rightarrow term list \rightarrow term list = fun seen permutations term terms \rightarrow List.fold_left (fun acc permutation \rightarrow splice_arrows seen permutation term :: acc) terms permutations and splice_arrows: A.factor list \rightarrow A.factor list \rightarrow term \rightarrow term = fun seen arrows term \rightarrow let coeff', arrows' = add_arrow_list arrows term in (coeff', List.rev_append seen arrows')
```



Here we would like to use the type system to prove that the two failing cases can't happen. In real life they can't happen, because *arrow* is never *A.Epsilon*. Can we use the version of GADTs that are available in O'Caml 4.05?

```
and add\_arrow\_list: A.factor\ list \to\ term\ \to\ term\ =\ fun\ arrows\ term\ \to\ match\ arrows\ with |\ []\ \to\ term\ |\ arrow\ ::\ rest\ \to\ begin\ match\ add\_arrow\ arrow\ term\ with |\ Some\ term\ \to\ add\_arrow\_list\ rest\ term\ |\ None\ \to\ failwith\ "add\_arrow\_list: $\sqcup$unexpected$$\sqcup$None"\ end
```

The return type is term list, because adding an ϵ (or $\bar{\epsilon}$) will turn a term to a sum of terms iff the term contains a $\bar{\epsilon}$ (or ϵ), since $\epsilon - \bar{\epsilon}$ pairs will be expanded by $add_determinant$.

```
let add\_arrow\_or\_epsilon: A.factor \rightarrow term \rightarrow term\ list = fun\ arrow\ (coeff,\ arrows\ as\ term) \rightarrow if A.is\_epsilon\ arrow\ then match\ A.match\_epsilon\ arrow\ arrows\ with | None <math>\rightarrow begin match add\_arrow'\ coeff\ []\ arrow\ arrows\ with | None <math>\rightarrow [] | Some\ term\ \rightarrow\ [term] end | Some\ (even,\ odd,\ arrows') \rightarrow add\_determinant\ []\ even\ odd\ (coeff,\ arrows') else match add\_arrow\ arrow\ term\ with | None <math>\rightarrow [] | Some\ term\ \rightarrow\ [term]
```

 $add_arrows_or_epsilons$ arrows term add the arrows to term by calling $add_arrow_or_epsilon$ for each one. Return an empty list if there are leftover summation indices in the end.

NB: we can reject the contributions with unsaturated summation indices from Ghost contributions to T_a only after adding all arrows that might saturate an open index.

```
let rec add\_arrows\_or\_epsilons: A.factor\ list \to term\ \to\ term\ list = fun\ arrows\ (\_,\ acc\_arrows\ as\ term)\ \to match\ arrows\ with
|\ []\ \to if\ List.for\_all\ A.is\_free\ acc\_arrows\ then\ [term]\ else\ []\ |\ arrow\ ::\ rest\ \to\ ThoList.flatmap\ (add\_arrows\_or\_epsilons\ rest)\ (add\_arrow\_or\_epsilon\ arrow\ term)
let logging\_add\_arrows\_or\_epsilons\ arrows\ term\ = let\ result\ =\ add\_arrows\_or\_epsilons\ arrows\ term\ in
```

```
Printf.eprintf
  "add\_arrows\_or\_epsilons_{\sqcup} \%s_{\sqcup} to_{\sqcup} \%s_{\sqcup} ==>_{\sqcup} \%s \ ""
  (factor\_to\_string1 \ term)
  (factor\_arrows\_to\_string\ arrows)
  (ThoList.to_string factor_to_string1 result);
```

Note that a negative index might be summed only later in a sequence of binary products and must therefore be treated as free in this product. Therefore, we have to classify the indices as summation indices not only based on their sign, but in addition based on whether they appear in both factors. Only then can we reject surviving ghosts.

```
module ESet =
    Set.Make
      (struct
         type t = A.endpoint
         let compare = pcompare
      end)
let negatives arrows =
    List.fold\_left
      (fun acc \ arrow \rightarrow
         List.fold\_left
           (fun \ acc' \ i \rightarrow ESet.add \ i \ acc')
            acc (A.negatives arrow))
      ESet.empty arrows
let times1 (coeff1, arrows1) (coeff2, arrows2) =
   let summations = ESet.inter (negatives arrows1) (negatives arrows2) in
   \mathsf{let}\ is\_sum\ i\ =\ ESet.mem\ i\ summations\ \mathsf{in}
   \mathsf{let}\ \mathit{arrows1'}\ =\ \mathit{List.map}\ (A.to\_\mathit{left\_factor}\ \mathit{is\_sum})\ \mathit{arrows1}
   and arrows2' = List.map (A.to\_right\_factor is\_sum) arrows2 in
   List.map
      (fun (coeff1, arrows) \rightarrow
         (L.mul\ coeff1\ coeff2,\ List.map\ A.of\_factor\ arrows))
      (add_arrows_or_epsilons arrows2' (coeff1, arrows1'))
{\tt let} \ logging\_times1 \ factor1 \ factor2 \ =
   let \ result = times1 \ factor1 \ factor2 \ in
    Printf.eprintf
      "%s_{\sqcup}times1_{\sqcup}%s_{\sqcup}==>_{\sqcup}%s\n"
      (to_string1 factor1)
      (to_string1 factor2)
      (ThoList.to_string to_string1 result);
   result
let sum terms =
    canonicalize (List.concat terms)
 let times term term' =
    can onicalize
      (Product.fold2
          (\text{fun } x \ y \rightarrow List.rev\_append \ (times1 \ x \ y))
          term term' [])
Is that more efficient than the following implementation?
```



```
let rec multiply1' acc = function
   | [] \rightarrow [acc]
   \mid factor :: factors \rightarrow
        List.fold_right multiply1' (times1 acc factor) factors
\mathsf{let}\ \mathit{multiply1}\ =\ \mathsf{function}
   [] \rightarrow [(L.unit, [])]
   | [factor] \rightarrow [factor]
```

canonicalize (List.rev_append term1 (scale (qc_int (-1)) term2))

 $\begin{array}{lll} (L.scale \ q \ coeff, \ arrows) \\ \text{let} \ scale \ q \ = \ List.map \ (scale1 \ q) \end{array}$

let diff_term1_term2_=

let (***) = times

let $scale1 \ q \ (coeff, \ arrows) =$

 $\begin{array}{lll} \text{module } \mathit{Infix} &= \\ \text{struct} & \text{let (} +++ \text{) } \mathit{term } \mathit{term'} \text{ } = \textit{ } \mathit{sum } [\mathit{term}; \; \mathit{term'}] \\ \text{let (} --- \text{) } &= \textit{ } \mathit{diff} \end{array}$

end

open *Infix*

Compute $\operatorname{tr}(r(T_a)r(T_b)r(T_c))$. NB: this uses the summation indices -1, -2 and -3. Therefore it must not appear unevaluated more than once in a product!

```
let trace3\ r\ a\ b\ c = r\ a\ (-1)\ (-2)\ ****\ r\ b\ (-2)\ (-3)\ ****\ r\ c\ (-3)\ (-1)
let f\_of\_rep\ r\ a\ b\ c = minus\ ***\ imag\ ***\ (trace3\ r\ a\ b\ c\ ---\ trace3\ r\ a\ c\ b)
d_{abc} = {\rm tr}(r(T_a)[r(T_b), r(T_c)]_+)
let d\_of\_rep\ r\ a\ b\ c = trace3\ r\ a\ b\ c\ +++\ trace3\ r\ a\ c\ b
```

Feynman Rules

```
module IMap =
   Map.Make (struct type t = int let compare = pcompare end)
let line\_map\ lines\ =
   \mathsf{let}\ \_,\ \mathit{map}\ =
      List.fold\_left
        (fun (i, acc) line \rightarrow
           (succ i,
             match line with
             \mid~P.W~\rightarrow~acc
             \downarrow \rightarrow IMap.add \ i \ line \ acc)
         (1, IMap.empty)
         lines in
   map
let lines_to_string lines =
   match IMap.bindings lines with
   \mid \; [\;] \; 
ightarrow \; "W"
   | lines \rightarrow
       String.concat
          "_"
```

```
(List.map
             (\text{fun } (i, c) \rightarrow Printf.sprintf "%s0%d" } (P.to\_string c) i)
             lines)
let clear = IMap.remove
let add_in i \ cf \ lines =
  match IMap.find_opt i lines with
    Some (P.O \ cf') \rightarrow IMap.add \ i \ (P.IO \ (cf, \ cf')) \ lines
   \downarrow \rightarrow IMap.add \ i \ (P.I \ cf) \ lines
let add\_out \ i \ cf' \ lines =
  match IMap.find_opt i lines with
     Some (P.I \ cf) \rightarrow IMap.add \ i (P.IO \ (cf, \ cf')) \ lines
    \_ \rightarrow IMap.add \ i \ (P.O \ cf') \ lines
let \ add\_ghost \ i \ lines =
   IMap.add i P.G lines
let connect1 \ n \ arrow \ lines =
  match arrow with
   A.Ghost g \rightarrow
      let g = A.position g in
      if g = n then
         Some (add\_ghost \ n \ lines)
      else
         begin match IMap.find_opt g lines with
           Some \ P.G \rightarrow Some \ (clear \ g \ lines)
         |  \rightarrow None
         end
   A.Arrow(i, o) \rightarrow
      let i = A.position i
      and o = A.position o in
      if o = n then
         begin match IMap.find_opt i lines with
           Some (P.I \ cfi) \rightarrow Some (add\_in \ o \ cfi \ (clear \ i \ lines))
           Some (P.IO (cfi, cfi')) \rightarrow Some (add\_in o cfi (add\_out i cfi' lines))
         |  \rightarrow None
         end
      else if i = n then
         begin match IMap.find_opt o lines with
           Some (P.O \ cfo') \rightarrow Some (add\_out \ i \ cfo' (clear \ o \ lines))
           Some\ (P.IO\ (cfo,\ cfo')) \rightarrow Some\ (add\_out\ i\ cfo'\ (add\_in\ o\ cfo\ lines))
              \rightarrow None
         end
      else
         begin match IMap.find_opt i lines, IMap.find_opt o lines with
         | Some (P.I cfi), Some (P.O cfo') when cfi = cfo' \rightarrow
             Some (clear o (clear i lines))
           Some (P.I cfi), Some (P.IO (cfo, cfo')) when cfi = cfo' \rightarrow
             Some (add_in o cfo (clear i lines))
         | Some (P.IO (cfi, cfi')), Some (P.O cfo') when cfi = cfo' \rightarrow
             Some (add_out i cfi' (clear o lines))
         | Some (P.IO (cfi, cfi')), Some (P.IO (cfo, cfo')) when cfi = cfo' \rightarrow
             Some (add_in o cfo (add_out i cfi' lines))
         \mid _ \rightarrow None
         end
  \mid A.Epsilon \_ \rightarrow
     failwith "Birdtracks.connect_not_yet_defined_for_Epsilon"
    A.Epsilon\_bar \_ \rightarrow
      failwith "Birdtracks.connectunotuyetudefineduforuEpsilon_bar"
let connect connections lines =
  let n = succ (List.length lines)
```

```
and lines = line\_map \ lines in
  let rec connect' acc = function
     \mid arrow :: arrows \rightarrow
         begin match connect1 n arrow acc with
         | None \rightarrow None |
         | Some acc \rightarrow connect' acc arrows
         end
     | [] \rightarrow Some \ acc \ in
  match connect' lines connections with
    None \rightarrow None
  \mid Some \ acc \rightarrow
      begin match IMap.bindings acc with
      [] \rightarrow Some P.W
      [(i, cf)] when i = n \rightarrow Some \ cf
      |  \rightarrow None
      end
let fuse1 nc lines (c, vertex) =
  match connect vertex lines with
    None \rightarrow []
   | Some cf \rightarrow [(L.eval (qc\_int nc) c, cf)]
let fuse nc vertex lines =
  \mathsf{match}\ \mathit{vertex}\ \mathsf{with}
  | [] \rightarrow
      if List.for_all is_white lines then
         [(QC.unit, P.W)]
      else
         |vertex \rightarrow
       ThoList.flatmap (fuse1 nc lines) vertex
module Test : Test =
  struct
     open OUnit
     let vertices\_equal \ v1 \ v2 =
        (canonicalize v1) = (canonicalize v2)
     let eq v1 v2 =
        assert_equal ~printer: (ThoList.to_string to_string1) ~cmp: vertices_equal v1 v2
     let suite\_times1 =
        "times1" >:::
          ["merge<sub>□</sub>two" >::
               (fun () \rightarrow
                  eq
                    [(L.unit, 1 ==> 2)]
                    (times1 \ (L.unit, 1 ==> -1) \ (L.unit, -1 ==> 2)));
             "merge_{\sqcup}two_{\sqcup}exchanged">::
               (fun () \rightarrow
                  eq
                    [(L.unit, 1 ==> 2)]
                    (times1 (L.unit, -1 ==> 2) (L.unit, 1 ==> -1));
             "ghost1" >::
               (fun () \rightarrow
                  eq
                    [(l\_over\_nc\ (-1),\ 1 ==>\ 2)]
                    (times 1)
                        (L.unit, [-1 => 2; ?? (-3)])
                        (L.unit, [1 => -1; ?? (-3)]));
             "ghost2" >::
```

```
(fun () \rightarrow
            eq
               (times1)
                  (L.unit, [1 => -1; ?? (-3)])
                  (L.unit, [-1 => 2; -3 => -4; -4 => -3]));
       "ghost2_exchanged" >::
          (fun () \rightarrow
            eq
               (times1)
                  (L.unit, [-1 => 2; -3 => -4; -4 => -3])
                  (L.unit, [1 => -1; ?? (-3)])))
let suite_canonicalize =
  "canonicalize" >:::
let line\_option\_to\_string = function
    None \rightarrow "no\sqcupmatch"
    Some line \rightarrow P.to_string line
let test_connect_msg vertex formatter (expected, result) =
  Format.fprintf
     formatter
     "[%s]:_{\sqcup}expected_{\sqcup}%s,_{\sqcup}got_{\sqcup}%s"
     (arrows_to_string vertex)
     (line_option_to_string expected)
     (line\_option\_to\_string\ result)
let \ test\_connect \ expected \ lines \ vertex =
  assert\_equal
     printer: line\_option\_to\_string
     expected (connect vertex lines)
let test_connect_permutations expected lines vertex =
  List.iter
     (fun v \rightarrow
       assert\_equal
          pp\_diff: (test\_connect\_msg\ v)
          expected (connect \ v \ lines))
     (Combinatorics.permute vertex)
let suite\_connect =
  "connect" >:::
     [ "delta" >::
          (fun () \rightarrow
            test\_connect\_permutations
               (Some\ (P.I\ 1))
               [P.I 1; P.W]
               (1 ==> 3);
       "f:_1->3->2->1" >::
          (\mathsf{fun}\ ()\ \to
            test\_connect\_permutations
               (Some\ (P.IO\ (1,\ 3)))
               [P.IO\ (1,\ 2);\ P.IO\ (2,\ 3)]
               (A.cycle [1; 3; 2]);
       "f:_{\perp}1->2->3->1" >::
          (fun () \rightarrow
            test\_connect\_permutations
               (Some\ (P.IO\ (1,\ 2)))
               [P.IO\ (3,\ 2);\ P.IO\ (1,\ 3)]
```

```
(A.cycle [1; 2; 3])) ]
let suite =
    "Color.Birdtracks" >:::
    [suite_times1;
        suite_canonicalize;
        suite_connect]
let suite_long =
    "Color.Birdtracks_long" >:::
    []
end
let vertices_equal v1 v2 =
    is_null (v1 --- v2)
let assert_zero_vertex v =
    OUnit.assert_equal ~printer : to_string ~cmp : vertices_equal null v
```

As an extra protection agains vacuous tests, we make sure that the LHS does not vanish.

```
 \begin{array}{ll} \textbf{let} \ eq \ v1 \ v2 \ = \\ OUnit.assert\_bool \ "LHS$_{\sqcup}=$_{\sqcup}0" \ (\neg \ (is\_null \ v1)); \\ OUnit.assert\_equal \ \tilde{\ } printer: to\_string \ \tilde{\ } cmp: vertices\_equal \ v1 \ v2 \end{array}
```

7.2.4
$$SU(N_C)$$

We're computing with a general N_C , but epsilon and epsilonbar make only sense for $N_C = 3$. Also some of the terminology alludes to $N_C = 3$: triplet, sextet, octet.

Using the normalization $\operatorname{tr}(T_a T_b) = \delta_{ab}$, we can check the selfconsistency of the completeness relation

$$T_a^{i_1j_1}T_a^{i_2j_2} = \left(\delta^{i_1j_2}\delta^{i_2j_1} - \frac{1}{N_C}\delta^{i_1j_1}\delta^{j_1j_2}\right)$$
 (7.12)

as

end

$$T_{a}^{i_{1}j_{1}}T_{a}^{i_{2}j_{2}} = \operatorname{tr}\left(T_{a_{1}}T_{a_{2}}\right)T_{a_{1}}^{i_{1}j_{1}}T_{a_{2}}^{i_{2}j_{2}} = T_{a_{1}}^{l_{1}l_{2}}T_{a_{2}}^{l_{2}l_{1}}T_{a_{1}}^{i_{1}j_{1}}T_{a_{2}}^{i_{2}j_{2}}$$

$$= \left(\delta^{l_{1}j_{1}}\delta^{i_{1}l_{2}} - \frac{1}{N_{C}}\delta^{l_{1}l_{2}}\delta^{i_{1}j_{1}}\right)\left(\delta^{l_{2}j_{2}}\delta^{i_{2}l_{1}} - \frac{1}{N_{C}}\delta^{l_{2}l_{1}}\delta^{i_{2}j_{2}}\right) = \left(\delta^{i_{1}j_{2}}\delta^{i_{2}j_{1}} - \frac{1}{N_{C}}\delta^{i_{1}i_{2}}\delta^{j_{2}j_{1}}\right)$$
(7.13)

With

$$if_{a_1 a_2 a_3} = tr(T_{a_1}[T_{a_2}, T_{a_3}]) = tr(T_{a_1} T_{a_2} T_{a_3}) - tr(T_{a_1} T_{a_3} T_{a_2})$$
 (7.14)

and

$$\operatorname{tr}\left(T_{a_{1}}T_{a_{2}}T_{a_{3}}\right)T_{a_{1}}^{i_{1}j_{1}}T_{a_{2}}^{i_{2}j_{2}}T_{a_{3}}^{i_{3}j_{3}} = T_{a_{1}}^{l_{1}l_{2}}T_{a_{2}}^{l_{2}l_{3}}T_{a_{3}}^{l_{3}l_{1}}T_{a_{1}}^{i_{1}j_{1}}T_{a_{2}}^{i_{2}j_{2}}T_{a_{3}}^{i_{3}j_{3}} = \left(\delta^{l_{1}j_{1}}\delta^{i_{1}l_{2}} - \frac{1}{N_{C}}\delta^{l_{1}l_{2}}\delta^{i_{1}j_{1}}\right)\left(\delta^{l_{2}j_{2}}\delta^{i_{2}l_{3}} - \frac{1}{N_{C}}\delta^{l_{2}l_{3}}\delta^{i_{2}j_{2}}\right)\left(\delta^{l_{3}j_{3}}\delta^{i_{3}l_{1}} - \frac{1}{N_{C}}\delta^{l_{3}l_{1}}\delta^{i_{3}j_{3}}\right)$$
(7.15)

we find the decomposition

$$if_{a_1 a_2 a_3} T_{a_1}^{i_1 j_1} T_{a_2}^{i_2 j_2} T_{a_3}^{i_3 j_3} = \delta^{i_1 j_2} \delta^{i_2 j_3} \delta^{i_3 j_1} - \delta^{i_1 j_3} \delta^{i_3 j_2} \delta^{i_2 j_1}.$$

$$(7.16)$$

Indeed,

```
#procedure TTT(sign)
local [TTT'sign'] =
           (j1(11) * i1(12) - d_(11,12) * i1.j1 / nc)
        * ( j2(12) * i2(13) - d_{(12,13)} * i2.j2 / nc )
        * ( j3(13) * i3(11) - d_(13,11) * i3.j3 / nc )
 'sign' ( j1(l1) * i1(l2) - d_(l1,l2) * i1.j1 / nc )
        * ( j3(12) * i3(13) - d_(12,13) * i3.j3 / nc )
        * ( j2(13) * i2(11) - d_{(13,11)} * i2.j2 / nc );
#endprocedure
#call TTT(-)
#call TTT(+)
bracket nc;
print;
.sort
.end
gives
    [TT] =
          + nc^-1 * ( - i1.j1*i2.j2 )
          + i1.j2*i2.j1;
    [TTT-] =
          + i1.j2*i2.j3*i3.j1 - i1.j3*i2.j1*i3.j2;
    [TTT+] =
          + nc^{-2} * (
                              4*i1.j1*i2.j2*i3.j3)
         + nc^-1 * (
                           - 2*i1.j1*i2.j3*i3.j2
                            - 2*i1.j2*i2.j1*i3.j3
                            - 2*i1.j3*i2.j2*i3.j1 )
          + i1.j2*i2.j3*i3.j1 + i1.j3*i2.j1*i3.j2;
module type SU3 =
  sig
     include Birdtracks
     val \ delta3 : int \rightarrow int \rightarrow t
     \mathsf{val}\ delta8\ :\ int\rightarrow\ int\rightarrow\ t
     val\ delta8\_loop : int \rightarrow int \rightarrow t
     \mathsf{val}\ gluon\ :\ int\rightarrow\ int\rightarrow\ t
     val \ delta6 : int \rightarrow int \rightarrow t
     val delta10 : int \rightarrow int \rightarrow t
     \mathsf{val}\ t\ :\ int \to\ int \to\ int \to\ t
     \mathsf{val}\ f\ :\ int \to\ int \to\ int \to\ t
     \mathsf{val}\ d\ :\ int \to\ int \to\ int \to\ t
     \mathsf{val}\ epsilon\ :\ int\ list\ \rightarrow\ t
     \mathsf{val}\ epsilon\_bar\ :\ int\ list\ \rightarrow\ t
     val t8 : int \rightarrow int \rightarrow int \rightarrow t
     val t6 : int \rightarrow int \rightarrow int \rightarrow t
     \mathsf{val}\ t10\ :\ int \to\ int \to\ int \to\ t
     \mathsf{val}\ k6\ :\ int \to\ int \to\ int \to\ t
     \mathsf{val}\ k6bar\ :\ int \to\ int \to\ int \to\ t
     val\ delta\_of\_tableau\ :\ int\ Young.tableau\ 	o\ int\ 	o\ int\ 	o\ t
     val\ t\_of\_tableau\ :\ int\ Young.tableau\ 	o\ int\ 	o\ int\ 	o\ int\ 	o\ t
  end
\mathsf{module}\ SU3\ :\ SU3\ =
  struct
     \mathsf{module}\ A\ =\ \mathit{Arrow}
     open Arrow.Infix
```

```
module B = Birdtracks
type t = B.t
let canonicalize = B.canonicalize
let to\_string = B.to\_string
let pp = B.pp
let trivial = B.trivial
let is_null = B.is_null
\mathsf{let}\ null\ =\ B.null
let const = B.const
let one = B.one
let two = B.two
let int = B.int
let half = B.half
let third = B.third
let fraction = B.fraction
let <math>nc = B.nc
let \ over\_nc = B.over\_nc
let minus = B.minus
let imag = B.imag
let ints = B.ints
let sum = B.sum
let diff = B.diff
\mathsf{let} \ \mathit{scale} \ = \ \mathit{B.scale}
let times = B.times
let multiply = B.multiply
let relocate = B.relocate
\mathsf{let}\ \mathit{fuse}\ =\ B.\mathit{fuse}
let f_of_rep = B.f_of_rep
let d_-of_-rep = B.d_-of_-rep
module Infix = B.Infix
```

Fundamental and Adjoint Representation

If the δ_{ab} originates from a $tr(T_aT_b)$, like an effective $gg \to H$ coupling, it makes a difference in the color flow basis and we must write the full expression (6.2) from [16] including the ghosts instead. Note that the sign for the terms with one ghost has not been spelled out in that reference.

```
let delta8\_loop\ a\ b = (LP.int\ 1,\ a <=> b);

(LP.int\ (-1),\ [a => a;\ ??\ b]);

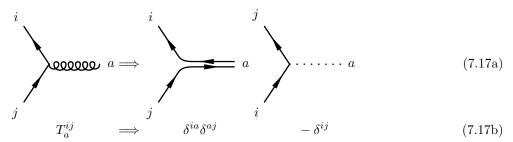
(LP.int\ (-1),\ [??\ a;\ b => b]);

(LP.nc\ 1,\ [??\ a;\ ??\ b])]
```

The following can be used for computing polarization sums (eventually, this could make the Flow module redundant). Note that we have $-N_C$ instead of $-1/N_C$ in the ghost contribution here, because two factors of $-1/N_C$ will be produced by add_arrow below, when contracting two ghost indices. Indeed, with this definition we can maintain multiply $[delta8\ 1\ (-1);\ gluon\ (-1)\ (-2);\ delta8\ (-2)\ 2] = delta8\ 1\ 2.$

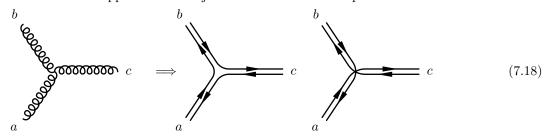
```
\begin{array}{lll} \text{let } ghost \ a \ b \ = \\ & [ \ (LP.nc \ (-1), \ [?? \ a; \ ?? \ b])] \\ \text{let } gluon \ a \ b \ = \\ & delta8 \ a \ b \ @ \ qhost \ a \ b \end{array}
```

Note that the arrow is directed from the second to the first index, opposite to our color flow paper [16]. Fortunately, this is just a matter of conventions.



```
let t \ a \ i \ j = (LP.int \ 1, \ [j => a; \ a => i]); (LP.int \ (-1), \ [j => i; \ ?? \ a])
```

Note that while we expect ${\rm tr}(T_a)=T_a^{ii}=0$, the evaluation of the expression t 1 (-1) (-1) will stop at $[-1=>1;\ 1=>-1]$ — $[-1=>-1;\ ??\ 1]$, because the summation index appears in a single term. However, a naive further evaluation would get stuck at [1=>1] — nc *** $[??\ 1]$. Fortunately, traces of single generators are never needed in our applications. We just have to resist the temptation to use them in unit tests.



```
let f \ a \ b \ c = [(LP.imag \ (1), \ A.cycle \ [a; \ b; \ c]); (LP.imag \ (-1), \ A.cycle \ [a; \ c; \ b])]
```

The generator in the adjoint representation $T_a^{bc} = -i f_{abc}$:

```
let t8 \ a \ b \ c = Birdtracks.Infix.(minus *** imag *** f a b c)
```

This d_{abc} is now compatible with (6.11) in our color flow paper [16]. The signs had been wrong in earlier versions of the code to match the missing sign in the ghost contribution to the generator T_a^{ij} above.

Decomposed Tensor Product Representations

```
let pass\_through \ m \ n \ incoming \ outgoing = List.rev\_map2 \ (fun \ i \ o \ \rightarrow \ (m, \ i) \ >=>> \ (n, \ o)) \ incoming \ outgoing let delta\_of\_permutations \ n \ permutations \ k \ l = let incoming = ThoList.range \ 0 \ (pred \ n) and normalization = List.length \ permutations in List.rev\_map (fun \ (eps, \ outgoing) \ \rightarrow \ (LP.fraction \ (eps \ \times \ normalization), \ pass\_through \ l \ k \ incoming \ outgoing))
```

```
permutations
    let totally\_symmetric n =
       List.map
          (\text{fun } p \rightarrow (1, p))
          (Combinatorics.permute\ (ThoList.range\ 0\ (pred\ n)))
    let totally\_antisymmetric\ n\ =
         (Combinatorics.permute\_signed\ (ThoList.range\ 0\ (pred\ n)))
    let delta\_S n k l =
       delta\_of\_permutations \ n \ (totally\_symmetric \ n) \ k \ l
    let delta_A n k l =
       delta\_of\_permutations n (totally\_antisymmetric n) k l
    let \ delta6 \ = \ delta\_S \ 2
    let \ delta10 = delta\_S \ 3
    let \ delta15 = delta\_S \ 4
    let delta3bar = delta\_A 2
Mixed symmetries, as in section 9.4 of the birdtracks book.
    module IM = Partial.Make (struct type t = int let compare = pcompare end)
    module P = Permutation.Default
Map the elements of original to permuted in all, with all a list of n integers from 0 to n-1 in order, and use
the resulting list to define a permutation. E.g. permute_partial [1, 3] [3, 1] [0, 1, 2, 3, 4] will define a permutation
that transposes the second and fourth element in a 5 element list.
    {\tt let} \ permute\_partial \ original \ permuted \ all \ =
       P.of_list (List.map (IM.auto (IM.of_lists original permuted)) all)
    let apply1 (sign, indices) (eps, p) =
       (eps \times sign, P.list \ p \ indices)
    let apply signed_permutations signed_indices =
       List.rev_map (apply1 signed_indices) signed_permutations
    let \ apply\_list \ signed\_permutations \ signed\_indices \ =
       ThoList.flatmap (apply signed_permutations) signed_indices
    let symmetrizer\_of\_permutations n original signed\_permutations =
       let incoming = ThoList.range \ 0 \ (pred \ n) in
       List.rev\_map
         (fun (eps, permuted) \rightarrow
            (eps, permute_partial original permuted incoming))
         signed\_permutations
    let symmetrizer n indices =
       symmetrizer\_of\_permutations
         n indices
         (List.rev\_map \ (fun \ p \rightarrow (1, \ p)) \ (Combinatorics.permute \ indices))
    let \ anti\_symmetrizer \ n \ indices =
       symmetrizer\_of\_permutations
          n indices
          (Combinatorics.permute\_signed\ indices)
    let symmetrize n elements indices =
       apply_list (symmetrizer n elements) indices
    let anti\_symmetrize \ n \ elements \ indices =
       apply_list (anti_symmetrizer n elements) indices
    let id \ n =
       [(1, ThoList.range\ 0\ (pred\ n))]
```



We can avoid the recursion here, if we use $Combinatorics.permute_tensor_signed$ in symmetrizer above.



Here we should at a sanity test for *tableau*: all integers should be consecutive starting from 0 with no duplicates. In additions the rows must not grow in length.

```
let delta\_of\_tableau\ tableau\ i\ j\ =
       \mathsf{let}\ n\ =\ Young.num\_cells\_tableau\ tableau
       and num, den = Young.normalization (Young.diagram_of_tableau tableau)
       and rows = tableau
       and cols = Young.conjugate\_tableau \ tableau \ in
       let permutations =
          apply\_tableau symmetrize n rows (apply\_tableau anti\_symmetrize n cols (id n)) in
       Birdtracks.Infix.(int num *** fraction den *** delta_of_permutations n permutations i j)
    let incomplete tensor =
       failwith \ ("Color.Vertex: \_" \ \hat{} \ tensor \ \hat{} \ "\_not\_supported\_yet!")
    let experimental tensor =
       Printf.eprintf
          "Color.Vertex: \( \)\%s\( \)\support \( \)\still \( \)\experimental \( \)\and \( \)\understant \( \)\n"
          tensor
    let \ distinct \ integers =
       let rec distinct' seen = function
          | [] \rightarrow \mathsf{true}
          i :: rest \rightarrow
             if Sets.Int.mem i seen then
                false
             else
                distinct' (Sets.Int.add i seen) rest in
       distinct' Sets.Int.empty integers
All lines start here: they point towards the vertex.
    let \ epsilon \ tips =
       if distinct tips then
          [(LP.int\ 1,\ [Arrow.epsilon\ tips])]
       else
          null
All lines end here: they point away from the vertex.
    let ensilon_bar tails =
       if distinct tails then
          [(LP.int\ 1,\ [Arrow.epsilon\_bar\ tails])]
       else
          null
```

In order to get the correct N_C dependence of quadratic Casimir operators, the arrows in the vertex must have the same permutation symmetry as the propagator. This is demonstrated by the unit tests involving Casimir operators on page 115 below. These tests also provide a check of our normalization.

The implementation takes a propagator and uses Arrow.tee to replace one arrow by the pair of arrows corresponding to the insertion of a gluon. This is repeated for each arrow. The normalization remains unchanged from the propagator. A minus sign is added for antiparallel arrows, since the conjugate representation is $-T_a^*$.

To this, we add the diagrams with a gluon connected to one arrow. Since these are identical, only one diagram multiplied by the difference of the number of parallel and antiparallel arrows is added.

```
\begin{array}{lll} \text{let } insert\_gluon \ a \ k \ l \ (norm, \ arrows) \ = \\ & \text{let } rec \ insert\_gluon' \ acc \ left \ = \ function \\ & | \ [] \ \rightarrow \ acc \\ & | \ arrow \ :: \ right \ \rightarrow \end{array}
```

```
insert\_qluon'
                ((Algebra.Laurent.mul (LP.int (A.dir k l arrow)) norm,
                   List.rev\_append\ left\ ((A.tee\ a\ arrow)\ @\ right))\ ::\ acc)
                (arrow :: left)
                right in
        insert_gluon' [] [] arrows
     let t_-of_-delta\ delta\ a\ k\ l\ =
       match delta \ k \ l with
        | [] \rightarrow []
        | (\_, arrows) :: \_ as delta\_kl \rightarrow
           let n =
              List.fold\_left
                (fun\ acc\ arrow\ 	o\ acc\ +\ A.dir\ k\ l\ arrow)
                0 arrows in
           let qhosts =
              List.rev\_map
                (fun (norm, arrows) \rightarrow
                   (Algebra.Laurent.mul\ (LP.int\ (-n))\ norm,\ ??\ a\ ::\ arrows))
                delta\_kl in
           List.fold\_left
              (fun\ acc\ arrows\ 	o\ insert\_gluon\ a\ k\ l\ arrows\ @\ acc)
              ghosts delta_kl
     let t_-of_-delta \ delta \ a \ k \ l =
        canonicalize (t\_of\_delta\ delta\ a\ k\ l)
     let t_-S n a k l =
        t\_of\_delta (delta\_S n) \ a \ k \ l
     let t_A n a k l =
        t\_of\_delta (delta\_A \ n) \ a \ k \ l
     \mathsf{let}\ t6\ =\ t \mathsf{\_} S\ 2
     let t10 = t_-S 3
     let t15 = t_-S 4
     let t3bar = t\_A 2
Equivalent definition:
     let t8' a b c =
        t\_of\_delta delta8 a b c
     let t_of_tableau tableau a k l =
       t\_of\_delta (delta\_of\_tableau tableau) a \ k \ l
    Check the following for a real live UFO file!
In the UFO paper, the Clebsh-Gordan is defined as K^{(6),ij}_{m}. Therefore, keeping our convention for the gener-
ators T_a^{(6),j}, the must arrows end at m.
     let k6 \ m \ i \ j =
        experimental "k6";
        [(LP.int 1, [i =>> (m, 0); j =>> (m, 1)]);
          (LP.int 1, [i =>> (m, 1); j =>> (m, 0)])
The arrow are reversed for \bar{K}^{(6),m}_{ij} and start at m.
     \mathsf{let}\ k6bar\ m\ i\ j\ =
        experimental "k6bar";
       [(LP.int 1, [(m, 0) >=> i; (m, 1) >=> j]);
          (LP.int 1, [(m, 1) >=> i; (m, 0) >=> j])
```

 $module \ Test : Test =$

Unit Tests

```
struct
           open OUnit
           module L = Algebra.Laurent
           module B = Birdtracks
           open Birdtracks
           open Birdtracks.Infix
           let \ exorcise \ vertex =
             List.filter
                (fun (\_, arrows) \rightarrow \neg (List.exists A.is\_ghost arrows))
                vertex
           let eqx v1 v2 =
             eq (exorcise v1) (exorcise v2)
                                                               Trivia
           let suite\_sum =
              "sum" >:::
                [ "atoms" >::
                      (fun () \rightarrow
                           (two *** delta3 1 2)
                           (delta3 \ 1 \ 2 \ +++ \ delta3 \ 1 \ 2))
           \mathsf{let} \ \mathit{suite} \, \_\mathit{diff} \ =
             "diff" >:::
                [ "atoms" >::
                      (\mathsf{fun}\ ()\ \to
                           (delta3 3 4)
                           (delta3 \ 1 \ 2 \ +++ \ delta3 \ 3 \ 4 \ --- \ delta3 \ 1 \ 2))
                                                              \prod_{k=i}^{j} f(k)
                                                                                                                              (7.19)
           let rec product f i j =
             if i > j then
                null
             else if i = j then
                f i
             else
                f \ i \ *** product f (succ i) j
In particular
  product\ (nc\_minus\_n\_plus\ n)\ i\ j\ \mapsto
                  \prod_{k=i}^{j} (N_C - n + k) = \frac{(N_C - n + j)!}{(N_C - n + i - 1)!} = (N_C - n + j)(N_C - n + j - 1) \cdots (N_C - n + i) \quad (7.20)
           let \ nc\_minus\_n\_plus \ n \ k \ =
             const\ (LP.ints\ [\ (1,\ 1);\ (-n\ +\ k,\ 0)\ ])
           let contractions \ rank \ k =
             product (nc\_minus\_n\_plus \ rank) \ 1 \ k
           let suite\_times =
              "times" >:::
```

```
[ "reorder_components_t1*t2" >:: (* trivial T_a^{ik}T_a^{kj} = T_a^{kj}T_a^{ik} *)
     (fun () \rightarrow
       let t1 = t(-1) 1(-2)
       and t2 = t(-1)(-2) 2 in
       eq(t1 *** t2)(t2 *** t1);
  "reorder_components_tr(t1*t2)" >:: (* trivial T_a^{ij}T_a^{ji} = T_a^{ji}T_a^{ij} *)
     (fun () \rightarrow
       let t1 = t 1 (-1) (-2)
       and t2 = t \ 2 \ (-2) \ (-1) in
       eq (t1 *** t2) (t2 *** t1));
  "reorderings" >::
     (fun () \rightarrow
       let v1 = [(L.unit, [1 => -2; -2 => -1; -1 => 1])]
       and v2 = [(L.unit, [-1 => 2; 2 => -2; -2 => -1])]
       and v' = [(L.unit, [1 => 1; 2 => 2])] in
       eq \ v' \ (v1 \ *** \ v2));
  "eps*epsbar" >::
     (fun () \rightarrow
       eq
          (delta3 \ 1 \ 2 \ *** \ delta3 \ 3 \ 4 \ --- \ delta3 \ 1 \ 4 \ *** \ delta3 \ 3 \ 2)
          (epsilon [1; 3] *** epsilon_bar [2; 4]));
  "eps*epsbar<sub>□</sub>-" >::
     (fun () \rightarrow
       eq
          (delta3\ 1\ 4\ ***\ delta3\ 3\ 2\ ---\ delta3\ 1\ 2\ ***\ delta3\ 3\ 4)
          (epsilon [1; 3] *** epsilon\_bar [4; 2]));
  "eps*epsbar<sub>□</sub>1" >::
     (\mathsf{fun}\ ()\ \to
        eq (* N_C - 3 + 1 = (N_C - 2), \text{ for } NC = 3: 1*)
          (contractions \ 3 \ 1 \ ***
              (delta3 \ 1 \ 2 \ *** \ delta3 \ 3 \ 4 \ --- \ delta3 \ 1 \ 4 \ *** \ delta3 \ 3 \ 2))
          (epsilon [-1; 1; 3] *** epsilon_bar [-1; 2; 4]));
  "eps*epsbar_cyclic_1">::
     (fun () \rightarrow
       eq (*N_C - 3 + 1 = (N_C - 2), \text{ for } NC = 3: 1*)
          (contractions \ 3 \ 1 \ ***
              (delta3\ 1\ 2\ ***\ delta3\ 3\ 4\ ---\ delta3\ 1\ 4\ ***\ delta3\ 3\ 2))
          (epsilon [3; -1; 1] *** epsilon_bar [-1; 2; 4]));
  "eps*epsbar_cyclic_2">::
     (fun () \rightarrow
       eq (*N_C - 3 + 1 = (N_C - 2), \text{ for } NC = 3: 1*)
          (contractions \ 3 \ 1 \ ***
              (delta3\ 1\ 2\ ***\ delta3\ 3\ 4\ ---\ delta3\ 1\ 4\ ***\ delta3\ 3\ 2))
          (epsilon [-1; 1; 3] *** epsilon_bar [4; -1; 2]));
  "eps*epsbar<sub>□</sub>2" >::
     (fun () \rightarrow
       eq (*(N_C - 3 + 2)(N_C - 3 + 1) = (N_C - 1)(N_C - 2), for NC = 3: 2*)
          (contractions \ 3 \ 2 \ *** \ delta3 \ 1 \ 2)
          (epsilon [-1; -2; 1] *** epsilon_bar [-1; -2; 2]));
  "eps*epsbar<sub>□</sub>3" >::
     (fun () \rightarrow
        eq (* (N_C - 3 + 3)(N_C - 3 + 2)(N_C - 3 + 1)) = N_C(N_C - 1)(N_C - 2), \text{ for } NC = 3: 3! *)
          (contractions 3 3)
          (epsilon [-1; -2; -3] *** epsilon\_bar [-1; -2; -3]));
  "eps*epsbar_big" >::
     (fun () \rightarrow
```

```
*)  eq \ (* \ (N_C - 5 + 3)(N_C - 5 + 2)(N_C - 5 + 1) = (N_C - 2)(N_C - 3)(N_C - 4), \text{ for } NC = 5: 3! 
 (contractions 5 \ 3 \ *** \\ (epsilon \ [4; \ 5] \ *** \ epsilon\_bar \ [6; \ 7])) \\ (epsilon \ [-1; \ -2; \ -3; \ 4; \ 5] \ *** \ epsilon\_bar \ [-1; \ -2; \ -3; \ 6; \ 7])); 
 "eps*epsbar_ubig_u-" > :: \\ (fun \ () \rightarrow eq \ (* \ (N_C - 5 + 3)(N_C - 5 + 2)(N_C - 5 + 1) = (N_C - 2)(N_C - 3)(N_C - 4), \text{ for } NC = 5: 3! 
 (contractions 5 \ 3 \ *** \\ (epsilon \ [5; \ 4] \ *** \ epsilon\_bar \ [6; \ 7])) \\ (epsilon \ [-1; \ 4; \ -3; \ -2; \ 5] \ *** \ epsilon\_bar \ [-1; \ -2; \ -3; \ 6; \ 7])) \ ] 
 Propagators
```

Verify the normalization of the propagators by making sure that $D^{ij}D^{jk} = D^{ik}$

Pass every arrow straight through, without (anti-)symmetrization.

```
let delta\_unsymmetrized\ n\ k\ l = \\ delta\_of\_permutations\ n\ [(1,\ ThoList.range\ 0\ (pred\ n))]\ k\ l
let completeness\ n\ tableaux\ = \\ eq \\ (delta\_unsymmetrized\ n\ 1\ 2) \\ (sum\ (List.map\ (fun\ t\ \to\ delta\_of\_tableau\ t\ 1\ 2)\ tableaux))
```

The following names are of historical origin. From the time, when we didn't have full support for Young tableaux and implemented figure 9.1 from the birdtrack book.

$$\begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix} \tag{7.21}$$

let $delta_ASA \ i \ j =$

$$\begin{bmatrix} 0 & 2 \\ 1 \end{bmatrix} \tag{7.22}$$

```
delta\_of\_tableau [[0; 2]; [1]] i j
let suite\_propagators =
   "propagators" >:::
     ["D*D=D" >:: (fun () \rightarrow projection_id delta3);
       "D8*D8=D8" >:: (fun () \rightarrow projection_id delta8);
       "G*G=G" >:: (fun () \rightarrow projection_id gluon);
       "D6*D6=D6" >:: (fun () \rightarrow projection_id delta6);
       "D10*D10=D10" >:: (fun () \rightarrow projection_id delta10);
       "D15*D15=D15" >:: (fun () \rightarrow projection_id delta15);
       "D3bar*D3bar=D3bar" >:: (fun () \rightarrow projection_id delta3bar);
       "D6*D3bar=0" >:: (fun () \rightarrow orthogonality delta6 delta3bar);
       "D_A3*D_A3=D_A3" >:: (fun () \rightarrow projection_id (delta_A 3));
       "D10*D_A3=0" >:: (fun () \rightarrow orthogonality delta10 (delta_A 3));
       "D_SAS*D_SAS=D_SAS" >:: (fun () \rightarrow projection_id delta_SAS);
       "D_ASA*D_ASA=D_ASA" >:: (fun () \rightarrow projection_id delta_ASA);
       "D_SAS*D_S3=0" >:: (fun () \rightarrow orthogonality delta_SAS (delta_S 3));
       "D_SAS*D_A3=0" >:: (fun () \rightarrow orthogonality delta_SAS (delta_A 3));
       "D_SAS*D_ASA=0" >:: (fun () \rightarrow orthogonality delta_SAS delta_ASA);
```

```
"D_ASA*D_SAS=0" >:: (fun () \rightarrow orthogonality delta_ASA delta_SAS);
                   "D_ASA*D_S3=0" >:: (fun () \rightarrow orthogonality delta_ASA (delta_S 3));
                   "D_ASA*D_A3=0" >:: (fun () \rightarrow orthogonality delta_ASA (delta_A 3));
                   "DU*DU=DU" >:: (fun () \rightarrow projection_id (delta_unsymmetrized 3));
                   "S3=[0123]" >::
                     (fun () \rightarrow
                        eq (delta_S 4 1 2) (delta_of_tableau [[0; 1; 2; 3]] 1 2));
                   "A3=[0,1,2,3]" >::
                     (fun () \rightarrow
                        eq (delta_A 4 1 2) (delta_of_tableau [[0]; [1]; [2]; [3]] 1 2));
                   "[0123]*[012,3]=0" >::
                     (fun () \rightarrow
                        orthogonality
                           (delta\_of\_tableau [[0; 1; 2; 3]])
                           (delta\_of\_tableau\ [[0;1;2];[3]]);
                   "[0123]*[01,23]=0">::
                      (fun () \rightarrow
                        orthogonality
                           (delta\_of\_tableau [[0;1;2;3]])
                           (delta\_of\_tableau\ [[0;1];[2;3]]);
                   "[012,3]*[012,3]=[012,3]">::
                     (fun () \rightarrow projection\_id (delta\_of\_tableau [[0; 1; 2]; [3]]));
                                                           01 + 0
                                                                                                                            (7.23)
                   "completeness2" >:: (fun () \rightarrow completeness 2 [ [[0;1]]; [[0];[1]] );
                   "completeness, 2' >::
                     (fun () \rightarrow
                        eq
                           (delta_unsymmetrized 2 1 2)
                           (delta\_S \ 2 \ 1 \ 2 \ +++ \ delta\_A \ 2 \ 1 \ 2));
The normalization factors are written for illustration. They are added by delta_of_tableau automatically.
                                              \boxed{012} + \frac{4}{3} \cdot \boxed{01}{2} + \frac{4}{3} \cdot \boxed{02}{1} + \boxed{0}{1}{2}
                                                                                                                            (7.24)
                   "completeness<sub>□</sub>3">::
                      (fun () \rightarrow completeness 3 [ [[0;1;2]]; [[0;1];[2]]; [[0;2];[1]]; [[0];[1];[2]] ));
                   "completeness_{\sqcup}3'" >::
                     (fun () \rightarrow
                        eq
```

 $\begin{array}{l} \text{"completeness}_{\square}3\text{'"}>::\\ & (\text{fun ()} \to \\ eq\\ & (\textit{delta_unsymmetrized } 3\ 1\ 2)\\ & (\textit{delta_S}\ 3\ 1\ 2\ +++\ delta_SAS\ 1\ 2\ +++\ delta_ASA\ 1\ 2\ +++\ delta_A\ 3\ 1\ 2)); \\ \hline \\ \boxed{01123} + \frac{3}{2} \cdot \frac{0112}{3} + \frac{3}{2} \cdot \frac{0133}{2} + \frac{3}{2} \cdot \frac{023}{2} + \frac{4}{3} \cdot \frac{01}{23} + \frac{4}{3} \cdot \frac{02}{13} + \frac{3}{2} \cdot \frac{01}{2} + \frac{3}{2} \cdot \frac{02}{13} + \frac{3}{2} \cdot \frac{03}{12} + \frac{3}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \\ \\ \text{"completeness}_{\square}4\text{"}>::\\ & (\text{fun ()} \to \\ & \textit{completeness 4} \\ & [[0;1;2;3]];\\ & [[0;1;2;3]];\\ & [[0;1;2];3]];\\ & [[0;2];[1;3]];\\ & [[0;2];[1];[2];[3]];\\ & [[0;2];[1];[2];[3]];\\ & [[0;1];[2];[3]];\\ & [[0;2];[1];[2];[3]];\\ & [[0;2];[1];[2];[3]];\\ & [[0;1];[2];[3];\\ & [[0;1];[2];[3]];\\ & [[0;1];[2];[3];\\ & [[0;1];[2];[3]];\\ & [[0;1];[2];[2];\\ & [[0;1];[2];[2];\\ & [[0;1];[2];[2];\\ & [[0;1];[2];[2];\\ & [[0;1];[2];[2];\\ & [[0;1];[2];[2];\\ & [[0;1];[2];[2];\\ & [[0;1];[2];[2];\\ & [[0;1];[2];[2];\\ & [[0;1];[2];[2];\\ & [[0;1];[2];[2];\\ & [[0;1];[2];[2];\\ & [[0;1];[2];[2];\\ & [[$

Normalization

```
\begin{split} &\texttt{let } \textit{suite\_normalization} &= \\ &\texttt{"normalization"} > \texttt{:::} \\ &\texttt{["tr(t*t)"} > \texttt{:::} (* \, \text{tr}(T_a T_b) = \delta_{ab} + \text{ghosts *}) \\ &\texttt{(fun ()} \rightarrow \\ &eq \\ &\texttt{(} \textit{delta8\_loop 1 2)} \\ &\texttt{(} t \ 1 \ (-1) \ (-2) \ *** \ t \ 2 \ (-2) \ (-1))); \\ &\texttt{"tr(t*t)\_sans\_ghosts"} > \texttt{::} \ (* \, \text{tr}(T_a T_b) = \delta_{ab} \ *) \\ &\texttt{(fun ()} \rightarrow \\ &eqx \\ &\texttt{(} \textit{delta8 1 2)} \\ &\texttt{(} t \ 1 \ (-1) \ (-2) \ *** \ t \ 2 \ (-2) \ (-1))); \\ \end{split}
```

The additional ghostly terms were unexpected, but arises like (6.2) in our color flow paper [16].

```
"t*t*t" >:: (* T_a T_b T_a = -T_b/N_C + \dots *)

(fun () \rightarrow

eq

(minus *** over_nc *** t 1 2 3

+++ [(LP.int \ 1, \ [1 => \ 1; \ 3 => \ 2]);

(LP.nc \ (-1), \ [3 => \ 2; \ ?? \ 1])])

(<math>t \ (-1) \ 2 \ (-2) \ *** \ t \ 1 \ (-2) \ (-3) \ *** \ t \ (-1) \ (-3) \ 3));
```

As expected, these ghostly terms cancel in the summed squares

$$tr(T_a T_b T_a T_c T_b T_c) = tr(T_b T_b) / N_C^2 = \delta_{bb} / N_C^2 = (N_C^2 - 1) / N_C^2 = 1 - 1 / N_C^2$$
(7.26)

As proposed in our color flow paper [16], we can get the correct (anti-)symmetrized generators by sandwiching the following unsymmetrized generators between the corresponding (anti-)symmetrized projectors. Therefore, the unsymmetrized generators work as long as they're used in Feynman diagrams, where they are connected by propagators that contain (anti-)symmetrized projectors. They even work in the Lie algebra relations and give the correct normalization there.

They fail however for more general color algebra expressions that can appear in UFO files. In particular, the Casimir operators come out really wrong.

```
 \begin{array}{lll} \text{let } t\_unsymmetrized \ n \ k \ l \ = \\ t\_of\_delta \ (delta\_unsymmetrized \ n) \ k \ l \end{array}
```

The following trivial vertices are *not* used anymore, since they don't get the normalization of the Ward identities right. For the quadratic casimir operators, they always produce a result proportional to $C_F = C_2(S_1)$. This can be understood because they correspond to a fundamental representation with spectators.

(Anti-)symmetrizing by sandwiching with projectors almost works, but they must be multiplied by hand by the number of arrows to get the normalization right. They're here just for documenting what doesn't work.

```
let t\_trivial\ n\ a\ k\ l =

let sterile =

List.map\ (fun\ i \to (l,\ i) >=>> (k,\ i))\ (ThoList.range\ 1\ (pred\ n))\ in

[ (LP.int\ (\ 1),\ ((l,\ 0)\ >=>\ a)\ ::\ (a\ =>>\ (k,\ 0))\ ::\ sterile);
```

```
(LP.int (-1), (?? a) :: ((l, 0) >=>> (k, 0)) :: sterile)
\mathsf{let}\ t6\_trivial\ =\ t\_trivial\ 2
let t10\_trivial = t\_trivial 3
let t15\_trivial = t\_trivial 4
let t_SAS = t_of_delta delta_SAS
let t_ASA = t_of_delta delta_ASA
let symmetrization ?rep_ts rep_tu rep_d =
  let rep_ts =
     \mathsf{match}\ \mathit{rep\_ts}\ \mathsf{with}
       None \rightarrow rep_tu
       Some \ rep_t \rightarrow rep_t \ in
  eq
     (rep_ts 1 2 3)
     (gluon\ 1\ (-1)\ ***\ rep\_d\ 2\ (-2)\ ***\ rep\_tu\ (-1)\ (-2)\ (-3)\ ***\ rep\_d\ (-3)\ 3)
let suite\_symmetrization =
   "symmetrization" >:::
     ["t6" >:: (fun () \rightarrow symmetrization t6 delta6);
        "t10" >:: (fun () \rightarrow symmetrization t10 delta10);
        "t15" >:: (fun () \rightarrow symmetrization t15 delta15);
        "t3bar" >:: (fun () \rightarrow symmetrization t3bar delta3bar);
        "t_SAS" >:: (fun () \rightarrow symmetrization t_SAS delta_SAS);
        "t_ASA" >:: (fun () \rightarrow symmetrization t_ASA delta_ASA);
        "t6'," >:: (fun () \rightarrow symmetrization ~rep_ts: t6 (t_unsymmetrized 2) delta6);
        "t10'" >:: (fun () \rightarrow symmetrization \sim rep_ts: t10 (t_unsymmetrized 3) delta10);
        "t15'" >:: (fun () \rightarrow symmetrization ~rep_ts: t15 (t_unsymmetrized 4) delta15);
        "t6'," >::
          (fun () \rightarrow
             eq
                (t6\ 1\ 2\ 3)
               (int\ 2^{***}\ delta6\ 2\ (-1)\ ***\ t6\_trivial\ 1\ (-1)\ (-2)\ ***\ delta6\ (-2)\ 3));
        "t10'," >::
          (\mathsf{fun}\ ()\ \to
             eq
                (t10\ 1\ 2\ 3)
               (int \ 3 *** \ delta10 \ 2 \ (-1) \ *** \ t10 \ trivial \ 1 \ (-1) \ (-2) \ *** \ delta10 \ (-2) \ 3));
        "t15'," >::
          (fun () \rightarrow
             eq
               (t15\ 1\ 2\ 3)
               (int 4 *** delta15 2 (-1) *** t15_trivial 1 (-1) (-2) *** delta15 (-2) 3))]
```

Traces

Compute (anti-)commutators of generators in the representation r, i. e. $[r(t_a)r(t_b)]_{ij} \mp [r(t_b)r(t_a)]_{ij}$, using isum < 0 as summation index in the matrix products.

```
let commutator rep_t i_sum a b i j =
    multiply [rep_t a i i_sum; rep_t b i_sum j]
    — multiply [rep_t b i i_sum; rep_t a i_sum j]
let anti_commutator rep_t i_sum a b i j =
    multiply [rep_t a i i_sum; rep_t b i_sum j]
    +++ multiply [rep_t b i i_sum; rep_t a i_sum j]
```

Trace of the product of three generators in the representation r, i.e. $\operatorname{tr}_r(r(t_a)r(t_b)r(t_c))$, using -1, -2, -3 as summation indices in the matrix products.

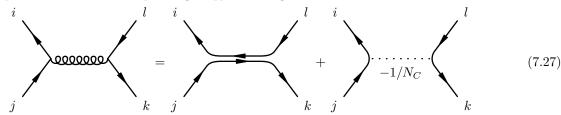
```
let trace3 \ rep_t \ a \ b \ c = rep_t \ a \ (-1) \ (-2) \ *** \ rep_t \ b \ (-2) \ (-3) \ *** \ rep_t \ c \ (-3) \ (-1)
```

```
let loop3 a b c =
               [ (LP.int\ 1,\ A.cycle\ (List.rev\ [a;\ b;\ c]));
                  (LP.int (-1), (a <=> b) @ [?? c]);
                  (LP.int (-1), (b <=> c) @ [?? a]);
                  (LP.int (-1), (c <=> a) @ [?? b]);
                  (LP.int 1, [a => a; ?? b; ?? c]);
                  (LP.int \ 1, \ [?? \ a; \ b \ => \ b; \ ?? \ c]);
                  (LP.int \ 1, \ [?? \ a; \ ?? \ b; \ c \ => \ c]);
                 (LP.nc\ (-1),\ [??\ a;\ ??\ b;\ ??\ c])
            let suite\_trace =
               "trace" >:::
                  [ "tr(ttt)" >::
                       (fun () \rightarrow eq (trace3 t 1 2 3) (loop3 1 2 3));
                    "tr(ttt)_{\square}cyclic_{\square}1" >:: (* tr(T_aT_bT_c) = tr(T_bT_cT_a) *)
                       (fun () \rightarrow eq (trace3 t 1 2 3) (trace3 t 2 3 1));
                    "tr(ttt)_{\square}cyclic_{\square}2" >:: (* tr(T_aT_bT_c) = tr(T_cT_aT_b) *)
                       (fun () \rightarrow eq (trace3 t 1 2 3) (trace3 t 3 1 2));
Do we expect this?
                    "tr(tttt)" >:: (* \operatorname{tr}(T_a T_b T_c T_d) = \dots *)
                       (fun () \rightarrow
                          eqx
                            [(LP.int 1, A.cycle [4; 3; 2; 1])]
                            (t\ 1\ (-1)\ (-2)\ ***\ t\ 2\ (-2)\ (-3)\ ***\ t\ 3\ (-3)\ (-4)\ ***\ t\ 4\ (-4)\ (-1)))\ ]
            let suite\_ghosts =
               "ghosts" >:::
                  [ "H->gg" >::
                       (fun () \rightarrow
                          eq
                            (delta8\_loop 1 2)
                            (t\ 1\ (-1)\ (-2)\ ***\ t\ 2\ (-2)\ (-1)));
                    "H->ggg_{\sqcup}f" >::
                       (fun () \rightarrow
                          eq
                            (imag *** f 1 2 3)
                            (trace3\ t\ 1\ 2\ 3\ ---\ trace3\ t\ 1\ 3\ 2));
                    "H->ggg_{\sqcup}d">::
                       (fun () \rightarrow
                          eq
                            (d\ 1\ 2\ 3)
                            (trace3\ t\ 1\ 2\ 3\ +++\ trace3\ t\ 1\ 3\ 2));
                    "H->ggg_{\sqcup}f'">::
                       (fun () \rightarrow
                            (imag **** f 1 2 3)
                            (t\ 1\ (-3)\ (-2)\ ***\ commutator\ t\ (-1)\ 2\ 3\ (-2)\ (-3)));
                    "H->ggg \sqcup d'" >::
                       (fun () \rightarrow
                          eq
                            (d\ 1\ 2\ 3)
                            (t\ 1\ (-3)\ (-2)\ ***\ anti\_commutator\ t\ (-1)\ 2\ 3\ (-2)\ (-3)));
                    "H->ggg⊔cyclic'">::
                       (fun () \rightarrow
                          \mathsf{let}\ trace\ a\ b\ c\ =
```

```
t \ a \ (-3) \ (-2) \ *** \ commutator \ t \ (-1) \ b \ c \ (-2) \ (-3) \ in
             eq (trace 1 2 3) (trace 2 3 1)) ]
let ff a1 a2 a3 a4 =
  [(LP.int (-1), A.cycle [a1; a2; a3; a4]);
     (LP.int\ (\ 1),\ A.cycle\ [a2;\ a1;\ a3;\ a4]);
     (LP.int\ (1),\ A.cycle\ [a1;\ a2;\ a4;\ a3]);
     (LP.int (-1), A.cycle [a2; a1; a4; a3])
\mathsf{let}\ \mathit{tf}\ \mathit{j}\ \mathit{i}\ \mathit{a}\ \mathit{b}\ =
  [(LP.imag(1), A.chain[i; a; b; j]);
     (LP.imag(-1), A.chain[i; b; a; j])
let suite_{-}ff =
   "f*f" >:::
     [ "1" > :: (fun () \rightarrow eq (ff 1 2 3 4) (f (-1) 1 2 * * * * f (-1) 3 4));
        "2" >:: (fun () \rightarrow eq (ff 1 2 3 4) (f (-1) 1 2 *** f 3 4 (-1)));
        "3" >:: (fun () \rightarrow eq (ff 1 2 3 4) (f (-1) 1 2 *** f 4 (-1) 3))
let suite_tf =
   "t*f" >:::
     ["1">:: (fun () \rightarrow eq (tf 1 2 3 4) (t (-1) 1 2 *** f (-1) 3 4))]
```

Completeness Relation

Check the completeness relation corresponding to $q\bar{q}$ -scattering:



 $T_a^{ij}T_a^{kl}$

$$\begin{array}{l} \text{let } tt \ i \ j \ k \ l = \\ t \ (-1) \ i \ j \ **** \ t \ (-1) \ k \ l \\ \delta^{il} \delta^{kj} - \delta^{ij} \delta^{kl} / N_C \\ \\ \text{let } tt_expected \ i \ j \ k \ l = \\ [\ (LP.int \ 1, \ [l => \ i; \ j => \ k]); \\ (LP.over_nc \ (-1), \ [j => \ i; \ l => \ k]) \] \\ \\ \text{let } suite_tt = \\ \text{"t*t"} > ::: \\ [\ "t*" > ::: \ (* \ T_a^{ij} T_a^{kl} = \delta^{il} \delta^{kj} - \delta^{ij} \delta^{kl} / N_C \ *) \\ (\text{fun } () \ \rightarrow \ eq \ (tt_expected \ 1 \ 2 \ 3 \ 4) \ (tt \ 1 \ 2 \ 3 \ 4)) \] \end{array}$$

$Lie\ Algebra$

Check the commutation relations $[T_a, T_b] = i f_{abc} T_c$ in various representations.

```
\begin{array}{lll} \text{let } \mathit{lie\_algebra\_id } \ \mathit{rep\_t} = \\ & \text{let } \mathit{lhs} = \mathit{imag} \ *** \ \mathit{f} \ 1 \ 2 \ (-1) \ *** \ \mathit{t} \ (-1) \ 3 \ 4 \\ & \text{and } \mathit{rhs} = \mathit{commutator} \ \mathit{t} \ (-1) \ 1 \ 2 \ 3 \ 4 \ \mathsf{in} \\ & \mathit{eq } \mathit{lhs } \mathit{rhs} \end{array}
```

Check the normalization of the structure consistants $\mathcal{N}f_{abc} = -i\operatorname{tr}(T_a[T_b, T_c])$

```
\begin{array}{lll} \text{let } f\_of\_rep\_id \ norm \ rep\_t = \\ \text{let } lhs = norm \ *** f \ 1 \ 2 \ 3 \\ \text{and } rhs = f\_of\_rep \ rep\_t \ 1 \ 2 \ 3 \ \text{in} \\ eq \ lhs \ rhs \end{array}
```



Are the normalization factors for the traces of the higher dimensional representations correct?



The traces don't work for the symmetrized generators that we need elsewhere!

```
let suite\_lie =
   "Lie\squarealgebra\squarerelations" >:::
     ["[t,t]=ift">:: (fun () \rightarrow lie\_algebra\_id t);
        "[t8,t8]=ift8" >:: (fun () \rightarrow lie\_algebra\_id \ t8);
        "[t6,t6]=ift6" >:: (fun () \rightarrow lie\_algebra\_id t6);
        "[t10,t10]=ift10" >:: (fun () \rightarrow lie\_algebra\_id t10);
        "[t15,t15]=ift15" >:: (fun () \rightarrow lie\_algebra\_id t15);
        "[t3bar,t3bar]=ift3bar" >:: (fun () \rightarrow lie\_algebra\_id t3bar);
        "[tSAS,tSAS]=iftSAS" >:: (fun () \rightarrow lie\_algebra\_id t\_SAS);
        "[tASA,tASA]=iftASA" >:: (fun () \rightarrow lie\_algebra\_id t\_ASA);
        "[t6,t6]=ift6'," >:: (fun () \rightarrow lie\_algebra\_id (t\_unsymmetrized 2));
        "[t10,t10]=ift10'," >:: (fun () \rightarrow lie_algebra_id (t_unsymmetrized 3));
        "[t15,t15]=ift15'" >:: (fun () \rightarrow lie\_algebra\_id (t\_unsymmetrized 4));
        "[t6,t6]=ift6'," >:: (fun () \rightarrow lie\_algebra\_id t6\_trivial);
        "[t10,t10]=ift10'," >:: (fun () \rightarrow lie\_algebra\_id t10\_trivial);
        "[t15,t15]=ift15'," >:: (fun () \rightarrow lie\_algebra\_id t15\_trivial);
        "if_{\sqcup}=_{\sqcup}tr(t[t,t])" >:: (fun () \rightarrow f_{-}of_{-}rep_{-}id one t);
        "2n*if_{\perp}=_{\perp}tr(t8[t8,t8])" >:: (fun () \rightarrow f_{-}of_{-}rep_{-}id (two *** nc) t8);
        "n*if_{=}tr(t6[t6,t6])" >:: (fun () \rightarrow f_{-}of_{-}rep_{-}id \ nc \ t6_{-}trivial);
        "\texttt{n^2*if}_{\sqcup} = _{\sqcup} \texttt{tr(t10[t10,t10])}" > :: (\texttt{fun} () \rightarrow f_{-}of_{-}rep_{-}id (nc *** nc) t10\_trivial);
        "n^3*if_{\perp}=_{\perp}tr(t15[t15,t15])" >:: (fun () \rightarrow f_{-}of_{-}rep_{-}id (nc *** nc *** nc) t15_{-}trivial)]
```

Ward Identities

Testing the color part of basic Ward identities is essentially the same as testing the Lie algebra equations above, but with generators sandwiched between propagators, as in Feynman diagrams, where the relative signs come from the kinematic part of the diagrams after applying the equations of motion.. First the diagram with the three gluon vertex $i f_{abc} D_{cd}^{gluon} D^{ik} T_d^{kl} D^{lj}$

```
let ward\_ft \ rep\_t \ rep\_d \ a \ b \ i \ j =
  imag *** f a b (-11) *** gluon (-11) (-12)
  *** rep_{-}d \ i \ (-1) \ *** \ rep_{-}t \ (-12) \ (-1) \ (-2) \ *** \ rep_{-}d \ (-2) \ j
```

then one diagram with two gauge couplings $D^{ik}T_c^{kl}D^{lm}T_c^{mn}D^{nj}$

```
let ward\_tt1 \ rep\_t \ rep\_d \ a \ b \ i \ j =
   rep_{-}d \ i \ (-1) \ *** \ rep_{-}t \ a \ (-1) \ (-2) \ *** \ rep_{-}d \ (-2) \ (-3)
   *** rep_t \ b \ (-3) \ (-4) \ *** \ rep_d \ (-4) \ j
```

finally the difference of exchanged orders: $D^{ik}T_a^{kl}D^{lm}T_b^{mn}D^{nj} - D^{ik}T_b^{kl}D^{lm}T_a^{mn}D^{nj}$

```
let ward\_tt \ rep\_t \ rep\_d \ a \ b \ i \ j =
   ward\_tt1 \ rep\_t \ rep\_d \ a \ b \ i \ j \ --- \ ward\_tt1 \ rep\_t \ rep\_d \ b \ a \ i \ j
```



The optional $\tilde{f}udge$ factor was used for debugging normalizations.

```
let ward\_id?(fudge = one) rep\_t rep\_d =
  let lhs = ward\_ft rep\_t rep\_d 1 2 3 4
  and rhs = ward\_tt \ rep\_t \ rep\_d \ 1 \ 2 \ 3 \ 4 in
   eq lhs (fudge *** rhs)
let suite\_ward =
   "Ward<sub>□</sub>identities" >:::
     ["fund." >:: (fun () \rightarrow ward_id t delta3);
        "adj." >:: (fun () \rightarrow ward\_id\ t8\ delta8);
        "S2" >:: (fun () \rightarrow ward\_id t6 delta6);
        "S3" >:: (fun () \rightarrow ward_id t10 delta10);
        "A2" >:: (fun () \rightarrow ward\_id\ t3bar\ delta3bar);
        "A3" >:: (fun () \rightarrow ward_id (t_A 3) (delta_A 3));
        "SAS" >:: (fun () \rightarrow ward\_id t\_SAS delta\_SAS);
        "ASA" >:: (fun () \rightarrow ward\_id t\_ASA delta\_ASA);
```

```
"S2'," >:: (fun () \rightarrow ward\_id \ \tilde{f}udge : two \ t6\_trivial \ delta6);
                    "S3'" >:: (fun () \rightarrow ward\_id \ \tilde{f}udge : (int 3) \ t10\_trivial \ delta10)
           let \ suite\_ward\_long =
               "Ward<sub>□</sub>identities" >:::
                 ["S4" >:: (fun () \rightarrow ward_id t15 delta15);
                    "S4'," >:: (fun () \rightarrow ward\_id \ \tilde{f}udge : (int 4) \ t15\_trivial \ delta15)
                                                            Jacobi Identities
T_aT_bT_c
           let prod3 \ rep_t \ a \ b \ c \ i \ j =
              rep_{-}t \ a \ i \ (-1) \ *** \ rep_{-}t \ b \ (-1) \ (-2) \ *** \ rep_{-}t \ c \ (-2) \ j
[T_a, [T_b, T_c]]
           let jacobi1 \ rep_t \ a \ b \ c \ i \ j =
              (prod 3 \ rep\_t \ a \ b \ c \ i \ j \ --- \ prod 3 \ rep\_t \ a \ c \ b \ i \ j)
                -(prod3 \ rep\_t \ b \ c \ a \ i \ j \ --- \ prod3 \ rep\_t \ c \ b \ a \ i \ j)
sum of cyclic permutations of [T_a, [T_b, T_c]]
           let jacobi rep_t =
              sum [jacobi1 \ rep_t \ 1 \ 2 \ 3 \ 4 \ 5;
                     jacobi1 \ rep_t \ 2 \ 3 \ 1 \ 4 \ 5;
                     jacobi1 rep_t 3 1 2 4 5]
           let jacobi_id rep_t =
               assert\_zero\_vertex\ (jacobi\ rep\_t)
           let suite\_jacobi =
               "Jacobi⊔identities" >:::
                 ["fund." >:: (fun () \rightarrow jacobi_idt);
                    "adj." >:: (fun () \rightarrow jacobi_id f);
                    "S2" >:: (fun () \rightarrow jacobi_id\ t6);
                    "S3" >:: (fun () \rightarrow jacobi_i t10);
                    "A2" >:: (fun () \rightarrow jacobi_i(t_A 2));
                    "A3" >:: (fun () \rightarrow jacobi_id (t_A 3));
                    "SAS" >:: (fun () \rightarrow jacobi_id t_SAS);
                    "ASA" >:: (fun () \rightarrow jacobi\_id t\_ASA);
                    "S2'," >:: (fun () \rightarrow jacobi\_id\ t6\_trivial);
                    "S3'" >:: (fun () \rightarrow jacobi\_id \ t10\_trivial)
           let suite\_jacobi\_long =
               "Jacobi⊔identities" >:::
                 [ "S4" >:: (fun () \rightarrow jacobi_id t15);
                    "S4'," >:: (fun () \rightarrow jacobi\_id \ t15\_trivial)
```

Casimir Operators

We can read of the eigenvalues of the Casimir operators for the adjoint, totally symmetric and totally antisymmetric representations of SU(N) from table II of hep-ph/0611341

$$C_2(\text{adj}) = 2N \tag{7.28a}$$

$$C_2(S_n) = \frac{n(N-1)(N+n)}{N}$$
 (7.28b)

$$C_2(A_n) = \frac{n(N-n)(N+1)}{N}$$
 (7.28c)

adjusted for our normalization. Also from arxiv:1912.13302

$$C_3(S_1) = (N^2 - 1)(N^2 - 4)/N^2 = \frac{N_C^4 - 5N_C^2 + 4}{N_C^2}$$
(7.29)

Building blocks n/N_C and $N_C + n$

```
let n\_over\_nc \ n = const \ (LP.ints \ [\ (n, -1)\ ])
          let nc_-plus \ n = const \ (LP.ints \ [ \ (1, \ 1); \ (n, 0) \ ])
C_2(S_n) = n/N_C(N_C - 1)(N_C + n)
          let c2\_S n = n\_over\_nc n *** nc\_plus (-1) *** nc\_plus n
C_2(A_n) = n/N_C(N_C - n)(N_C + 1)
          let c2\_A n = n\_over\_nc n *** nc\_plus (-n) *** nc\_plus 1
          let \ casimir\_tt \ i \ j \ = \ c2\_S \ 1 \ *** \ delta3 \ i \ j
          let casimir\_t6t6 i j = c2\_S 2 *** delta6 i j
          let casimir_t10t10 \ i \ j = c2\_S \ 3 \ *** \ delta10 \ i \ j
          let casimir\_t15t15 i j = c2\_S 4 *** delta15 i j
          let casimir\_t3bart3bar\ i\ j\ =\ c2\_A\ 2\ ***\ delta3bar\ i\ j
          let casimir_tA3tA3 i j = c2\_A 3 *** delta\_A 3 i j
C_2(\text{adj}) = 2N_C
          \mathsf{let}\ \mathit{ca}\ =\ \mathit{LP.ints}\ [(2,\ 1)]
           \mbox{let } casimir\_ff \  \, a \  \, b \  \, = \  \, [(ca, \  1 \  <=> \  2); \  \, (LP.int \  (-2), \  \, [1 => 1; \  2 => 2])] 
C_3(S_1) = N_C^2 - 5 + 4/N_C^2
          let c3f = LP.ints [(1, 2); (-5, 0); (4, -2)]
          let \ casimir\_ttt \ i \ j \ = \ const \ c3f \ *** \ delta3 \ i \ j
          let suite\_casimir =
             "Casimir_operators" >:::
                [ "t*t" >::
                     (fun () \rightarrow
                       eq
                          (casimir_{-}tt \ 1 \ 2)
                          (t (-1) 1 (-2) *** t (-1) (-2) 2));
                  "t*t*t" >::
                     (\mathsf{fun}\ ()\ \to
                       eq
                          (casimir_ttt 1 2)
                          (d(-1)(-2)(-3)***
                              t(-1) \ 1(-4) *** t(-2) (-4) (-5) *** t(-3) (-5) 2);
                  "f*f" >::
                     (fun () \rightarrow
                       eq
                          (casimir_{-}ff \ 1 \ 2)
                          (minus *** f (-1) 1 (-2) *** f (-1) (-2) 2));
                  "t6*t6" >::
                     (fun () \rightarrow
                       eq
                          (casimir_t6t6 1 2)
                          (t6 (-1) 1 (-2) *** t6 (-1) (-2) 2));
                  "t3bar*t3bar" >::
                     (fun () \rightarrow
                       eq
                          (casimir\_t3bart3bar\ 1\ 2)
                          (t3bar (-1) 1 (-2) *** t3bar (-1) (-2) 2));
                  "tA3*tA3" >::
                     (fun () \rightarrow
                       eq
                          (casimir_tA3tA3 \ 1 \ 2)
                          (t_{-}A\ 3\ (-1)\ 1\ (-2)\ ***\ t_{-}A\ 3\ (-1)\ (-2)\ 2));
                  "t_SAS*t_SAS" >::
                     (fun () \rightarrow
```

```
(const\ (LP.ints\ [(3,1);\ (-9,-1)])\ ***\ delta\_SAS\ 1\ 2)
              (t\_SAS\ (-1)\ 1\ (-2)\ ***\ t\_SAS\ (-1)\ (-2)\ 2));
       "t_ASA*t_ASA" >::
         (fun () \rightarrow
            eq
               (const\ (LP.ints\ [(3,1);\ (-9,-1)])\ ***\ delta\_ASA\ 1\ 2)
              (t_{-}ASA\ (-1)\ 1\ (-2)\ ***\ t_{-}ASA\ (-1)\ (-2)\ 2));
       "t10*t10" >::
         (fun () \rightarrow
            eq
               (casimir_t10t10 \ 1 \ 2)
              (t10 \ (-1) \ 1 \ (-2) \ *** \ t10 \ (-1) \ (-2) \ 2))
let suite\_casimir\_long =
  "Casimir_operators" >:::
     [ "t15*t15" >::
         (fun () \rightarrow
            eq
               (casimir_t15t15 \ 1 \ 2)
              (t15 (-1) 1 (-2) *** t15 (-1) (-2) 2))
                                           Color Sums
let suite\_colorsums =
  "(squared) \( color \( sums" > :::
     [ "gluon_normalization" >::
         (fun () \rightarrow
            eq
              (delta8 1 2)
               (delta8\ 1\ (-1)\ ***\ gluon\ (-1)\ (-2)\ ***\ delta8\ (-2)\ 2));
       "f*f" >::
         (fun () \rightarrow
            let sum_{-}ff =
              multiply [f(-11)(-12)(-13);
                            f(-21)(-22)(-23);
                            gluon (-11) (-21);
                            gluon (-12) (-22);
                            gluon (-13) (-23) ]
            and expected = ints [(2, 3); (-2, 1)] in
            eq expected sum_ff);
       "d*d" >::
         (fun () \rightarrow
            let sum_{-}dd =
              multiply [d(-11)(-12)(-13);
                            d(-21)(-22)(-23);
                            gluon (-11) (-21);
                            gluon (-12) (-22);
                            gluon (-13) (-23)
            and expected = ints [(2, 3); (-10, 1); (8, -1)] in
            eq \ expected \ sum_{-}dd);
       "f*d" >::
         (fun () \rightarrow
            let sum_{-}fd =
              multiply [f(-11)(-12)(-13);
                            d(-21)(-22)(-23);
                            gluon (-11) (-21);
```

```
gluon (-12) (-22);
                              gluon (-13) (-23) ] in
               assert_zero_vertex sum_fd);
          "Hgg" >::
            (fun () \rightarrow
              let sum\_hgg =
                 multiply [ delta8\_loop (-11) (-12);
                               delta8\_loop\ (-21)\ (-22);
                              gluon (-11) (-21);
                              gluon (-12) (-22) ]
              and expected = ints [(1, 2); (-1, 0)] in
               eq expected sum_hgg) ]
  let suite =
     "Color.SU3" >:::
       [suite\_sum;
        suite\_diff;
        suite\_times;
        suite\_normalization;
        suite\_symmetrization;
        suite\_ghosts;
        suite\_propagators;
        suite\_trace;
        suite\_ff;
        suite\_tf;
        suite_{-}tt;
        suite\_lie;
        suite\_ward;
        suite\_jacobi;
        suite\_casimir;
        suite\_colorsums]
  let suite\_long =
     "Color.SU3<sub>□</sub>long" >:::
       [suite\_ward\_long;
        suite\_jacobi\_long;
        suite\_casimir\_long]
end
```

 $7.2.5 \ \mathrm{U}(N_C)$



end

This must not be used, because it has not yet been updated to the correctly symmetrized version!

```
\begin{array}{lll} \operatorname{module} \ U3 \ : \ SU3 \ = \\ \operatorname{struct} \\ & \operatorname{module} \ A \ = \ Arrow \\ \operatorname{open} \ Arrow.Infix \\ & \operatorname{module} \ B \ = \ Birdtracks \\ \operatorname{type} \ t \ = \ B.t \\ \operatorname{let} \ canonicalize \ = \ B.canonicalize \\ \operatorname{let} \ to \ -string \ = \ B.to \ -string \\ \operatorname{let} \ pp \ = \ B.pp \\ \operatorname{let} \ trivial \ = \ B.trivial \\ \operatorname{let} \ is \ -null \ = \ B.is \ -null \\ \operatorname{let} \ null \ = \ B.null \\ \operatorname{let} \ const \ = \ B.const \\ \operatorname{let} \ one \ = \ B.one \\ \end{array}
```

```
let two = B.two
let int = B.int
let half = B.half
let third = B.third
let fraction = B.fraction
let nc = B.nc
let over\_nc = B.over\_nc
\mathsf{let}\ minus\ =\ B.minus
let imag = B.imag
let ints = B.ints
let sum = B.sum
let diff = B.diff
\mathsf{let} \ scale \ = \ B.scale
let times = B.times
let multiply = B.multiply
\mathsf{let}\ \mathit{relocate}\ =\ B.\mathit{relocate}
let fuse = B.fuse
\mathsf{let}\ f\_of\_rep\ =\ B.f\_of\_rep
let d_-of_-rep = B.d_-of_-rep
module Infix = B.Infix
let delta3 i j =
  [(LP.int 1, j ==> i)]
let delta8 a b =
  [(LP.int 1, a \ll b)]
let delta8\_loop = delta8
let gluon \ a \ b =
  delta8 a b
let delta6 \ n \ m =
  [(LP.fraction\ 2,\ [(m,\ 0)\ >=>>\ (n,\ 0);\ (m,\ 1)\ >=>>\ (n,\ 1)]);
     (LP.fraction\ 2,\ [(m,\ 0)\ >=>>\ (n,\ 1);\ (m,\ 1)\ >=>>\ (n,\ 0)])\ ]
let triples =
  [(0, 1, 2); (1, 2, 0); (2, 0, 1);
   (2, 1, 0); (0, 2, 1); (1, 0, 2)]
let delta10 \ n \ m =
  List.map
     (\text{fun }(i, j, k) \rightarrow
       (LP.fraction 6, [(m, 0) >=>> (n, i);
                            (m, 1) >=>> (n, j);
                            (m, 2) > = > (n, k))
     triples
\mathsf{let}\ t\ a\ i\ j\ =
  [(LP.int 1, [j => a; a => i])]
let f a b c =
  [(LP.imag(1), A.cycle[a; b; c]);
     (LP.imag (-1), A.cycle [a; c; b])
let t8 \ a \ b \ c =
  Birdtracks.Infix.(minus *** imag *** f a b c)
  [ (LP.int 1, A.cycle [a; b; c]);
     (LP.int 1, A.cycle [a; c; b])
let incomplete tensor =
  failwith \ ("Color.Vertex: \_" \ \hat tensor \ \hat " \_not \_supported \_yet!")
let \ experimental \ tensor =
  Printf.eprintf
     "Color.Vertex: \( \)\%s\\\ support\( \)\ still\( \)\ experimental\( \)\ and\( \)\ untested!\\ n\"
```

```
tensor
    let epsilon tips = incomplete "epsilon-tensor"
    let \ epsilon\_bar \ tails \ = \ incomplete \ \texttt{"epsilon-tensor"}
    let t6 \ a \ m \ n =
       [(LP.int (1), [(n, 0) >=> a; a =>> (m, 0); (n, 1) >=>> (m, 1)]);
         (LP.int (1), [(n, 1) >=> a; a =>> (m, 0); (n, 0) >=>> (m, 1)])]
    let t10 \ a \ m \ n =
       [(LP.int (1), [(n, 0) >=> a; a =>> (m, 0);
                           (n, 1) > = > > (m, 1);
                           (n, 2) >=>> (m, 2)]);
         (LP.int (-1), [(n, 0) >=>> (m, 0);
                           (n, 1) >=>> (m, 1);
                           (n, 2) > = > > (m, 2)
    \mathsf{let}\ k6\ m\ i\ j\ =
       experimental "k6-tensor";
       [(LP.int 1, [(m, 0) >=> i; (m, 1) >=> j]);
         (LP.int 1, [(m, 1) >=> i; (m, 0) >=> j])]
    let k6bar m i j =
       experimental "k6-tensor";
       [(LP.int 1, [i =>> (m, 0); j =>> (m, 1)]);
         (LP.int 1, [i =>> (m, 1); j =>> (m, 0)])]
    let delta\_of\_tableau\ t\ i\ j\ =
       incomplete "delta_of_tableau"
    let t_of_tableau tableau a k l =
       incomplete "t_of_tableau"
                                                    Unit Tests
    module \ Test : Test =
       struct
         open OUnit
         open Birdtracks
         open Infix
         let suite\_lie =
            "\mathtt{Lie}_{\sqcup}\mathtt{algebra}_{\sqcup}\mathtt{relations}">:::
              [ "if_=utr(t[t,t])" >::
                   (fun () \rightarrow eq (f 1 2 3) (f_-of_-rep t 1 2 3))]
N_C = N_C^2/N_C
         let cf = LP.ints [(1, 1)]
         \mathsf{let}\ \mathit{casimir} \_\mathit{tt}\ i\ j\ =
            [(cf, i ==> j)]
         let \ \mathit{suite\_casimir} \ =
            "Casimir_operators" >:::
              [ "t*t" >::
                   (fun () \rightarrow
                     eq (casimir_tt 2 1) (t (-1) (-2) 2 *** t (-1) 1 (-2)))
         let suite =
            "Color.U3" >:::
              [suite\_lie;
               suite\_casimir
         let suite\_long =
            "Color.U3⊔long" >:::
```

_ Implementation of *Color*

end end

 $\mathsf{module}\ \mathit{Vertex}\ =\ \mathit{SU3}$

—8— Fusions

8.1 Interface of Fusion

```
\begin{array}{ll} \mbox{module type} \ T \ = \\ \mbox{sig} \\ \mbox{val} \ options \ : \ Options.t \end{array}
```

JRR's implementation of Majoranas needs a special case.

val vintage : bool

Wavefunctions are an abstract data type, containing a momentum p and additional quantum numbers, collected in flavor.

```
\begin{array}{lll} \text{type } wf \\ \text{val } conjugate \ : \ wf \ \rightarrow \ wf \end{array}
```

Obviously, flavor is not restricted to the physical notion of flavor, but can carry spin, color, etc.

```
type flavor val flavor: wf \rightarrow flavor type flavor\_sans\_color val flavor\_sans\_color: wf \rightarrow flavor\_sans\_color
```

Momenta are represented by an abstract datatype (defined in *Momentum*) that is optimized for performance. They can be accessed either abstractly or as lists of indices of the external momenta. These indices are assigned sequentially by *amplitude* below.

```
\begin{array}{lll} \text{type } p \\ \text{val } momentum \ : \ wf \ \rightarrow \ p \\ \text{val } momentum\_list \ : \ wf \ \rightarrow \ int \ list \end{array}
```

At tree level, the wave functions are uniquely specified by *flavor* and momentum. If loops are included, we need to distinguish among orders. Also, if we build a result from an incomplete sum of diagrams, we need to add a distinguishing mark. At the moment, we assume that a *string* that can be attached to the symbol suffices.

```
val wf_tag : wf \rightarrow string option
```

Coupling constants

```
type constant
```

and right hand sides of assignments. The latter are formed from a sign from Fermi statistics, a coupling (constand and Lorentz structure) and wave functions.

```
type coupling type rhs type \alpha children val sign: rhs \rightarrow int val coupling: rhs \rightarrow constant Coupling.t val coupling\_tag: rhs \rightarrow string option type exclusions val no\_exclusions: exclusions
```

In renormalized perturbation theory, couplings come in different orders of the loop expansion. Be prepared: val $order: rhs \rightarrow int$



This is here only for the benefit of Target and shall become val $children: rhs \rightarrow wf \ children$ later . . .

 $\mathsf{val}\ children\ :\ rhs\ \to\ wf\ list$

Fusions come in two types: fusions of wave functions to off-shell wave functions:

$$\phi(p+q) = \phi(p)\phi(q)$$

 $\mathsf{type}\; \mathit{fusion}$

 $\begin{array}{lll} \mathsf{val} \ \mathit{lhs} \ : \ \mathit{fusion} \ \to \ \mathit{wf} \\ \mathsf{val} \ \mathit{rhs} \ : \ \mathit{fusion} \ \to \ \mathit{rhs} \ \mathit{list} \end{array}$

and products at the keystones:

$$\phi(-p-q)\cdot\phi(p)\phi(q)$$

 $type \ braket$

 $\begin{array}{lll} \mathsf{val} \ bra \ : \ braket \ \to \ wf \\ \mathsf{val} \ ket \ : \ braket \ \to \ rhs \ list \end{array}$

amplitude goldstones incoming outgoing calculates the amplitude for scattering of incoming to outgoing. If goldstones is true, also non-propagating off-shell Goldstone amplitudes are included to allow the checking of Slavnov-Taylor identities.

type amplitude

type $amplitude_sans_color$

type selectors

 $\mathsf{val}\ amplitudes\ :\ bool \to\ exclusions\ \to\ selectors\ \to$

 $flavor_sans_color\ list
ightarrow\ flavor_sans_color\ list
ightarrow\ amplitude\ list$

 $val\ amplitude_sans_color: bool \rightarrow exclusions \rightarrow selectors \rightarrow$

 $flavor_sans_color\ list
ightarrow\ flavor_sans_color\ list
ightarrow\ amplitude_sans_color$

val dependencies: amplitude \rightarrow wf \rightarrow (wf, coupling) Tree2.t

We should be precise regarding the semantics of the following functions, since modules implementating Target must not make any mistakes interpreting the return values. Instead of calculating the amplitude

$$\langle f_3, p_3, f_4, p_4, \dots | T | f_1, p_1, f_2, p_2 \rangle$$
 (8.1a)

directly, O'Mega calculates the—equivalent, but more symmetrical—crossed amplitude

$$\langle \bar{f}_1, -p_1, \bar{f}_2, -p_2, f_3, p_3, f_4, p_4, \dots | T | 0 \rangle$$
 (8.1b)

Internally, all flavors are represented by their charge conjugates

$$A(f_1, -p_1, f_2, -p_2, \bar{f}_3, p_3, \bar{f}_4, p_4, \dots)$$
 (8.1c)

The correspondence of vertex and term in the lagrangian



suggests to denote the *outgoing* particle by the flavor of the *anti*particle and the *outgoing anti*particle by the flavor of the particle, since this choice allows to represent the vertex by a triple

$$\bar{\psi} \mathcal{A} \psi : (\mathbf{e}^+, A, \mathbf{e}^-) \tag{8.3}$$

which is more intuitive than the alternative (e^-, A, e^+) . Also, when thinking in terms of building wavefunctions from the outside in, the outgoing *antiparticle* is represented by a *particle* propagator and vice versa¹. *incoming* and *outgoing* are the physical flavors as in (8.1a)

¹Even if this choice will appear slightly counter-intuitive on the Target side, one must keep in mind that much more people are expected to prepare Models.

```
val\ incoming\ :\ amplitude\ 	o\ flavor\ list
     val\ outgoing\ :\ amplitude\ 	o\ flavor\ list
externals are flavors and momenta as in (8.1c)
     val\ externals\ :\ amplitude\ 	o\ wf\ list
     val\ variables\ :\ amplitude\ 	o\ wf\ list
    \mathsf{val}\ \mathit{fusions}\ :\ \mathit{amplitude}\ \to\ \mathit{fusion}\ \mathit{list}
    val\ brakets\ :\ amplitude\ 	o\ braket\ list
    val\ on\_shell\ :\ amplitude\ 	o\ (wf\ 	o\ bool)
    val\ is\_gauss: amplitude \rightarrow (wf \rightarrow bool)
    val\ constraints\ :\ amplitude\ 	o\ string\ option
    \mathsf{val}\ symmetry\ :\ amplitude\ \to\ int
     val\ allowed: amplitude 
ightarrow bool
                                                     Diagnostics
    val\ check\_charges:\ unit 
ightarrow\ flavor\_sans\_color\ list\ list
     val\ count\_fusions\ :\ amplitude\ 	o\ int
     val\ count\_propagators\ :\ amplitude\ 	o\ int
    val\ count\_diagrams\ :\ amplitude\ 	o\ int
     val forest: wf \rightarrow amplitude \rightarrow ((wf \times coupling option, wf) Tree.t) list
    val\ poles\ :\ amplitude\ 	o\ wf\ list\ list
    val s\_channel : amplitude \rightarrow wf list
    val\ tower\_to\_dot:\ out\_channel\ 	o\ amplitude\ 	o\ unit
     \mathsf{val}\ amplitude\_to\_dot\ :\ out\_channel\ \to\ amplitude\ \to\ unit
                                                      WHIZARD
    val\ phase\_space\_channels\ :\ out\_channel\ 	o\ amplitude\_sans\_color\ 	o\ unit
     val\ phase\_space\_channels\_flipped: out\_channel 
ightarrow amplitude\_sans\_color 
ightarrow unit
There is more than one way to make fusions.
module type Maker =
     functor (P:Momentum.T) \rightarrow functor (M:Model.T) \rightarrow
       T with type p = P.t
       and type flavor = Colorize.It(M).flavor
       and type flavor\_sans\_color = M.flavor
       and type constant = M.constant
       and type selectors = Cascade.Make(M)(P).selectors
Straightforward Dirac fermions vs. slightly more complicated Majorana fermions:
exception Majorana
module Binary: Maker
module Binary\_Majorana : Maker
module Mixed23 : Maker
module Mixed 23 \_Majorana : Maker
module Nary: functor (B : Tuple.Bound) \rightarrow Maker
module Nary\_Majorana: functor (B : Tuple.Bound) \rightarrow Maker
We can also proceed á la [2]. Empirically, this will use slightly (O(10\%)) fewer fusions than the symmetric
factorization. Our implementation uses significantly (O(50\%)) fewer fusions than reported by [2]. Our pruning
of the DAG might be responsible for this.
module Helac\_Binary : Maker
module\ Helac\_Binary\_Majorana\ :\ Maker
module\ Helac\_Mixed23: Maker
```

```
\begin{array}{lll} \operatorname{module} \ Helac\_Mixed23\_Majorana \ : \ Maker \\ \operatorname{module} \ Helac \ : \ \operatorname{functor} \ (B \ : \ Tuple.Bound) \ \rightarrow \ Maker \\ \operatorname{module} \ Helac\_Majorana \ : \ \operatorname{functor} \ (B \ : \ Tuple.Bound) \ \rightarrow \ Maker \\ \end{array}
```

8.1.1 Multiple Amplitudes

```
 \begin{array}{lll} \text{module type } \textit{Multi} &= \\ \text{sig} & \text{exception } \textit{Mismatch} \\ \text{val } \textit{options} : \textit{Options.t} \\ \\ \text{type } \textit{flavor} \\ \text{type } \textit{process} &= \textit{flavor list} \times \textit{flavor list} \\ \text{type } \textit{amplitude} \\ \text{type } \textit{fusion} \\ \text{type } \textit{exclusions} \\ \text{val } \textit{no\_exclusions} : \textit{exclusions} \\ \text{type } \textit{selectors} \\ \text{type } \textit{amplitudes} \\  \end{array}
```

Construct all possible color flow amplitudes for a given process.

```
val amplitudes: bool \rightarrow int\ option \rightarrow exclusions \rightarrow selectors \rightarrow process\ list \rightarrow amplitudes val empty: amplitudes
```

The list of all combinations of incoming and outgoing particles with a nonvanishing scattering amplitude.

```
val\ flavors: amplitudes \rightarrow process\ list
```

The list of all combinations of incoming and outgoing particles that don't lead to any color flow with non vanishing scattering amplitude.

```
val vanishing\_flavors: amplitudes \rightarrow process\ list
```

The list of all color flows with a nonvanishing scattering amplitude.

```
val\ color\_flows: amplitudes \rightarrow Color.Flow.t\ list
```

The list of all valid helicity combinations.

```
val\ helicities: amplitudes \rightarrow (int\ list \times int\ list)\ list
```

The list of all amplitudes.

```
val\ processes\ :\ amplitudes\ 	o\ amplitude\ list
```

 $(process_table\ a).(f).(c)$ returns the amplitude for the fth allowed flavor combination and the cth allowed color flow as an $amplitude\ option$.

```
val process\_table: amplitudes \rightarrow amplitude option array array
```

The list of all non redundant fusions together with the amplitudes they came from.

```
val\ fusions\ :\ amplitudes\ 	o\ (fusion\ 	imes\ amplitude)\ list
```

If there's more than external flavor state, the wavefunctions are *not* uniquely specified by *flavor* and *Momentum.t.* This function can be used to determine how many variables must be allocated.

```
val\ multiplicity: amplitudes \rightarrow wf \rightarrow int
```

This function can be used to disambiguate wavefunctions with the same combination of flavor and Momentum.t.

```
\mathsf{val}\ dictionary\ :\ amplitudes\ 	o\ amplitude\ 	o\ wf\ 	o\ int
```

 $(color_factors\ a).(c1).(c2)$ power of N_C for the given product of color flows.

```
val\ color\_factors\ :\ amplitudes\ 	o\ Color.Flow.factor\ array\ array
```

A description of optional diagram selectors.

```
val\ constraints\ :\ amplitudes\ 	o\ string\ option
```

end

```
\begin{array}{lll} \operatorname{module\ type\ } \textit{Multi\_Maker} &= \operatorname{functor\ } (Fusion\_Maker : \textit{Maker}) \to \\ \operatorname{functor\ } (P : \textit{Momentum.}T) \to \\ \operatorname{functor\ } (M : \textit{Model.}T) \to \\ \operatorname{\textit{Multi\ }} \text{with\ } \operatorname{type\ } \textit{flavor} &= M.flavor \\ \operatorname{and\ } \operatorname{type\ } \textit{amplitude\ } &= Fusion\_Maker(P)(M).amplitude \\ \operatorname{and\ } \operatorname{type\ } \textit{fusion\ } &= Fusion\_Maker(P)(M).fusion \\ \operatorname{and\ } \operatorname{type\ } \textit{wf\ } &= Fusion\_Maker(P)(M).wf \\ \operatorname{and\ } \operatorname{type\ } \textit{selectors\ } &= Fusion\_Maker(P)(M).selectors \\ \\ \operatorname{module\ } \textit{Multi\ } &: \textit{Multi\_Maker\ } \\ \end{array}
```

8.1.2 Tags

It appears that there are useful applications for tagging couplings and wave functions, e.g. skeleton expansion and diagram selections. We can abstract this in a *Tags* signature:

```
module type Tags =
  sig
    type wf
    type coupling
    type \alpha children
    val null_wf : wf
    val null_coupling : coupling
    \mathsf{val}\ fuse\ :\ coupling\ 	o\ wf\ children\ 	o\ wf
    val wf_to_string : wf \rightarrow string option
    val\ coupling\_to\_string\ :\ coupling\ 	o\ string\ option
  end
module type Tagger =
    functor (PT: Tuple.Poly) \rightarrow Tags with type \alpha \ children = \alpha \ PT.t
module type Tagged\_Maker =
    functor (Tagger : Tagger) \rightarrow
       functor (P:Momentum.T) \rightarrow functor (M:Model.T) \rightarrow
          T with type p = P.t
         and type flavor = Colorize.It(M).flavor
         and type flavor\_sans\_color = M.flavor
         and type constant = M.constant
module Tagged\_Binary : Tagged\_Maker
```

8.2 Implementation of Fusion

Avoid referring to *Pervasives.compare*, because *Pervasives* will become *Stdlib.Pervasives* in O'Caml 4.07 and *Stdlib* in O'Caml 4.08.

```
let pcompare = compare

module type T = sig

val options : Options.t

val vintage : bool

type wf

val conjugate : wf \rightarrow wf

type flavor

type flavor : wf \rightarrow flavor

val flavor : wf \rightarrow flavor : wf
```

```
type constant
     type coupling
     type rhs
     type \alpha children
     val \ sign : rhs \rightarrow int
     val\ coupling\ :\ rhs\ 	o\ constant\ Coupling.t
     val\ coupling\_tag\ :\ rhs\ 	o\ string\ option
     type exclusions
     val no_exclusions : exclusions
     val\ children\ :\ rhs\ 	o\ wf\ list
     type fusion
     val\ lhs: fusion \rightarrow wf
     val \ rhs : fusion \rightarrow rhs \ list
     type braket
     val bra : braket \rightarrow wf
     \mathsf{val}\ ket : \mathit{braket} \ 	o \ \mathit{rhs}\ \mathit{list}
     type amplitude
     type amplitude\_sans\_color
     type selectors
     \mathsf{val}\ amplitudes\ :\ bool 	o\ exclusions\ 	o\ selectors\ 	o
        flavor\_sans\_color\ list 
ightarrow\ flavor\_sans\_color\ list 
ightarrow\ amplitude\ list
     \verb|val|| amplitude\_sans\_color : bool \rightarrow exclusions \rightarrow selectors \rightarrow \\
        flavor\_sans\_color\ list 
ightarrow\ flavor\_sans\_color\ list 
ightarrow\ amplitude\_sans\_color
     val dependencies : amplitude \rightarrow wf \rightarrow (wf, coupling) Tree2.t
     val\ incoming\ :\ amplitude\ 	o\ flavor\ list
     val\ outgoing\ :\ amplitude\ 	o\ flavor\ list
     val\ externals\ :\ amplitude\ 	o\ wf\ list
     val\ variables\ :\ amplitude\ 	o\ wf\ list
     \mathsf{val}\ fusions\ :\ amplitude\ 	o\ fusion\ list
     \mathsf{val}\ \mathit{brakets}\ :\ \mathit{amplitude}\ \to\ \mathit{braket}\ \mathit{list}
     val\ on\_shell\ :\ amplitude\ 	o\ (wf\ 	o\ bool)
     val\ is\_gauss: amplitude \rightarrow (wf \rightarrow bool)
     val\ constraints\ :\ amplitude\ 	o\ string\ option
     val\ symmetry\ :\ amplitude\ 	o\ int
     val \ allowed : amplitude \rightarrow bool
     val\ check\_charges\ :\ unit 
ightarrow\ flavor\_sans\_color\ list\ list
     \mathsf{val}\ count\_fusions\ :\ amplitude\ \to\ int
     val\ count\_propagators\ :\ amplitude\ 	o\ int
     val\ count\_diagrams\ :\ amplitude\ 	o\ int
     val forest: wf \rightarrow amplitude \rightarrow ((wf \times coupling \ option, \ wf) \ Tree.t) \ list
     val\ poles\ :\ amplitude\ 	o\ wf\ list\ list
     val s\_channel : amplitude \rightarrow wf list
     val\ tower\_to\_dot: out\_channel \rightarrow amplitude \rightarrow unit
     val\ amplitude\_to\_dot:\ out\_channel\ 	o\ amplitude\ 	o\ unit
     val\ phase\_space\_channels\ :\ out\_channel\ 	o\ amplitude\_sans\_color\ 	o\ unit
     \verb|val|| phase\_space\_channels\_flipped : out\_channel \rightarrow amplitude\_sans\_color \rightarrow unit|
  end
module type Maker =
     functor (P:Momentum.T) \rightarrow functor (M:Model.T) \rightarrow
        T with type p = P.t
        and type flavor = Colorize.It(M).flavor
        and type flavor\_sans\_color = M.flavor
        and type constant = M.constant
        and type selectors = Cascade.Make(M)(P).selectors
```

8.2.1 Fermi Statistics

```
module type Stat = sig
```

This will be *Model*. T. flavor.

type flavor

A record of the fermion lines in the 1POW.

type stat

Vertices with an odd number of fermion fields.

exception Impossible

External lines.

```
val\ stat\ :\ flavor\ 	o\ int\ 	o\ stat
```

 $stat_fuse$ (Some flines) slist f combines the fermion lines in the elements of slist according to the connections listed in flines. On the other hand, $stat_fuse$ None slist f corresponds to the legacy mode with at most two fermions. The resulting flavor f of the 1POW can be ignored for models with only Dirac fermions, except for debugging, since the direction of the arrows is unambiguous. However, in the case of Majorana fermions and/or fermion number violating interactions, the flavor f must be used.

```
val stat_fuse :
```

 $Coupling.fermion_lines\ option
ightarrow \ stat\ list
ightarrow \ flavor
ightarrow \ stat$

Analogous to stat_fuse, but for the finalizing keystone instead of the 1POW.

```
val stat\_keystone :
```

 $Coupling.fermion_lines\ option
ightarrow \ stat\ list
ightarrow \ flavor
ightarrow \ stat$

Compute the sign corresponding to the fermion lines in a 1POW or keystone.

```
val stat\_sign : stat \rightarrow int
```

Debugging and consistency checks ...

```
\begin{array}{lll} \mathsf{val} \ stat\_to\_string \ : \ stat \ \to \ string \\ \mathsf{val} \ equal \ : \ stat \ \to \ stat \ \to \ bool \\ \mathsf{val} \ saturated \ : \ stat \ \to \ bool \end{array}
```

end

```
module type Stat\_Maker = functor (M : Model.T) \rightarrow Stat with type flavor = M.flavor
```

8.2.2 Dirac Fermions

```
\begin{array}{lll} \mbox{let } dirac\_log \ silent \ logging \ = \ logging \\ \mbox{let } dirac\_log \ silent \ logging \ = \ silent \end{array}
```

exception Majorana

```
\label{eq:module Stat_Dirac} \mbox{module } Stat\_Dirac \; (M \; : \; Model. T) \; : \; (Stat \; \mbox{with type } flavor \; = \; M.flavor) \; = \; \\ \mbox{struct}
```

type flavor = M.flavor

$$\gamma_{\mu}\psi(1) G^{\mu\nu} \bar{\psi}(2)\gamma_{\nu}\psi(3) - \gamma_{\mu}\psi(3) G^{\mu\nu} \bar{\psi}(2)\gamma_{\nu}\psi(1)$$
(8.4)

```
type stat =
```

```
 | Fermion of int \times (int \ option \times int \ option) \ list \\ | AntiFermion of int \times (int \ option \times int \ option) \ list \\ | Boson of (int \ option \times int \ option) \ list \\ | let \ lines\_to\_string \ lines =
```

```
ThoList.to\_string
```

```
(function
```

```
| Some i, Some j \rightarrow Printf.sprintf "%d>%d" i j | Some i, None \rightarrow Printf.sprintf "%d>*" i | None, Some j \rightarrow Printf.sprintf "*>%d" j | None, None \rightarrow "*>*")
```

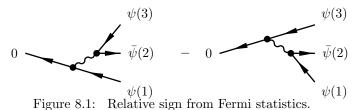
```
lines
```

```
let stat\_to\_string = function
    Boson\ lines \rightarrow Printf.sprintf "Boson_\" (lines\_to\_string\ lines)
    Fermion (p, lines) \rightarrow
      Printf.sprintf "Fermion (%d, %s)" p (lines\_to\_string\ lines)
  \mid AntiFermion (p, lines) \rightarrow
      Printf.sprintf "AntiFermion_(%d,_%s)" p (lines_to_string lines)
let equal \ s1 \ s2 =
  match s1, s2 with
    Boson l1, Boson l2 \rightarrow
      List.sort\ compare\ l1\ =\ List.sort\ compare\ l2
    Fermion (p1, l1), Fermion (p2, l2)
  | AntiFermion (p1, l1), AntiFermion (p2, l2) \rightarrow
      p1 = p2 \land List.sort compare l1 = List.sort compare l2
  \mid \_ \rightarrow false
let saturated = function
    Boson \_ \rightarrow true
  \perp \rightarrow false
let stat f p =
  match M.fermion f with
    0 \rightarrow Boson[]
    1 \rightarrow Fermion(p, [])
     -1 \rightarrow AntiFermion(p, [])
    2 \rightarrow raise Majorana
    \_ \rightarrow invalid\_arg "Fusion.Stat_Dirac:\_invalid\_fermion\_number"
exception Impossible
let stat\_fuse\_pair\_legacy f s1 s2 =
  match s1, s2 with
    Boson l1, Boson l2 \rightarrow Boson (l1 @ l2)
    Boson l1, Fermion (p, l2) \rightarrow Fermion (p, l1 @ l2)
    Boson l1, AntiFermion (p, l2) \rightarrow AntiFermion (p, l1 @ l2)
    Fermion (p, l1), Boson l2 \rightarrow Fermion (p, l1 @ l2)
    AntiFermion (p, l1), Boson l2 \rightarrow AntiFermion (p, l1 @ l2)
    AntiFermion\ (pbar,\ l1),\ Fermion\ (p,\ l2)\ \rightarrow
       Boson ((Some pbar, Some p) :: l1 @ l2)
    Fermion (p, l1), AntiFermion (pbar, l2) \rightarrow
       Boson\ ((Some\ pbar,\ Some\ p)\ ::\ l1\ @\ l2)
    Fermion \_, Fermion \_ | AntiFermion \_, AntiFermion \_ \rightarrow
       raise\ Impossible
let stat\_fuse\_legacy s1 s23\_\_n f =
  List.fold\_right\ (stat\_fuse\_pair\_legacy\ f)\ s23\_\_n\ s1
let stat\_fuse\_legacy\_logging \ s1 \ s23\_\_n \ f =
  let s = stat\_fuse\_legacy s1 s23\_\_n f in
  Printf.eprintf
     (M.flavor\_to\_string f)
     (ThoList.to\_string\ stat\_to\_string\ (s1\ ::\ s23\_\_n))
     (stat\_to\_string\ s);
let stat\_fuse\_legacy =
  dirac_log stat_fuse_legacy stat_fuse_legacy_logging
module IMap = Map.Make (struct type t = int let compare = compare end)
type partial =
  \{ stat : stat (* the stat accumulated so far *); 
     fermions: int IMap.t (* a map from the indices in the vertex to open fermion lines *);
     antifermions: int IMap.t (* a map from the indices in the vertex to open antifermion lines *);
```

```
n : int (* the number of incoming propagators *) }
let partial_to_string p =
   Printf.sprintf
     "{_{\sqcup}fermions=%s,_{\sqcup}antifermions=%s,_{\sqcup}state=%s,_{\sqcup}#=%d_{\sqcup}}"
     (ThoList.to\_string)
         (fun (i, f) \rightarrow Printf.sprintf "%d0%d" f i)
         (IMap.bindings p.fermions))
     (ThoList.to_string
         (\mathsf{fun}\ (i,\ f)\ \to\ \mathit{Printf.sprintf}\ \texttt{"%d0\%d"}\ f\ i)
         (IMap.bindings p.antifermions))
     (stat\_to\_string\ p.stat)
     p.n
let \ add\_lines \ l \ = \ function
     Boson\ l'\ \rightarrow\ Boson\ (List.rev\_append\ l\ l')
     Fermion (n, l') \rightarrow Fermion (n, List.rev\_append l l')
    AntiFermion (n, l') \rightarrow AntiFermion (n, List.rev\_append l l')
\mathsf{let}\ partial\_of\_slist\ slist\ =
   List.fold\_left
     (fun acc s \rightarrow
        let n = succ \ acc.n \ in
        \mathsf{match}\ s\ \mathsf{with}
        \mid Boson \ l \rightarrow
            { acc with
               stat = add\_lines \ l \ acc.stat;
               n \}
        | Fermion (p, l) \rightarrow
            { acc with
              fermions = IMap.add n p acc. fermions;
               stat = add\_lines \ l \ acc.stat;
               n
        \mid AntiFermion (p, l) \rightarrow
            \{ acc \text{ with }
               antifermions = IMap.add n p acc.antifermions;
               stat = add\_lines \ l \ acc.stat;
               n \})
     \{ stat = Boson []; 
        fermions = IMap.empty;
        antifermions = IMap.empty;
        n = 0
     slist
let find_opt p map =
  try Some \ (IMap.find \ p \ map) with Not\_found \ 	o \ None
let match\_fermion\_line\ p\ (i,\ j)\ =
  if i \leq p.n \wedge j \leq p.n then
     match \ find\_opt \ i \ p.fermions, \ find\_opt \ j \ p.antifermions \ with
     | (Some \_ as f), (Some \_ as fbar) \rightarrow
         \{p \text{ with }
            stat = add\_lines [fbar, f] p.stat;
            fermions = IMap.remove i p.fermions;
            antifermions = IMap.remove j p.antifermions }
         invalid\_arg "match_fermion_line:_\mismatched_\doson"
  else if i \leq p.n then
     match find_opt i p.fermions, p.stat with
     \mid Some f, Boson l \rightarrow
         \{ p \text{ with }
            stat = Fermion(f, l);
           fermions = IMap.remove i p.fermions 
     |  \rightarrow
```

```
invalid\_arg "match_fermion_line:_mismatched_fermion"
  else if i < p.n then
     match find_opt j p.antifermions, p.stat with
     | Some fbar, Boson l \rightarrow
        \{p \text{ with }
           stat = AntiFermion (fbar, l);
           antifermions = IMap.remove j p.antifermions }
        invalid_arg "match_fermion_line: _mismatched_antifermion"
  else
     failwith "match_fermion_line:_impossible"
let match\_fermion\_line\_logging \ p \ (i, j) =
  Printf.eprintf
     \verb|"match_fermion_line|| \% s_{\sqcup} (\% d, _{\sqcup} \% d) \, "
     (partial\_to\_string \ p) \ i \ j;
  let p' = match\_fermion\_line p(i, j) in
  p'
let match\_fermion\_line =
  dirac_log match_fermion_line match_fermion_line_logging
let match\_fermion\_lines\ flines\ s1\ s23\_\_n\ =
  let p = partial\_of\_slist (s1 :: s23\_\_n) in
  List.fold_left match_fermion_line p flines
let stat\_fuse\_new flines s1 s23\_\_n f =
  (match\_fermion\_lines\ flines\ s1\ s23\_\_n).stat
let stat_fuse_new_checking flines s1 \ s23_n f =
  let stat = stat\_fuse\_new flines s1 s23\_\_n f in
  if List.length\ flines\ <\ 2 then
     begin
       let legacy = stat\_fuse\_legacy s1 s23\_\_n f in
       if \neg (equal stat legacy) then
         failwith
            (Printf.sprintf)
                "Fusion.Stat_Dirac.stat_fuse_new:_{\square}%s_{\parallel}<>_{\square}%s!"
                (stat\_to\_string\ stat)
                (stat\_to\_string\ legacy))
     end:
  stat
let stat\_fuse\_new\_logging\ flines\ s1\ s23\_\_n\ f =
  Printf.eprintf
     "stat_fuse_new:_connecting_fermion_lines_\%s_in_\%s_<-\\%s\n"
     (UFO_Lorentz.fermion_lines_to_string flines)
     (M.flavor\_to\_string \ f)
     (ThoList.to\_string\ stat\_to\_string\ (s1\ ::\ s23\_\_n));
  stat_fuse_new_checking flines s1 s23__n f
let stat_fuse_new =
  dirac\_log\ stat\_fuse\_new\ stat\_fuse\_new\_logging
let stat\_fuse\ flines\_opt\ slist\ f\ =
  match slist with
   \mid \; [\;] \; 
ightarrow \; invalid\_arg \; "Fusion.Stat_Dirac.stat_fuse:\sqcupempty"
  \mid s1 :: s23\_\_n \rightarrow
      begin match flines\_opt with
        Some flines \rightarrow stat_fuse_new flines s1 s23_n f
       None \rightarrow stat\_fuse\_legacy s1 s23\_\_n f
let stat\_fuse\_logging\ flines\_opt\ slist\ f\ =
  Printf.eprintf
```

```
(M.flavor\_to\_string \ f)
     (ThoList.to_string stat_to_string slist);
   stat_fuse flines_opt slist f
let stat_fuse =
   dirac_log stat_fuse stat_fuse_logging
let stat\_keystone\_legacy \ s1 \ s23\_\_n \ f =
  let s2 = List.hd s23\_\_n
  and s34 - n = List.tl \ s23 - n in
  stat\_fuse\_legacy \ s1 \ [stat\_fuse\_legacy \ s2 \ s34\_\_n \ (M.conjugate \ f)] \ f
let stat\_keystone\_legacy\_logging \ s1 \ s23\_\_n \ f =
  let s = stat\_keystone\_legacy \ s1 \ s23\_\_n \ f \ in
  Printf.eprintf
     (stat\_to\_string \ s1)
     (M.flavor\_to\_string \ f)
     (ThoList.to\_string\ stat\_to\_string\ s23\_\_n)
     (stat\_to\_string\ s);
let stat_keystone_legacy =
   dirac_log stat_keystone_legacy stat_keystone_legacy_logging
let stat\_keystone flines\_opt slist f =
  match slist with
    [] 
ightarrow invalid\_arg "Fusion.Stat_Dirac.stat_keystone:\_empty"
    [s] \rightarrow invalid\_arg "Fusion.Stat_Dirac.stat_keystone:\sqcupsingleton"
  | s1 :: (s2 :: s34\_n \text{ as } s23\_n) \rightarrow
      begin match flines_opt with
        None \rightarrow stat\_keystone\_legacy s1 s23\_\_n f
      \mid Some flines \rightarrow
          (* The fermion line indices in flines must match the lines on one side of the keystone. *)
          let stat =
             stat\_fuse\_legacy \ s1 \ [stat\_fuse\_new \ flines \ s2 \ s34\_\_n \ f] \ f \ in
          if saturated stat then
             stat
          else
            fail with
               (Printf.sprintf
                   "Fusion.Stat_Dirac.stat_keystone: incomplete %s!"
                   (stat\_to\_string\ stat))
      end
let stat_keystone_logging flines_opt slist f =
  let s = stat\_keystone flines\_opt slist f in
   Printf.eprintf
     "stat_keystone:\square\square\square\square\square\square\square%s\square(%s)\square%s\square->\square%s\n"
     (stat\_to\_string (List.hd slist))
     (M.flavor\_to\_string \ f)
     (ThoList.to\_string\ stat\_to\_string\ (List.tl\ slist))
     (stat\_to\_string\ s);
let stat_k eystone =
   dirac\_log\ stat\_keystone\ stat\_keystone\_logging
                                   \epsilon(\{(0,1),(2,3)\}) = -\epsilon(\{(0,3),(2,1)\})
                                                                                                                (8.5)
let \ permutation \ lines =
  let fout, fin = List.split lines in
```



```
let eps\_in, \_ = Combinatorics.sort\_signed fin and eps\_out, \_ = Combinatorics.sort\_signed fout in (eps\_in \times eps\_out)
```

\$

This comparing of permutations of fermion lines is a bit tedious and takes a macroscopic fraction of time. However, it's less than 20 %, so we don't focus on improving on it yet.

```
\begin{array}{lll} \text{let } stat\_sign &=& \text{function} \\ & \mid Boson \ lines & \rightarrow \ permutation \ lines \\ & \mid Fermion \ (p, \ lines) \ \rightarrow \ permutation \ ((None, \ Some \ p) \ :: \ lines) \\ & \mid AntiFermion \ (pbar, \ lines) \ \rightarrow \ permutation \ ((Some \ pbar, \ None) \ :: \ lines) \end{array} end
```

8.2.3 Tags

```
module type Tags =
  sig
    type wf
    type coupling
    type \alpha children
    val null_wf : wf
    val null_coupling : coupling
    \mathsf{val}\ fuse\ :\ coupling\ 	o\ wf\ children\ 	o\ wf
    val wf_to_string : wf \rightarrow string option
    val\ coupling\_to\_string\ :\ coupling\ 	o\ string\ option
module type Tagger =
     functor (PT : Tuple.Poly) \rightarrow Tags with type \alpha \ children = \alpha \ PT.t
module type Tagged\_Maker =
    functor (Tagger : Tagger) \rightarrow
       \mathsf{functor}\;(P\;:\;Momentum.\,T)\;\to\;\mathsf{functor}\;(M\;:\;Model.\,T)\;\to
          T with type p = P.t
         and type flavor = Colorize.It(M).flavor
         and type flavor\_sans\_color = M.flavor
         and type constant = M.constant
No tags is one option for good tags ...
module No\_Tags (PT : Tuple.Poly) =
  struct
     type wf = unit
     \mathsf{type} \ coupling \ = \ unit
     type \alpha children = \alpha PT.t
    let null_w f = ()
    let null\_coupling = ()
    let fuse () _{-} = ()
    let wf\_to\_string() = None
    let coupling\_to\_string() = None
  end
```



Here's a simple additive tag that can grow into something useful for loop calculations.

```
module Loop\_Tags (PT : Tuple.Poly) =
  struct
     type wf = int
     type coupling = int
     \mathsf{type} \,\, \alpha \,\, \mathit{children} \,\, = \,\, \alpha \,\, \mathit{PT.t}
     let null_wf = 0
     let null\_coupling = 0
     let fuse \ c \ wfs = PT.fold\_left \ (+) \ c \ wfs
     let wf\_to\_string \ n = Some \ (string\_of\_int \ n)
     let coupling\_to\_string \ n = Some (string\_of\_int \ n)
  end
module \ Order\_Tags \ (PT : Tuple.Poly) =
  struct
     type wf = int
     type coupling = int
     type \alpha children = \alpha PT.t
     let null_wf = 0
     let null\_coupling = 0
     let fuse \ c \ wfs = PT.fold\_left \ (+) \ c \ wfs
     let wf\_to\_string \ n = Some \ (string\_of\_int \ n)
     let coupling\_to\_string \ n = Some (string\_of\_int \ n)
  end
                                    8.2.4
                                              Tagged, the Fusion. Make Functor
module Tagged (Tagger: Tagger) (PT: Tuple.Poly)
     (Stat : Stat\_Maker) (T : Topology.T with type <math>\alpha \ children = \alpha \ PT.t)
     (P : Momentum.T) (M : Model.T) =
  struct
     let \ vintage = false
     type cache_mode = Cache_Use | Cache_Ignore | Cache_Overwrite
     let cache_option = ref Cache_Ignore
     type qcd\_order =
       | QCD_order of int
     type ew\_order =
        \mid EW\_order \text{ of } int
     let \ qcd\_order \ = \ ref \ (QCD\_order \ 99)
     let ew\_order = ref (EW\_order 99)
     let \ options = Options.create
             "qcd", Arg.Int (fun n \rightarrow qcd\_order := QCD\_order n),
             "_{\sqcup} set_{\sqcup} QCD_{\sqcup} order_{\sqcup} n_{\sqcup} [>=_{\sqcup} 0,_{\sqcup} default_{\sqcup} =_{\sqcup} 99]_{\sqcup} (ignored)";
             "ew", Arg.Int (fun n \rightarrow ew\_order := EW\_order n),
             "_{\sqcup}set_{\sqcup}QCD_{\sqcup}order_{\sqcup}n_{\sqcup}[>=0,_{\sqcup}default_{\sqcup}=_{\sqcup}99]_{\sqcup}(ignored)"]
     exception Negative_QCD_order
     exception Negative_EW_order
     exception Vanishing\_couplings
     exception Negative_QCD_EW_orders
     let int\_orders =
       match !qcd_order, !ew_order with
          |QCD\_order \ n, \ EW\_order \ n' \ when \ n < 0 \land n' \geq 0 \rightarrow
                raise\ Negative\_QCD\_order
            QCD\_order \ n, \ EW\_order \ n' \ when \ n \ge 0 \ \land \ n' < 0 \ \rightarrow
                raise Negative_EW_order
          | QCD\_order \ n, \ EW\_order \ n' \ when \ n < 0 \land n' < 0 \rightarrow
                raise Negative_QCD_EW_orders
          | QCD\_order \ n, \ EW\_order \ n' \rightarrow (n, \ n')
```

```
open Coupling

module S = Stat(M)

type stat = S.stat

let stat = S.stat

let stat\_sign = S.stat\_sign
```



This will do something for 4-, 6-, ... fermion vertices, but not necessarily the right thing ...



This is copied from *Colorize* and should be factored!



In the long run, it will probably be beneficial to apply the permutations in $Modeltools.add_vertexn!$

```
module PosMap =
  Partial.Make (struct type t = int let compare = compare end)
let partial\_map\_undoing\_permutation \ l \ l' =
  let module P = Permutation.Default in
  let p = P.of\_list (List.map pred l') in
  PosMap.of\_lists\ l\ (P.list\ p\ l)
let partial_map_undoing_fuse fuse =
  partial\_map\_undoing\_permutation
    (ThoList.range\ 1\ (List.length\ fuse))
let undo_permutation_of_fuse fuse =
  PosMap.apply\_with\_fallback
     (fun _ → invalid_arg "permutation_of_fuse")
     (partial_map_undoing_fuse fuse)
let fermion\_lines = function
    Coupling. V3 = | Coupling. V4 = \rightarrow None
    Coupling. Vn (Coupling. UFO (\_, \_, \_, fl, \_), fuse, \_) \rightarrow
      Some (UFO_Lorentz.map_fermion_lines (undo_permutation_of_fuse fuse) ft)
type constant = M.constant
```

Wave Functions

\$

The code below is not yet functional. Too often, we assign to Tags.null_wf instead of calling Tags.fuse.

We will need two types of amplitudes: with color and without color. Since we can build them using the same types with only *flavor* replaced, it pays to use a functor to set up the scaffolding.

```
module \ Tags = Tagger(PT)
```

In the future, we might want to have *Coupling* among the functor arguments. However, for the moment, *Coupling* is assumed to be comprehensive.

```
type sign = int
     type t =
           \{ sign : sign;
              coupling: constant\ Coupling.t;
              coupling_tag : Tags.coupling }
     let sign c = c.sign
     let coupling c = c.coupling
     let coupling\_tag\_raw c = c.coupling\_tag
     let coupling_tag rhs = Tags.coupling_to_string (coupling_tag_raw rhs)
                                        Amplitudes: Monochrome and Colored
module type Amplitude =
  sig
     module \ Tags : Tags
     type flavor
     type p
     type wf =
           { flavor : flavor;
              momentum : p;
              wf\_tag : Tags.wf }
     \mathsf{val}\; \mathit{flavor}\;:\; \mathit{wf}\; \rightarrow\; \mathit{flavor}
     val conjugate : wf \rightarrow wf
     \mathsf{val}\ momentum\ :\ wf\ \to\ p
     \mathsf{val}\ momentum\_list\ :\ wf\ \to\ int\ list
     val wf_tag : wf \rightarrow string option
     val \ wf \_tag \_raw : wf \rightarrow Tags.wf
     \mathsf{val}\ \mathit{order\_wf}\ :\ \mathit{wf}\ \to\ \mathit{wf}\ \to\ \mathit{int}
     \mathsf{val}\ external\_wfs\ :\ int \to\ (\mathit{flavor}\ \times\ int)\ \mathit{list} \to\ \mathit{wf}\ \mathit{list}
     type \alpha children
     \mathsf{type}\ coupling\ =\ Tagged\_Coupling.t
     type rhs = coupling \times wf \ children
     val \ sign : rhs \rightarrow int
     \mathsf{val}\ coupling\ :\ rhs\ 	o\ constant\ Coupling.t
     val\ coupling\_tag\ :\ rhs\ 	o\ string\ option
     type exclusions
     val no_exclusions : exclusions
     val\ children\ :\ rhs\ 	o\ wf\ list
     \mathsf{type}\;\mathit{fusion}\;=\;\mathit{wf}\;\times\;\mathit{rhs}\;\mathit{list}
     val lhs : fusion \rightarrow wf
     val \ rhs : fusion \rightarrow rhs \ list
     type braket = wf \times rhs \ list
     val bra : braket \rightarrow wf
     \mathsf{val}\ ket\ :\ \mathit{braket}\ 	o\ \mathit{rhs}\ \mathit{list}
     \mathsf{module}\ D :
           DAG.T with type node = wf and type edge = coupling and type children = wf children
     val wavefunctions : braket list \rightarrow wf list
     type amplitude =
           { fusions : fusion list;
              brakets: braket list;
              on\_shell : (wf \rightarrow bool);
              is\_gauss : (wf \rightarrow bool);
              constraints: string option;
              incoming: flavor list;
```

```
outgoing: flavor list;
            externals: wf list;
            symmetry : int;
            dependencies : (wf \rightarrow (wf, coupling) Tree2.t);
            fusion\_tower : D.t;
            fusion\_dag : D.t 
    \mathsf{val}\ incoming\ :\ amplitude\ 	o\ flavor\ list
    \mathsf{val}\ outgoing\ :\ amplitude\ \to\ flavor\ list
    val\ externals\ :\ amplitude\ 	o\ wf\ list
    val\ variables\ :\ amplitude\ 	o\ wf\ list
    val\ fusions\ :\ amplitude\ 	o\ fusion\ list
    \mathsf{val}\ brakets\ :\ amplitude\ \to\ braket\ list
     val \ on\_shell : amplitude \rightarrow (wf \rightarrow bool)
    \mathsf{val}\ is\_gauss\ :\ amplitude\ \to\ (wf\ \to\ bool)
    val\ constraints\ :\ amplitude\ 	o\ string\ option
    val\ symmetry\ :\ amplitude\ 	o\ int
    val dependencies: amplitude \rightarrow wf \rightarrow (wf, coupling) Tree2.t
    val\ fusion\_dag\ :\ amplitude\ 	o\ D.t
module \ Amplitude \ (PT : Tuple.Poly) \ (P : Momentum.T) \ (M : Model.T) :
     Amplitude
    with type p = P.t
    and type flavor = M.flavor
    and type \alpha children = \alpha PT.t
    and module Tags = Tags =
  struct
     type flavor = M.flavor
     \mathsf{type}\ p\ =\ P.t
    module Tags = Tags
     type wf =
          \{ flavor : flavor; 
            momentum : p;
            wf\_tag : Tags.wf }
    let flavor wf = wf.flavor
    let conjugate \ wf = \{ wf \ with \ flavor = M.conjugate \ wf.flavor \}
    let momentum \ wf = wf.momentum
    let momentum\_list \ wf = P.to\_ints \ wf.momentum
    let wf\_tag \ wf = Tags.wf\_to\_string \ wf.wf\_tag
    let wf_tag_raw wf = wf.wf_tag
    let external_wfs rank particles =
       List.map
          (\mathsf{fun}\ (f,\ p)\ \to
            \{ flavor = f; \}
               momentum = P.singleton \ rank \ p;
               wf\_tag = Tags.null\_wf \})
         particles
```

Order wavefunctions so that the external come first, then the pairs, etc. Also put possible Goldstone bosons before their gauge bosons.

```
\begin{array}{lll} \text{let } lorentz\_ordering \ f &= \\ & \text{match } M.lorentz \ f \ \text{with} \\ & \mid Coupling.Scalar \ \rightarrow \ 0 \\ & \mid Coupling.Spinor \ \rightarrow \ 1 \\ & \mid Coupling.ConjSpinor \ \rightarrow \ 2 \\ & \mid Coupling.Majorana \ \rightarrow \ 3 \\ & \mid Coupling.Vector \ \rightarrow \ 4 \\ & \mid Coupling.Massive\_Vector \ \rightarrow \ 5 \end{array}
```

```
Coupling. Tensor_2 \rightarrow 6
    Coupling. Tensor_1 \rightarrow 7
    Coupling. Vectorspinor \rightarrow 8
    Coupling.BRS\ Coupling.Scalar\ 	o\ 9
    Coupling.BRS\ Coupling.Spinor\ 	o\ 10
    Coupling.BRS\ Coupling.ConjSpinor \rightarrow 11
    Coupling.BRS\ Coupling.Majorana\ 	o\ 12
    Coupling.BRS\ Coupling.Vector\ 	o\ 13
    Coupling.BRS\ Coupling.Massive\_Vector \rightarrow 14
    Coupling.BRS\ Coupling.Tensor\_2\ 	o\ 15
    Coupling.BRS\ Coupling.Tensor\_1 \rightarrow 16
    Coupling.BRS\ Coupling.Vectorspinor \rightarrow 17
    Coupling.BRS \rightarrow invalid\_arg "Fusion.lorentz_ordering:_not_needed"
    Coupling.Maj\_Ghost \rightarrow 18
let order\_flavor\ f1\ f2\ =
  let c = compare (lorentz\_ordering f1) (lorentz\_ordering f2) in
  if c \neq 0 then
    c
  else
     compare f1 f2
```

Note that Momentum().compare guarantees that wavefunctions will be ordered according to $increasing\ Momentum().rank$ of their momenta.

```
\begin{array}{lll} \text{let } \mathit{order\_wf} \ \mathit{wf1} \ \mathit{wf2} &= \\ & \text{let } c &= P.\mathit{compare} \ \mathit{wf1}.\mathit{momentum} \ \mathit{wf2}.\mathit{momentum} \ \mathit{in} \\ & \text{if } c \neq 0 \ \mathsf{then} \\ & c \\ & \text{else} \\ & \text{let } c &= \mathit{order\_flavor} \ \mathit{wf1}.\mathit{flavor} \ \mathit{wf2}.\mathit{flavor} \ \mathit{in} \\ & \text{if } c \neq 0 \ \mathsf{then} \\ & c \\ & \text{else} \\ & \mathit{compare} \ \mathit{wf1}.\mathit{wf\_tag} \ \mathit{wf2}.\mathit{wf\_tag} \\ \end{array}
```

This must be a pair matching the edge \times node children pairs of DAG. Forest!

```
type coupling = Tagged\_Coupling.t
\mathsf{type} \,\, \alpha \,\, \mathit{children} \,\, = \,\, \alpha \,\, \mathit{PT}.t
\mathsf{type}\ \mathit{rhs}\ =\ \mathit{coupling}\ \times\ \mathit{wf}\ \mathit{children}
let sign(c, \bot) = Tagged\_Coupling.sign(c)
let coupling(c, \_) = Tagged\_Coupling.couplingc
let coupling\_tag(c, \_) = Tagged\_Coupling\_coupling\_tag(c, \_)
type exclusions =
  { x\_flavors : flavor list;}
     x\_couplings : coupling list 
let no\_exclusions = \{ x\_flavors = []; x\_couplings = [] \}
let children(\_, wfs) = PT.to\_list wfs
type fusion = wf \times rhs \ list
let lhs (l, \_) = l
let rhs(-, r) = r
type braket = wf \times rhs \ list
let bra(b, \_) = b
let ket (\_, k) = k
\mathsf{module}\ D\ =\ DAG.Make
     (DAG.Forest(PT))
         (struct type t = wf let compare = order_wf end)
         (struct type t = coupling let compare = compare end))
module WFSet =
  Set.Make (struct type t = wf let compare = order\_wf end)
```

```
let wavefunctions brakets =
            WFSet.elements (List.fold_left (fun set (wf1, wf23) \rightarrow
               WFSet.add\ wf1\ (List.fold\_left\ (fun\ set'\ (\_,\ wfs)\ 	o
                 PT.fold_right WFSet.add wfs set') set wf23)) WFSet.empty brakets)
         type amplitude =
              \{\ fusions\ :\ fusion\ list;
                 brakets: braket list;
                 on\_shell : (wf \rightarrow bool);
                 is\_gauss : (wf \rightarrow bool);
                 constraints : string option;
                 incoming : flavor \ list;
                 outgoing: flavor list;
                 externals: wf list;
                 symmetry : int;
                 dependencies: (wf \rightarrow (wf, coupling) Tree2.t);
                 fusion\_tower : D.t;
                 fusion\_dag : D.t 
         let incoming a = a.incoming
         let outgoing \ a = a.outgoing
         let externals a = a.externals
         \mathsf{let}\ \mathit{fusions}\ a\ =\ a.\mathit{fusions}
         \mathsf{let}\ brakets\ a\ =\ a.brakets
         let symmetry a = a.symmetry
         let on\_shell a = a.on\_shell
         let is\_gauss \ a = a.is\_gauss
         let constraints a = a.constraints
         let variables \ a = List.map \ lhs \ a.fusions
         let dependencies a = a.dependencies
         let fusion\_dag \ a = a.fusion\_dag
       end
     module A = Amplitude(PT)(P)(M)
Operator insertions can be fused only if they are external.
    let is\_source \ wf =
       match M.propagator wf.A.flavor with
         Only\_Insertion \rightarrow P.rank \ wf.A.momentum = 1
       \mid _{-} 
ightarrow true
is\_goldstone\_of g v is true if and only if g is the Goldstone boson corresponding to the gauge particle v.
    let is\_goldstone\_of g v =
       match M.goldstone v with
       None \rightarrow false
       Some (g', \_) \rightarrow g = g'
   In the end, PT.to\_list should become redudant!
    let fuse\_rhs rhs = M.fuse (PT.to\_list rhs)
                                                       Vertices
Compute the set of all vertices in the model from the allowed fusions and the set of all flavors:
   One could think of using M.vertices instead of M.fuse2, M.fuse3 and M.fuse...
    module VSet = Map.Make(struct type t = A.flavor let compare = compare end)
    let add\_vertices\ f\ rhs\ m\ =
       VSet.add\ f\ (\mathsf{try}\ rhs\ ::\ VSet.find\ f\ m\ \mathsf{with}\ Not\_found\ 	o\ [rhs])\ m
```

 $let collect_vertices rhs =$

```
List.fold\_right (fun (f1, c) \rightarrow add\_vertices (M.conjugate f1) (c, rhs))
  (fuse_rhs rhs)
```

The set of all vertices with common left fields factored.

I used to think that constant initializers are a good idea to allow compile time optimizations. The down side turned out to be that the constant initializers will be evaluated every time the functor is applied. Relying on the fact that the functor will be called only once is not a good idea!

```
type vertices = (A.flavor \times (constant\ Coupling.t \times A.flavor\ PT.t)\ list)\ list
```



This is very inefficient for max_degree > 6. Find a better approach that avoids precomputing the huge lookup table!



I should revive the above Idea to use M.vertices instead directly, instead of rebuilding it from M.fuse2, M.fuse3 and M.fuse!

```
let vertices_nocache max_degree flavors : vertices =
  VSet.fold (fun f rhs v \rightarrow (f, rhs) :: v)
     (PT.power\_fold)
         \tilde{truncate}: (pred\ max\_degree)
         collect_vertices flavors VSet.empty) []
```

Performance hack:

```
type vertex\_table =
          ((A.flavor \times A.flavor \times A.flavor) \times constant Coupling.vertex3 \times constant) list
        \times ((A.flavor \times A.flavor \times A.flavor \times A.flavor)
               \times constant Coupling.vertex4 \times constant) list
        \times (A.flavor list \times constant Coupling.vertexn \times constant) list
let \ vertices = \ vertices\_nocache
let vertices' max\_degree flavors =
   Printf.eprintf ">>> \sqcup vertices \sqcup %d \sqcup \ldots " max\_degree;
  flush stderr;
  let v = vertices max\_degree flavors in
   Printf.eprintf "

done.\n";
  flush stderr;
```

Note that we must perform any filtering of the vertices after caching, because the restrictions must not influence the cache (unless we tag the cache with model and restrictions).

```
let filter_vertices select_vtx vertices =
   List.fold\_left
      (fun acc\ (f,\ cfs)\ \rightarrow
         \mathsf{let}\ f'\ =\ M.\mathit{conjugate}\ f\ \mathsf{in}
         let cfs =
             List.filter
                (\text{fun } (c, fs) \rightarrow select\_vtx \ c \ f' \ (PT.to\_list \ fs))
         match cfs with
         | [] \rightarrow acc
         cfs \rightarrow (f, cfs) :: acc)
      [] vertices
```

Partitions

Vertices that are not crossing invariant need special treatment so that they're only generated for the correct combinations of momenta.

NB: the crossing checks here are a bit redundant, because CM.fuse below will bring the killed vertices back to life and will have to filter once more. Nevertheless, we keep them here, for the unlikely case that anybody ever wants to use uncolored amplitudes directly.

NB: the analogous problem does not occur for $select_wf$, because this applies to momenta instead of vertices.



This approach worked before the colorize, but has become *futile*, because *CM.fuse* will bring the killed vertices back to life. We need to implement the same checks there again!!!



Using PT. Mismatched_arity is not really good style . . .

Tho's approach doesn't work since he does not catch charge conjugated processes or crossed processes. Another very strange thing is that O'Mega seems always to run in the q2 q3 timelike case, but not in the other two. (Property of how the DAG is built?). For the ZZZZ vertex I add the same vertex again, but interchange 1 and 3 in the crossing vertex

```
let kmatrix_cuts c momenta =
  match c with
     V4 (Vector4_K_Matrix_tho (disc, _), fusion, _)
     V4 \ (Vector4\_K\_Matrix\_jr \ (disc, \_), \ fusion, \_) \rightarrow
        let s12, s23, s13 =
          begin match PT.to_list momenta with
          [q1; q2; q3] \rightarrow (P.Scattering.timelike (P.add q1 q2),
                                   P.Scattering.timelike\ (P.add\ q2\ q3),
                                   P.Scattering.timelike (P.add q1 q3))
          \downarrow \rightarrow raise PT.Mismatched_arity
          end in
        begin match disc, s12, s23, s13, fusion with
         0, true, false, false, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
          0, false, true, false, (F134 | F143 | F234 | F243 | F312 | F321 |
                                                                                  F412 \mid F421)
        \mid 0, false, false, true, (F314 \mid F413 \mid F324 \mid F423 \mid F132 \mid F231 \mid F142 \mid F241) <math>\rightarrow
            true
        | 1, true, false, false, (F341 | F431 | F342 | F432)
        | 1, false, true, false, (F134 | F143 | F234 | F243)
        \mid 1, false, false, true, (F314 \mid F413 \mid F324 \mid F423) \rightarrow
             true
          2, true, false, false, (F123 | F213 | F124 | F214)
          2, false, true, false, (F312 | F321 | F412 | F421)
        \mid 2, false, false, true, (F132 \mid F231 \mid F142 \mid F241) \rightarrow
             true
          3, true, false, false, (F143 | F413 | F142 | F412 | F321 | F231 | F324 | F234)
          3, false, true, false, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423)
         3, false, false, true, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow
             true
             \rightarrow false
         _
        end
   V4 (Vector_4 K_Matrix_cf_t0 (disc, \_), fusion, \_) \rightarrow
       let s12, s23, s13 =
          begin match PT.to_list momenta with
          [q1; q2; q3] \rightarrow (P.Scattering.timelike (P.add q1 q2),
                                   P.Scattering.timelike (P.add q2 q3),
                                   P.Scattering.timelike\ (P.add\ q1\ q3))
          \mid \_ \rightarrow raise\ PT.Mismatched\_arity
          end in
        begin match disc, s12, s23, s13, fusion with
         0, true, false, false, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
          0, false, true, false, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
        \mid 0, false, false, true, (F314 \mid F413 \mid F324 \mid F423 \mid F132 \mid F231 \mid F142 \mid F241) \rightarrow
             true
         1, true, false, false, (F341 \mid F431 \mid F342 \mid F432)
          1, false, true, false, (F134 \mid F143 \mid F234 \mid F243)
        \mid 1, false, false, true, (F314 \mid F413 \mid F324 \mid F423) \rightarrow
         2, true, false, false, (F123 | F213 | F124 | F214)
          2, false, true, false, (F312 | F321 | F412 | F421)
        \mid 2, false, false, true, (F132 \mid F231 \mid F142 \mid F241) \rightarrow
             true
```

```
3, true, false, false, (F143 | F413 | F142 | F412 | F321 | F231 | F324 | F234)
     3, false, true, false, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423)
     \mid 3, false, false, true, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) <math>\rightarrow
          true
          \rightarrow false
     end
V4 (Vector_4 K_Matrix_cf_t1 (disc, \_), fusion, \_) \rightarrow
     let s12, s23, s13 =
        begin match PT.to_list momenta with
        | [q1; q2; q3] \rightarrow (P.Scattering.timelike (P.add q1 q2),
                                 P.Scattering.timelike (P.add q2 q3),
                                 P.Scattering.timelike (P.add q1 q3))
        \downarrow \rightarrow raise PT.Mismatched_arity
        end in
     begin match disc, s12, s23, s13, fusion with
       0, true, false, false, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
       0, false, true, false, (F134 \mid F143 \mid F234 \mid F243 \mid F312 \mid F321 \mid F412 \mid F421)
     \mid 0, false, false, true, (F314 \mid F413 \mid F324 \mid F423 \mid F132 \mid F231 \mid F142 \mid F241) \rightarrow
          true
     | 1, true, false, false, (F341 | F431 | F342 | F432)
       1, false, true, false, (F134 \mid F143 \mid F234 \mid F243)
     \mid 1, false, false, true, (F314 \mid F413 \mid F324 \mid F423) \rightarrow
          true
       2, true, false, false, (F123 | F213 | F124 | F214)
       2, false, true, false, (F312 | F321 | F412 | F421)
     \mid 2, false, false, true, (F132 \mid F231 \mid F142 \mid F241) \rightarrow
       3, true, false, false, (F143 | F413 | F142 | F412 | F321 | F231 | F324 | F234)
       3, false, true, false, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423)
     \mid 3, false, false, true, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) <math>\rightarrow
          true
     \mid \ \_ \ 	o \ \mathsf{false}
     end
V4 (Vector4\_K\_Matrix\_cf\_t2 (disc, \_), fusion, \_) \rightarrow
     let s12, s23, s13 =
        begin match PT.to_list momenta with
        [q1; q2; q3] \rightarrow (P.Scattering.timelike (P.add q1 q2),
                                 P.Scattering.timelike (P.add q2 q3),
                                 P.Scattering.timelike (P.add q1 q3))
        \mid \_ \rightarrow raise\ PT.Mismatched\_arity
       end in
     begin match disc, s12, s23, s13, fusion with
     0, true, false, false, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
       0, false, true, false, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
     \mid 0, false, false, true, (F314 \mid F413 \mid F324 \mid F423 \mid F132 \mid F231 \mid F142 \mid F241) \rightarrow
          true
     | 1, true, false, false, (F341 | F431 | F342 | F432)
     | 1, false, true, false, (F134 | F143 | F234 | F243)
     \mid 1, false, false, true, (F314 \mid F413 \mid F324 \mid F423) \rightarrow
          true
       2, true, false, false, (F123 | F213 | F124 | F214)
       2, false, true, false, (F312 \mid F321 \mid F412 \mid F421)
     \mid 2, false, false, true, (F132 \mid F231 \mid F142 \mid F241) \rightarrow
          true
       3, true, false, false, (F143 | F413 | F142 | F412 | F321 | F231 | F234 | F234)
     3, false, true, false, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423)
     \mid 3, false, false, true, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) 
ightarrow
          true
     \mid \ \_ \rightarrow \mathsf{false}
     end
V4 (Vector_4\_K\_Matrix\_cf\_t\_rsi (disc, \_), fusion, \_) \rightarrow
```

```
let s12, s23, s13 =
        begin match PT.to_list momenta with
       [q1; q2; q3] \rightarrow (P.Scattering.timelike (P.add q1 q2),
                                 P.Scattering.timelike\ (P.add\ q2\ q3),
                                 P.Scattering.timelike\ (P.add\ q1\ q3))
        \downarrow \rightarrow raise PT.Mismatched_arity
       end in
     begin match disc, s12, s23, s13, fusion with
     | 0, true, false, false, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
     0, false, true, false, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
     \mid 0, false, false, true, (F314 \mid F413 \mid F324 \mid F423 \mid F132 \mid F231 \mid F142 \mid F241) \rightarrow
       1, true, false, false, (F341 \mid F431 \mid F342 \mid F432)
       1, false, true, false, (F134 | F143 | F234 | F243)
     \mid 1, false, false, true, (F314 \mid F413 \mid F324 \mid F423) \rightarrow
          true
     | 2, true, false, false, (F123 | F213 | F124 | F214)
       2, false, true, false, (F312 \mid F321 \mid F412 \mid F421)
     \mid 2, false, false, true, (F132 \mid F231 \mid F142 \mid F241) \rightarrow
          true
       3, true, false, false, (F143 | F413 | F142 | F412 | F321 | F231 | F234 | F234)
       3, false, true, false, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423)
     \mid 3, false, false, true, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow
          true
     \mid \_ \rightarrow \mathsf{false}
     end
V4 (Vector4\_K\_Matrix\_cf\_m0 (disc, \_), fusion, \_) \rightarrow
     let s12, s23, s13 =
        begin match PT.to_list momenta with
       [q1; q2; q3] \rightarrow (P.Scattering.timelike (P.add q1 q2),
                                 P.Scattering.timelike (P.add q2 q3),
                                 P.Scattering.timelike\ (P.add\ q1\ q3))
       \downarrow \rightarrow raise\ PT.Mismatched\_arity
       end in
     begin match disc, s12, s23, s13, fusion with
       0, true, false, false, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
       0, false, true, false, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
     \mid 0, false, false, true, (F314 \mid F413 \mid F324 \mid F423 \mid F132 \mid F231 \mid F142 \mid F241) 
ightarrow
          true
     | 1, true, false, false, (F341 | F431 | F342 | F432)
     | 1, false, true, false, (F134 | F143 | F234 | F243)
     \mid 1, false, false, true, (F314 \mid F413 \mid F324 \mid F423) \rightarrow
          true
       2, true, false, false, (F123 | F213 | F124 | F214)
       2, false, true, false, (F312 | F321 | F412 | F421)
     \mid 2, false, false, true, (F132 \mid F231 \mid F142 \mid F241) \rightarrow
          true
     | \ 3, true, false, false, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234)
       3, false, true, false, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423)
     \mid 3, false, false, true, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow
          true
          \rightarrow false
     | _
     end
V4 (Vector4\_K\_Matrix\_cf\_m1 (disc, \_), fusion, \_) \rightarrow
     let s12, s23, s13 =
       begin match PT.to_list momenta with
       [q1; q2; q3] \rightarrow (P.Scattering.timelike (P.add q1 q2),
                                 P.Scattering.timelike\ (P.add\ q2\ q3),
                                 P.Scattering.timelike\ (P.add\ q1\ q3))
        \mid \_ \rightarrow raise\ PT.Mismatched\_arity
       end in
```

```
begin match disc, s12, s23, s13, fusion with
     | \ 0, true, false, false, (F341 \mid F431 \mid F342 \mid F432 \mid F123 \mid F213 \mid F124 \mid F214)
       0, false, true, false, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
     \mid 0, false, false, true, (F314 \mid F413 \mid F324 \mid F423 \mid F132 \mid F231 \mid F142 \mid F241) \rightarrow
          true
      1, true, false, false, (F341 \mid F431 \mid F342 \mid F432)
       1, false, true, false, (F134 \mid F143 \mid F234 \mid F243)
     \mid 1, false, false, true, (F314 \mid F413 \mid F324 \mid F423) \rightarrow
          true
      2, true, false, false, (F123 | F213 | F124 | F214)
       2, false, true, false, (F312 | F321 | F412 | F421)
     \mid 2, false, false, true, (F132 \mid F231 \mid F142 \mid F241) \rightarrow
          true
       3, true, false, false, (F143 | F413 | F142 | F412 | F321 | F231 | F234 | F234)
       3, false, true, false, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423)
     \mid 3, false, false, true, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) <math>\rightarrow
          true
     | _

ightarrow false
     end
V4 (Vector_4 \_K\_Matrix\_cf\_m7 (disc, \_), fusion, \_) \rightarrow
     let s12, s23, s13 =
       begin match PT.to\_list momenta with
       [q1; q2; q3] \rightarrow (P.Scattering.timelike\ (P.add\ q1\ q2),
                                 P.Scattering.timelike\ (P.add\ q2\ q3),
                                  P.Scattering.timelike\ (P.add\ q1\ q3))
       \mid \_ \rightarrow raise\ PT.Mismatched\_arity
       end in
     begin match disc, s12, s23, s13, fusion with
       0, true, false, false, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
       0, false, true, false, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
     \mid 0, false, false, true, (F314 \mid F413 \mid F324 \mid F423 \mid F132 \mid F231 \mid F142 \mid F241) 
ightarrow
          true
     | 1, true, false, false, (F341 | F431 | F342 | F432)
     | 1, false, true, false, (F134 | F143 | F234 | F243)
     \mid 1, false, false, true, (F314 \mid F413 \mid F324 \mid F423) \rightarrow
          true
      2, true, false, false, (F123 \mid F213 \mid F124 \mid F214)
       2, false, true, false, (F312 | F321 | F412 | F421)
     \mid 2, false, false, true, (F132 \mid F231 \mid F142 \mid F241) \rightarrow
     |\ 3, true, false, false, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234)
       3, false, true, false, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423)
     \mid 3, false, false, true, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) <math>\rightarrow
          true
     \mid \_ \rightarrow false
     end
V4 (DScalar2\_Vector2\_K\_Matrix\_ms (disc, \_), fusion, \_) \rightarrow
     let s12, s23, s13 =
       begin match PT.to\_list momenta with
       [q1; q2; q3] \rightarrow (P.Scattering.timelike\ (P.add\ q1\ q2),
                                  P.Scattering.timelike\ (P.add\ q2\ q3),
                                  P.Scattering.timelike\ (P.add\ q1\ q3))
        \mid \_ \rightarrow raise\ PT.Mismatched\_arity
       end in
     begin match disc, s12, s23, s13, fusion with
     | \ 0, true, false, false, (F341 \mid F431 \mid F342 \mid F432 \mid F123 \mid F213 \mid F124 \mid F214)
     0, false, true, false, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
     \mid 0, false, false, true, (F314 \mid F413 \mid F324 \mid F423 \mid F132 \mid F231 \mid F142 \mid F241) \rightarrow
       1, true, false, false, (F341 \mid F432 \mid F123 \mid F214)
     | 1, false, true, false, (F134 | F243 | F312 | F421)
```

```
\mid 1, false, false, true, (F314 \mid F423 \mid F132 \mid F241) \rightarrow
    2, true, false, false, (F431 \mid F342 \mid F213 \mid F124)
     2, false, true, false, (F143 | F234 | F321 | F412)
   \mid 2, false, false, true, (F413 \mid F324 \mid F231 \mid F142) \rightarrow
        true
    3, true, false, false, (F143 | F413 | F142 | F412 | F321 | F231 | F324 | F234)
     3, false, true, false, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423)
   \mid 3, false, false, true, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) <math>\rightarrow
   4, true, false, false, (F142 | F413 | F231 | F324)
     4, false, true, false, (F214 | F341 | F123 | F432)
   \mid 4, false, false, true, (F124 \mid F431 \mid F213 \mid F342) \rightarrow
    5, true, false, false, (F143 \mid F412 \mid F321 \mid F234)
    5, false, true, false, (F314 | F241 | F132 | F423)
   | 5, false, false, true, (F134 \mid F421 \mid F312 \mid F243) \rightarrow
   | 6, true, false, false, (F134 | F132 | F314 | F312 | F241 | F243 | F421 | F423)
     6, false, true, false, (F213 | F413 | F231 | F431 | F124 | F324 | F142 | F342)
   \mid 6, false, false, true, (F143 \mid F123 \mid F341 \mid F321 \mid F412 \mid F214 \mid F432 \mid F234) <math>\rightarrow
        true
    7, true, false, false, (F134 | F312 | F421 | F243)
    7, false, true, false, (F413 \mid F231 \mid F142 \mid F324)
   | 7, false, false, true, (F143 \mid F321 \mid F412 \mid F432) \rightarrow
        true
   | 8, true, false, false, (F132 | F314 | F241 | F423)
    8, false, true, false, (F213 | F431 | F124 | F342)
   \mid 8, false, false, true, (F123 \mid F341 \mid F214 \mid F234) \rightarrow
        true
   \mid \ \_ \ 	o \ \mathsf{false}
   end
V4 (DScalar2\_Vector2\_m\_0\_K\_Matrix\_cf (disc, \_), fusion, \_) \rightarrow
   let s12, s23, s13 =
     begin match PT.to_list momenta with
      [q1; q2; q3] \rightarrow (P.Scattering.timelike\ (P.add\ q1\ q2),
                                P.Scattering.timelike (P.add q2 q3),
                                P.Scattering.timelike (P.add q1 q3))
      \mid \_ \rightarrow raise\ PT.Mismatched\_arity
     end in
   begin match disc, s12, s23, s13, fusion with
   0, true, false, false, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
    0, false, true, false, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
   \mid 0, false, false, true, (F314 \mid F413 \mid F324 \mid F423 \mid F132 \mid F231 \mid F142 \mid F241) 
ightarrow
        true
    1, true, false, false, (F341 \mid F432 \mid F123 \mid F214)
    1, false, true, false, (F134 \mid F243 \mid F312 \mid F421)
   \mid 1, false, false, true, (F314 \mid F423 \mid F132 \mid F241) \rightarrow
        true
   |\ 2,\ \text{true},\ \text{false},\ \text{false},\ (F431\ |\ F342\ |\ F213\ |\ F124)
     2, false, true, false, (F143 | F234 | F321 | F412)
   \mid 2, false, false, true, (F413 \mid F324 \mid F231 \mid F142) \rightarrow
        true
    3, true, false, false, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234)
     3, false, true, false, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423)
   \mid 3, false, false, true, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) 
ightarrow
   | 4, true, false, false, (F142 | F413 | F231 | F324)
     4, false, true, false, (F214 \mid F341 \mid F123 \mid F432)
   \mid 4, false, false, true, (F124 \mid F431 \mid F213 \mid F342) \rightarrow
        true
```

```
| 5, true, false, false, (F143 | F412 | F321 | F234)
     5, false, true, false, (F314 | F241 | F132 | F423)
     \mid 5, false, false, true, (F134 \mid F421 \mid F312 \mid F243) \rightarrow
          true
     | 6, true, false, false, (F134 | F132 | F314 | F312 | F241 | F243 | F421 | F423)
       6, false, true, false, (F213 \mid F413 \mid F231 \mid F431 \mid F124 \mid F324 \mid F142 \mid F342)
     \mid 6, false, false, true, (F143 \mid F123 \mid F341 \mid F321 \mid F412 \mid F214 \mid F432 \mid F234) <math>\rightarrow
          true
     | 7, true, false, false, (F134 | F312 | F421 | F243)
     | 7, false, true, false, (F413 | F231 | F142 | F324)
     | 7, false, false, true, (F143 \mid F321 \mid F412 \mid F432) \rightarrow
       8, true, false, false, (F132 \mid F314 \mid F241 \mid F423)
       8, false, true, false, (F213 | F431 | F124 | F342)
     \mid 8, false, false, true, (F123 \mid F341 \mid F214 \mid F234) \rightarrow
          true
     \mid \_ \rightarrow \mathsf{false}
     end
V4\ (DScalar2\_Vector2\_m\_1\_K\_Matrix\_cf\ (disc,\ \_),\ fusion,\ \_) \rightarrow
     let s12, s23, s13 =
        begin match PT.to_list momenta with
        [q1; q2; q3] \rightarrow (P.Scattering.timelike (P.add q1 q2),
                                  P.Scattering.timelike (P.add q2 q3),
                                  P.Scattering.timelike\ (P.add\ q1\ q3))
        \downarrow \rightarrow raise PT.Mismatched_arity
        end in
     begin match disc, s12, s23, s13, fusion with
     | 0, true, false, false, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
       0, false, true, false, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
     \mid 0, false, false, true, (F314 \mid F413 \mid F324 \mid F423 \mid F132 \mid F231 \mid F142 \mid F241) \rightarrow
          true
     | 1, true, false, false, (F341 | F432 | F123 | F214)
     | 1, false, true, false, (F134 | F243 | F312 | F421)
     \mid 1, false, false, true, (F314 \mid F423 \mid F132 \mid F241) \rightarrow
          true
       2, true, false, false, (F431 \mid F342 \mid F213 \mid F124)
       2, false, true, false, (F143 | F234 | F321 | F412)
     \mid 2, false, false, true, (F413 \mid F324 \mid F231 \mid F142) \rightarrow
          true
     |\ 3, true, false, false, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234)
       3, false, true, false, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423)
     \mid 3, false, false, true, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow
          true
       4, true, false, false, (F142 \mid F413 \mid F231 \mid F324)
       4, false, true, false, (F214 | F341 | F123 | F432)
     \mid 4, false, false, true, (F124 \mid F431 \mid F213 \mid F342) \rightarrow
          true
     | 5, true, false, false, (F143 | F412 | F321 | F234)
       5, false, true, false, (F314 \mid F241 \mid F132 \mid F423)
     | 5, false, false, true, (F134 \mid F421 \mid F312 \mid F243) \rightarrow
          true
     \mid 6, true, false, false, (F134 \mid F132 \mid F314 \mid F312 \mid F241 \mid F243 \mid F421 \mid F423)
       6, false, true, false, (F213 | F413 | F231 | F431 | F124 | F324 | F142 | F342)
     \mid 6, false, false, true, (F143 \mid F123 \mid F341 \mid F321 \mid F412 \mid F214 \mid F432 \mid F234) \rightarrow
     | 7, true, false, false, (F134 | F312 | F421 | F243)
     | 7, false, true, false, (F413 | F231 | F142 | F324)
     | 7, false, false, true, (F143 \mid F321 \mid F412 \mid F432) \rightarrow
       8, true, false, false, (F132 | F314 | F241 | F423)
     8, false, true, false, (F213 | F431 | F124 | F342)
```

```
\mid 8, false, false, true, (F123 \mid F341 \mid F214 \mid F234) \rightarrow
          true
          \rightarrow false
     end
V4 (DScalar2\_Vector2\_m\_7\_K\_Matrix\_cf (disc, \_), fusion, \_) \rightarrow
     let s12, s23, s13 =
       begin match PT.to\_list\ momenta with
       [q1; q2; q3] \rightarrow (P.Scattering.timelike\ (P.add\ q1\ q2),
                                 P.Scattering.timelike\ (P.add\ q2\ q3),
                                 P.Scattering.timelike (P.add q1 q3))
       \downarrow \rightarrow raise PT.Mismatched_arity
       end in
     begin match disc, s12, s23, s13, fusion with
       0, true, false, false, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
       0, false, true, false, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
     \mid 0, false, false, true, (F314 \mid F413 \mid F324 \mid F423 \mid F132 \mid F231 \mid F142 \mid F241) 
ightarrow
          true
     | 1, true, false, false, (F341 | F432 | F123 | F214)
      1, false, true, false, (F134 | F243 | F312 | F421)
     | 1, false, false, true, (F314 \mid F423 \mid F132 \mid F241) \rightarrow
          true
       2, true, false, false, (F431 \mid F342 \mid F213 \mid F124)
       2, false, true, false, (F143 | F234 | F321 | F412)
     \mid 2, false, false, true, (F413 \mid F324 \mid F231 \mid F142) \rightarrow
      3, true, false, false, (F143 | F413 | F142 | F412 | F321 | F231 | F324 | F234)
       3, false, true, false, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423)
     \mid 3, false, false, true, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) <math>\rightarrow
       4, true, false, false, (F142 | F413 | F231 | F324)
       4, false, true, false, (F214 | F341 | F123 | F432)
     \mid 4, false, false, true, (F124 \mid F431 \mid F213 \mid F342) \rightarrow
     | 5, true, false, false, (F143 | F412 | F321 | F234)
       5, false, true, false, (F314 | F241 | F132 | F423)
     \mid 5, false, false, true, (F134 \mid F421 \mid F312 \mid F243) \rightarrow
          true
     \mid 6, true, false, false, (F134 \mid F132 \mid F314 \mid F312 \mid F241 \mid F243 \mid F421 \mid F423)
       6, false, true, false, (F213 | F413 | F231 | F431 | F124 | F324 | F142 | F342)
     \mid 6, false, false, true, (F143 \mid F123 \mid F341 \mid F321 \mid F412 \mid F214 \mid F432 \mid F234) <math>\rightarrow
          true
     7, true, false, false, (F134 | F312 | F421 | F243)
     | 7, false, true, false, (F413 | F231 | F142 | F324)
     | 7, false, false, true, (F143 \mid F321 \mid F412 \mid F432) \rightarrow
          true
       8, true, false, false, (F132 | F314 | F241 | F423)
       8, false, true, false, (F213 \mid F431 \mid F124 \mid F342)
     \mid 8, false, false, true, (F123 \mid F341 \mid F214 \mid F234) \rightarrow
          true
          \rightarrow false
     end
V4 (DScalar4\_K\_Matrix\_ms (disc, \_), fusion, \_) \rightarrow
     let s12, s23, s13 =
       begin match PT.to_list momenta with
       [q1; q2; q3] \rightarrow (P.Scattering.timelike (P.add q1 q2),
                                 P.Scattering.timelike (P.add q2 q3),
                                 P.Scattering.timelike\ (P.add\ q1\ q3))
       \downarrow \rightarrow raise PT.Mismatched_arity
       end in
     begin match disc, s12, s23, s13, fusion with
     |\ 0, true, false, false, (F341 \mid F431 \mid F342 \mid F432 \mid F123 \mid F213 \mid F124 \mid F214)
```

```
0, false, true, false, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
             \mid 0, false, false, true, (F314 \mid F413 \mid F324 \mid F423 \mid F132 \mid F231 \mid F142 \mid F241) <math>\rightarrow
               3, true, false, false, (F143 | F413 | F142 | F412 | F321 | F231 | F324 | F234)
               3, false, true, false, (F314 | F341 | F214 | F241 | F132 | F123 |
                                                                                            F432 \mid F423)
             \mid 3, false, false, true, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) <math>\rightarrow
                   true
               4, true, false, false, (F142 | F413 | F231 | F324)
             | 4, false, true, false, (F214 | F341 | F123 | F432)
             \mid 4, false, false, true, (F124 \mid F431 \mid F213 \mid F342) \rightarrow
                   true
               5, true, false, false, (F143 | F412 | F321 | F234)
               5, false, true, false, (F314 | F241 | F132
              \mid 5, false, false, true, (F134 \mid F421 \mid F312 \mid F243) \rightarrow
                   true
               6, true, false, false, (F134 | F132 | F314 | F312 | F241 | F243 | F421 | F423)
               6, false, true, false, (F213 \mid F413 \mid F231 \mid F431 \mid F124 \mid F324 \mid F142 \mid F342)
             \mid 6, false, false, true, (F143 \mid F123 \mid F341 \mid F321 \mid F412 \mid F214 \mid F432 \mid F234) \rightarrow
                   true
             7, true, false, false, (F134 | F312 | F421 | F243)
               7, false, true, false, (F413 | F231 | F142 |
                                                                  F324)
             | 7, false, false, true, (F143 \mid F321 \mid F412 \mid F432) \rightarrow
                   true
               8, true, false, false, (F132 | F314 | F241 | F423)
               8, false, true, false, (F213 \mid F431 \mid F124 \mid F342)
               8, false, false, true, (F123 \mid F341 \mid F214 \mid F234) \rightarrow
                   true
                   \rightarrow false
             | _
             end
        \mid \_ \rightarrow true
Counting QCD and EW orders.
     let \ qcd_ew_check \ orders =
        if fst\ (orders) \le fst\ (int\_orders) \land
            snd (orders) \leq snd (int\_orders) then
          true
        else
          false
```

Match a set of flavors to a set of momenta. Form the direct product for the lists of momenta two and three with the list of couplings and flavors two and three.

```
let flavor\_keystone\ select\_p\ dim\ (f1,\ f23)\ (p1,\ p23)\ =\ (\{\ A.flavor\ =\ f1;\ A.momentum\ =\ P.of\_ints\ dim\ p1;\ A.wf\_tag\ =\ A.Tags.null\_wf\ \},
Product.fold2\ (fun\ (c,\ f)\ p\ acc\ \to\ try\ |\ let\ p'\ =\ PT.map\ (P.of\_ints\ dim\ p1)\ (PT.to\_list\ p')\ \land\ kmatrix\_cuts\ c\ p'\ then\ (c,\ PT.map2\ (fun\ f''\ p''\ \to\ \{\ A.flavor\ =\ f'';\ A.momentum\ =\ p'';\ A.wf\_tag\ =\ A.Tags.null\_wf\ \})\ f\ p')\ ::\ acc\ else\ acc\ with\ |\ PT.Mismatched\_arity\ \to\ acc)\ f23\ p23\ [])
```

Produce all possible combinations of vertices (flavor keystones) and momenta by forming the direct product. The semantically equivalent Product.list2 ($flavor_keystone$ select_wf n) vertices keystones with subsequent filtering would be a very bad idea, because a potentially huge intermediate list is built for large models. E. g. for the MSSM this would lead to non-termination by thrashing for $2 \rightarrow 4$ processes on most PCs.

```
let flavor\_keystones filter select\_p dim vertices keystones = Product.fold2 (fun v k acc \rightarrow filter (flavor\_keystone select\_p dim v k) acc) vertices keystones []
```

Flatten the nested lists of vertices into a list of attached lines.

```
\begin{array}{lll} \text{let } \textit{flatten\_keystones} \ t &= \\ \textit{ThoList.flatmap} \ (\text{fun} \ (p1, \ p23) \ \rightarrow \\ p1 \ :: \ (\textit{ThoList.flatmap} \ (\text{fun} \ (\_, \ rhs) \ \rightarrow \ PT.to\_list \ rhs) \ p23)) \ t \end{array}
```

Subtrees

Fuse a tuple of wavefunctions, keeping track of Fermi statistics. Record only the sign *relative* to the children. (The type annotation is only for documentation.)

```
let fuse \ select\_wf \ select\_vtx \ wfss : (A.wf \times stat \times A.rhs) \ list =
       if PT.for\_all (fun (wf, \_) \rightarrow is\_source \ wf) wfss then
          try
            let wfs, ss = PT.split wfss in
            let flavors = PT.map A.flavor wfs
            and momenta = PT.map A.momentum wfs
in
            let p = PT.fold\_left\_internal P.add momenta in
            List.fold\_left
               (fun acc\ (f,\ c)\ \rightarrow
                 if select\_wf f p (PT.to\_list momenta)
                    \land select_vtx c f (PT.to_list flavors)
                    \land kmatrix_cuts c momenta then
                    (* let \_ = Printf.eprintf.fusion.fuse: \_%s = -.%s = (M.flavor_to_string f) (ThoList.to_string M.flavor_to_string f)
*)
                    let s = S.stat\_fuse (fermion\_lines c) (PT.to\_list ss) f in
                    let flip =
                       PT.fold\_left (fun acc \ s' \rightarrow acc \times stat\_sign \ s') (stat\_sign \ s) \ ss in
                    (\{ A.flavor = f;
                        A.momentum = p;
                        A.wf_tag = A.Tags.null_wf \}, s,
                     (\{ Tagged\_Coupling.sign = flip; \}
                         Tagged\_Coupling.coupling = c;
                         Tagged\_Coupling\_coupling\_tag = A.Tags.null\_coupling \}, wfs)) :: acc
                 else
                    acc)
               [] (fuse_rhs flavors)
          with
          \mid P.Duplicate \_ \mid S.Impossible \rightarrow []
       else
```

Ś

Eventually, the pairs of tower and dag in $fusion_tower'$ below could and should be replaced by a graded DAG. This will look like, but currently tower containts statistics information that is missing from dag:

```
Type node = flavor * p is not compatible with type wf * stat
```

This should be easy to fix. However, replacing type t = wf with type $t = wf \times stat$ is not a good idea because the variable stat makes it impossible to test for the existence of a particular wf in a DAG.



In summary, it seems that $(wf \times stat)$ list $array \times A.D.t$ should be replaced by $(wf \rightarrow stat) \times A.D.t$.

```
\begin{array}{lll} \operatorname{module} \ GF &= \\ \operatorname{struct} & \operatorname{module} \ Nodes \ = \\ \operatorname{struct} & \operatorname{type} \ t \ = \ A.wf \\ \operatorname{module} \ G \ = \ \operatorname{struct} \ \operatorname{type} \ t \ = \ int \ \operatorname{let} \ compare \ = \ compare \ \operatorname{end} \end{array}
```

```
let compare = A.order\_wf
         let rank \ wf = P.rank \ wf.A.momentum
    module Edges = struct type t = A.coupling let compare = compare end
    module F = DAG.Forest(PT)(Nodes)(Edges)
    type node = Nodes.t
    type edge = F.edge
    type \ children = F.children
    type t = F.t
    let compare = F.compare
    \mathsf{let}\ \mathit{for\_all}\ =\ \mathit{F.for\_all}
    let fold = F.fold
  end
module D' = DAG.Graded(GF)
let tower\_of\_dag dag =
  let _, max\_rank = D'.min\_max\_rank dag in
  Array.init \ max\_rank \ (fun \ n \rightarrow D'.ranked \ n \ dag)
```

The function $fusion_tower'$ recursively builds the tower of all fusions from bottom up to a chosen level. The argument tower is an array of lists, where the i-th sublist (counting from 0) represents all off shell wave functions depending on i + 1 momenta and their Fermistatistics.

$$\left[\{ \phi_{1}(p_{1}), \phi_{2}(p_{2}), \phi_{3}(p_{3}), \dots \},
\{ \phi_{12}(p_{1} + p_{2}), \phi'_{12}(p_{1} + p_{2}), \dots, \phi_{13}(p_{1} + p_{3}), \dots, \phi_{23}(p_{2} + p_{3}), \dots \},
\dots
\{ \phi_{1 \dots n}(p_{1} + \dots + p_{n}), \phi'_{1 \dots n}(p_{1} + \dots + p_{n}), \dots \} \right]$$
(8.6)

The argument dag is a DAG representing all the fusions calculated so far. NB: The outer array in tower is always very short, so we could also have accessed a list with List.nth. Appending of new members at the end brings no loss of performance. NB: the array is supposed to be immutable.

The towers must be sorted so that the combinatorical functions can make consistent selections.



Intuitively, this seems to be correct. However, one could have expected that no element appears twice and that this ordering is not necessary . . .

```
let grow select\_wf select\_vtx tower =
  let rank = succ (Array.length tower) in
   List.sort pcompare
     (PT.graded\_sym\_power\_fold\ rank
         (fun wfs\ acc \rightarrow fuse\ select\_wf\ select\_vtx\ wfs\ @\ acc)\ tower\ [\ ]
let add\_offspring \ dag \ (wf, \_, \ rhs) =
   A.D.add\_offspring \ wf \ rhs \ dag
let filter_offspring fusions =
   List.map (fun (wf, s, \_) \rightarrow (wf, s)) fusions
let rec fusion\_tower' n\_max select\_wf select\_vtx tower dag : (A.wf \times stat) list array \times A.D.t =
  if Array.length\ tower\ \geq\ n\_max then
     (tower, dag)
     let \ tower' = grow \ select\_wf \ select\_vtx \ tower \ in
     fusion\_tower' n\_max select\_wf select\_vtx
        (Array.append\ tower\ [|filter\_offspring\ tower'|])
        (List.fold_left add_offspring dag tower')
```

Discard the tower and return a map from wave functions to Fermistatistics together with the DAG.

```
 \begin{array}{lll} \text{let} \ make\_external\_dag \ wfs &= \\ \ List.fold\_left \ (\text{fun} \ m \ (wf, \ \_) \ \rightarrow \ A.D.add\_node \ wf \ m) \ A.D.empty \ wfs \\ \text{let} \ mixed\_fold\_left \ f \ acc \ lists \ = \\ \end{array}
```

```
Array.fold\_left\ (List.fold\_left\ f)\ acc\ lists \mathsf{module}\ Stat\_Map\ =\ Map.Make\ (\mathsf{struct}\ \mathsf{type}\ t\ =\ A.wf\ \mathsf{let}\ compare\ =\ A.order\_wf\ \mathsf{end}) \mathsf{let}\ fusion\_tower\ height\ select\_wf\ select\_vtx\ wfs\ :\ (A.wf\ \to\ stat)\ \times\ A.D.t\ =\ \mathsf{let}\ tower,\ dag\ =\ fusion\_tower'\ height\ select\_wf\ select\_vtx\ [|wfs|]\ (make\_external\_dag\ wfs)\ \mathsf{in}\ \mathsf{let}\ stats\ =\ mixed\_fold\_left\ (\mathsf{fun}\ m\ (wf,\ s)\ \to\ Stat\_Map.add\ wf\ s\ m)\ Stat\_Map.empty\ tower\ \mathsf{in}\ ((\mathsf{fun}\ wf\ \to\ Stat\_Map.find\ wf\ stats),\ dag)
```

Calculate the minimal tower of fusions that suffices for calculating the amplitude.

```
let minimal\_fusion\_tower\ n\ select\_wf\ select\_vtx\ wfs: (A.wf \rightarrow stat) \times A.D.t = fusion\_tower\ (T.max\_subtree\ n)\ select\_wf\ select\_vtx\ wfs
```

Calculate the complete tower of fusions. It is much larger than required, but it allows a complete set of gauge checks.

```
let complete\_fusion\_tower\ select\_wf\ select\_vtx\ wfs: (A.wf \rightarrow stat) \times A.D.t = fusion\_tower\ (List.length\ wfs\ -\ 1)\ select\_wf\ select\_vtx\ wfs
```



There is a natural product of two DAGs using fuse. Can this be used in a replacement for $fusion_tower$? The hard part is to avoid double counting, of course. A straight forward solution could do a diagonal sum (in order to reject flipped offspring representing the same fusion) and rely on the uniqueness in DAG otherwise. However, this will (probably) slow down the procedure significanty, because most fusions (including Fermi signs!) will be calculated before being rejected by $DAG().add_offspring$.

Add to dag all Goldstone bosons defined in tower that correspond to gauge bosons in dag. This is only required for checking Slavnov-Taylor identities in unitarity gauge. Currently, it is not used, because we use the complete tower for gauge checking.

```
let harvest\_goldstones tower dag = A.D.fold\_nodes (fun wf dag' \rightarrow match M.goldstone wf.A.flavor with |Some (g, \_) \rightarrow let wf' = \{ wf \text{ with } A.flavor = g \} in if <math>A.D.is\_node wf' tower then begin A.D.harvest tower wf' dag' end else begin dag' end log
```

Calculate the sign from Fermi statistics that is not already included in the children.

```
let strip\_fermion\_lines = function
    (Coupling. V3 \_ | Coupling. V4 \_ as v) \rightarrow v
    Coupling. Vn (Coupling. UFO (c, l, s, fl, col), f, x) \rightarrow
      Coupling. Vn (Coupling. UFO (c, l, s, [], col), f, x)
let num\_fermion\_lines\_v3 = function
    FBF \ \_ \ | \ PBP \ \_ \ | \ BBB \ \_ \ | \ GBG \ \_ \ \rightarrow \ 1
     \rightarrow 0
let num\_fermion\_lines = function
     Coupling. Vn (Coupling. UFO (c, l, s, fl, col), f, x) \rightarrow List.length fl
     Coupling. \ V3 \ (v3, \ \_, \ \_) \ \rightarrow \ num\_fermion\_lines\_v3 \ v3
    Coupling. V4 = 0
let stat\_keystone \ v \ stats \ wf1 \ wfs =
  let wf1' = stats wf1
  and wfs' = PT.map \ stats \ wfs in
  let f = A.flavor wf1 in
  let slist = wf1' :: PT.to\_list wfs' in
  let stat = S.stat\_keystone (fermion_lines v) slist f in
```

```
(* We can compare with the legacy implementation only if there are no fermion line ambiguities possible,
i.e. for at most one line. *)
       if num\_fermion\_lines v < 2 then
         begin
            let \ legacy = S.stat\_keystone \ None \ slist f \ in
            if \neg (S.equal \ stat \ legacy) then
              failwith
                 (Printf.sprintf
                    "Fusion.stat_keystone: \_\%s_\<>\_\%s!"
                    (S.stat\_to\_string\ legacy)
                    (S.stat\_to\_string\ stat));
            if \neg (S.saturated legacy) then
              failwith
                 (Printf.sprintf
                    "Fusion.stat_keystone: ulegacy incomplete: u%s!"
                    (S.stat\_to\_string\ legacy))
         end;
       if \neg (S.saturated stat) then
         fail with
            (Printf.sprintf
               "Fusion.stat_keystone: incomplete: %s!"
               (S.stat\_to\_string\ stat));
       stat\_sign \ stat
          	imes PT.fold\_left (fun acc \ wf \ 	o \ acc \ 	imes \ stat\_sign \ wf) (stat\_sign \ wf1') wfs'
    let stat\_keystone\_logging v stats wf1 wfs =
       let sign = stat\_keystone \ v \ stats \ wf1 \ wfs in
       Printf.eprintf
         (M.flavor\_to\_string\ (A.flavor\ wf1))
         (ThoList.to_string
             (\text{fun } wf \rightarrow M.flavor\_to\_string (A.flavor wf))
             (PT.to\_list \ wfs))
         sign;
       sign
Test all members of a list of wave functions are defined by the DAG simultaneously:
    let test\_rhs dag(\_, wfs) =
       PT.for\_all (fun wf \rightarrow is\_source \ wf \land A.D.is\_node \ wf \ dag) wfs
Add the keystone (wf1, pairs) to acc only if it is present in dag and calculate the statistical factor depending
on stats en passant:
    let filter_keystone stats dag (wf1, pairs) acc =
       if is\_source \ wf1 \land A.D.is\_node \ wf1 \ dag then
         match List.filter (test_rhs dag) pairs with
          | [] \rightarrow acc
         | pairs' \rightarrow (wf1, List.map (fun (c, wfs) \rightarrow
              \{ Tagged\_Coupling.sign = stat\_keystone \ c \ stats \ wf1 \ wfs; \}
                  Tagged\_Coupling.coupling = c;
                  Tagged\_Coupling\_coupling\_tag = A.Tags.null\_coupling \},
               wfs)) pairs') :: acc
       else
          acc
                                                     Amplitudes
    module C = Cascade.Make(M)(P)
    type selectors = C.selectors
    let external\_wfs n particles =
```

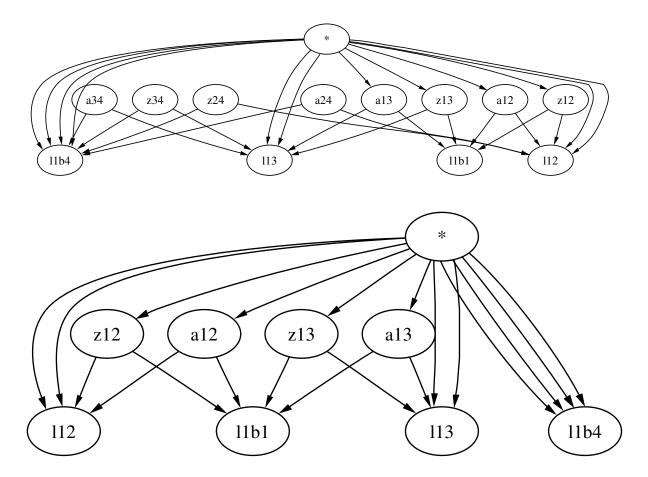


Figure 8.2: The DAGs for Bhabha scattering before and after weeding out unused nodes. The blatant asymmetry of these DAGs is caused by our prescription for removing doubling counting for an even number of external lines.

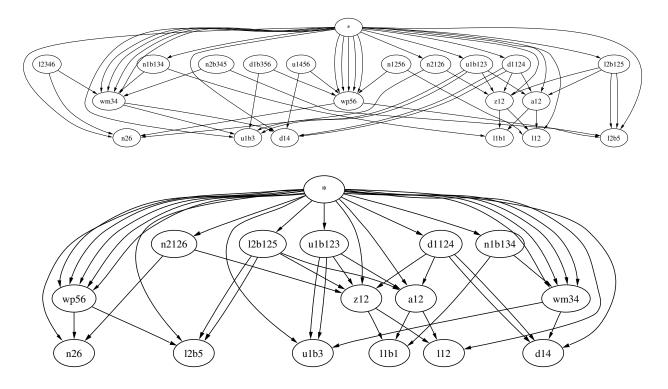


Figure 8.3: The DAGs for $e^+e^- \to u\bar{d}\mu^-\bar{\nu}_{\mu}$ before and after weeding out unused nodes.

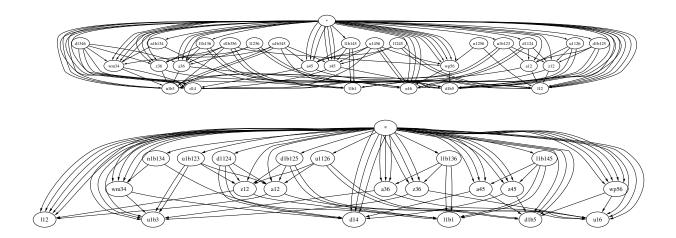


Figure 8.4: The DAGs for $e^+e^- \to u\bar{d}d\bar{u}$ before and after weeding out unused nodes.

```
 \begin{array}{lll} \textit{List.map} & (\text{fun } (f, \ p) \ \rightarrow \\ & (\{ \ \textit{A.flavor} = f; \\ & \textit{A.momentum} = \textit{P.singleton } n \ p; \\ & \textit{A.wf\_tag} = \textit{A.Tags.null\_wf} \ \}, \\ & \textit{stat } f \ p)) \ \textit{particles} \end{array}
```

Main Function

```
module WFMap = Map.Make (struct type t = A.wf let compare = compare end)
map\_amplitude\_wfs\ f\ a applies the function f: wf \to wf to all wavefunctions appearing in the amplitude a.
    let map\_amplitude\_wfs f a =
      let map\_rhs (c, wfs) = (c, PT.map f wfs) in
      let map\_braket (wf, rhs) = (f wf, List.map map\_rhs rhs)
      and map\_fusion (lhs, rhs) = (f lhs, List.map map\_rhs rhs) in
      let map\_dag = A.D.map f (fun node \ rhs \rightarrow map\_rhs \ rhs) in
      let tower = map\_dag a.A.fusion\_tower
      and dag = map\_dag \ a.A.fusion\_dag \ in
      let dependencies\_map =
        A.D.fold (fun wf \longrightarrow WFMap.add wf (A.D.dependencies dag wf)) dag WFMap.empty in
      \{A.fusions = List.map\ map\_fusion\ a.A.fusions;
         A.brakets = List.map\ map\_braket\ a.A.brakets;
        A.on\_shell = a.A.on\_shell;
        A.is\_gauss = a.A.is\_gauss;
         A.constraints = a.A.constraints;
         A.incoming = a.A.incoming;
        A.outgoing = a.A.outgoing;
        A.externals = List.map f a.A.externals;
        A.symmetry = a.A.symmetry;
        A.dependencies = (fun \ wf \rightarrow WFMap.find \ wf \ dependencies\_map);
        A.fusion\_tower = tower;
        A.fusion\_dag = dag
```

This is the main function that constructs the amplitude for sets of incoming and outgoing particles and returns the results in conveniently packaged pieces.

let amplitude goldstones selectors fin fout =

Set up external lines and match flavors with numbered momenta.

```
let f = fin @ List.map \ M.conjugate \ fout in let nin, \ nout = List.length \ fin, \ List.length \ fout in let n = nin + nout in let n = nout in let n = nout in let n = nout in let nout
```

Build the full fusion tower (including nodes that are never needed in the amplitude).

```
let stats, tower =
  if goldstones then
    complete_fusion_tower select_wf select_vtx wfs
  else
    minimal_fusion_tower n select_wf select_vtx wfs in
```

Find all vertices for which all off shell wavefunctions are defined by the tower.

```
\begin{array}{ll} \mathsf{let}\ \mathit{brakets}\ = \\ \mathit{flavor\_keystones}\ (\mathit{filter\_keystone}\ \mathit{stats}\ \mathit{tower})\ \mathit{select\_p}\ \mathit{n} \end{array}
```

```
(filter_vertices select_vtx
               (vertices\ (min\ n\ (M.max\_degree\ ()))\ (M.flavors\ ())))
           (T.keystones (ThoList.range 1 n)) in
Remove the part of the DAG that is never needed in the amplitude.
      let dag =
         if goldstones then
           tower
         else
           A.D.harvest_list tower (A.wavefunctions brakets) in
Remove the leaf nodes of the DAG, corresponding to external lines.
      let fusions =
         List.filter (function (_, []) \rightarrow false | _ \rightarrow true) (A.D.lists\ dag) in
Calculate the symmetry factor for identical particles in the final state.
      let symmetry =
         Combinatorics.symmetry fout in
      let dependencies\_map =
         A.D. fold (fun wf \rightarrow WFMap. add wf (A.D. dependencies dag wf)) dag WFMap. empty in
Finally: package the results:
       \{ A.fusions = fusions; \}
         A.brakets = brakets;
         A.on\_shell = (fun \ wf \rightarrow C.on\_shell \ selectors \ (A.flavor \ wf) \ wf.A.momentum);
         A.is\_gauss = (fun \ wf \rightarrow C.is\_gauss \ selectors \ (A.flavor \ wf) \ wf.A.momentum);
         A.constraints = C.description selectors;
         A.incoming = fin;
         A.outgoing = fout;
         A.externals = List.map fst wfs;
         A.symmetry = symmetry;
         A.dependencies = (fun \ wf \rightarrow WFMap.find \ wf \ dependencies\_map);
         A.fusion\_tower = tower;
         A.fusion\_dag = dag 
                                                     Color
    module CM = Colorize.It(M)
    module CA = Amplitude(PT)(P)(CM)
    let \ colorize_wf \ flavor \ wf =
       \{ CA.flavor = flavor; \}
         CA.momentum = wf.A.momentum;
         CA.wf\_tag = wf.A.wf\_tag 
    let uncolorize\_wf \ wf =
       \{A.flavor = CM.flavor\_sans\_color wf.CA.flavor;
         A.momentum = wf.CA.momentum;
         A.wf_tag = wf.CA.wf_tag
```

At the end of the day, I shall want to have some sort of fibered DAG as abstract data type, with a projection of colored nodes to their uncolored counterparts.

```
module \ CWFBundle = Bundle.Make
     (struct
       \mathsf{type}\ \mathit{elt}\ =\ \mathit{CA.wf}
       let compare\_elt = compare
       type base = A.wf
       let compare\_base = compare
       let pi wf =
         \{A.flavor = CM.flavor\_sans\_color wf.CA.flavor;
```

```
Implementation of Fusion
                 A.momentum = wf.CA.momentum;
                 A.wf_taq = wf.CA.wf_taq
         end)
   For now, we can live with simple aggregation:
    type fibered\_dag = \{ dag : CA.D.t; bundle : CWFBundle.t \}
Not yet(?) needed: module CS = Stat(CM)
    let colorize_sterile_nodes dag f wf fibered_dag =
       if A.D.is\_sterile wf dag then
         let wf', wf\_bundle' = f wf fibered\_dag in
          \{ dag = CA.D.add\_node wf' fibered\_dag.dag; \}
            bundle = wf\_bundle' }
       else
         fibered\_dag
    let colorize_nodes f wf rhs fibered_dag =
       let wf\_rhs\_list', wf\_bundle' = f wf rhs fibered\_dag in
       let dag' =
         List.fold\_right
            (\text{fun } (wf', rhs') \rightarrow CA.D.add\_offspring wf' rhs')
            wf\_rhs\_list'\ fibered\_dag.dag\ in
       \{ daq = daq'; 
          bundle = wf\_bundle' }
O'Caml (correctly) infers the type val colorize\_dag: (D.node \rightarrow D.edge \times D.children \rightarrow fibered\_dag \rightarrow
 (CA.D.node \times (CA.D.edge \times CA.D.children)) list \times CWFBundle.t) \rightarrow (D.node \rightarrow fibered_dag \rightarrow
 CA.D.node \times CWFBundle.t) \rightarrow D.t \rightarrow CWFBundle.t \rightarrow fibered\_dag.
    let colorize_dag f_node f_ext dag wf_bundle =
       A.D.fold\ (colorize\_nodes\ f\_node)\ dag
          (A.D.fold\_nodes\ (colorize\_sterile\_nodes\ dag\ f\_ext)\ dag
             \{ dag = CA.D.empty; bundle = wf\_bundle \} \}
    let colorize_external wf fibered_dag =
       match CWFBundle.inv_pi wf fibered_dag.bundle with
       | [c_-wf] \rightarrow (c_-wf, fibered\_dag.bundle)|
       [] \rightarrow failwith "colorize_external: \_not \_found"
       _ → failwith "colorize_external:_not_unique"
    let fuse\_c\_wf \ rhs =
       let momenta = PT.map (fun wf \rightarrow wf.CA.momentum) rhs in
       List.filter
          (fun (\_, c) \rightarrow kmatrix\_cuts \ c \ momenta)
         (CM.fuse\ (List.map\ (fun\ wf\ 	o\ wf.CA.flavor)\ (PT.to\_list\ rhs)))
    let colorize_coupling c coupling =
          \{ coupling with Tagged\_Coupling.coupling = c \}
    let colorize_fusion wf (coupling, children) fibered_dag =
       let match\_flavor (f, \_) = (CM.flavor\_sans\_color f = A.flavor wf)
       and find\_colored \ wf' = CWFBundle.inv\_pi \ wf' \ fibered\_dag.bundle in
       let fusions =
          Tho List. flat map
            (fun c\_children \rightarrow
```

 $(colorize_wf\ f\ wf,\ (colorize_coupling\ c\ coupling,\ c_children)))$

 $(List.filter\ match_flavor\ (fuse_c_wf\ c_children)))$

(PT.product (PT.map find_colored children)) in

 $(\text{fun } (c_wf, _) \rightarrow CWFBundle.add c_wf)$

List.map

let bundle = $List.fold_right$

 $(fun (f, c) \rightarrow$

```
fusions fibered_dag.bundle in
       (fusions, bundle)
    let colorize_braket1 (wf, (coupling, children)) fibered_dag =
       let find\_colored\ wf' = CWFBundle.inv\_pi\ wf'\ fibered\_dag.bundle\ in
       Product.fold2
          (fun bra \ ket \ acc \rightarrow
            List.fold\_left
               (fun brakets (f, c) \rightarrow
                 if CM.conjugate \ bra.CA.flavor = f then
                    (bra, (colorize\_coupling \ c \ coupling, \ ket)) :: brakets
                 else
                    brakets)
               acc (fuse\_c\_wf \ ket))
          (find_colored wf) (PT.product (PT.map find_colored children)) []
     module \ CWFMap =
       Map.Make (struct type t = CA.wf let compare = CA.order\_wf end)
     module \ CKetSet =
       Set.Make (struct type t = CA.rhs let compare = compare end)
Find a set of kets in map that belong to bra. Return the empty set, if nothing is found.
    let lookup\_ketset bra map =
       try CWFMap.find bra map with Not_found → CKetSet.empty
Return the set of kets belonging to bra in map, augmented by ket.
     let \ addto\_ketset \ bra \ ket \ map =
       CKetSet.add ket (lookup_ketset bra map)
Augment or update map with a new (bra, ket) relation.
     let \ addto\_ketset\_map \ map \ (bra, \ ket) =
       CWFMap.add bra (addto_ketset bra ket map) map
Take a list of (bra, ket) pairs and group the kets according to bra. This is very similar to ThoList.factorize
on page 624, but the latter keeps duplicate copies, while we keep only one, with equality determined by
CA.order\_wf.
   Isn't Bundle L.1 the correct framework for this?
     let factorize_brakets brakets =
       CWFMap.fold
          (\textit{fun }\textit{bra }\textit{ket }\textit{acc} \ \rightarrow \ (\textit{bra}, \ \textit{CKetSet.elements }\textit{ket}) \ :: \ \textit{acc})
          (List.fold_left addto_ketset_map CWFMap.empty brakets)
         {\tt let} \ colorize\_braket \ (w\!f, \ rhs\_list) \ fibered\_dag \ =
       factorize\_brakets
          (ThoList.flatmap
             (fun \ rhs \rightarrow (colorize\_braket1 \ (wf, \ rhs) \ fibered\_dag))
             rhs\_list)
     let colorize_amplitude a fin fout =
       let f = fin @ List.map CM.conjugate fout in
       let nin, nout = List.length fin, List.length fout in
       \mathsf{let}\ n\ =\ nin\ +\ nout\ \mathsf{in}
       let externals = List.combine f (ThoList.range 1 n) in
       let \ external\_wfs = CA.external\_wfs \ n \ externals \ in
       let \ wf\_bundle = CWFBundle.of\_list \ external\_wfs \ in
       let fibered\_dag =
          colorize\_dag
            colorize_fusion colorize_external a.A.fusion_dag wf_bundle in
```

let brakets =

```
ThoList.flatmap
       (fun \ braket \rightarrow \ colorize\_braket \ braket \ fibered\_dag)
       a.A.brakets in
  let dag = CA.D.harvest\_list fibered\_dag.dag (CA.wavefunctions brakets) in
  let fusions =
     List.filter (function (_, []) \rightarrow false | _ \rightarrow true) (CA.D.lists\ dag) in
  let dependencies\_map =
     CA.D.fold
       (\text{fun } wf \perp \rightarrow CWFMap.add \ wf \ (CA.D.dependencies \ dag \ wf))
       dag CWFMap.empty in
  \{ CA.fusions = fusions; \}
     CA.brakets = brakets;
     CA.constraints = a.A.constraints;
     CA.incoming = fin;
     CA.outgoing = fout;
     CA.externals = external\_wfs;
     CA.fusion\_dag = dag;
     CA.fusion\_tower = dag;
     CA.symmetry = a.A.symmetry;
     CA.on\_shell = (fun \ wf \rightarrow a.A.on\_shell \ (uncolorize\_wf \ wf));
     CA.is\_gauss = (fun \ wf \rightarrow a.A.is\_gauss (uncolorize\_wf \ wf));
     CA.dependencies = \{fun \ wf \rightarrow CWFMap.find \ wf \ dependencies\_map\} \}
let \ allowed \ amplitude =
  match amplitude. CA. brakets with
    [] 
ightarrow \mathsf{false}
    _{	extsf{-}} 
ightarrow true
let \ colorize\_amplitudes \ a =
  List.fold\_left
     (fun amps (fin, fout) \rightarrow
       let amp = colorize\_amplitude \ a \ fin \ fout \ in
       if allowed amp then
          amp :: amps
       else
          amps)
    [] (CM.amplitude a.A.incoming a.A.outgoing)
let amplitudes goldstones exclusions selectors fin fout =
  colorize_amplitudes (amplitude goldstones selectors fin fout)
let amplitude_sans_color goldstones exclusions selectors fin fout =
  amplitude\ goldstones\ selectors\ fin\ fout
type flavor = CA.flavor
type flavor\_sans\_color = A.flavor
type p = A.p
type wf = CA.wf
let conjugate = CA.conjugate
let flavor = CA.flavor
let flavor\_sans\_color \ wf = CM.flavor\_sans\_color \ (CA.flavor \ wf)
let momentum = CA.momentum
let momentum\_list = CA.momentum\_list
let wf_tag = CA.wf_tag
type coupling = CA.coupling
let \ sign = CA.sign
let coupling = CA.coupling
let coupling\_tag = CA.coupling\_tag
type exclusions = CA.exclusions
let no\_exclusions = CA.no\_exclusions
```

```
type \alpha children = \alpha CA.children
type rhs = CA.rhs
let children = CA.children
type fusion = CA.fusion
let lhs = CA.lhs
let rhs = CA.rhs
type braket = CA.braket
let bra = CA.bra
let ket = CA.ket
type amplitude = CA.amplitude
type amplitude\_sans\_color = A.amplitude
let incoming = CA.incoming
let outgoing = CA.outgoing
let \ externals = CA.externals
let fusions = CA.fusions
let brakets = CA.brakets
let symmetry = CA.symmetry
let \ on\_shell = CA.on\_shell
let is\_gauss = CA.is\_gauss
let \ constraints = CA.constraints
let \ variables \ a \ = \ List.map \ lhs \ (fusions \ a)
let \ dependencies = CA. dependencies
                                         Checking Conservation Laws
let check\_charges() =
   let vlist3, vlist4, vlistn = M.vertices () in
   List.filter
      (\text{fun } flist \rightarrow \neg (M.Ch.is\_null (M.Ch.sum (List.map M.charges flist))))
      (List.map (fun ((f1, f2, f3), \_, \_) \rightarrow [f1; f2; f3]) vlist3)
       @ List.map (fun ((f1, f2, f3, f4), -, -) \rightarrow [f1; f2; f3; f4]) <math>vlist4
       @ List.map (fun (flist, \_, \_) \rightarrow flist) vlistn)
                                                  Diagnostics
let count\_propagators a =
   List.length \ a.CA.fusions
let count\_fusions \ a =
   List.fold\_left (fun n (\_, a) \rightarrow n + List.length a) 0 a.CA.fusions
     + List.fold\_left (fun \ n \ (\_, \ t) \rightarrow n + List.length \ t) \ 0 \ a.CA.brakets
     + List.length a.CA.brakets
This brute force approach blows up for more than ten particles. Find a smarter algorithm.
let count\_diagrams \ a =
   List.fold\_left (fun n (wf1, wf23) \rightarrow
     n + CA.D.count\_trees \ wf1 \ a.CA.fusion\_dag \ \times
        (List.fold\_left (fun n' (\_, wfs) \rightarrow
           n' + PT.fold\_left (fun n'' wf \rightarrow
             n'' \ \times \ CA.D.count\_trees \ wf \ a.CA.fusion\_dag) \ 1 \ wfs) \ 0 \ wf23))
     0 a.CA.brakets
exception Impossible
let forest' a =
   let below \ wf = CA.D.forest\_memoized \ wf \ a.CA.fusion\_dag in
   ThoList.flatmap
      (fun (bra, ket) \rightarrow
```

```
(Product.list2 (fun bra' ket' \rightarrow bra' :: ket')
            (below bra)
            (ThoList.flatmap
                (\text{fun } (\_, wfs) \rightarrow
                   Product.list (fun \ w \rightarrow w) (PT.to\_list (PT.map \ below \ wfs)))
      a.CA.brakets
let cross \ wf =
    \{ CA.flavor = CM.conjugate wf.CA.flavor; \}
      CA.momentum = P.neg \ wf.CA.momentum;
      CA.wf\_tag = wf.CA.wf\_tag 
let fuse\_trees \ wf \ ts =
    Tree.fuse (fun (wf', e) \rightarrow (cross \ wf', e))
      wf (fun t \rightarrow List.mem \ wf (Tree.leafs t)) ts
let forest \ wf \ a =
    List.map\ (fuse\_trees\ wf)\ (forest'\ a)
{\tt let} \ poles\_beneath \ wf \ dag \ =
    CA.D.eval\_memoized (fun \ wf' \rightarrow [[]])
      (\text{fun } wf' \perp p \rightarrow List.map (\text{fun } p' \rightarrow wf' :: p') p)
      (fun wf1 wf2 \rightarrow
         Product.fold2 (fun wf' wfs' wfs'' \rightarrow (wf' @ wfs') :: wfs'') <math>wf1 wf2 [])
      (@) [[]] [[]] wf dag
let poles a =
    ThoList.flatmap (fun (wf1, wf23) \rightarrow
      \mathsf{let}\ poles\_wf1\ =\ poles\_beneath\ wf1\ a.CA.fusion\_dag\ \mathsf{in}
      (ThoList.flatmap (fun (\_, wfs) \rightarrow
         Product.list List.flatten
           (PT.to\_list\ (PT.map\ (fun\ wf\ 
ightarrow
              poles\_wf1 @ poles\_beneath wf a.CA.fusion\_dag) wfs)))
          wf23)
      a.CA.brakets
module WFSet =
    Set.Make (struct type t = CA.wf let compare = CA.order\_wf end)
let s\_channel a =
    WFSet.elements
      (ThoList.fold\_right2
          (fun wf \ wfs \rightarrow
            if P.Scattering.timelike\ wf.CA.momentum\ then
                WFSet.add wf wfs
            else
               wfs) (poles a) WFSet.empty)
This should be much faster! Is it correct? Is it faster indeed?
let poles' a =
   List.map CA.lhs a.CA.fusions
let s\_channel \ a =
    WFSet.elements
      (List.fold\_right
          (fun wf \ wfs \rightarrow
            if P.Scattering.timelike wf.CA.momentum then
                WFSet.add wf wfs
            else
               wfs) (poles' a) WFSet.empty)
```

Pictures

Export the DAG in the dot(1) file format so that we can draw pretty pictures to impress audiences ...

```
let p2s p =
  if p \geq 0 \land p \leq 9 then
    string\_of\_int p
  else if p~\leq~36 then
     String.make\ 1\ (Char.chr\ (Char.code\ 'A'\ +\ p\ -\ 10))
     "_"
let variable wf =
  CM.flavor\_symbol\ wf.CA.flavor\ \hat{}
  String.concat "" (List.map p2s (P.to_ints wf.CA.momentum))
module Int = Map.Make (struct type t = int let compare = compare end)
let add\_to\_list i n m =
  Int.add\ i\ (n\ ::\ try\ Int.find\ i\ m\ with\ Not\_found\ 
ightarrow\ [])\ m
let \ classify\_nodes \ dag =
  Int.fold (fun i \ n \ acc \rightarrow (i, \ n) :: acc)
     (CA.D.fold\_nodes (fun \ wf \rightarrow add\_to\_list (P.rank \ wf.CA.momentum) \ wf)
        dag Int.empty) []
let dag_to_dot ch brakets dag =
  Printf.fprintf\ ch\ "digraph_OMEGA_{\cup}\{\n";
  CA.D.iter\_nodes (fun wf \rightarrow
     Printf.fprintf\ ch\ "$\sqcup$\sqcup ``$s\"$\_[$\sqcup$label$\_=$\sqcup\"%s\"$\_];\n"
       (variable \ wf) \ (variable \ wf)) \ dag;
  List.iter (fun (_, wfs) \rightarrow
     Printf.fprintf ch "⊔⊔{⊔rank⊔=⊔same;";
     List.iter (fun n \rightarrow
       Printf.fprintf\ ch\ "

"%s\";" (variable\ n)) wfs;
     Printf.fprintf ch " (classify\_nodes dag);
  List.iter (fun n \rightarrow
     Printf.fprintf\ ch\ "$\""*\"->$\""%s\"; \n"\ (variable\ n))
     (flatten\_keystones\ brakets);
  CA.D.iter (fun n (_, ns) \rightarrow
    \mathsf{let}\ p\ =\ variable\ n\ \mathsf{in}
     PT.iter (fun n' \rightarrow
       Printf.fprintf.ch."
let tower_to_dot ch a =
  dag_to_dot ch a.CA.brakets a.CA.fusion_tower
let \ amplitude\_to\_dot \ ch \ a =
  dag_to_dot ch a.CA.brakets a.CA.fusion_dag
                                                Phase space
let variable \ wf =
  M.flavor\_to\_string\ wf.A.flavor ^
     "[" ^ String.concat "/" (List.map p2s (P.to_ints wf.A.momentum)) ^ "]"
let below_to_channel transform ch dag wf =
  let n2s \ wf = variable \ (transform \ wf)
  and e2s c = "" in
  Tree2.to_channel ch n2s e2s (A.D.dependencies dag wf)
let bra\_to\_channel\ transform\ ch\ dag\ wf\ =
  let tree = A.D.dependencies dag wf in
  if Tree2.is\_singleton\ tree\ then
    let n2s \ wf = variable \ (transform \ wf)
```

```
and e2s c = "" in
     Tree2.to_channel ch n2s e2s tree
  else
     failwith "Fusion.phase_space_channels:\squarewrong\squaretopology!"
let ket_to_channel transform ch dag ket =
  Printf.fprintf ch "(";
  begin match A.children\ ket with
    [] \rightarrow ()
    [child] \rightarrow below\_to\_channel\ transform\ ch\ dag\ child
   child::children \rightarrow
      below\_to\_channel\ transform\ ch\ dag\ child;
      List.iter
        (fun child \rightarrow
           Printf.fprintf ch ",";
           below_to_channel transform ch dag child)
         children
  end;
  Printf.fprintf ch ")"
let phase\_space\_braket\ transform\ ch\ (bra,\ ket)\ dag\ =
  bra_to_channel transform ch dag bra;
  Printf.fprintf \ ch \ ": \sqcup \{";
  begin match ket with
   [] \rightarrow ()
  |[ket1]| \rightarrow
      Printf.fprintf ch "";
      ket_to_channel transform ch dag ket1
  \mid ket1 :: kets \rightarrow
      Printf.fprintf\ ch\ "";
      ket_to_channel transform ch dag ket1;
      List.iter
        (fun k \rightarrow
           ket\_to\_channel\ transform\ ch\ dag\ k)
        kets
  end;
  Printf.fprintf ch "⊔}\n"
let\ phase\_space\_channels\_transformed\ transform\ ch\ a\ =
  List.iter
     (fun \ braket \rightarrow phase\_space\_braket \ transform \ ch \ braket \ a.A.fusion\_dag)
     a.A.brakets
let phase\_space\_channels ch a =
  phase\_space\_channels\_transformed (fun wf \rightarrow wf) ch a
let exchange\_momenta\_list p1 p2 p =
  List.map
     (fun pi \rightarrow
       if pi = p1 then
         p2
       else if pi = p2 then
          p1
       else
          pi)
let exchange\_momenta p1 p2 p =
  P.of\_ints\ (P.dim\ p)\ (exchange\_momenta\_list\ p1\ p2\ (P.to\_ints\ p))
let flip\_momenta \ wf =
  \{ wf \text{ with } A.momentum = exchange\_momenta 1 2 wf.A.momentum \} 
let phase\_space\_channels\_flipped \ ch \ a =
```

```
phase_space_channels_transformed flip_momenta ch a
  end
module Make = Tagged(No\_Tags)
module\ Binary\ =\ Make(Tuple.Binary)(Stat\_Dirac)(Topology.Binary)
module Tagged\_Binary (T : Tagger) =
  Tagged(T)(Tuple.Binary)(Stat\_Dirac)(Topology.Binary)
                                8.2.5 Fusions with Majorana Fermions
let majorana\_log silent logging = logging
let majorana\_log silent logging = silent
let force\_legacy = true
let force\_legacy = false
Model.T: (Stat with type flavor = M.flavor) =
    exception Impossible
    type flavor = M.flavor
                                        Keeping Track of Fermion Lines
JRR's algorithm doesn't use lists of pairs representing directed arrows as in Stat_Dirac().stat above, but a list
of integers denoting the external leg a fermion line connects to:
    type stat =
         Fermion of int \times int list
         AntiFermion of int \times int list
         Boson of int list
         Majorana of int \times int list
    let \ sign\_of\_permutation \ lines = fst \ (Combinatorics.sort\_signed \ lines)
    let lines\_equivalent l1 l2 =
       sign\_of\_permutation\ l1\ =\ sign\_of\_permutation\ l2
    let stat_to_string s =
       let open Printf in
       let \ \mathit{l2s} \ = \ \mathit{ThoList.to\_string \ string\_of\_int} \ in
       \mathsf{match}\ s \ \mathsf{with}
         Boson\ lines \rightarrow sprintf "B\s" (l2s\ lines)
         Fermion (p, lines) \rightarrow sprintf "F(%d, \_%s)" p (l2s lines)
         AntiFermion\ (p,\ lines)\ 	o\ sprintf\ "A(%d, _\%s)"\ p\ (l2s\ lines)
        Majorana (p, lines) \rightarrow sprintf "M(%d, _%s)" p (l2s lines)
Writing all cases explicitly is tedious, but allows exhaustiveness checking.
    let equal \ s1 \ s2 =
       match s1, s2 with
       | Boson l1, Boson l2 \rightarrow
          lines_equivalent l1 l2
         Majorana (p1, l1), Majorana (p2, l2)
         Fermion (p1, l1), Fermion (p2, l2)
         AntiFermion (p1, l1), AntiFermion (p2, l2) \rightarrow
          p1 = p2 \land lines\_equivalent l1 l2
         Boson _, (Fermion _ | AntiFermion _ | Majorana _ )
         (Fermion _ | AntiFermion _ | Majorana _ ), Boson _
         Majorana _, (Fermion _ | AntiFermion _)
         (Fermion \_ | AntiFermion \_), Majorana \_
         Fermion _ , AntiFermion _
```

The final amplitude must not be fermionic!

 $AntiFermion _$, $Fermion _$ \rightarrow false

```
|\ boson\ \_ \to \ true \\ |\ Fermion\ \_ |\ AntiFermion\ \_ |\ Majorana\ \_ \to \ false stat\ f\ p\ interprets\ the\ numeric\ fermion\ numbers\ of\ flavor\ f\ at\ external\ leg\ p\ at\ creates\ a\ leaf: |\ bet\ stat\ f\ p\ = \\ match\ M.fermion\ f\ with \\ |\ 0\ \to\ Boson\ [] \\ |\ 1\ \to\ Fermion\ (p,\ []) \\ |\ -1\ \to\ AntiFermion\ (p,\ []) \\ |\ 2\ \to\ Majorana\ (p,\ []) \\ |\ \_\ \to\ invalid\_arg\ "Fusion.Stat\_Majorana: $\sqcup invalid\_fermion $\sqcup number"$
```

The formalism of [7] does not distinguish spinors from conjugate spinors, it is only important to know in which direction a fermion line is calculated. So the sign is made by the calculation together with an aditional one due to the permuation of the pairs of endpoints of fermion lines in the direction they are calculated. We propose a "canonical" direction from the right to the left child at a fusion point so we only have to keep in mind which external particle hangs at each side. Therefore we need not to have a list of pairs of conjugate spinors and spinors but just a list in which the pairs are right-left-right-left and so on. Unfortunately it is unavoidable to have couplings with clashing arrows in supersymmetric theories so we need transmutations from fermions in antifermions and vice versa as well.

Merge Fermion Lines for Legacy Models with Implied Fermion Connections

In the legacy case with at most one fermion line, it was straight forward to determine the kind of outgoing line from the corresponding flavor. In the general case, it is not possible to maintain this constraint, when constructing the n-ary fusion from binary ones.

We can break up the process into two steps however: first perform unconstrained fusions pairwise ...

let $stat_fuse_pair_unconstrained\ s1\ s2\ =$

```
match s1, s2 with
         Boson l1, Boson l2 \rightarrow Boson (l1 @ l2)
       | (Majorana (p1, l1) | Fermion (p1, l1) | AntiFermion (p1, l1)),
         (Majorana (p2, l2) \mid Fermion (p2, l2) \mid AntiFermion (p2, l2)) \rightarrow
            Boson ([p2; p1] @ l1 @ l2)
         Boson 11, Majorana (p, l2) \rightarrow Majorana (p, l1 @ l2)
         Boson l1, Fermion (p, l2) \rightarrow Fermion (p, l1 @ l2)
         Boson l1, AntiFermion (p, l2) \rightarrow AntiFermion (p, l1 @ l2)
         Majorana (p, l1), Boson l2 \rightarrow Majorana (p, l1 @ l2)
         Fermion (p, l1), Boson l2 \rightarrow Fermion (p, l1 @ l2)
         AntiFermion (p, l1), Boson l2 \rightarrow AntiFermion (p, l1 @ l2)
... and only apply the constraint to the outgoing leg.
    let constrain\_stat\_fusion \ s \ f =
       match s, M.lorentz f with
       | (Majorana (p, l) | Fermion (p, l) | AntiFermion (p, l)),
         (Coupling.Majorana \mid Coupling.Vectorspinor \mid Coupling.Maj\_Ghost) \rightarrow
          Majorana (p, l)
       | (Majorana (p, l) | Fermion (p, l) | AntiFermion (p, l)),
         Coupling.Spinor \rightarrow Fermion (p, l)
       | (Majorana (p, l) | Fermion (p, l) | AntiFermion (p, l)),
         Coupling.ConjSpinor \rightarrow AntiFermion (p, l)
       | (Majorana \perp | Fermion \perp | AntiFermion \perp as s),
         (Coupling.Scalar | Coupling.Vector | Coupling.Massive_Vector
          \mid Coupling.Tensor\_1 \mid Coupling.Tensor\_2 \mid Coupling.BRS\_) \rightarrow
          invalid\_arg
             (Printf.sprintf
                "Fusion.stat_fuse_pair_constrained: |expected|boson, |got| %s"
                (stat\_to\_string\ s))
       \mid Boson \ l \ as \ s,
         (Coupling.Majorana | Coupling.Vectorspinor | Coupling.Maj_Ghost
          | Coupling.Spinor | Coupling.ConjSpinor | \rightarrow
```

```
invalid\_arg
        (Printf.sprintf
           "Fusion.stat\_fuse\_pair\_constrained: \_expected\_fermion, \_got\_\%s"
           (stat\_to\_string\ s))
    Boson l,
    (Coupling.Scalar | Coupling.Vector | Coupling.Massive_Vector
       Coupling.Tensor\_1 \mid Coupling.Tensor\_2 \mid Coupling.BRS\_) \rightarrow
let stat\_fuse\_pair\_legacy f s1 s2 =
  stat_fuse_pair_unconstrained s1 s2
let stat\_fuse\_pair\_legacy\_logging \ f \ s1 \ s2 =
  let stat = stat\_fuse\_pair\_legacy f s1 s2 in
  Printf.eprintf
     (stat\_to\_string \ s1) \ (stat\_to\_string \ s2) \ (stat\_to\_string \ stat)
    (M.flavor\_to\_string \ f);
  stat
let stat_fuse_pair_legacy =
  majorana_log stat_fuse_pair_legacy stat_fuse_pair_legacy_logging
```

Note that we are using $List.fold_left$, therefore we perform the fusions as $f(f(\ldots(f(s_1,s_2),s_3),\ldots),s_n)$. Had we used $List.fold_right$ instead, we would compute $f(s_1,f(s_2,\ldots f(s_{n-1},s_n)))$. For our Dirac algorithm, this makes no difference, but JRR's Majorana algorithm depends on the order!

Also not that we must not apply constrain_stat_fusion here, because stat_fuse_legacy will be used in stat_keystone_legacy again, where we always expect Boson _.

Merge Fermion Lines using Explicit Fermion Connections

We need to match the fermion lines in the incoming propagators using the connection information in the vertex. This used to be trivial in the old omega, because there was at most one fermion line in a vertex.

```
module IMap = Map.Make (struct type t = int let compare = compare end) From version 4.05 on, this is just IMap.find\_opt. let imap\_find\_opt p map = try Some (IMap.find p map) with Not\_found \rightarrow None
```

Partially combined stats of the incoming propagators and keeping track of the fermion lines, while we're scanning them.

```
type partial = \{ stat : stat \ (* the stat accumulated so far *); \\ fermions : int IMap.t \ (* a map from the indices in the vertex to open (anti)fermion lines *); \\ n : int \ (* the number of incoming propagators *) \}
```

We will perform two passes:

1. collect the saturated fermion lines in a *Boson*, while building a map from the indices in the vertex to the open fermion lines

2. connect the open fermion lines using the $int \rightarrow int$ map fermions.

```
{\sf let}\ empty\_partial\ =
        \{ stat = Boson []; 
           fermions = IMap.empty;
           n = 0
Only for debugging:
     let partial\_to\_string p =
        Printf.sprintf
           "\{ \Box fermions = \%s, \Box stat = \%s, \Box # = \%d \Box \}"
           (ThoList.to_string
               (\mathsf{fun}\ (i,\ particle)\ \to\ Printf.sprintf\ \verb"%d@%d"\ particle\ i)
               (IMap.bindings p.fermions))
           (stat\_to\_string\ p.stat)
```

Add a list of saturated fermion lines at the top of the list of lines in a stat.

```
let \ add\_lines \ l = function
    Boson l' \rightarrow Boson (l @ l')
    Fermion (n, l') \rightarrow Fermion (n, l @ l')
     AntiFermion (n, l') \rightarrow AntiFermion (n, l @ l')
    Majorana (n, l') \rightarrow Majorana (n, l @ l')
```

Process one line in the first pass: add the saturated fermion lines to the partial stat p.stat and add a pointer to an open fermion line in case of a fermion.

```
let \ add\_lines\_to\_partial \ p \ stat =
  let n = succ p.n in
  match stat with
  \mid Boson \ l \rightarrow
      \{ fermions = p.fermions; \}
         stat = add\_lines \ l \ p.stat;
         n
  \mid Majorana (f, l) \rightarrow
      \{ fermions = IMap.add \ n \ f \ p.fermions; \}
         stat = add\_lines \ l \ p.stat;
         n
  | Fermion (p, l) \rightarrow
      invalid\_arg
         "add_lines_to_partial:_unexpected_Fermion"
  \mid AntiFermion(p, l) \rightarrow
      invalid\_arg
         "add_lines_to_partial:_unexpected_AntiFermion"
let partial\_of\_slist stat\_list =
```

Do it for all lines:

```
List.fold\_left\ add\_lines\_to\_partial\ empty\_partial\ stat\_list
let partial\_of\_rev\_slist stat\_list =
   List.fold_left add_lines_to_partial empty_partial (List.rev stat_list)
```

The building blocks for a single step of the second pass: saturate a fermion line or pass it through. The indices i and j refer to incoming lines: add a saturated line to p.stat and remove the corresponding open

lines from the map.

```
let saturate\_fermion\_line p i j =
  match imap_find_opt i p.fermions, imap_find_opt j p.fermions with
  | Some f, Some f' \rightarrow
      \{ stat = add\_lines [f'; f] p.stat; \}
        fermions = IMap.remove i (IMap.remove j p.fermions);
        n = p.n
  \mid Some \_, None \rightarrow
      invalid\_arg "saturate_fermion_line:\sqcupno\sqcupopen\sqcupoutgoing\sqcupfermion\sqcupline"
```

```
\mid None, Some \rightarrow
           invalid_arq "saturate_fermion_line: | no | open | incoming | fermion | line"
       | None, None \rightarrow
           invalid\_arg "saturate_fermion_line:\Boxno\Boxopen\Boxfermion\Boxlines"
The index i refers to an incoming line: add the open line to p. stat and remove it from the map.
     let pass_through_fermion_line p i =
       match \ imap\_find\_opt \ i \ p.fermions, \ p.stat \ with
       \mid Some f, Boson l \rightarrow
           \{ stat = Majorana (f, l); \}
             fermions = IMap.remove i p.fermions;
              n = p.n
       | Some \_, (Majorana \_ | Fermion \_ | AntiFermion \_) \rightarrow
           invalid\_arg "pass_through_fermion_line:\_more\_than\_one\_open\_line"
       | None, \_ \rightarrow
           invalid\_arg "pass_through_fermion_line:_\(\text{expected}\)\(\text{fermion}\(\text{Lond}\)\(\text{bund}'')
Ignoring the direction of the fermion line reproduces JRR's algorithm.
     let sort_pair(i, j) =
       if i < j then
         (i, j)
       else
          (j, i)
The index p.n + 1 corresponds to the outgoing line:
    let is\_incoming p i =
       i \leq p.n
     let match\_fermion\_line\ p\ (i,\ j)\ =
       let i, j = sort_pair(i, j) in
       if is\_incoming p i \land is\_incoming p j then
          saturate_fermion_line p i j
       else if is\_incoming p i then
          pass\_through\_fermion\_line\ p\ i
       else if is\_incoming p j then
          pass_through_fermion_line p j
       else
          failwith "match_fermion_line:_\u00cdboth\u00cdlines\u00cdotngumuoutgoing"
     let match\_fermion\_line\_logging \ p \ (i, j) =
       Printf.eprintf
          "match\_fermion\_line_{\sqcup \sqcup \sqcup \sqcup \sqcup} \%s_{\sqcup} [\%d->\%d]"
          (partial\_to\_string \ p) \ i \ j;
       let p' = match\_fermion\_line p(i, j) in
       Printf.eprintf "_{\sqcup} >>_{\sqcup} %s \ (partial\_to\_string p');
       p'
    let match\_fermion\_line =
       majorana_log match_fermion_line match_fermion_line_logging
Combine the passes ...
     let match\_fermion\_lines\ flines\ s1\ s23\_\_n\ =
       List.fold\_left\ match\_fermion\_line\ (partial\_of\_slist\ (s1\ ::\ s23\_\_n))\ flines
... and keep only the stat.
     let stat\_fuse\_new flines s1 s23\_\_n \_
       (match\_fermion\_lines\ flines\ s1\ s23\_\_n).stat
If there is at most a single fermion line, we can compare stat against the result of stat_fuse_legacy for checking
stat_fuse_new (admittedly, this case is rather trivial) ...
     let stat\_fuse\_new\_check stat flines s1 s23\_\_n f
       if List.length\ flines\ <\ 2 then
          begin
```

```
let legacy = stat\_fuse\_legacy s1 s23\_\_n f in
           if \neg (equal stat legacy) then
              failwith
                (Printf.sprintf
                    (stat\_to\_string\ stat)
                    (stat\_to\_string\ legacy))
         end
... do it, but only when we are writing debugging output.
    let stat\_fuse\_new\_logging\ flines\ s1\ s23\_\_n\ f\ =
      let stat = stat\_fuse\_new flines s1 s23\_\_n f in
       Printf.eprintf
         (UFO_Lorentz.fermion_lines_to_string flines)
         (ThoList.to\_string\ stat\_to\_string\ (s1\ ::\ s23\_\_n))
         (stat\_to\_string\ stat)
         (M.flavor\_to\_string \ f);
       stat\_fuse\_new\_check stat flines s1 s23\_\_n f;
      stat
    let stat_fuse_new =
       majorana_log stat_fuse_new stat_fuse_new_logging
Use stat\_fuse\_new, whenever fermion connections are available. NB: Some [] is not the same as None!
    let stat_fuse flines_opt slist f =
      \mathsf{match}\ \mathit{slist}\ \mathsf{with}
        [] 
ightarrow invalid\_arg "stat_fuse:\_empty"
       | s1 :: s23\_n \rightarrow
          constrain\_stat\_fusion
             (match flines_opt with
                Some\ flines\ 	o\ stat\_fuse\_new\ flines\ s1\ s23\_\_n\ f
                None \rightarrow stat\_fuse\_legacy s1 s23\_\_n f)
    let stat\_fuse\_logging\ flines\_opt\ slist\ f\ =
      let stat = stat\_fuse flines\_opt slist f in
       Printf.eprintf
         (ThoList.to_string stat_to_string slist)
         (stat\_to\_string\ stat)
         (M.flavor\_to\_string \ f);
       stat
    let stat_fuse =
       majorana_log stat_fuse stat_fuse_logging
                                 Final Step using Implied Fermion Connections
    let stat\_keystone\_legacy \ s1 \ s23\_\_n \ f =
       stat_fuse_legacy s1 s23__n f
    let stat\_keystone\_legacy\_logging \ s1 \ s23\_\_n \ f =
      let s = stat\_keystone\_legacy s1 s23\_\_n f in
       Printf.eprintf
         "stat_keystone_legacy:_{\square}%s_{\square}(%s)_{\square}%s_{\square}->_{\square}%s\n"
         (stat\_to\_string \ s1)
         (M.flavor\_to\_string \ f)
         (ThoList.to\_string\ stat\_to\_string\ s23\_\_n)
         (stat\_to\_string\ s);
    let stat_keystone_legacy =
       majorana_log stat_keystone_legacy stat_keystone_legacy_logging
```

Final Step using Explicit Fermion Connections

```
let stat_keystone_new flines slist f =
        \mathsf{match}\ \mathit{slist}\ \mathsf{with}
          [] \rightarrow invalid\_arg "stat_keystone:\sqcupempty"
          [s] \rightarrow invalid\_arg "stat_keystone:\sqcupsingleton"
        | s1 :: s2 :: s34 \_\_n \rightarrow
           let stat =
              stat\_fuse\_pair\_unconstrained\ s1\ (stat\_fuse\_new\ flines\ s2\ s34\_\_n\ f) in
            if saturated stat then
              stat
            else
              failwith
                 (Printf.sprintf
                     "stat_keystone: uincomplete u%s!"
                     (stat\_to\_string\ stat))
     let stat_keystone_new_check stat slist f =
        match slist with
         [] \rightarrow invalid\_arg "stat_keystone_check:_\muempty"
        | s1 :: s23 \_n \rightarrow
            let legacy = stat\_keystone\_legacy s1 s23\_\_n f in
            if \neg (equal \ stat \ legacy) then
              failwith
                 (Printf.sprintf
                     "stat_keystone_check:_{\sqcup}%s_{\sqcup}<>_{\sqcup}%s!"
                     (stat\_to\_string\ stat)
                     (stat\_to\_string\ legacy))
     let stat\_keystone\ flines\_opt\ slist\ f\ =
        match flines\_opt with
          Some\ flines\ 	o\ stat\_keystone\_new\ flines\ slist\ f
         None \rightarrow
            begin match slist with
            [] \rightarrow invalid\_arg "stat_keystone: \_empty"
            s1 :: s23\_n \rightarrow stat\_keystone\_legacy s1 s23\_n f
            end
     let stat\_keystone\_logging\ flines\_opt\ slist\ f\ =
        \mathsf{let}\ stat\ =\ stat\_keystone\ flines\_opt\ slist\ f\ \mathsf{in}
        Printf.eprintf
           "stat_keystone:_{\cup\cup\cup\cup\cup\cup\cup\cup}%s_{\cup}(%s)_{\cup}%s_{\cup}->_{\cup}%s\n"
          (stat\_to\_string (List.hd slist))
           (M.flavor\_to\_string \ f)
          (ThoList.to\_string\ stat\_to\_string\ (List.tl\ slist))
          (stat\_to\_string\ stat);
        stat\_keystone\_new\_check \ stat \ slist \ f;
        stat
     let stat_k eystone =
        majorana_log stat_keystone stat_keystone_logging
Force the legacy version w/o checking against the new implementation for comparing generated code against
the hard coded models:
     let stat\_fuse flines\_opt slist f =
        if force\_legacy then
          stat_fuse_legacy (List.hd slist) (List.tl slist) f
        else
          stat_fuse flines_opt slist f
     let stat\_keystone\ flines\_opt\ slist\ f\ =
        if force\_legacy then
           stat\_keystone\_legacy (List.hd slist) (List.tl slist) f
        else
```

$stat_keystone\ flines_opt\ slist\ f$

Evaluate Signs from Fermion Permuations

```
let stat\_sign = function
        Boson\ lines\ 	o\ sign\_of\_permutation\ lines
        Fermion(p, lines) \rightarrow sign\_of\_permutation(p :: lines)
        AntiFermion\ (pbar,\ lines) \rightarrow sign\_of\_permutation\ (pbar\ ::\ lines)
        Majorana (pm, lines) \rightarrow sign\_of\_permutation (pm :: lines)
    let stat\_sign\_logging stat =
      let sign = stat\_sign stat in
       Printf.eprintf
         (stat\_to\_string\ stat)\ sign;
      sign
    \mathsf{let}\ stat\_sign\ =
       majorana_log stat_sign stat_sign_logging
  end
module Binary\_Majorana =
  Make(Tuple.Binary)(Stat\_Majorana)(Topology.Binary)
module Nary (B : Tuple.Bound) =
  Make(Tuple.Nary(B))(Stat\_Dirac)(Topology.Nary(B))
module\ Nary\_Majorana\ (B:\ Tuple.Bound) =
  Make(Tuple.Nary(B))(Stat\_Majorana)(Topology.Nary(B))
module Mixed23 =
  Make(Tuple.Mixed23)(Stat\_Dirac)(Topology.Mixed23)
module Mixed23\_Majorana =
  Make(Tuple.Mixed23)(Stat\_Majorana)(Topology.Mixed23)
module \ Helac \ (B: Tuple.Bound) =
  Make(Tuple.Nary(B))(Stat\_Dirac)(Topology.Helac(B))
module\ Helac\_Majorana\ (B:\ Tuple.Bound) =
  Make(Tuple.Nary(B))(Stat\_Majorana)(Topology.Helac(B))
module B2 = \text{struct let } max\_arity () = 2 \text{ end}
module B3 = \text{struct let } max\_arity () = 3 \text{ end}
module Helac\_Binary = Helac(B2)
module\ Helac\_Binary\_Majorana\ =\ Helac(B2)
module\ Helac\_Mixed23 = Helac(B3)
module\ Helac\_Mixed23\_Majorana\ =\ Helac(B3)
                                      8.2.6 Multiple Amplitudes
module type Multi =
```

```
odule type Multi = sig

exception Mismatch

val options : Options.t

type flavor

type process = flavor \ list \times flavor \ list

type amplitude

type flavor

type flavor
```

```
val empty : amplitudes
    val\ flavors: amplitudes \rightarrow process\ list
    val\ vanishing\_flavors\ :\ amplitudes\ 	o\ process\ list
    val\ color\_flows\ :\ amplitudes\ 	o\ Color.Flow.t\ list
    val\ helicities: amplitudes \rightarrow (int\ list \times int\ list)\ list
    val\ processes\ :\ amplitudes\ 	o\ amplitude\ list
    val\ process\_table\ :\ amplitudes\ 	o\ amplitude\ option\ array\ array
    \mathsf{val}\ \mathit{fusions}\ :\ \mathit{amplitudes}\ \to\ (\mathit{fusion}\ \times\ \mathit{amplitude})\ \mathit{list}
    val multiplicity: amplitudes \rightarrow wf \rightarrow int
    \mathsf{val}\ dictionary\ :\ amplitudes\ 	o\ amplitude\ 	o\ wf\ 	o\ int
    val\ color\_factors\ :\ amplitudes\ 	o\ Color.Flow.factor\ array\ array
    val\ constraints\ :\ amplitudes\ 	o\ string\ option
  end
module type Multi\_Maker = functor (Fusion\_Maker : Maker) \rightarrow
  functor (P : Momentum. T) \rightarrow
    functor (M : Model.T) \rightarrow
       Multi with type flavor = M.flavor
       and type amplitude = Fusion\_Maker(P)(M).amplitude
       and type fusion = Fusion\_Maker(P)(M).fusion
       and type wf = Fusion\_Maker(P)(M).wf
       and type selectors = Fusion\_Maker(P)(M).selectors
module\ Multi\ (Fusion\_Maker\ :\ Maker)\ (P\ :\ Momentum.T)\ (M\ :\ Model.T)\ =
  struct
    exception Mismatch
    type progress\_mode =
         Quiet
         Channel of out_channel
         File of string
    let progress\_option = ref Quiet
    module CM = Colorize.It(M)
    module F = Fusion\_Maker(P)(M)
    module C = Cascade.Make(M)(P)
   A kludge, at best ...
    let \ options = Options.extend \ F.options
         [ "progress", Arg.Unit (fun () \rightarrow progress_option := Channel stderr),
            "report_progress_to_the_standard_error_stream";
            "progress_file", Arg.String (fun s \rightarrow progress\_option := File s),
            "report_progress_to_a_file"]
    type flavor = M.flavor
    type p = F.p
    type process = flavor \ list \times flavor \ list
    type amplitude = F.amplitude
    type fusion = F.fusion
    type wf = F.wf
    type exclusions = F.exclusions
    let no\_exclusions = F.no\_exclusions
    type selectors = F.selectors
    type flavors = flavor \ list \ array
    type helicities = int list array
    type \ colors = Color.Flow.t \ array
    type amplitudes' = amplitude array array array
    type amplitudes =
         { flavors : process list;
            vanishing_flavors : process list;
```

```
color_flows : Color.Flow.t list;
            helicities: (int list \times int list) list;
            processes: amplitude list;
            process_table : amplitude option array array;
            fusions : (fusion \times amplitude) \ list;
            multiplicity: (wf \rightarrow int);
            dictionary: (amplitude \rightarrow wf \rightarrow int);
            color_factors : Color.Flow.factor array array;
            constraints : string option }
    let flavors a = a.flavors
    let \ vanishing\_flavors \ a = a.vanishing\_flavors
    let \ color\_flows \ a \ = \ a.color\_flows
    let helicities \ a = a.helicities
    let processes a = a.processes
    let process\_table a = a.process\_table
    let fusions \ a = a.fusions
    let multiplicity \ a = a.multiplicity
    let dictionary a = a.dictionary
    let color\_factors \ a = a.color\_factors
    let constraints a = a.constraints
    let sans\_colors f =
       List.map CM.flavor_sans_color f
    let colors (fin, fout) =
       List.map M.color (fin @ fout)
    let process\_sans\_color a =
       (sans\_colors\ (F.incoming\ a),\ sans\_colors\ (F.outgoing\ a))
    let \ color flow \ a =
       CM.flow (F.incoming a) (F.outgoing a)
    let process_to_string fin fout =
       String.concat ", " (List.map M.flavor_to_string fin)
         " \_ -> \_ " ^ String.concat " \_ " (List.map M.flavor_to_string fout)
    let count_processes colored_processes =
       List.length colored_processes
    module FMap =
       Map.Make (struct type t = process let compare = compare end)
       Map.Make (struct type t = Color.Flow.t let compare = compare end)
Recently Product.list began to guarantee lexicographic order for sorted arguments. Anyway, we still force a
lexicographic order.
    let rec order\_spin\_table1 \ s1 \ s2 =
       match s1, s2 with
       \mid h1 :: t1, h2 :: t2 \rightarrow
           let c = compare \ h1 \ h2 in
            if c \neq 0 then
              c
            else
              order_spin_table1 t1 t2
       | [], [] \rightarrow 0
       \bot \rightarrow invalid\_arg "order_spin_table:\sqcupinconsistent\sqcuplengths"
    let order\_spin\_table\ (s1\_in,\ s1\_out)\ (s2\_in,\ s2\_out)\ =
       let c = compare s1\_in s2\_in in
       if c \neq 0 then
         c
       else
         order_spin_table1 s1_out s2_out
```

```
\begin{array}{lll} \text{let } sort\_spin\_table \ table \ = \\ List.sort \ order\_spin\_table \ table \\ \\ \text{let } id \ x \ = \ x \\ \\ \text{let } pair \ x \ y \ = \ (x, \ y) \end{array}
```



Improve support for on shell Ward identities: $Coupling.Vector \rightarrow [4]$ for one and only one external vector.

```
let rec hs\_of\_lorentz = function
    Coupling.Scalar \rightarrow [0]
    Coupling.Spinor \mid Coupling.ConjSpinor
    Coupling.Majorana \mid Coupling.Maj\_Ghost \rightarrow [-1; 1]
    Coupling. Vector \rightarrow [-1; 1]
    Coupling.Massive\_Vector \rightarrow [-1; 0; 1]
    Coupling. Tensor_1 \rightarrow [-1; 0; 1]
    Coupling. Vectorspinor \rightarrow [-2; -1; 1; 2]
     Coupling.Tensor_2 \rightarrow [-2; -1; 0; 1; 2]
    Coupling.BRS\ f \rightarrow hs\_of\_lorentz\ f
let hs\_of\_flavor\ f =
  hs\_of\_lorentz (M.lorentz f)
let hs\_of\_flavors (fin, fout) =
  (List.map hs_of_flavor fin, List.map hs_of_flavor fout)
let rec unphysical\_of\_lorentz = function
    Coupling. Vector \rightarrow [4]
    Coupling.Massive\_Vector \rightarrow [4]
  | \_ \rightarrow invalid\_arg "unphysical_of_lorentz:\_not\_a\_vector\_particle"
let unphysical\_of\_flavor f =
  unphysical\_of\_lorentz\ (M.lorentz\ f)
let unphysical\_of\_flavors1 \ n \ f\_list =
  Tho List.mapi
     (fun i f \rightarrow if i = n then unphysical_of_flavor f else hs_of_flavor f)
     1 f_{-}list
let unphysical\_of\_flavors\ n\ (fin,\ fout) =
  (unphysical\_of\_flavors1 \ n \ fin, \ unphysical\_of\_flavors1 \ (n - List.length \ fin) \ fout)
let helicity_table unphysical flavors =
  let hs =
     begin match unphysical with
      None \rightarrow List.map \ hs\_of\_flavors \ flavors
     | Some n \rightarrow List.map (unphysical\_of\_flavors n) flavors
     end in
  if \neg (ThoList.homogeneous\ hs) then
     invalid\_arg "Fusion.helicity_table:\_not\_all\_flavors\_have\_the\_same\_helicity\_states!"
     match hs with
     | [] \rightarrow []
     | (hs_in, hs_out) :: \_ \rightarrow
         sort_spin_table (Product.list2 pair (Product.list id hs_in) (Product.list id hs_out))
module\ Proc = Process.Make(M)
module WFMap = Map.Make (struct type t = F.wf let compare = compare end)
module WFSet2 =
  Set.Make (struct type t = F.wf \times (F.wf, F.coupling) Tree2.t let compare = compare end)
module WFMap2 =
  Map.Make (struct type t = F.wf \times (F.wf, F.coupling) Tree2.t let compare = compare end)
module WFTSet =
  Set.Make (struct type t = (F.wf, F.coupling) Tree2.t let compare = compare end)
```

All wavefunctions are unique per amplitude. So we can use per-amplitude dependency trees without additional *internal* tags to identify identical wave functions.

NB: we miss potential optimizations, because we assume all coupling to be different, while in fact we have horizontal/family symmetries and non abelian gauge couplings are universal anyway.

```
{\tt let} \ \textit{disambiguate\_fusions} \ \textit{amplitudes} \ = \\
  let fusions =
     ThoList.flatmap (fun amplitude \rightarrow
       List.map
          (fun\ fusion\ 	o\ (fusion,\ F.dependencies\ amplitude\ (F.lhs\ fusion)))
          (F.fusions\ amplitude))
       amplitudes in
  let duplicates =
     List.fold\_left
       (fun map\ (fusion,\ dependencies) \rightarrow
         let wf = F.lhs fusion in
         let set = try WFMap.find wf map with Not_found <math>\rightarrow WFTSet.empty in
          WFMap.add wf (WFTSet.add dependencies set) map)
       WFMap.empty fusions in
  let multiplicity\_map =
     WFMap.fold (fun wf dependencies acc \rightarrow
       let cardinal = WFTSet.cardinal \ dependencies in
       if cardinal \leq 1 then
          acc
       else
          WFMap.add wf cardinal acc)
       duplicates WFMap.empty
  and dictionary\_map =
     WFMap.fold (fun wf dependencies acc \rightarrow
       let \ cardinal = \ WFTSet.cardinal \ dependencies \ in
       if cardinal \leq 1 then
          acc
       else
         snd\ (\mathit{WFTSet.fold}
                  (fun dependency (i', acc') \rightarrow
                    (succ\ i',\ WFMap2.add\ (wf,\ dependency)\ i'\ acc'))
                  dependencies (1, acc)))
       duplicates WFMap2.empty in
  let multiplicity \ wf =
     WFMap.find wf multiplicity_map
  and dictionary amplitude wf =
     WFMap2.find (wf, F.dependencies amplitude wf) dictionary_map in
  (multiplicity, dictionary)
let \ eliminate\_common\_fusions1 \ seen\_wfs \ amplitude =
  List.fold\_left
     (fun (seen, acc) f \rightarrow
       let wf = F.lhs f in
       let dependencies = F.dependencies amplitude wf in
       if WFSet2.mem (wf, dependencies) seen then
         (seen, acc)
       else
         (WFSet2.add\ (wf,\ dependencies)\ seen,\ (f,\ amplitude)\ ::\ acc))
    seen\_wfs (F.fusions amplitude)
let eliminate_common_fusions processes =
  let _-, rev_fusions =
    List.fold\_left
       eliminate\_common\_fusions1
       (WFSet2.empty, []) processes in
  List.rev\ rev\_fusions
```

Calculate All The Amplitudes

 $let \ amplitudes \ goldstones \ unphysical \ exclusions \ select_wf \ processes =$



Eventually, we might want to support inhomogeneous helicities. However, this makes little physics sense for external particles on the mass shell, unless we have a model with degenerate massive fermions and bosons.

```
if \neg (ThoList.homogeneous (List.map hs\_of\_flavors processes)) then
  invalid_arg "Fusion.Multi.amplitudes: uincompatible uhelicities";
let unique\_uncolored\_processes =
  Proc.remove_duplicate_final_states (C.partition select_wf) processes in
let progress =
  match !progress_option with
    Quiet \rightarrow Progress.dummy
    Channel\ oc\ 	o \ Progress.channel\ oc\ (count\_processes\ unique\_uncolored\_processes)
    \mathit{File}\ \mathit{name}\ 	o\ \mathit{Progress.file}\ \mathit{name}\ (\mathit{count\_processes}\ \mathit{unique\_uncolored\_processes})\ \mathsf{in}
let allowed =
   ThoList.flatmap
     (\text{fun } (fi, fo) \rightarrow
       Progress.begin_step progress (process_to_string fi fo);
       let amps = F.amplitudes goldstones exclusions select_wf fi fo in
       begin match amps with
       | [] → Progress.end_step progress "forbidden"
       \mid \rightarrow Progress.end\_step\ progress "allowed"
       end:
       amps) unique_uncolored_processes in
Progress.summary progress "all_processes_done";
let color\_flows =
   ThoList.uniq (List.sort compare (List.map color_flow allowed))
and flavors =
   ThoList.uniq (List.sort compare (List.map process_sans_color allowed)) in
let vanishing\_flavors =
  Proc.diff processes flavors in
let \ helicities =
  helicity_table unphysical flavors in
let f_index =
  fst (List.fold_left
           (\text{fun } (m, i) f \rightarrow (FMap.add f i m, succ i))
           (FMap.empty, 0) flavors)
and c\_index =
  fst (List.fold\_left
           (fun (m, i) c \rightarrow (CMap.add c i m, succ i))
           (CMap.empty, 0) color_flows) in
let table =
  Array.make_matrix (List.length flavors) (List.length color_flows) None in
List.iter
  (fun a \rightarrow
    let f = FMap.find (process\_sans\_color a) f\_index
     and c = CMap.find (color\_flow a) c\_index in
     table.(f).(c) \leftarrow Some(a)
  allowed;
let cf\_array = Array.of\_list color\_flows in
let ncf = Array.length \ cf\_array in
let color_factor_table = Array.make_matrix ncf ncf Color.Flow.zero in
\quad \text{for } i \ = \ 0 \ \text{to} \ \mathit{pred} \ \mathit{ncf} \ \text{do}
```

```
for j = 0 to i do
       color\_factor\_table.(i).(j) \leftarrow
         Color.Flow.factor\ cf\_array.(i)\ cf\_array.(j);
       color\_factor\_table.(j).(i) \leftarrow
         color\_factor\_table.(i).(j)
    done
  done;
  let \ fusions = eliminate\_common\_fusions \ allowed
  and multiplicity, dictionary = disambiguate_fusions allowed in
  \{ flavors = flavors; \}
     vanishing\_flavors = vanishing\_flavors;
     color\_flows = color\_flows;
     helicities = helicities;
    processes = allowed;
     process\_table = table;
    fusions = fusions;
     multiplicity = multiplicity;
     dictionary = dictionary;
     color\_factors = color\_factor\_table;
     constraints = C.description select\_wf }
let empty =
  \{ flavors = [];
     vanishing\_flavors = [];
     color\_flows = [];
     helicities = [];
    processes = [];
     process_table = Array.make_matrix 0 0 None;
    fusions = [];
     multiplicity = (fun _ \rightarrow 1);
     dictionary = (fun \_ \_ \rightarrow 1);
     color\_factors = Array.make\_matrix \ 0 \ 0 \ Color.Flow.zero;
     constraints = None
```

end

—9—

LORENTZ REPRESENTATIONS, COUPLINGS, MODELS AND TARGETS

9.1 Interface of Coupling

The enumeration types used for communication from *Models* to *Targets*. On the physics side, the modules in *Models* must implement the Feynman rules according to the conventions set up here. On the numerics side, the modules in *Targets* must handle all cases according to the same conventions.

9.1.1 Propagators

The Lorentz representation of the particle. NB: O'Mega treats all lines as *outgoing* and particles are therefore transforming as *ConjSpinor* and antiparticles as *Spinor*.

```
\label{eq:content} \begin{array}{l} \operatorname{type}\ lorentz = \\ \mid \ Scalar \\ \mid \ Spinor\ (*\ \psi\ *) \\ \mid \ ConjSpinor\ (*\ \psi\ *) \\ \mid \ Majorana\ (*\ \chi\ *) \\ \mid \ Maj\_Ghost\ (*\ SUSY\ ghosts\ *) \\ \mid \ Vector \\ \mid \ Massive\_Vector \\ \mid \ Vectorspinor\ (*\ supersymmetric\ currents\ and\ gravitinos\ *) \\ \mid \ Tensor\_1 \\ \mid \ Tensor\_2\ (*\ massive\ gravitons\ (large\ extra\ dimensions)\ *) \\ \mid \ BRS\ of\ lorentz \\ \text{type}\ lorentz3 = lorentz\ \times\ lorentz\ \times\ lorentz\ \times\ lorentz\ \\ \text{type}\ lorentz4 = lorentz\ \times\ lorentz\ \times\ lorentz\ \times\ lorentz\ \\ \text{type}\ lorentzn = lorentz\ list \\ \text{type}\ fermion\_lines} = \ (int\ \times\ int)\ list \\ \end{array}
```

If there were no vectors or auxiliary fields, we could deduce the propagator from the Lorentz representation. While we're at it, we can introduce "propagators" for the contact interactions of auxiliary fields as well. $Prop_Gauge$ and $Prop_Feynman$ are redundant as special cases of $Prop_Rxi$.

The special case Only_Insertion corresponds to operator insertions that do not correspond to a propagating field all. These are used for checking Slavnov-Taylor identities

$$\partial_{\mu} \langle \text{out} | W^{\mu}(x) | \text{in} \rangle = m_W \langle \text{out} | \phi(x) | \text{in} \rangle$$
 (9.1)

of gauge theories in unitarity gauge where the Goldstone bosons are not propagating. Numerically, it would suffice to use a vanishing propagator, but then superflous fusions would be calculated in production code in which the Slavnov-Taylor identities are not tested.

	only Dirac fermions	incl. Majorana fermions	
Prop_Scalar	$\phi(p) \leftarrow \frac{\mathrm{i}}{p^2 - m^2 + \mathrm{i}m\Gamma}\phi(p)$		
Prop_Spinor	$\psi(p) \leftarrow \frac{\mathrm{i}(-\not p + m)}{p^2 - m^2 + \mathrm{i}m\Gamma} \psi(p)$	$\psi(p) \leftarrow \frac{\mathrm{i}(-\not p + m)}{p^2 - m^2 + \mathrm{i}m\Gamma} \psi(p)$	
Prop_ConjSpinor	$\bar{\psi}(p) \leftarrow \bar{\psi}(p) \frac{\mathrm{i}(\not p + m)}{p^2 - m^2 + \mathrm{i} m \Gamma}$	$\psi(p) \leftarrow \frac{\mathrm{i}(-\not p + m)}{p^2 - m^2 + \mathrm{i}m\Gamma} \psi(p)$	
Prop_Majorana	N/A	$\chi(p) \leftarrow \frac{\mathrm{i}(-\not p + m)}{p^2 - m^2 + \mathrm{i}m\Gamma}\chi(p)$	
$Prop_Unitarity$	$\epsilon_{\mu}(p) \leftarrow \frac{\mathrm{i}}{p^2 - m^2 + \mathrm{i}m\Gamma} \left(-g_{\mu\nu} + \frac{p_{\mu}p_{\nu}}{m^2} \right) \epsilon^{\nu}(p)$		
Prop_Feynman	$\epsilon^{\nu}(p) \leftarrow \frac{-\mathrm{i}}{p^2 - m^2 + \mathrm{i}m\Gamma} \epsilon^{\nu}(p)$		
Prop_Gauge	$\epsilon_{\mu}(p) \leftarrow \frac{\mathrm{i}}{p^2} \left(-g_{\mu\nu} + (1-\xi) \frac{p_{\mu}p_{\nu}}{p^2} \right) \epsilon^{\nu}(p)$		
Prop_Rxi	$\epsilon_{\mu}(p) \leftarrow \frac{\mathrm{i}}{p^2 - m^2 + \mathrm{i}m\Gamma} \left(-g_{\mu\nu} + (1 - \xi) \frac{p_{\mu}p_{\nu}}{p^2 - \xi m^2} \right) \epsilon^{\nu}(p)$		

Table 9.1: Propagators. NB: The sign of the momenta in the spinor propagators comes about because O'Mega treats all momenta as *outgoing* and the charge flow for *Spinor* is therefore opposite to the momentum, while the charge flow for *ConjSpinor* is parallel to the momentum.

Aux_Scalar	$\phi(p) \leftarrow \mathrm{i}\phi(p)$
Aux_Spinor	$\psi(p) \leftarrow \mathrm{i} \psi(p)$
$Aux_ConjSpinor$	$\bar{\psi}(p) \leftarrow i\bar{\psi}(p)$
Aux_Vector	$\epsilon^{\mu}(p) \leftarrow i\epsilon^{\mu}(p)$
Aux_Tensor_1	$T^{\mu\nu}(p) \leftarrow iT^{\mu\nu}(p)$
$Only_Insertion$	N/A

Table 9.2: Auxiliary and non propagating fields

```
Prop_Col_Unitarity
Aux_Scalar | Aux_Vector | Aux_Tensor_1
Aux_Col_Scalar | Aux_Col_Vector | Aux_Col_Tensor_1
Aux_Spinor | Aux_ConjSpinor | Aux_Majorana
Only_Insertion
Prop_UFO of string
```



JR sez' (regarding the Majorana Feynman rules): We don't need different fermionic propagators as supposed by the variable names $Prop_Spinor$, $Prop_ConjSpinor$ or $Prop_Majorana$. The propagator in all cases has to be multiplied on the left hand side of the spinor out of which a new one should be built. All momenta are treated as outgoing, so for the propagation of the different fermions the following table arises, in which the momentum direction is always downwards and the arrows show whether the momentum and the fermion line, respectively are parallel or antiparallel to the direction of calculation:

Fermion type	fermion arrow	mom.	calc.	sign
Dirac fermion	†	↑ ↓	↑ ↑	negative
Dirac antifermion	↓	+ +	\uparrow \downarrow	negative
Majorana fermion	-	\uparrow \downarrow	-	negative

So the sign of the momentum is always negative and no further distinction is needed. (JR's probably right, but I need to check myself...)

```
\begin{tabular}{ll} type \ width = & | \ Vanishing \\ | \ Constant \\ | \ Timelike \\ | \ Running \\ | \ Fudged \\ | \ Complex\_Mass \\ | \ Custom \ of \ string \\ \end{tabular}
```

9.1.2 Vertices

The combined S-P and V-A couplings (see tables 9.5, 9.6, 9.8 and 9.12) are redundant, of course, but they allow some targets to create more efficient numerical code.¹ Choosing VA2 over VA will cause the FORTRAN backend to pass the coupling as a whole array

The integer is an additional coefficient that multiplies the respective coupling constant. This allows to reduce the number of required coupling constants in manifestly symmetric cases. Most of times it will be equal unity, though.

The two vertex types PBP and BBB for the couplings of two fermions or two antifermions ("clashing arrows") is unavoidable in supersymmetric theories.



... the doesn't like the names and has promised to find a better mnemonics!

```
type \alpha \ vertex3 =
 | FBF \ of \ int \times fermionbar \ \times \ boson \ \times fermion 
 | PBP \ of \ int \times fermion \ \times \ boson \ \times \ fermion 
 | BBB \ of \ int \times fermionbar \ \times \ boson \ \times \ fermionbar 
 | GBG \ of \ int \times fermionbar \ \times \ boson \ \times \ fermion \ (* \ gravitino-boson-fermion *)
```

¹An additional benefit is that the counting of Feynman diagrams is not upset by a splitting of the vectorial and axial pieces of gauge bosons.

```
Gauge_Gauge_Gauge of int | Aux_Gauge_Gauge of int
I\_Gauge\_Gauge\_Gauge of int
Scalar_Vector_Vector of int
Aux\_Vector\_Vector of int \mid Aux\_Scalar\_Vector of int
Scalar\_Scalar\_Scalar of int \mid Aux\_Scalar\_Scalar of int
 Vector_Scalar_Scalar of int
Graviton_Scalar_Scalar of int
Graviton_Vector_Vector of int
Graviton_Spinor_Spinor of int
Dim4_Vector_Vector_Vector_T of int
Dim4\_Vector\_Vector\_Vector\_L of int
Dim4\_Vector\_Vector\_Vector\_T5 of int
Dim4_Vector_Vector_L5 of int
Dim6_Gauge_Gauge_Gauge of int
Dim6_Gauge_Gauge_5 of int
Aux\_DScalar\_DScalar of int \mid Aux\_Vector\_DScalar of int
Dim5\_Scalar\_Gauge2 of int (* \frac{1}{2}\phi F_{1,\mu\nu}F_2^{\mu\nu} = -\frac{1}{2}\phi(i\partial_{[\mu},V_{1,\nu]})(i\partial^{[\mu},V_2^{\nu]}) *)
Dim5_Scalar_Gauge2_Skew of int
     (* \frac{1}{4}\phi F_{1,\mu\nu}\tilde{F}_2^{\mu\nu} = -\phi(\mathrm{i}\partial_{\mu}V_{1,\nu})(\mathrm{i}\partial_{\rho}V_{2,\sigma})\epsilon^{\mu\nu\rho\sigma} *)
Dim5\_Scalar\_Scalar2 of int (* \phi_1 \partial_\mu \phi_2 \partial^\mu \phi_3 *)
Dim5\_Scalar\_Vector\_Vector\_T of int (* \phi(i\partial_{\mu}V_{1}^{\nu})(i\partial_{\nu}V_{2}^{\mu}) *)
Dim5\_Scalar\_Vector\_Vector\_TU of int (* (i\partial_{\nu}\phi)(i\partial_{\mu}V_{1}^{\nu})V_{2}^{\mu} *)
Dim5\_Scalar\_Vector\_Vector\_U \text{ of } int \ (* \ (\mathrm{i}\partial_{\nu}\phi)(\mathrm{i}\partial_{\mu}V^{\nu})V^{\mu} \ *)
Scalar\_Vector\_Vector\_t of int (* (\partial_{\mu}V_{\nu} - \partial_{\nu}V_{\mu})^2 *)
Dim6\_Vector\_Vector\_T of int (*V_1^{\mu}((i\partial_{\nu}V_2^{\rho})i\overrightarrow{\partial_{\mu}}(i\partial_{\rho}V_3^{\nu})) *)
 Tensor_2 - Vector_Vector of int (* T^{\mu\nu}(V_{1,\mu}V_{2,\nu} + V_{1,\nu}V_{2,\mu}) *)
 Tensor\_2\_Vector\_Vector\_1 \  \, \text{of} \  \, int \, \left(* \, T^{\mu\nu}(V_{1,\mu}V_{2,\nu} + V_{1,\nu}V_{2,\mu} - g_{\mu,\nu}V_1^{\rho}V_{2,\rho}) \, *\right)
 Tensor_2 - Vector_1 - Vector_2 - cf of int (*T^{\mu\nu}(-\frac{c_f}{2}g_{\mu,\nu}V_1^{\rho}V_{2,\rho}) *)
 Tensor_2 - Scalar_S calar of int (*T^{\mu\nu}(\partial_\mu \phi_1 \partial_\nu \phi_2 + \partial_\nu \phi_1 \partial_\mu \phi_2) *)
Tensor_2\_Scalar\_Scalar\_cf of int (*T^{\mu\nu}(-\frac{c_f}{2}g_{\mu,\nu}\partial_{\rho}\phi_1\partial_{\rho}\phi_2)*)
Tensor\_2\_Vector\_Vector\_t \ \ of \ \ int \ \ (*\ T^{\mu\nu}(V_{1,\bar{\mu}}V_{2,\nu} + \underbrace{V_{1,\bar{\nu}}}_{2,\nu}V_{\underline{2,\mu}} - g_{\mu,\nu}V_1^{\rho}V_{2,\rho}) \ *)
Dim5\_Tensor\_2\_Vector\_Vector\_1 of int (*T^{\alpha\beta}(V_1^{\mu}i\overleftrightarrow{\partial}_{\alpha}i\overleftrightarrow{\partial}_{\beta}V_{2,\mu}*)
Dim5\_Tensor\_2\_Vector\_Vector\_2 of int
     (* T^{\alpha\beta}(V_1^{\mu} i \overleftrightarrow{\partial}_{\beta} (i\partial_{\mu} V_{2,\alpha}) + V_1^{\mu} i \overleftrightarrow{\partial}_{\alpha} (i\partial_{\mu} V_{2,\beta})) *)
Dim 7\_Tensor\_2\_Vector\_Vector\_T \ \ \text{of} \ \ int \ (*\ T^{\alpha\beta}((\mathrm{i}\partial^\mu V_1^\nu)\mathrm{i} \overleftrightarrow{\partial}_\alpha \mathrm{i} \overleftrightarrow{\partial}_\beta (\mathrm{i}\partial_\nu V_{2,\mu}))\ *)
Dim6_Scalar_Vector_Vector_D of int
 (*i\phi(-(\partial^{\mu}\partial^{\nu}W_{\mu}^{-})W_{\nu}^{+} - (\partial^{\mu}\partial^{\nu}W_{\nu}^{+})W
  + \left( (\partial^{\rho} \partial_{\rho} W_{\mu}^{-}) \dot{W}_{\nu}^{+} + (\partial^{\rho} \partial_{\rho} W_{\nu}^{+}) W_{\mu}^{-} \right) g^{\dot{\mu}\nu} \right) *)
Dim6_Scalar_Vector_Vector_DP of int
 (*i((\partial^{\mu}H)(\partial^{\nu}W_{\mu}^{-})W_{\nu}^{+}+(\partial^{\nu}H)(\partial^{\mu}W_{\nu}^{+})W_{\nu}^{-})
  -((\partial^{\rho}H)(\partial_{\rho}W_{\mu}^{-\rho})W_{\nu}^{+}(\partial^{\rho}H)(\partial^{\rho}W_{\nu}^{+})W_{\mu}^{-})g^{\mu\nu})*)
Dim6\_HAZ\_D of int (*i((\partial^{\mu}\partial^{\nu}A_{\mu})Z_{\nu} + (\partial^{\rho}\partial_{\rho}A_{\mu})Z_{\nu}g^{\mu\nu}) *)
Dim6\_HAZ\_DP of int (*i((\partial^{\nu}A_{\mu})(\partial^{\mu}H)Z_{\nu} - (\partial^{\rho}A_{\mu})(\partial_{\rho}H)Z_{\nu}g^{\mu\nu}) *)
Dim6\_AWW\_DP \text{ of } int \ (*i((\partial^{\rho}A_{\mu})W_{\nu}^{-}W_{\rho}^{+}g^{\mu\nu} - (\partial^{\nu}A_{\mu})W_{\nu}^{-}W_{\rho}^{+}g^{\mu\rho}) \ *)
Dim6\_AWW\_DW of int
 (*\mathrm{i}[(3(\partial^\rho A_\mu)W_\nu^-W_\rho^+ - (\partial^\rho W_\nu^-)A_\mu W_\rho^+ + (\partial^\rho W_\rho^+)A_\mu W_\nu^-)g^{\mu\nu}
  + \left( -3(\partial^{\nu}A_{\mu})W_{\nu}^{-}W_{\rho}^{+} - (\partial^{\nu}W_{\nu}^{-})A_{\mu}W_{\rho}^{+} + (\partial^{\nu}W_{\rho}^{+})A_{\mu}W_{\nu}^{-})g^{\mu\rho} + (2(\partial^{\mu}W_{\nu}^{-})A_{\mu}W_{\rho}^{+} - 2(\partial^{\mu}W_{\rho}^{+})A_{\mu}W_{\nu}^{-})g^{\nu\rho} \right] *)
Dim6\_HHH of int (*i(-(\partial^{\mu}H_1)(\partial_{\mu}H_2)H_3 - (\partial^{\mu}H_1)H_2(\partial_{\mu}H_3) - H_1(\partial^{\mu}H_2)(\partial_{\mu}H_3)) *)
Dim6\_Gauge\_Gauge\_i of int
 (*i(-(\partial^{\nu}V_{\mu})(\partial^{\rho}V_{\nu})(\partial^{\mu}V_{\rho}) + (\partial^{\rho}V_{\mu})(\partial^{\mu}V_{\nu})(\partial^{\nu}V_{\rho})
  +(-\partial^{\nu}V_{\rho}g^{\mu\rho}+\partial^{\mu}V_{\rho}g^{\nu\rho})(\partial^{\sigma}V_{\mu})(\partial_{\sigma}V_{\nu})+(\partial^{\rho}V_{\nu}g^{\mu\nu}-\partial^{\mu}V_{\nu}g^{\nu\rho})(\partial^{\sigma}V_{\mu})(\partial_{\sigma}V_{\rho})
  + (-\partial^{\rho}V_{\mu}g^{\mu\nu} + \partial^{\mu}V_{\mu}g^{\mu\rho})(\partial^{\sigma}V_{\nu})(\partial_{\sigma}V_{\rho})) *)
Gauge_Gauge_i of int
Dim6\_GGG of int
Dim6\_WWZ\_DPWDW of int
 (\ast \ \mathrm{i}(((\partial^{\rho}V_{\mu})V_{\nu}V_{\rho}-(\partial^{\rho}V_{\nu})V_{\mu}V_{\rho})g^{\mu\nu}-(\partial^{\nu}V_{\mu})V_{\nu}V_{\rho}g^{\mu\rho}+(\partial^{\mu}V_{\nu})V_{\mu}V_{\rho})g^{\rho\nu})\ \ast)
Dim6\_WWZ\_DW of int
 (\ast \ \mathrm{i}(((\partial^{\mu}V_{\mu})V_{\nu}V_{\rho}+V_{\mu}(\partial^{\mu}V_{\nu})V_{\rho})g^{\nu\rho}-((\partial^{\nu}V_{\mu})V_{\nu}V_{\rho}+V_{\mu}(\partial^{\nu}V_{\nu})V_{\rho})g^{\mu\rho})\ \ast)
Dim6\_WWZ\_D of int (*i(V_{\mu})V_{\nu}(\partial^{\nu}V_{\rho})g^{\mu\rho} + V_{\mu}V_{\nu}(\partial^{\mu}V_{\rho})g^{\nu\rho}) *)
```

```
| TensorVector_Vector_Vector of int
| TensorVector_Vector_Cf of int
| TensorVector_Scalar_Scalar of int
| TensorVector_Scalar_Scalar_cf of int
| TensorScalar_Vector_Vector of int
| TensorScalar_Vector_Vector_cf of int
| TensorScalar_Scalar_Scalar of int
| TensorScalar_Scalar_Scalar_cf of int
```

As long as we stick to renormalizable couplings, there are only three types of quartic couplings: Scalar4, $Scalar2_Vector2$ and Vector4. However, there are three inequivalent contractions for the latter and the general vertex will be a linear combination with integer coefficients:

Scalar
$$41: \phi_1\phi_2\phi_3\phi_4$$
 (9.2a)

$$Scalar 2 - Vector 2 1: \quad \phi_1 \phi_2 V_3^{\mu} V_{4,\mu}$$
 (9.2b)

Vector4 [1,
$$C_{-}12_{-}34$$
]: $V_{1}^{\mu}V_{2,\mu}V_{3}^{\nu}V_{4,\nu}$ (9.2c)

$$Vector 4 [1, C_1 3_4 2] : V_1^{\mu} V_2^{\nu} V_{3,\mu} V_{4,\nu}$$
 (9.2d)

$$Vector 4 [1, C_{-}14_{-}23] : V_{1}^{\mu} V_{2}^{\nu} V_{3,\nu} V_{4,\mu}$$
 (9.2e)

type $contract4 = C_12_34 \mid C_13_42 \mid C_14_23$

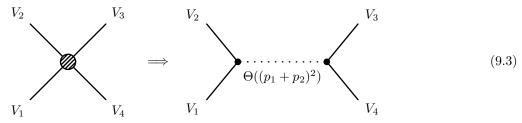
type $\alpha \ vertex4 =$ Scalar4 of int Scalar2_Vector2 of int Vector4 of $(int \times contract4)$ list DScalar4 of $(int \times contract4)$ list $DScalar2_Vector2$ of $(int \times contract4)$ list Dim8_Scalar2_Vector2_1 of int Dim8_Scalar2_Vector2_2 of int $Dim8_Scalar2_Vector2_m_0$ of int $Dim8_Scalar2_Vector2_m_1$ of int $Dim8_Scalar2_Vector2_m_7$ of intDim8_Scalar4 of int $Dim8_Vector4_t_0$ of $(int \times contract4)$ list $Dim8_Vector4_t_1$ of $(int \times contract4)$ list Dim8 - Vector4 - t - 2 of $(int \times contract4)$ list $Dim8_Vector4_m_0$ of $(int \times contract4)$ list

 $Dim8_Vector4_m_1$ of $(int \times contract4)$ list $Dim8_Vector4_m_7$ of $(int \times contract4)$ list GBBG of $int \times fermionbar \times boson2 \times fermion$

In some applications, we have to allow for contributions outside of perturbation theory. The most prominent example is heavy gauge boson scattering at very high energies, where the perturbative expression violates unitarity.

One solution is the 'K-matrix' ansatz. Such unitarizations typically introduce effective propagators and/or vertices that violate crossing symmetry and vanish in the t-channel. This can be taken care of in Fusion by filtering out vertices that have the wrong momenta.

In this case the ordering of the fields in a vertex of the Feynman rules becomes significant. In particular, we assume that (V_1, V_2, V_3, V_4) implies



The list of pairs of parameters denotes the location and strengths of the poles in the K-matrix ansatz:

$$(c_1, a_1, c_2, a_2, \dots, c_n, a_n) \Longrightarrow f(s) = \sum_{i=1}^n \frac{c_i}{s - a_i}$$
 (9.4)

```
Vector 4 \_K \_Matrix\_tho of int \times (\alpha \times \alpha) list
    Vector 4 \_K \_Matrix \_jr \text{ of } int \times (int \times contract 4) \ list
    Vector 4 \_K \_Matrix \_cf \_t0 of int \times (int \times contract 4) list
    Vector 4 \_K \_Matrix \_cf \_t1 of int \times (int \times contract 4) list
    Vector 4 K_Matrix_c f_t 2 of int \times (int \times contract 4) list
    Vector4\_K\_Matrix\_cf\_t\_rsi of int \times (int \times contract4) list
    Vector 4 \_K \_Matrix \_cf \_m0 of int \times (int \times contract 4) list
    Vector 4 \_K \_Matrix \_cf \_m1 of int \times (int \times contract 4) list
    Vector4\_K\_Matrix\_cf\_m7 of int \times (int \times contract4) list
   DScalar2\_Vector2\_K\_Matrix\_ms of int \times (int \times contract4) list
   DScalar2\_Vector2\_m\_0\_K\_Matrix\_cf of int \times (int \times contract4) list
   DScalar2\_Vector2\_m\_1\_K\_Matrix\_cf of int \times (int \times contract4) list
    DScalar2\_Vector2\_m\_7\_K\_Matrix\_cf of int \times (int \times contract4) list
   DScalar4\_K\_Matrix\_ms of int \times (int \times contract4) list
   Dim6_H_4_P_2 of int
    (* \mathrm{i} (-(\partial^{\mu} H_1)(\partial_{\mu} H_2) H_3 H_4 - (\partial^{\mu} H_1) H_2 (\partial_{\mu} H_3) H_4 - (\partial^{\mu} H_1) H_2 H_3 (\partial_{mu} H_4)
     -H_1(\partial^{\mu}H_2)(\partial_{\mu}H_3)H_4 - H_1(\partial^{\mu}H_2)H_3(\partial_{\mu}H_4) - H_1H_2(\partial^{\mu}H_3)(\partial_{\mu}H_4)) *
   Dim6\_AHWW\_DPB of int (*iH((\partial^{\rho}A_{\mu})W_{\nu}W_{\rho}g^{\mu\nu} - (\partial^{\nu}A_{\mu})W_{\nu}W_{\rho}g^{\mu\rho}) *)
   Dim6\_AHWW\_DPW of int
    (*i(((\partial^{\rho}A_{\mu})W_{\nu}W_{\rho}-(\partial^{\rho}H)A_{\mu}W_{\nu}W_{\rho})g^{\mu\nu}
     (-(\partial^{\nu}A_{\mu})W_{\nu}W_{\rho} + (\partial^{\nu}H)A_{\mu}W_{\nu}W_{\rho})g^{\mu\rho}) *)
  Dim6\_AHWW\_DW of int
    (*iH((3(\partial^{\rho}A_{\mu})W_{\nu}W_{\rho}-A_{\mu}(\partial^{\rho}W_{\nu})W_{\rho}+A_{\mu}W_{\nu}(\partial^{\rho}W_{\rho}))g^{\mu\nu})
     +(-3(\partial^{\nu}A_{\mu})W_{\nu}W_{\rho}-A_{\mu}(\partial^{\nu}W_{\nu})W_{\rho}+A_{\mu}W_{\nu}(\partial^{\nu}W_{\rho}))g^{\mu\rho}
     +2(A_{\mu}(\partial^{\mu}W_{\nu})W_{\rho}+A_{\mu}W_{\nu}(\partial^{\mu}W_{\rho})))g^{\nu\rho})*)
| Dim6_Vector4_DW of int (*i(-V_{1,\mu}V_{2,\nu}V^{3,\nu}V^{4,\mu}-V_{1,\mu}V_{2,\nu}V^{3,\mu}V^{4,\nu})
     +2V_{1,\mu}V^{2,\mu}V_{3,\nu}V^{4,\nu} *)
   Dim6\_Vector4\_W of int
    (*i(((\partial^{\rho}V_{1,\mu})V_{2}^{\mu}(\partial^{\sigma}V_{3,\rho})V_{4,\sigma}+V_{1,\mu}(\partial^{\rho}V_{2}^{\mu})(\partial^{\sigma}V_{3,\rho})V_{4,\sigma})
     +\left.(\partial^{\sigma}V_{1,\mu})V_{2}^{\mu}V_{3,\rho}(\partial^{\rho}V_{4,\sigma})+V_{1,\mu}(\partial^{\sigma}V_{2}^{\mu})V_{3,\rho}(\partial^{\rho}V_{4,\sigma})\right)
     + ((\partial^{\sigma}V_{1,\mu})V_{2,\nu}(\partial^{\nu}V_{3}^{\mu})V_{4,\sigma} - V_{1,\mu}(\partial^{\sigma}V_{2,\nu})(\partial^{\nu}V_{3}^{\mu})V_{4,\sigma}
     -(\partial^{\nu}V_{1}^{\mu})V_{2,\nu}(\partial^{\sigma}V_{3,\mu})V_{4,\sigma}-(\partial^{\sigma}V_{1,\mu})V_{2,\nu}V_{3}^{\mu}(\partial^{\nu}V_{4,\sigma}))
     +(-(\partial^{\rho}V_{1,\mu})V_{2,\nu}(\partial^{\nu}V_{3,\rho})V_{4}^{\mu}+(\partial^{\rho}V_{1,\mu})V_{2,\nu}V_{3,\rho}(\partial^{\nu}V_{4}^{\mu})
     -V_{1,\mu}(\partial^{\rho}V_{2,\nu})V_{3,\rho}(\partial^{\nu}V_{4}^{\mu}) - (\partial^{\nu}V_{1,\mu})V_{2,\nu}V_{3,\rho}(\partial^{\rho}V_{4}^{\mu}))
     + (-(\partial^{\sigma}V_{1,\mu})V_{2,\nu}(\partial^{\mu}V_{3}^{\nu})V_{4,\sigma} + V_{1,\mu}(\partial^{\sigma}V_{2,\nu})(\partial^{\mu}V_{3}^{\nu})V_{4,\sigma}
     -V_{1,\mu}(\partial^{\mu}V_{2,\nu})(\partial^{\sigma}V_{3}^{\nu})V_{4,\sigma}-V_{1,\mu}(\partial^{\sigma}V_{2,\nu})V_{3}^{\nu}(\partial^{\mu}V_{4,\sigma})
     + \left( -V_{1,\mu} (\partial^{\rho} V_{2,\nu}) (\partial^{\mu} V_{3,\rho}) V_{4}^{\nu} - (\partial^{\rho} V_{1,\mu}) V_{2,\nu} V_{3,\rho} (\partial^{\mu} V_{4}^{\nu}) \right)
     + V_{1,\mu}(\partial^{\rho}V_{2,\nu})V_{3,\rho}(\partial^{\mu}V_{4}^{\nu}) - V_{1,\mu}(\partial^{\mu}V_{2,\nu})V_{3,\rho}(\partial^{\rho}V_{4}^{\nu}))
     + ((\partial^{\nu}V_{1,\mu})V_{2,\nu}(\partial^{\mu}V_{3,\rho})V_{4}^{\rho} + V_{1,\mu}(\partial^{\mu}V_{2,\nu})(\partial^{\nu}V_{3,\rho})V_{4}^{\rho}
     + (\partial^{\nu} V_{1,\mu}) V_{2,\nu} V_{3,\rho} (\partial^{\mu} V_{4}^{\rho}) + V_{1,\mu} (\partial^{\mu} V_{2,\nu}) V_{3,\rho} (\partial^{\nu} V_{4}^{\rho}))
     + (\partial^{\rho}V_{1,\mu})V_{2,\nu}V_{3}^{\mu}(\partial_{\rho}V_{4}^{\nu}) - (\partial^{\rho}V_{1,\mu})V_{2}^{\mu}V_{3,\nu}(\partial_{\rho}V_{4}^{\nu})
     +V_{1,\mu}(\partial^{\rho}V_{2,\nu})(\partial_{\rho}V_{3}^{\mu})V_{4}^{\nu}-V_{1,\mu}(\partial^{\rho}V_{2}^{\mu})(\partial_{\rho}V_{3,\nu})V_{4}^{\nu}
     + \, (\partial^{\rho} V_{1,\mu}) V_{2,\nu}^{\rho} (\partial_{\rho} V_{3}^{\rho}) V_{4}^{\mu} - (\partial^{\rho} V_{1,\mu}) V_{2}^{\mu} (\partial_{\rho} V_{3,\nu}) V_{4}^{\nu}
     + V_{1,\mu}(\partial^{\rho}V_{2,\nu})V_{3}^{\nu}(\partial_{\rho}V_{4}^{\mu}) - V_{1,\mu}(\partial^{\rho}V_{2}^{\mu})V_{3,\nu}(\partial_{\rho}V_{4}^{\nu})) *
  Dim6\_Scalar2\_Vector2\_D of int
    (*iH_1H_2(-(\partial^{\mu}\partial^{\nu}V_{3,\mu})V_{4,\nu}+(\partial^{\mu}\partial_{\mu}V_{3,\nu})V_4^{\nu})
     -V_{3,\mu}(\partial^{\mu}\partial^{\nu}V_{4,\nu})+V_{3,\mu}(\partial^{\nu}\partial_{\nu}V_{4}^{\mu}))*)
   Dim6_Scalar2_Vector2_DP of int
    (*i((\partial^{\mu}H_1)H_2(\partial^{\nu}V_{3,\mu})V_{4,\nu} - (\partial^{\nu}H_1)H_2(\partial_{\nu}V_{3,\mu})V^{4,\mu} + H_1(\partial^{\mu}H_2)(\partial^{\nu}V_{3,\mu})V_{4,\nu})
     -H_1(\partial^{\nu} H_2)(\partial_{\nu} V_{3,\mu})V^{4,\mu} + (\partial^{\nu} H_1)H_2V_{3,\mu}(\partial^{\mu} V_{4,\nu}) - (\partial^{\nu} H_1)H_2V_{3,\mu}(\partial_{\nu} V^{4,\mu})
     +H_1(\partial^{\nu}H_2)V_{3,\mu}(\partial^{\mu}V_{4,\nu})-H_1(\partial^{\nu}H_2)V_{3,\mu}(\partial_{\nu}V^{4,\mu}))*
   Dim6_Scalar2_Vector2_PB of int
    (*i(H_1H_2(\partial^{\nu}V_{3,\mu})(\partial^{\mu}V_{4,\nu}) - H_1H_2(\partial^{\nu}V_{3,\mu})(\partial_{\nu}V^{4,\mu})) *)
   Dim6\_HHZZ\_T of int (*iH_1H_2V_{3,\mu}V^{4,\mu} *)
  Dim6\_HWWZ\_DW of int
    (*i(H_1(\partial^{\rho}W_{2,\mu})W^{3,\mu}Z_{4,\rho} - H_1W_{2,\mu}(\partial^{\rho}W^{3,\mu})Z_{4,\rho} - 2H_1(\partial^{\nu}W_{2,\mu})W_{3,\nu}Z^{4,\mu})
     -H_1W_{2,\mu}(\partial^{\nu}W_{3,\nu})Z^{4,\mu}+H_1(\partial^{\mu}W_{2,\mu})W_{3,\nu}Z^{4,\nu}+2H_1W_{2,\mu}(\partial^{\mu}W_{3,\nu})Z^{4,\nu})*
  Dim6_HWWZ_DPB of int
    (*i(-H_1W_{2,\mu}W_{3,\nu}(\partial^{\nu}Z^{4,\mu}) + H_1W_{2,\mu}W_{3,\nu}(\partial^{\mu}Z^{4,\nu})) *)
   Dim6\_HWWZ\_DDPW of int
    (*i(H_1(\partial^{\nu}W_{2,\mu})W^{3,\mu}Z_{4,\nu}-H_1W_{2,\mu}(\partial^{\nu}W^{3,\mu})Z_{4,\nu}-H_1(\partial^{\nu}W_{2,\mu})W_{3,\nu}Z^{4,\mu})
```

```
+H_1W_{2,\mu}W_{3,\nu}(\partial^{\nu}Z^{4,\mu})+H_1W_{2,\mu}(\partial^{\mu}W_{3,\nu})Z^{4,\nu}-H_1W_{2,\mu}W_{3,\nu}(\partial^{\mu}Z^{4,\nu}))*
Dim6\_HWWZ\_DPW of int
 \begin{array}{l} (* \mathrm{i}(H_1(\partial^\nu W_{2,\mu})W^{3,\mu}Z_{4,\nu} - H_1W_{2,\mu}(\partial^\nu W^{3,\mu})Z_{4,\nu} + (\partial^\nu H_1)W_{2,\mu}W_{3,\nu}Z^{4,\mu} \\ - H_1(\partial^\nu W_{2,\mu})W_{3,\nu}Z^{4,\mu} - (\partial^\mu H_1)W_{2,\mu}W_{3,\nu}Z^{4,\nu} + H_1W_{2,\mu}(\partial^\mu W_{3,\nu})Z^{4,\nu}) \ *) \end{array}
Dim6\_AHHZ\_D of int
 (*i(H_1H_2(\partial^{\mu}\partial^{\nu}A_{\mu})Z_{\nu} - H_1H_2(\partial^{\nu}\partial_{\nu}A_{\mu})Z^{\mu}) *)
Dim6\_AHHZ\_DP of int
 (*i((\partial^{\mu}H_1)H_2(\partial^{\nu}A_{\mu})Z_{\nu} + H_1(\partial^{\mu}H_2)(\partial^{\nu}A_{\mu})Z_{\nu}
   -(\partial^{\nu}H_1)H_2(\partial_{\nu}A_{\mu})Z^{\mu} - H_1(\partial^{\nu}H_2)(\partial_{\nu}A_{\mu})Z^{\mu}) *)
Dim6_AHHZ_PB of int
 (*i(H_1H_2(\partial^{\nu}A_{\mu})(\partial_{\nu}Z^{\mu}) - H_1H_2(\partial^{\nu}A_{\mu})(\partial^{\mu}Z_{\nu})) *)
```

type $\alpha \ vertexn =$

| UFO of Algebra. $QC.t \times string \times lorentzn \times fermion_lines \times Color.Vertex.t$

An obvious candidate for addition to boson is T, of course.



This list is sufficient for the minimal standard model, but not comprehensive enough for most of its extensions, supersymmetric or otherwise. In particular, we need a general parameterization for all trilinear vertices. One straightforward possibility are polynomials in the momenta for each combination of fields.



JR sez' (regarding the Majorana Feynman rules): Here we use the rules which can be found in [7] and are more properly described in Targets where the performing of the fusion rules in analytical expressions is encoded. (JR's probably right, but I need to check myself...)

Signify which two of three fields are fused:

type
$$fuse2 = F23 | F32 | F31 | F13 | F12 | F21$$

Signify which three of four fields are fused:

type fuse3 =

F123	F231	F312	F132	F321	F213
F124	F241	F412	F142	F421	F214
F134	F341	F413	F143	F431	F314
F234	F342	F423	F243	F432	F324

Explicit enumeration types make no sense for higher degrees.

type fusen = int list

The third member of the triplet will contain the coupling constant:

type $\alpha t =$ $V3 ext{ of } \alpha ext{ } vertex3 ext{ } ext{ } ext{ } fuse2 ext{ } ext{ } ext{ } lpha$ V4 of α vertex4 \times fuse3 \times α $Vn ext{ of } \alpha ext{ } vertexn ext{ } ext{ } ext{ } fusen ext{ } ext{ } ext{ } lpha$

9.1.3 Gauge Couplings

Dimension-4 trilinear vector boson couplings

$$f_{abc}\partial^{\mu}A^{a,\nu}A^{b}_{\mu}A^{c}_{\nu} \to if_{abc}k^{\mu}_{1}A^{a,\nu}(k_{1})A^{b}_{\mu}(k_{2})A^{c}_{\nu}(k_{3})$$

$$= -\frac{i}{3!}f_{a_{1}a_{2}a_{3}}C^{\mu_{1}\mu_{2}\mu_{3}}(k_{1},k_{2},k_{3})A^{a_{1}}_{\mu_{1}}(k_{1})A^{a_{2}}_{\mu_{2}}(k_{2})A^{a_{3}}_{\mu_{3}}(k_{3}) \quad (9.5a)$$

with the totally antisymmetric tensor (under simultaneous permutations of all quantum numbers μ_i and k_i) and all momenta outgoing

$$C^{\mu_1\mu_2\mu_3}(k_1, k_2, k_3) = (g^{\mu_1\mu_2}(k_1^{\mu_3} - k_2^{\mu_3}) + g^{\mu_2\mu_3}(k_2^{\mu_1} - k_3^{\mu_1}) + g^{\mu_3\mu_1}(k_3^{\mu_2} - k_1^{\mu_2}))$$
(9.5b)

Since $f_{a_1a_2a_3}C^{\mu_1\mu_2\mu_3}(k_1,k_2,k_3)$ is totally symmetric (under simultaneous permutations of all quantum numbers a_i , μ_i and k_i), it is easy to take the partial derivative

$$A^{a,\mu}(k_2 + k_3) = -\frac{\mathrm{i}}{2!} f_{abc} C^{\mu\rho\sigma}(-k_2 - k_3, k_2, k_3) A^b_{\rho}(k_2) A^c_{\sigma}(k_3)$$
(9.6a)

	only Dirac fermions	incl. Majorana fermions
$FBF (Psibar, S, Psi): \mathcal{L}_I = g_S y$		-
F12	$\overline{\bar{\psi}_2 \leftarrow i \cdot g_S \bar{\psi}_1 S}$	$\psi_2 \leftarrow \mathbf{i} \cdot g_S \psi_1 S$
F21	$\bar{\psi}_2 \leftarrow i \cdot g_S S \bar{\psi}_1$	$\psi_2 \leftarrow \mathbf{i} \cdot g_S S \psi_1$
F13	$S \leftarrow \mathbf{i} \cdot g_S \bar{\psi}_1 \psi_2$	$S \leftarrow \mathbf{i} \cdot g_S \psi_1^T \mathbf{C} \psi_2$
F31	$S \leftarrow i \cdot g_S \psi_{2,\alpha} \bar{\psi}_{1,\alpha}$	$S \leftarrow \mathbf{i} \cdot g_S \psi_2^T \mathbf{C} \psi_1$
F23	$\psi_1 \leftarrow \mathbf{i} \cdot g_S S \psi_2$	$\psi_1 \leftarrow \mathbf{i} \cdot g_S S \psi_2$
F32	$\psi_1 \leftarrow \mathbf{i} \cdot g_S \psi_2 S$	$\psi_1 \leftarrow \mathbf{i} \cdot g_S \psi_2 S$
FBF	$(Psibar, P, Psi): \mathcal{L}_I = g_P \psi$	$\bar{\psi}_1 P \gamma_5 \psi_2$
F12	$\bar{\psi}_2 \leftarrow i \cdot g_P \bar{\psi}_1 \gamma_5 P$	$\psi_2 \leftarrow \mathbf{i} \cdot g_P \gamma_5 \psi_1 P$
F21	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot g_P P \bar{\psi}_1 \gamma_5$	$\psi_2 \leftarrow \mathbf{i} \cdot g_P P \gamma_5 \psi_1$
F13	$P \leftarrow \mathbf{i} \cdot g_P \bar{\psi}_1 \gamma_5 \psi_2$	$P \leftarrow \mathbf{i} \cdot g_P \psi_1^T \mathbf{C} \gamma_5 \psi_2$
F31	$P \leftarrow i \cdot g_P[\gamma_5 \psi_2]_{\alpha} \bar{\psi}_{1,\alpha}$	$P \leftarrow \mathbf{i} \cdot g_P \psi_2^T \mathbf{C} \gamma_5 \psi_1$
F23	$\psi_1 \leftarrow \mathbf{i} \cdot g_P P \gamma_5 \psi_2$	$\psi_1 \leftarrow \mathbf{i} \cdot g_P P \gamma_5 \psi_2$
F32	$\psi_1 \leftarrow \mathbf{i} \cdot g_P \gamma_5 \psi_2 P$	$\psi_1 \leftarrow \mathbf{i} \cdot g_P \gamma_5 \psi_2 P$
FBF	(Psibar, V , Psi): $\mathcal{L}_I = g_V v$	$ar{\psi}_1 V \psi_2$
F12	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot g_V \bar{\psi}_1 V$	$\psi_{2,\alpha} \leftarrow i \cdot (-g_V) \psi_{1,\beta} V_{\alpha\beta}$
F21	$\bar{\psi}_{2,\beta} \leftarrow i \cdot g_V V_{\alpha\beta} \bar{\psi}_{1,\alpha}$	$\psi_2 \leftarrow \mathbf{i} \cdot (-g_V) V \psi_1$
F13	$V_{\mu} \leftarrow \mathbf{i} \cdot g_V \bar{\psi}_1 \gamma_{\mu} \psi_2$	$V_{\mu} \leftarrow \mathbf{i} \cdot g_{V}(\psi_{1})^{T} \mathbf{C} \gamma_{\mu} \psi_{2}$
F31	$V_{\mu} \leftarrow i \cdot g_{V}[\gamma_{\mu}\psi_{2}]_{\alpha}\bar{\psi}_{1,\alpha}$	$V_{\mu} \leftarrow \mathbf{i} \cdot (-g_V)(\psi_2)^T \mathbf{C} \gamma_{\mu} \psi_1$
F23	$\psi_1 \leftarrow \mathbf{i} \cdot g_V V \psi_2$	$\psi_1 \leftarrow \mathbf{i} \cdot g_V V \psi_2$
F32	$\psi_{1,\alpha} \leftarrow \mathbf{i} \cdot g_V \psi_{2,\beta} V_{\alpha\beta}$	$\psi_{1,\alpha} \leftarrow \mathbf{i} \cdot g_V \psi_{2,\beta} V_{\alpha\beta}$
FBF	(Psibar, A, Psi): $\mathcal{L}_I = g_A \bar{\psi}$	$\overline{\psi}_1\gamma_5 A\psi_2$
F12	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot g_A \bar{\psi}_1 \gamma_5 A$	$\psi_{2,\alpha} \leftarrow \mathbf{i} \cdot g_A \psi_\beta [\gamma_5 A]_{\alpha\beta}$
F21	$\bar{\psi}_{2,\beta} \leftarrow i \cdot g_A [\gamma_5 A]_{\alpha\beta} \bar{\psi}_{1,\alpha}$	$\psi_2 \leftarrow \mathbf{i} \cdot g_A \gamma_5 A \psi$
F13	$A_{\mu} \leftarrow \mathbf{i} \cdot g_A \bar{\psi}_1 \gamma_5 \gamma_{\mu} \psi_2$	$A_{\mu} \leftarrow \mathbf{i} \cdot g_A \psi_1^T \mathbf{C} \gamma_5 \gamma_{\mu} \psi_2$
F31	$A_{\mu} \leftarrow i \cdot g_A [\gamma_5 \gamma_{\mu} \psi_2]_{\alpha} \bar{\psi}_{1,\alpha}$	$A_{\mu} \leftarrow \mathbf{i} \cdot g_A \psi_2^T \mathbf{C} \gamma_5 \gamma_{\mu} \psi_1$
F23	$\psi_1 \leftarrow \mathbf{i} \cdot g_A \gamma_5 A \psi_2$	$\psi_1 \leftarrow \mathbf{i} \cdot g_A \gamma_5 A \psi_2$
F32	$\psi_{1,\alpha} \leftarrow i \cdot g_A \psi_{2,\beta} [\gamma_5 A]_{\alpha\beta}$	$\psi_{1,\alpha} \leftarrow \mathbf{i} \cdot g_A \psi_{2,\beta} [\gamma_5 A]_{\alpha\beta}$

Table 9.3: Dimension-4 trilinear fermionic couplings. The momenta are unambiguous, because there are no derivative couplings and all participating fields are different.

	only Dirac fermions	incl. Majorana fermions
FBF (Psibar, T , Psi): $\mathcal{L}_I = g_T T_{\mu\nu} \bar{\psi}$		$[\gamma^{\mu}, \gamma^{\nu}]_{-}\psi_2$
F12	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot g_T \bar{\psi}_1 [\gamma^\mu, \gamma^\nu] T_{\mu\nu}$	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot g_T \cdots$
F21	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot g_T T_{\mu\nu} \bar{\psi}_1 [\gamma^\mu, \gamma^\nu]$	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot g_T \cdots$
F13	$T_{\mu\nu} \leftarrow \mathrm{i} \cdot g_T \bar{\psi}_1 [\gamma_\mu, \gamma_\nu] \psi_2$	$T_{\mu\nu} \leftarrow \mathbf{i} \cdot g_T \cdots$
F31	$T_{\mu\nu} \leftarrow \mathbf{i} \cdot g_T[[\gamma_\mu, \gamma_\nu] \psi_2]_\alpha \bar{\psi}_{1,\alpha}$	$T_{\mu\nu} \leftarrow \mathbf{i} \cdot g_T \cdots$
F23	$\psi_1 \leftarrow i \cdot g_T T_{\mu\nu} [\gamma^\mu, \gamma^\nu] \psi_2$	$\psi_1 \leftarrow \mathbf{i} \cdot g_T \cdots$
F32	$\psi_1 \leftarrow \mathbf{i} \cdot g_T[\gamma^\mu, \gamma^\nu] \psi_2 T_{\mu\nu}$	$\psi_1 \leftarrow \mathbf{i} \cdot g_T \cdots$

Table 9.4: Dimension-5 trilinear fermionic couplings (NB: the coefficients and signs are not fixed yet). The momenta are unambiguous, because there are no derivative couplings and all participating fields are different.

	only Dirac fermions	incl. Majorana fermions		
FBF	FBF (Psibar, SP, Psi): $\mathcal{L}_I = \bar{\psi}_1 \phi(g_S + g_P \gamma_5) \psi_2$			
F12	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot \bar{\psi}_1(g_S + g_P \gamma_5) \phi$	$\psi_2 \leftarrow i \cdots$		
F21	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot \phi \bar{\psi}_1 (g_S + g_P \gamma_5)$	$\psi_2 \leftarrow i \cdots$		
F13	$\phi \leftarrow i \cdot \bar{\psi}_1(g_S + g_P \gamma_5) \psi_2$	$\phi \leftarrow i \cdot \cdots$		
F31	$\phi \leftarrow i \cdot [(g_S + g_P \gamma_5) \psi_2]_{\alpha} \bar{\psi}_{1,\alpha}$	$\phi \leftarrow i \cdot \cdots$		
F23	$\psi_1 \leftarrow \mathbf{i} \cdot \phi(g_S + g_P \gamma_5) \psi_2$	$\psi_1 \leftarrow \mathrm{i} \cdot \cdots$		
F32	$\psi_1 \leftarrow \mathrm{i} \cdot (g_S + g_P \gamma_5) \psi_2 \phi$	$\psi_1 \leftarrow i \cdots$		
FBF	(Psibar, SL, Psi): $\mathcal{L}_I = g_L \bar{\psi}_1 \phi$	$\phi(1-\gamma_5)\psi_2$		
F12	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot g_L \bar{\psi}_1 (1 - \gamma_5) \phi$	$\psi_2 \leftarrow \mathrm{i} \cdot \cdots$		
F21	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot g_L \phi \bar{\psi}_1 (1 - \gamma_5)$	$\psi_2 \leftarrow \mathrm{i} \cdot \cdots$		
F13	$\phi \leftarrow i \cdot g_L \bar{\psi}_1 (1 - \gamma_5) \psi_2$	$\phi \leftarrow i \cdots$		
F31	$\phi \leftarrow i \cdot g_L[(1-\gamma_5)\psi_2]_{\alpha}\bar{\psi}_{1,\alpha}$	$\phi \leftarrow \mathrm{i} \cdot \cdots$		
F23	$\psi_1 \leftarrow \mathbf{i} \cdot g_L \phi (1 - \gamma_5) \psi_2$	$\psi_1 \leftarrow \mathrm{i} \cdot \cdots$		
F32	$\psi_1 \leftarrow \mathbf{i} \cdot g_L(1 - \gamma_5)\psi_2\phi$	$\psi_1 \leftarrow \mathrm{i} \cdot \cdots$		
FBF	(Psibar, SR, Psi): $\mathcal{L}_I = g_R \bar{\psi}_1 e^{i \bar{\psi}_1}$	$\phi(1+\gamma_5)\psi_2$		
F12	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot g_R \bar{\psi}_1 (1 + \gamma_5) \phi$	$\psi_2 \leftarrow \mathrm{i} \cdot \cdots$		
F21	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot g_R \phi \bar{\psi}_1 (1 + \gamma_5)$	$\psi_2 \leftarrow \mathrm{i} \cdot \cdots$		
F13	$\phi \leftarrow i \cdot g_R \bar{\psi}_1 (1 + \gamma_5) \psi_2$	$\phi \leftarrow i \cdot \cdots$		
F31	$\phi \leftarrow i \cdot g_R[(1+\gamma_5)\psi_2]_{\alpha}\bar{\psi}_{1,\alpha}$	$\phi \leftarrow i \cdots$		
F23	$\psi_1 \leftarrow \mathbf{i} \cdot g_R \phi (1 + \gamma_5) \psi_2$	$\psi_1 \leftarrow i \cdots$		
F32	$\psi_1 \leftarrow \mathbf{i} \cdot g_R(1+\gamma_5)\psi_2\phi$	$\psi_1 \leftarrow \mathrm{i} \cdot \cdots$		
FBF	FBF (Psibar, SLR, Psi): $\mathcal{L}_I = g_L \bar{\psi}_1 \phi (1 - \gamma_5) \psi_2 + g_R \bar{\psi}_1 \phi (1 + \gamma_5) \psi_2$			

 ${\it Table 9.5:} \quad {\it Combined dimension-4 trilinear fermionic couplings}.$

	only Dirac fermions	incl. Majorana fermions
FBF (Psibar, VA, Psi): $\mathcal{L}_I = \bar{\psi}_1 \mathbb{Z}(g_V - g_A \gamma_5) \psi_2$		
F12	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot \bar{\psi}_1 \mathbf{Z} (g_V - g_A \gamma_5)$	$\psi_2 \leftarrow \mathrm{i} \cdot \cdots$
F21	$\bar{\psi}_{2,\beta} \leftarrow \mathrm{i} \cdot [\mathbf{Z}(g_V - g_A \gamma_5)]_{\alpha\beta} \bar{\psi}_{1,\alpha}$	$\psi_2 \leftarrow \mathrm{i} \cdot \cdots$
F13	$Z_{\mu} \leftarrow \mathrm{i} \cdot \bar{\psi}_1 \gamma_{\mu} (g_V - g_A \gamma_5) \psi_2$	$Z_{\mu} \leftarrow \mathrm{i} \cdots$
F31	$Z_{\mu} \leftarrow \mathrm{i} \cdot [\gamma_{\mu} (g_V - g_A \gamma_5) \psi_2]_{\alpha} \bar{\psi}_{1,\alpha}$	$Z_{\mu} \leftarrow \mathrm{i} \cdot \cdots$
F23	$\psi_1 \leftarrow \mathrm{i} \cdot \mathbb{Z}(g_V - g_A \gamma_5) \psi_2$	$\psi_1 \leftarrow \mathbf{i} \cdot \cdots$
F32	$\psi_{1,\alpha} \leftarrow \mathrm{i} \cdot \psi_{2,\beta} [Z(g_V - g_A \gamma_5)]_{\alpha\beta}$	$\psi_1 \leftarrow \mathrm{i} \cdot \cdots$
FBF	(Psibar, VL, Psi): $\mathcal{L}_I = g_L \bar{\psi}_1 \mathbb{Z}(1)$	$-\gamma_5)\psi_2$
F12	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot g_L \bar{\psi}_1 \mathbf{Z} (1 - \gamma_5)$	$\psi_2 \leftarrow \mathrm{i} \cdot \cdots$
F21	$\bar{\psi}_{2,\beta} \leftarrow \mathrm{i} \cdot g_L[\mathbf{Z}(1-\gamma_5)]_{\alpha\beta}\bar{\psi}_{1,\alpha}$	$\psi_2 \leftarrow \mathrm{i} \cdot \cdots$
F13	$Z_{\mu} \leftarrow \mathrm{i} \cdot g_L \bar{\psi}_1 \gamma_{\mu} (1 - \gamma_5) \psi_2$	$Z_{\mu} \leftarrow \mathrm{i} \cdot \cdots$
F31	$Z_{\mu} \leftarrow \mathrm{i} \cdot g_L [\gamma_{\mu} (1 - \gamma_5) \psi_2]_{\alpha} \bar{\psi}_{1,\alpha}$	$Z_{\mu} \leftarrow \mathrm{i} \cdot \cdots$
F23	$\psi_1 \leftarrow \mathrm{i} \cdot g_L \mathbf{Z} (1 - \gamma_5) \psi_2$	$\psi_1 \leftarrow \mathrm{i} \cdot \cdots$
F32	$\psi_{1,\alpha} \leftarrow \mathrm{i} \cdot g_L \psi_{2,\beta} [Z(1-\gamma_5)]_{\alpha\beta}$	$\psi_1 \leftarrow \mathrm{i} \cdot \cdots$
FBF	(Psibar, VR, Psi): $\mathcal{L}_I = g_R \bar{\psi}_1 \mathbb{Z}(1$	$+\gamma_5)\psi_2$
F12	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot g_R \bar{\psi}_1 \mathbf{Z} (1 + \gamma_5)$	$\psi_2 \leftarrow \mathrm{i} \cdot \cdots$
F21	$\bar{\psi}_{2,\beta} \leftarrow \mathrm{i} \cdot g_R[\mathbf{Z}(1+\gamma_5)]_{\alpha\beta}\bar{\psi}_{1,\alpha}$	$\psi_2 \leftarrow \mathrm{i} \cdot \cdots$
F13	$Z_{\mu} \leftarrow \mathrm{i} \cdot g_R \bar{\psi}_1 \gamma_{\mu} (1 + \gamma_5) \psi_2$	$Z_{\mu} \leftarrow \mathrm{i} \cdot \cdots$
F31	$Z_{\mu} \leftarrow \mathrm{i} \cdot g_R [\gamma_{\mu} (1 + \gamma_5) \psi_2]_{\alpha} \bar{\psi}_{1,\alpha}$	$Z_{\mu} \leftarrow \mathrm{i} \cdot \cdots$
F23	$\psi_1 \leftarrow \mathrm{i} \cdot g_R Z(1 + \gamma_5) \psi_2$	$\psi_1 \leftarrow i \cdots$
F32	$\psi_{1,\alpha} \leftarrow i \cdot g_R \psi_{2,\beta} [Z(1+\gamma_5)]_{\alpha\beta}$	$\psi_1 \leftarrow i \cdots$
FBF (Psibar, VLR, Psi): $\mathcal{L}_I = g_L \bar{\psi}_1 \mathbf{Z} (1 - \gamma_5) \psi_2 + g_R \bar{\psi}_1 \mathbf{Z} (1 + \gamma_5) \psi_2$		

Table 9.6: Combined dimension-4 trilinear fermionic couplings continued.

FBF (Psibar, S, Chi): $\bar{\psi}S\chi$			
F12:	$\chi \leftarrow \psi S$	F21:	$\chi \leftarrow S\psi$
F13:	$S \leftarrow \psi^T \mathbf{C} \chi$	F31:	$S \leftarrow \chi^T \mathbf{C} \psi$
F23:	$\psi \leftarrow S\chi$	F32:	$\psi \leftarrow \chi S$
FBF (Psib	ar, P, Chi): $\bar{\psi}P\gamma_5\gamma_5$	ζ	
F12:	$\chi \leftarrow \gamma_5 \psi P$	F21:	$\chi \leftarrow P\gamma_5\psi$
F13:	$P \leftarrow \psi^T \mathbf{C} \gamma_5 \chi$	F31:	$P \leftarrow \chi^T \mathbf{C} \gamma_5 \psi$
F23:	$\psi \leftarrow P\gamma_5\chi$	F32:	$\psi \leftarrow \gamma_5 \chi P$
FBF (Psib	$ar, V, Chi): \bar{\psi}V\chi$		
F12:	$\chi_{\alpha} \leftarrow -\psi_{\beta} V_{\alpha\beta}$	F21:	$\chi \leftarrow -V\psi$
F13:	$V_{\mu} \leftarrow \psi^T \mathbf{C} \gamma_{\mu} \chi$	F31:	$V_{\mu} \leftarrow \chi^T \mathbf{C}(-\gamma_{\mu} \psi)$
F23:	$\psi \leftarrow V \chi$	F32:	$\psi_{\alpha} \leftarrow \chi_{\beta} V_{\alpha\beta}$
FBF (Psib	ar, A, Chi): $\bar{\psi}\gamma^5 A\gamma$	ζ	
F12:	$\chi_{\alpha} \leftarrow \psi_{\beta} [\gamma^5 A]_{\alpha\beta}$	F21:	$\chi \leftarrow \gamma^5 A \psi$
F13:	$A_{\mu} \leftarrow \psi^T C \gamma^5 \gamma_{\mu} \chi$	F31:	$A_{\mu} \leftarrow \chi^T \mathbf{C}(\gamma^5 \gamma_{\mu} \psi)$
F23:	$\psi \leftarrow \gamma^5 A \chi$	F32:	$\psi_{\alpha} \leftarrow \chi_{\beta} [\gamma^5 A]_{\alpha\beta}$

Table 9.7: Dimension-4 trilinear couplings including one Dirac and one Majorana fermion

$FBF \ (Psibar, \ SP, \ Chi): \ \bar{\psi}\phi(g_S + g_P\gamma_5)\chi$				
F12: y	$\chi \leftarrow (g_S + g_P \gamma_5) \psi \phi$	F21:	$\chi \leftarrow \phi(g_S + g_P \gamma_5) \psi$	
F13: q	$\phi \leftarrow \psi^T \mathbf{C}(g_S + g_P \gamma_5) \chi$	F31:	$\phi \leftarrow \chi^T \mathbf{C}(g_S + g_P \gamma_5) \chi$	
F23: 4	$\psi \leftarrow \phi(g_S + g_P \gamma_5) \chi$	F32:	$\psi \leftarrow (g_S + g_P \gamma_5) \chi \phi$	
FBF (Psibar	FBF (Psibar, VA, Chi): $\bar{\psi} Z(g_V - g_A \gamma_5) \chi$			
F12: y	$\chi_{\alpha} \leftarrow \psi_{\beta} [Z(-g_V - g_A \gamma_5)]_{\alpha\beta}$	F21:	$\chi \leftarrow \mathcal{Z}(-g_V - g_A \gamma_5)]\psi$	
F13: 2	$Z_{\mu} \leftarrow \psi^T C \gamma_{\mu} (g_V - g_A \gamma_5) \chi$	F31:	$Z_{\mu} \leftarrow \chi^T C \gamma_{\mu} (-g_V - g_A \gamma_5) \psi$	
F23: 4	$\psi \leftarrow \mathcal{Z}(g_V - g_A \gamma_5) \chi$	F32:	$\psi_{\alpha} \leftarrow \chi_{\beta} [Z(g_V - g_A \gamma_5)]_{\alpha\beta}$	

Table 9.8: Combined dimension-4 trilinear fermionic couplings including one Dirac and one Majorana fermion.

FBF (Chibar, S, Psi): $\bar{\chi}S\psi$			
F12:	$\psi \leftarrow \chi S$	F21:	$\psi \leftarrow S\chi$
F13:	$S \leftarrow \chi^T \mathbf{C} \psi$	F31:	$S \leftarrow \psi^T \mathbf{C} \chi$
F23:	$\chi \leftarrow S\psi$	F32:	$\chi \leftarrow \psi S$
FBF (Chil	bar, P, Psi): $\bar{\chi}P\gamma_5\psi$		
F12:	$\psi \leftarrow \gamma_5 \chi P$	F21:	$\psi \leftarrow P\gamma_5\chi$
F13:	$P \leftarrow \chi^T \mathbf{C} \gamma_5 \psi$	F31:	$P \leftarrow \psi^T \mathbf{C} \gamma_5 \chi$
F23:	$\chi \leftarrow P\gamma_5\psi$	F32:	$\chi \leftarrow \gamma_5 \psi P$
FBF (Chil	bar, V , Psi): $\bar{\chi}V\psi$		
F12:	$\psi_{\alpha} \leftarrow -\chi_{\beta} V_{\alpha\beta}$	F21:	$\psi \leftarrow - V \chi$
F13:	$V_{\mu} \leftarrow \chi^T \mathbf{C} \gamma_{\mu} \psi$	F31:	$V_{\mu} \leftarrow \psi^T \mathbf{C}(-\gamma_{\mu} \chi)$
F23:	$\chi \leftarrow V \psi$	F32:	$\chi_{\alpha} \leftarrow \psi_{\beta} V_{\alpha\beta}$
FBF (Chil	bar, A, Psi): $\bar{\chi}\gamma^5 A\psi$		
F12:	$\psi_{\alpha} \leftarrow \chi_{\beta} [\gamma^5 A]_{\alpha\beta}$	F21:	$\psi \leftarrow \gamma^5 A \hspace{-0.1cm}/ \chi$
F13:	$A_{\mu} \leftarrow \chi^T \mathbf{C}(\gamma^5 \gamma_{\mu} \psi)$	F31:	$A_{\mu} \leftarrow \psi^T \mathbf{C} \gamma^5 \gamma_{\mu} \chi$
F23:	$\chi \leftarrow \gamma^5 A \psi$	F32:	$\chi_{\alpha} \leftarrow \psi_{\beta} [\gamma^5 A]_{\alpha\beta}$

Table 9.9: Dimension-4 trilinear couplings including one Dirac and one Majorana fermion

$FBF \ (Chibar, \ SP, \ Psi): \ \bar{\chi}\phi(g_S + g_P\gamma_5)\psi$			
F12: $\psi \leftarrow (g_S + g_P \gamma_5) \chi \phi$	F21: $\psi \leftarrow \phi(g_S + g_P \gamma_5) \chi$		
F13: $\phi \leftarrow \chi^T C(g_S + g_P \gamma_5) \psi$	F31: $\phi \leftarrow \psi^T C(g_S + g_P \gamma_5) \chi$		
$F23: \chi \leftarrow \phi(g_S + g_P \gamma_5)\psi$	F32: $\chi \leftarrow (g_S + g_P \gamma_5) \psi \phi$		
FBF (Chibar, VA, Psi): $\bar{\chi} Z (g_V - g_A \gamma_5) \psi$			
F12: $\psi_{\alpha} \leftarrow \chi_{\beta} [Z(-g_V - g_A \gamma_5)]_{\alpha\beta}$	F21: $\psi \leftarrow \mathbb{Z}(-g_V - g_A \gamma_5)\chi$		
F13: $Z_{\mu} \leftarrow \chi^T C \gamma_{\mu} (g_V - g_A \gamma_5) \psi$	F31: $Z_{\mu} \leftarrow \psi^T C \gamma_{\mu} (-g_V - g_A \gamma_5) \chi$		
$F23: \chi \leftarrow \mathbb{Z}(g_V - g_A \gamma_5)]\psi$	F32: $\chi_{\alpha} \leftarrow \psi_{\beta} [Z(g_V - g_A \gamma_5)]_{\alpha\beta}$		

Table 9.10: Combined dimension-4 trilinear fermionic couplings including one Dirac and one Majorana fermion.

FBF (Chibar, S , Chi): $\bar{\chi}_a S \chi_b$			
F12:	$\chi_b \leftarrow \chi_a S$	F21:	$\chi_b \leftarrow S\chi_a$
F13:	$S \leftarrow \chi_a^T \mathbf{C} \chi_b$	F31:	$S \leftarrow \chi_b^T \mathbf{C} \chi_a$
F23:	$\chi_a \leftarrow S\chi_b$	F32:	$\chi_a \leftarrow \chi S_b$
FBF (Chil	par, P , Chi): $\bar{\chi}_a P \gamma_5 \psi$	b	
F12:	$\chi_b \leftarrow \gamma_5 \chi_a P$	F21:	$\chi_b \leftarrow P\gamma_5\chi_a$
F13:	$P \leftarrow \chi_a^T \mathbf{C} \gamma_5 \chi_b$	F31:	$P \leftarrow \chi_b^T \mathbf{C} \gamma_5 \chi_a$
F23:	$\chi_a \leftarrow P\gamma_5\chi_b$	F32:	$\chi_a \leftarrow \gamma_5 \chi_b P$
FBF (Chil	par, V , Chi): $\bar{\chi}_a V \chi_b$		
F12:	$\chi_{b,\alpha} \leftarrow -\chi_{a,\beta} V_{\alpha\beta}$	F21:	$\chi_b \leftarrow -V \chi_a$
F13:	$V_{\mu} \leftarrow \chi_a^T C \gamma_{\mu} \chi_b$	F31:	$V_{\mu} \leftarrow -\chi_b^T \mathbf{C} \gamma_{\mu} \chi_a$
F23:	$\chi_a \leftarrow V \chi_b$	F32:	$\chi_{a,\alpha} \leftarrow \chi_{b,\beta} V_{\alpha\beta}$
FBF (Chibar, A, Chi): $\bar{\chi}_a \gamma^5 A \chi_b$			
F12:	$\chi_{b,\alpha} \leftarrow \chi_{a,\beta} [\gamma^5 A]_{\alpha\beta}$	F21:	$\chi_b \leftarrow \gamma^5 A \chi_a$
F13:	$A_{\mu} \leftarrow \chi_a^T C \gamma^5 \gamma_{\mu} \chi_b$	F31:	$A_{\mu} \leftarrow \chi_b^T C(\gamma^5 \gamma_{\mu} \chi_a)$
F23:	$\chi_a \leftarrow \gamma^5 A \chi_b$	F32:	$\chi_{a,\alpha} \leftarrow \chi_{b,\beta} [\gamma^5 A]_{\alpha\beta}$

Table 9.11: Dimension-4 trilinear couplings of two Majorana fermions

FBF (Chibar, SP , Chi): $\bar{\chi}\phi_a(g_S + g_P\gamma_5)\chi_b$				
F12:	$\chi_b \leftarrow (g_S + g_P \gamma_5) \chi_a \phi$	F21:	$\chi_b \leftarrow \phi(g_S + g_P \gamma_5) \chi_a$	
F13:	$\phi \leftarrow \chi_a^T \mathbf{C}(g_S + g_P \gamma_5) \chi_b$	F31:	$\phi \leftarrow \chi_b^T C(g_S + g_P \gamma_5) \chi_a$	
F23:	$\chi_a \leftarrow \phi(g_S + g_P \gamma_5) \chi_b$	F32:	$\chi_a \leftarrow (g_S + g_P \gamma_5) \chi_b \phi$	
FBF (Chil	FBF (Chibar, VA, Chi): $\bar{\chi}_a Z(g_V - g_A \gamma_5) \chi_b$			
F12:	$\chi_{b,\alpha} \leftarrow \chi_{a,\beta} [Z(-g_V - g_A \gamma_5)]_{\alpha\beta}$	F21:	$\chi_b \leftarrow \mathcal{Z}(-g_V - g_A \gamma_5)]\chi_a$	
F13:	$Z_{\mu} \leftarrow \chi_a^T C \gamma_{\mu} (g_V - g_A \gamma_5) \chi_b$	F31:	$Z_{\mu} \leftarrow \chi_b^T C \gamma_{\mu} (-g_V - g_A \gamma_5) \chi_a$	
F23:	$\chi_a \leftarrow Z(g_V - g_A \gamma_5) \chi_b$	F32:	$\chi_{a,\alpha} \leftarrow \chi_{b,\beta} [Z(g_V - g_A \gamma_5)]_{\alpha\beta}$	

Table 9.12: Combined dimension-4 trilinear fermionic couplings of two Majorana fermions.

$Gauge_Gauge_Gauge: \mathcal{L}_I = gf_{abc}A_a^{\mu}A_b^{\nu}\partial_{\mu}A_{c,\nu}$
$: A_a^{\mu} \leftarrow \mathrm{i} \cdot (-\mathrm{i}g/2) \cdot C_{abc}^{\mu\rho\sigma}(-k_2 - k_3, k_2, k_3) A_{\rho}^b A_{\sigma}^c $
$Aux_Gauge_Gauge: \mathcal{L}_{I} = gf_{abc}X_{a,\mu\nu}(k_{1})(A_{b}^{\mu}(k_{2})A_{c}^{\nu}(k_{3}) - A_{b}^{\nu}(k_{2})A_{c}^{\mu}(k_{3}))$
$F23 \vee F32: X_a^{\mu\nu}(k_2 + k_3) \leftarrow i \cdot g f_{abc}(A_b^{\mu}(k_2) A_c^{\nu}(k_3) - A_b^{\nu}(k_2) A_c^{\mu}(k_3))$
F12 \vee F13: $A_{a,\mu}(k_1 + k_{2/3}) \leftarrow i \cdot g f_{abc} X_{b,\nu\mu}(k_1) A_c^{\nu}(k_{2/3})$
F21 \vee F31: $A_{a,\mu}(k_{2/3} + k_1) \leftarrow i \cdot g f_{abc} A_b^{\nu}(k_{2/3}) X_{c,\mu\nu}(k_1)$

Table 9.13: Dimension-4 Vector Boson couplings with *outgoing* momenta. See (11.1b) and (9.6b) for the definition of the antisymmetric tensor $C^{\mu_1\mu_2\mu_3}(k_1,k_2,k_3)$.

$Scalar_Vector_Vector: \mathcal{L}_I = g\phi V_1^{\mu} V_{2,\mu}$			
$F13: \leftarrow i \cdot g \cdots$	F31:	$\leftarrow i \cdot g \cdots$	
$F12: \leftarrow i \cdot g \cdots$	F21:	$\leftarrow i \cdot g \cdots$	
$F23: \phi \leftarrow i \cdot gV_1^{\mu}V_{2,\mu}$	F32:	$\phi \leftarrow \mathbf{i} \cdot g V_{2,\mu} V_1^{\mu}$	
$Aux_Vector_Vector: \mathcal{L}_I = gX$	$(V_1^{\mu}V_{2,\mu})$		
F13: $\leftarrow i \cdot g \cdots$	F31:	$\leftarrow i \cdot g \cdots$	
$F12: \leftarrow i \cdot g \cdots$	F21:	$\leftarrow i \cdot g \cdots$	
F23: $X \leftarrow \mathbf{i} \cdot gV_1^{\mu}V_{2,\mu}$	F32:	$X \leftarrow \mathbf{i} \cdot g V_{2,\mu} V_1^{\mu}$	
$Aux_Scalar_Vector: \mathcal{L}_I = gX^{\mu}\phi V_{\mu}$			
F13: $\leftarrow i \cdot g \cdots$	F31:	$\leftarrow i \cdot g \cdots$	
$F12: \leftarrow i \cdot g \cdots$	F21:	$\leftarrow i \cdot g \cdots$	
$F23: \leftarrow \mathbf{i} \cdot g \cdots$	F32:	$\leftarrow i \cdot g \cdots$	

Table 9.14: ...

$Scalar_Scalar$: $\mathcal{L}_I = g\phi_1\phi_2\phi_3$					
F13:	$\phi_2 \leftarrow \mathbf{i} \cdot g \phi_1 \phi_3$	F31:	$\phi_2 \leftarrow \mathbf{i} \cdot g \phi_3 \phi_1$		
F12:	$\phi_3 \leftarrow \mathbf{i} \cdot g \phi_1 \phi_2$	F21:	$\phi_3 \leftarrow \mathbf{i} \cdot g \phi_2 \phi_1$		
F23:	$\phi_1 \leftarrow \mathbf{i} \cdot g \phi_2 \phi_3$	F32:	$\phi_1 \leftarrow \mathbf{i} \cdot g \phi_3 \phi_2$		
Aux_Scala	Aux_Scalar_Scalar : $\mathcal{L}_I = gX\phi_1\phi_2$				
F13:	$\leftarrow \mathbf{i} \cdot g \cdots$	F31:	$\leftarrow \mathbf{i} \cdot g \cdots$		
F12:	$\leftarrow \mathbf{i} \cdot g \cdots$	F21:	$\leftarrow \mathbf{i} \cdot g \cdots$		
F23:	$X \leftarrow \mathbf{i} \cdot g\phi_1\phi_2$	F32:	$X \leftarrow \mathbf{i} \cdot g\phi_2\phi_1$		

Table 9.15: ...

Vector_Sce	$alar_Scalar: \mathcal{L}_I = gV^{\mu}\phi_1 i \overleftrightarrow{\partial_{\mu}}\phi_2$
F23:	$V^{\mu}(k_2 + k_3) \leftarrow i \cdot g(k_2^{\mu} - k_3^{\mu})\phi_1(k_2)\phi_2(k_3)$
F32:	$V^{\mu}(k_2 + k_3) \leftarrow i \cdot g(k_2^{\mu} - k_3^{\mu})\phi_2(k_3)\phi_1(k_2)$
F12:	$\phi_2(k_1 + k_2) \leftarrow i \cdot g(k_1^{\mu} + 2k_2^{\mu})V_{\mu}(k_1)\phi_1(k_2)$
F21:	$\phi_2(k_1 + k_2) \leftarrow i \cdot g(k_1^{\mu} + 2k_2^{\mu})\phi_1(k_2)V_{\mu}(k_1)$
F13:	$\phi_1(k_1 + k_3) \leftarrow i \cdot g(-k_1^{\mu} - 2k_3^{\mu})V_{\mu}(k_1)\phi_2(k_3)$
F31:	$\phi_1(k_1 + k_3) \leftarrow i \cdot g(-k_1^{\mu} - 2k_3^{\mu})\phi_2(k_3)V_{\mu}(k_1)$

Table 9.16: ...

Aux_DSca	$lar_DScalar: \mathcal{L}_I = g\chi(i\partial_\mu\phi_1)(i\partial^\mu\phi_2)$
F23:	$\chi(k_2 + k_3) \leftarrow i \cdot g(k_2 \cdot k_3) \phi_1(k_2) \phi_2(k_3)$
F32:	$\chi(k_2 + k_3) \leftarrow i \cdot g(k_3 \cdot k_2) \phi_2(k_3) \phi_1(k_2)$
F12:	$\phi_2(k_1 + k_2) \leftarrow i \cdot g((-k_1 - k_2) \cdot k_2) \chi(k_1) \phi_1(k_2)$
F21:	$\phi_2(k_1 + k_2) \leftarrow i \cdot g(k_2 \cdot (-k_1 - k_2))\phi_1(k_2)\chi(k_1)$
F13:	$\phi_1(k_1 + k_3) \leftarrow i \cdot g((-k_1 - k_3) \cdot k_3) \chi(k_1) \phi_2(k_3)$
F31:	$\phi_1(k_1 + k_3) \leftarrow i \cdot g(k_3 \cdot (-k_1 - k_3))\phi_2(k_3)\chi(k_1)$

Table 9.17: ...

Aux_Vecto	$r_{-}DScalar: \mathcal{L}_{I} = g\chi V_{\mu}(\mathrm{i}\partial^{\mu}\phi)$
F23:	$\chi(k_2 + k_3) \leftarrow i \cdot g k_3^{\mu} V_{\mu}(k_2) \phi(k_3)$
F32:	$\chi(k_2 + k_3) \leftarrow i \cdot g\phi(k_3) k_3^{\mu} V_{\mu}(k_2)$
F12:	$\phi(k_1 + k_2) \leftarrow i \cdot g\chi(k_1)(-k_1 - k_2)^{\mu}V_{\mu}(k_2)$
F21:	$\phi(k_1 + k_2) \leftarrow i \cdot g(-k_1 - k_2)^{\mu} V_{\mu}(k_2) \chi(k_1)$
F13:	$V_{\mu}(k_1 + k_3) \leftarrow i \cdot g(-k_1 - k_3)_{\mu} \chi(k_1) \phi(k_3)$
F31:	$V_{\mu}(k_1 + k_3) \leftarrow i \cdot g(-k_1 - k_3)_{\mu} \phi(k_3) \chi(k_1)$

Table 9.18: ...

with

$$C^{\mu\rho\sigma}(-k_2 - k_3, k_2, k_3) = (g^{\rho\sigma}(k_2^{\mu} - k_3^{\mu}) + g^{\mu\sigma}(2k_3^{\rho} + k_2^{\rho}) - g^{\mu\rho}(2k_2^{\sigma} + k_3^{\sigma}))$$

$$(9.6b)$$

i.e.

$$A^{a,\mu}(k_2 + k_3) = -\frac{\mathrm{i}}{2!} f_{abc} \left((k_2^{\mu} - k_3^{\mu}) A^b(k_2) \cdot A^c(k_3) + (2k_3 + k_2) \cdot A^b(k_2) A^{c,\mu}(k_3) - A^{b,\mu}(k_2) A^c(k_3) \cdot (2k_2 + k_3) \right)$$
(9.6c)



Investigate the rearrangements proposed in [5] for improved numerical stability.

Non-Gauge Vector Couplings

As a basis for the dimension-4 couplings of three vector bosons, we choose "transversal" and "longitudinal" (with respect to the first vector field) tensors that are odd and even under permutation of the second and third argument

$$\mathcal{L}_{T}(V_{1}, V_{2}, V_{3}) = V_{1}^{\mu}(V_{2,\nu} i \overleftrightarrow{\partial_{\mu}} V_{3}^{\nu}) = -\mathcal{L}_{T}(V_{1}, V_{3}, V_{2})$$
(9.7a)

$$\mathcal{L}_L(V_1, V_2, V_3) = (i\partial_\mu V_1^\mu) V_{2,\nu} V_3^\nu = \mathcal{L}_L(V_1, V_3, V_2)$$
(9.7b)

Using partial integration in \mathcal{L}_L , we find the convenient combinations

$$\mathcal{L}_T(V_1, V_2, V_3) + \mathcal{L}_L(V_1, V_2, V_3) = -2V_1^{\mu} i \partial_{\mu} V_{2,\nu} V_3^{\nu}$$
(9.8a)

$$\mathcal{L}_T(V_1, V_2, V_3) - \mathcal{L}_L(V_1, V_2, V_3) = 2V_1^{\mu} V_{2,\nu} i \partial_{\mu} V_3^{\nu}$$
(9.8b)

As an important example, we can rewrite the dimension-4 "anomalous" triple gauge couplings

$$i\mathcal{L}_{TGC}(g_1, \kappa, g_4)/g_{VWW} = g_1 V^{\mu} (W_{\mu\nu}^- W^{+,\nu} - W_{\mu\nu}^+ W^{-,\nu}) + \kappa W_{\mu}^+ W_{\nu}^- V^{\mu\nu} + g_4 W_{\mu}^+ W_{\nu}^- (\partial^{\mu} V^{\nu} + \partial^{\nu} V^{\mu})$$
(9.9)

as

$$\mathcal{L}_{TGC}(g_1, \kappa, g_4) = g_1 \mathcal{L}_T(V, W^-, W^+)$$

$$- \frac{\kappa + g_1 - g_4}{2} \mathcal{L}_T(W^-, V, W^+) + \frac{\kappa + g_1 + g_4}{2} \mathcal{L}_T(W^+, V, W^-)$$

$$- \frac{\kappa - g_1 - g_4}{2} \mathcal{L}_L(W^-, V, W^+) + \frac{\kappa - g_1 + g_4}{2} \mathcal{L}_L(W^+, V, W^-)$$
 (9.10)

CP Violation

$$\mathcal{L}_{\tilde{T}}(V_1, V_2, V_3) = V_{1,\mu}(V_{2,\rho} i \overleftrightarrow{\partial_{\nu}} V_{3,\sigma}) \epsilon^{\mu\nu\rho\sigma} = +\mathcal{L}_T(V_1, V_3, V_2)$$
(9.11a)

$$\mathcal{L}_{\tilde{L}}(V_1, V_2, V_3) = (i\partial_{\mu} V_{1,\nu}) V_{2,\rho} V_{3,\sigma} \epsilon^{\mu\nu\rho\sigma} = -\mathcal{L}_L(V_1, V_3, V_2)$$
(9.11b)

Here the notations \tilde{T} and \tilde{L} are clearly *abuse de langage*, because $\mathcal{L}_{\tilde{L}}(V_1, V_2, V_3)$ is actually the transversal combination, due to the antisymmetry of ϵ . Using partial integration in $\mathcal{L}_{\tilde{L}}$, we could again find combinations

$$\mathcal{L}_{\tilde{T}}(V_1, V_2, V_3) + \mathcal{L}_{\tilde{L}}(V_1, V_2, V_3) = -2V_{1,\mu}V_{2,\nu} i\partial_{\rho}V_{3,\sigma}\epsilon^{\mu\nu\rho\sigma}$$
(9.12a)

	/
Dim4_Vec	$tor_Vector_Vector_T: \mathcal{L}_I = gV_1^{\mu}V_{2,\nu}i\overrightarrow{\partial_{\mu}}V_3^{\nu}$
F23:	$V_1^{\mu}(k_2 + k_3) \leftarrow i \cdot g(k_2^{\mu} - k_3^{\mu}) V_{2,\nu}(k_2) V_3^{\nu}(k_3)$
F32:	$V_1^{\mu}(k_2 + k_3) \leftarrow i \cdot g(k_2^{\mu} - k_3^{\mu})V_3^{\nu}(k_3)V_{2,\nu}(k_2)$
F12:	$V_3^{\mu}(k_1 + k_2) \leftarrow i \cdot g(2k_2^{\nu} + k_1^{\nu})V_{1,\nu}(k_1)V_2^{\mu}(k_2)$
F21:	$V_3^{\mu}(k_1 + k_2) \leftarrow i \cdot g(2k_2^{\nu} + k_1^{\nu})V_2^{\mu}(k_2)V_{1,\nu}(k_1)$
F13:	$V_2^{\mu}(k_1 + k_3) \leftarrow i \cdot g(-k_1^{\nu} - 2k_3^{\nu})V_1^{\nu}(k_1)V_3^{\mu}(k_3)$
F31:	$V_2^{\mu}(k_1 + k_3) \leftarrow i \cdot g(-k_1^{\nu} - 2k_3^{\nu})V_3^{\mu}(k_3)V_1^{\nu}(k_1)$
Dim4_Vec	$tor_Vector_Vector_L$: $\mathcal{L}_I = gi\partial_\mu V_1^\mu V_{2,\nu} V_3^\nu$
F23:	$V_1^{\mu}(k_2 + k_3) \leftarrow i \cdot g(k_2^{\mu} + k_3^{\mu}) V_{2,\nu}(k_2) V_3^{\nu}(k_3)$
F32:	$V_1^{\mu}(k_2 + k_3) \leftarrow i \cdot g(k_2^{\mu} + k_3^{\mu})V_3^{\nu}(k_3)V_{2,\nu}(k_2)$
F12:	$V_3^{\mu}(k_1 + k_2) \leftarrow i \cdot g(-k_1^{\nu}) V_{1,\nu}(k_1) V_2^{\mu}(k_2)$
F21:	$V_3^{\mu}(k_1 + k_2) \leftarrow i \cdot g(-k_1^{\nu}) V_2^{\mu}(k_2) V_{1,\nu}(k_1)$
F13:	$V_2^{\mu}(k_1 + k_3) \leftarrow i \cdot g(-k_1^{\nu})V_1^{\nu}(k_1)V_3^{\mu}(k_3)$
F31:	$V_2^{\mu}(k_1+k_3) \leftarrow \mathbf{i} \cdot g(-k_1^{\nu})V_3^{\mu}(k_3)V_1^{\nu}(k_1)$

Table 9.19: ...

Dim4_Vec	$tor_Vector_Vector_T5: \mathcal{L}_I = gV_{1,\mu}V_{2,\rho}i\overleftrightarrow{\partial_{\nu}}V_{3,\sigma}\epsilon^{\mu\nu\rho\sigma}$
F23:	$V_1^{\mu}(k_2 + k_3) \leftarrow i \cdot g \epsilon^{\mu\nu\rho\sigma}(k_{2,\nu} - k_{3,\nu}) V_{2,\rho}(k_2) V_{3,\sigma}(k_3)$
F32:	$V_1^{\mu}(k_2 + k_3) \leftarrow i \cdot g \epsilon^{\mu\nu\rho\sigma}(k_{2,\nu} - k_{3,\nu}) V_{3,\sigma}(k_3) V_{2,\rho}(k_2)$
F12:	$V_3^{\mu}(k_1 + k_2) \leftarrow i \cdot g \epsilon^{\mu\nu\rho\sigma} (2k_{2,\nu} + k_{1,\nu}) V_{1,\rho}(k_1) V_{2,\sigma}(k_2)$
F21:	$V_3^{\mu}(k_1 + k_2) \leftarrow i \cdot g \epsilon^{\mu\nu\rho\sigma} (2k_{2,\nu} + k_{1,\nu}) V_{2,\sigma}(k_2) V_{1,\rho}(k_1)$
F13:	$V_2^{\mu}(k_1 + k_3) \leftarrow i \cdot g \epsilon^{\mu\nu\rho\sigma}(-k_{1,\nu} - 2k_{3,\nu})V_{1,\rho}(k_1)V_{3,\sigma}(k_3)$
F31:	$V_2^{\mu}(k_1 + k_3) \leftarrow i \cdot g \epsilon^{\mu\nu\rho\sigma}(-k_{1,\nu} - 2k_{3,\nu})V_{3,\sigma}(k_3)V_{1,\rho}(k_1)$
Dim4_Vec	$tor_Vector_Vector_L5: \mathcal{L}_I = gi\partial_{\mu}V_{1,\nu}V_{2,\nu}V_{3,\sigma}\epsilon^{\mu\nu\rho\sigma}$
F23:	$V_1^{\mu}(k_2 + k_3) \leftarrow i \cdot g \epsilon^{\mu\nu\rho\sigma}(k_{2,\nu} + k_{3,\nu}) V_{2,\rho}(k_2) V_{3,\sigma}(k_3)$
F32:	$V_1^{\mu}(k_2 + k_3) \leftarrow i \cdot g \epsilon^{\mu\nu\rho\sigma}(k_{2,\nu} + k_{3,\nu}) V_{2,\rho}(k_2) V_{3,\sigma}(k_3)$
F12:	$V_3^{\mu}(k_1 + k_2) \leftarrow i \cdot g \epsilon^{\mu\nu\rho\sigma}(-k_{1,\nu}) V_{1,\rho}(k_1) V_{2,\sigma}(k_2)$
F21:	$V_3^{\mu}(k_1 + k_2) \leftarrow i \cdot g \epsilon^{\mu\nu\rho\sigma}(-k_{1,\nu}) V_{2,\sigma}(k_2) V_{1,\rho}(k_1)$
F13:	$V_2^{\mu}(k_1 + k_3) \leftarrow i \cdot g \epsilon^{\mu\nu\rho\sigma}(-k_{1,\nu}) V_{1,\rho}(k_1) V_{3,\sigma}(k_3)$
F31:	$V_2^{\mu}(k_1 + k_3) \leftarrow i \cdot g \epsilon^{\mu\nu\rho\sigma}(-k_{1,\nu}) V_{3,\sigma}(k_3) V_{1,\rho}(k_1)$

Table 9.20: ...

$$\mathcal{L}_{\tilde{T}}(V_1, V_2, V_3) - \mathcal{L}_{\tilde{L}}(V_1, V_2, V_3) = -2V_{1,\mu} \mathrm{i}\partial_{\nu} V_{2,\rho} V_{3,\sigma} \epsilon^{\mu\nu\rho\sigma}$$

$$\tag{9.12b}$$

but we don't need them, since

$$i\mathcal{L}_{TGC}(g_5, \tilde{\kappa})/g_{VWW} = g_5 \epsilon_{\mu\nu\rho\sigma} (W^{+,\mu} i \overleftrightarrow{\partial^{\rho}} W^{-,\nu}) V^{\sigma}$$

$$-\frac{\tilde{\kappa}_V}{2} W_{\mu}^{-} W_{\nu}^{+} \epsilon^{\mu\nu\rho\sigma} V_{\rho\sigma} \quad (9.13)$$

is immediately recognizable as

$$\mathcal{L}_{TGC}(g_5, \tilde{\kappa})/g_{VWW} = -ig_5 \mathcal{L}_{\tilde{L}}(V, W^-, W^+) + \tilde{\kappa} \mathcal{L}_{\tilde{T}}(V, W^-, W^+)$$
(9.14)

Dim6_Gauge_Gauge_Gauge:
$$\mathcal{L}_{I} = gF_{1}^{\mu\nu}F_{2,\nu\rho}F_{3,\mu}^{\rho}$$

 $\therefore A_{1}^{\mu}(k_{2}+k_{3}) \leftarrow -i \cdot \Lambda^{\mu\rho\sigma}(-k_{2}-k_{3},k_{2},k_{3})A_{2,\rho}A_{c,\sigma}$

Table 9.21: ...

Dim6_Gav	$uge_Gauge_Gauge_5: \mathcal{L}_I = g/2 \cdot \epsilon^{\mu\nu\lambda\tau} F_{1,\mu\nu} F_{2,\tau\rho} F_{3,\lambda}^{\rho}$
F23:	$A_1^{\mu}(k_2 + k_3) \leftarrow -i \cdot \Lambda_5^{\mu\rho\sigma}(-k_2 - k_3, k_2, k_3) A_{2,\rho} A_{3,\sigma}$
F32:	$A_1^{\mu}(k_2 + k_3) \leftarrow -i \cdot \Lambda_5^{\mu\rho\sigma}(-k_2 - k_3, k_2, k_3) A_{3,\sigma} A_{2,\rho}$
F12:	$A_3^{\mu}(k_1+k_2) \leftarrow -\mathrm{i}\cdot$
F21:	$A_3^{\mu}(k_1+k_2) \leftarrow -\mathrm{i}\cdot$
F13:	$A_2^{\mu}(k_1+k_3) \leftarrow -\mathrm{i}\cdot$
F31:	$A_2^{\mu}(k_1+k_3) \leftarrow -\mathrm{i}\cdot$

Table 9.22: ...

9.1.4 SU(2) Gauge Bosons

An important special case for table 9.13 are the two usual coordinates of SU(2)

$$W_{\pm} = \frac{1}{\sqrt{2}} \left(W_1 \mp i W_2 \right) \tag{9.15}$$

i.e.

$$W_1 = \frac{1}{\sqrt{2}} \left(W_+ + W_- \right) \tag{9.16a}$$

$$W_2 = \frac{i}{\sqrt{2}} \left(W_+ - W_- \right) \tag{9.16b}$$

and

$$W_1^{\mu}W_2^{\nu} - W_2^{\mu}W_1^{\nu} = i\left(W_-^{\mu}W_+^{\nu} - W_+^{\mu}W_-^{\nu}\right) \tag{9.17}$$

Thus the symmtry remains after the change of basis:

$$\epsilon^{abc} W_a^{\mu_1} W_b^{\mu_2} W_c^{\mu_3} = i W_-^{\mu_1} (W_+^{\mu_2} W_3^{\mu_3} - W_3^{\mu_2} W_+^{\mu_3})
+ i W_+^{\mu_1} (W_3^{\mu_2} W_-^{\mu_3} - W_-^{\mu_2} W_3^{\mu_3}) + i W_3^{\mu_1} (W_-^{\mu_2} W_+^{\mu_3} - W_+^{\mu_2} W_-^{\mu_3})$$
(9.18)

9.1.5 Quartic Couplings and Auxiliary Fields

Quartic couplings can be replaced by cubic couplings to a non-propagating auxiliary field. The quartic term should get a negative sign so that it the energy is bounded from below for identical fields. In the language of functional integrals

$$\mathcal{L}_{\phi^4} = -g^2 \phi_1 \phi_2 \phi_3 \phi_4 \Longrightarrow$$

$$\mathcal{L}_{X\phi^2} = X^* X \pm g X \phi_1 \phi_2 \pm g X^* \phi_3 \phi_4 = (X^* \pm g \phi_1 \phi_2)(X \pm g \phi_3 \phi_4) - g^2 \phi_1 \phi_2 \phi_3 \phi_4 \quad (9.19a)$$

and in the language of Feynman diagrams

$$-ig^2 \implies \pm ig \qquad +i \qquad \pm ig \qquad (9.19b)$$

The other choice of signs

$$\mathcal{L}'_{X\phi^2} = -X^*X \pm gX\phi_1\phi_2 \mp gX^*\phi_3\phi_4 = -(X^* \pm g\phi_1\phi_2)(X \mp g\phi_3\phi_4) - g^2\phi_1\phi_2\phi_3\phi_4$$
(9.20)

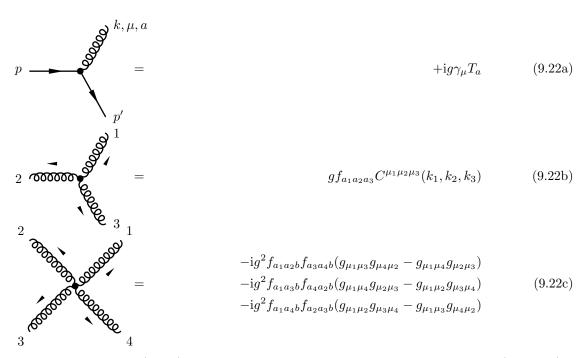


Figure 9.1: Gauge couplings. See (11.1b) for the definition of the antisymmetric tensor $C^{\mu_1\mu_2\mu_3}(k_1,k_2,k_3)$.

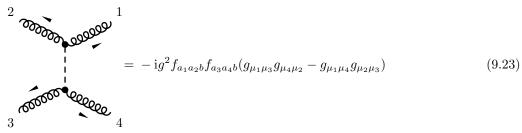


Figure 9.2: Gauge couplings.

can not be extended easily to identical particles and is therefore not used. For identical particles we have

$$\mathcal{L}_{\phi^4} = -\frac{g^2}{4!}\phi^4 \Longrightarrow$$

$$\mathcal{L}_{X\phi^2} = \frac{1}{2}X^2 \pm \frac{g}{2}X\phi^2 \pm \frac{g}{2}X\phi^2 = \frac{1}{2}\left(X \pm \frac{g}{2}\phi^2\right)\left(X \pm \frac{g}{2}\phi^2\right) - \frac{g^2}{4!}\phi^4 \quad (9.21)$$

Explain the factor 1/3 in the functional setting and its relation to the three diagrams in the graphical setting?

Quartic Gauge Couplings

The three crossed versions of figure 9.2 reproduces the quartic coupling in figure 9.1, because

$$-ig^{2}f_{a_{1}a_{2}b}f_{a_{3}a_{4}b}(g_{\mu_{1}\mu_{3}}g_{\mu_{4}\mu_{2}} - g_{\mu_{1}\mu_{4}}g_{\mu_{2}\mu_{3}})$$

$$= (igf_{a_{1}a_{2}b}T_{\mu_{1}\mu_{2},\nu_{1}\nu_{2}})\left(\frac{ig^{\nu_{1}\nu_{3}}g^{\nu_{2}\nu_{4}}}{2}\right)(igf_{a_{3}a_{4}b}T_{\mu_{3}\mu_{4},\nu_{3}\nu_{4}}) \quad (9.24)$$

with $T_{\mu_1\mu_2,\mu_3\mu_4} = g_{\mu_1\mu_3}g_{\mu_4\mu_2} - g_{\mu_1\mu_4}g_{\mu_2\mu_3}$.

9.1.6 Gravitinos and supersymmetric currents

In supergravity theories there is a fermionic partner of the graviton, the gravitino. Therefore we have introduced the Lorentz type Vectorspinor.

GBG (Fermbar, MOM, Ferm): $\bar{\psi}_1(i\partial \!\!\!/ \pm m)\phi\psi_2$				
F12:	$\psi_2 \leftarrow -(k \mp m)\psi_1 S$	F21:	$\psi_2 \leftarrow -S(\not k \mp m)\psi_1$	
F13:	$S \leftarrow \psi_1^T \mathbf{C}(k \pm m) \psi_2$	F31:	$S \leftarrow \psi_2^T \mathbf{C}(-(\not k \mp m)\psi_1)$	
F23:	$\psi_1 \leftarrow S(\not k \pm m)\psi_2$	F32:	$\psi_1 \leftarrow (\not k \pm m) \psi_2 S$	
GBG (Ferr	mbar, MOM5, Ferm): $\bar{\psi}_1(i\partial \!\!\!/ \pm$	$m)\phi\gamma^5$	ψ_2	
F12:	$\psi_2 \leftarrow (k \pm m) \gamma^5 \psi_1 P$	F21:	$\psi_2 \leftarrow P(k \pm m) \gamma^5 \psi_1$	
F13:	$P \leftarrow \psi_1^T \mathbf{C}(k \pm m) \gamma^5 \psi_2$	F31:	$P \leftarrow \psi_2^T \mathbf{C}(k \pm m) \gamma^5 \psi_1$	
F23:	$\psi_1 \leftarrow P(k \pm m) \gamma^5 \psi_2$	F32:	$\psi_1 \leftarrow (k \pm m) \gamma^5 \psi_2 P$	
GBG (Ferr	mbar, MOML, Ferm): $\bar{\psi}_1(i\partial \!\!\!/) \pm$	$m)\phi(1$	$(-\gamma^5)\psi_2$	
F12:	$\psi_2 \leftarrow -(1-\gamma^5)(k \mp m)\psi_1 \phi$	F21:	$\psi_2 \leftarrow -\phi(1-\gamma^5)(\not k \mp m)\psi_1$	
F13:	$\phi \leftarrow \psi_1^T \mathbf{C}(k \pm m)(1 - \gamma^5)\psi_2$	F31:	$\phi \leftarrow \psi_2^T C(1 - \gamma^5)(-(\cancel{k} \mp m)\psi_1)$	
F23:	$\psi_1 \leftarrow \phi(\not k \pm m)(1 - \gamma^5)\psi_2$	F32:	$\psi_1 \leftarrow (\not k \pm m)(1 - \gamma^5)\psi_2 \phi$	
GBG (Ferr	mbar, LMOM, Ferm): $\bar{\psi}_1\phi(1-\bar{\psi}_1)$	$-\gamma^5)(\mathrm{i}\phi$	$(\theta\pm m)\psi_2$	
F12:	$\psi_2 \leftarrow -(\cancel{k} \mp m)\psi_1(1-\gamma^5)\phi$	F21:	$\psi_2 \leftarrow -\phi(k \mp m)(1 - \gamma^5)\psi_1$	
F13:	$\phi \leftarrow \psi_1^T \mathbf{C} (1 - \gamma^5) (\mathbf{k} \pm m) \psi_2$	F31:	$\phi \leftarrow \psi_2^T \mathbf{C}(-(k \mp m)(1 - \gamma^5)\psi_1)$	
F23:	$\psi_1 \leftarrow \phi(1 - \gamma^5)(\not k \pm m)\psi_2$	F32:	$\psi_1 \leftarrow (1 - \gamma^5)(\not k \pm m)\psi_2 \phi$	
GBG (Fermbar, VMOM, Ferm): $\bar{\psi}_1 i \partial_{\alpha} V_{\beta} [\gamma^{\alpha}, \gamma^{\beta}] \psi_2$			$^{eta}]\psi_2$	
F12:	$\psi_2 \leftarrow -[k, \gamma^\alpha] \psi_1 V_\alpha$	F21:	$\psi_2 \leftarrow -[k, V]\psi_1$	
F13:	$V_{\alpha} \leftarrow \psi_1^T \mathbf{C}[k, \gamma_{\alpha}] \psi_2$	F31:	$V_{\alpha} \leftarrow \psi_2^T \mathbf{C}(-[k, \gamma_{\alpha}] \psi_1)$	
F23:	$\psi_1 \leftarrow] \not k, \not V] \psi_2$	F32:	$\psi_1 \leftarrow [k, \gamma^{\alpha}] \psi_2 V_{\alpha}$	

Table 9.23: Combined dimension-4 trilinear fermionic couplings including a momentum. Ferm stands for Psi and Chi. The case of MOMR is identical to MOML if one substitutes $1 + \gamma^5$ for $1 - \gamma^5$, as well as for LMOM and RMOM. The mass term forces us to keep the chiral projector always on the left after "inverting the line" for MOML while on the right for LMOM.

GBBG (Fermbar, S2LR, Ferm): $\bar{\psi}_1 S_1 S_2$	$(g_L P_L + g_R P_R)\psi_2$
F123 F213 F132 F231 F312 F321:	$\psi_2 \leftarrow S_1 S_2 (g_R P_L + g_L P_R) \psi_1$
F423 F243 F432 F234 F342 F324:	$\psi_1 \leftarrow S_1 S_2 (g_L P_L + g_R P_R) \psi_2$
F134 F143 F314:	$S_1 \leftarrow \psi_1^T C S_2 (g_L P_L + g_R P_R) \psi_2$
F124 F142 F214:	$S_2 \leftarrow \psi_1^T C S_1 (g_L P_L + g_R P_R) \psi_2$
F413 F431 F341:	$S_1 \leftarrow \psi_2^T C S_2 (g_R P_L + g_L P_R) \psi_1$
F412 F421 F241:	$S_2 \leftarrow \psi_2^T C S_1 (g_R P_L + g_L P_R) \psi_1$
GBBG (Fermbar, S2, Ferm): $\bar{\psi}_1 S_1 S_2 \gamma^5 \psi$	'2
F123 F213 F132 F231 F312 F321:	$\psi_2 \leftarrow S_1 S_2 \gamma^5 \psi_1$
F423 F243 F432 F234 F342 F324:	$\psi_1 \leftarrow S_1 S_2 \gamma^5 \psi_2$
F134 F143 F314:	$S_1 \leftarrow \psi_1^T C S_2 \gamma^5 \psi_2$
F124 F142 F214:	$S_2 \leftarrow \psi_1^T C S_1 \gamma^5 \psi_2$
F413 F431 F341:	$S_1 \leftarrow \psi_2^T C S_2 \gamma^5 \psi_1$
F412 F421 F241:	$S_2 \leftarrow \psi_2^T C S_1 \gamma^5 \psi_1$
GBBG (Fermbar, V2, Ferm): $\bar{\psi}_1[V_1, V_2]$	ψ_2
F123 F213 F132 F231 F312 F321:	$\psi_2 \leftarrow -[V_1, V_2]\psi_1$
F423 F243 F432 F234 F342 F324:	$\psi_1 \leftarrow [V_1, V_2]\psi_2$
F134 F143 F314:	$V_{1\alpha} \leftarrow \psi_1^T C[\gamma_\alpha, V_2] \psi_2$
F124 F142 F214:	$V_{2\alpha} \leftarrow \psi_1^T C(-[\gamma_\alpha, V_1]) \psi_2$
F413 F431 F341:	$V_{1\alpha} \leftarrow \psi_2^T C(-[\gamma_\alpha, V_2])\psi_1$
F412 F421 F241:	$V_{2\alpha} \leftarrow \psi_2^T C[\gamma_\alpha, V_1] \psi_1$

Table 9.24: Vertices with two fermions (Ferm stands for Psi and Chi, but not for Grav) and two bosons (two scalars, scalar/vector, two vectors) for the BRST transformations. Part I

GBBG (Fermbar, SV, Ferm): $\bar{\psi}_1 V S \psi_2$	
F123 F213 F132 F231 F312 F321:	$\psi_2 \leftarrow -VS\psi_1$
F423 F243 F432 F234 F342 F324:	$\psi_1 \leftarrow VS\psi_2$
F134 F143 F314:	$V_{\alpha} \leftarrow \psi_1^T C \gamma_{\alpha} S \psi_2$
F124 F142 F214:	$S \leftarrow \psi_1^T C V \psi_2$
F413 F431 F341:	$V_{\alpha} \leftarrow \psi_2^T C(-\gamma_{\alpha} S \psi_1)$
F412 F421 F241:	$S \leftarrow \psi_2^T C(-V \psi_1)$
GBBG (Fermbar, PV, Ferm): $\bar{\psi}_1 V \gamma^5 P \psi$	2
F123 F213 F132 F231 F312 F321:	$\psi_2 \leftarrow V \gamma^5 P \psi_1$
F423 F243 F432 F234 F342 F324:	$\psi_1 \leftarrow V \gamma^5 P \psi_2$
F134 F143 F314:	$V_{\alpha} \leftarrow \psi_1^T C \gamma_{\alpha} \gamma^5 P \psi_2$
F124 F142 F214:	$P \leftarrow \psi_1^T C V \gamma^5 \psi_2$
F413 F431 F341:	$V_{\alpha} \leftarrow \psi_2^T C \gamma_{\alpha} \gamma^5 P \psi_1$
F412 F421 F241:	$P \leftarrow \psi_2^T C V \gamma^5 \psi_1$
GBBG (Fermbar, $S(L/R)V$, Ferm): $\bar{\psi}_1V$	$r(1 \mp \gamma^5)\phi\psi_2$
F123 F213 F132 F231 F312 F321:	$\psi_2 \leftarrow -V(1 \pm \gamma^5)\phi\psi_1$
F423 F243 F432 F234 F342 F324:	$\psi_1 \leftarrow V(1 \mp \gamma^5)\phi\psi_2$
F134 F143 F314:	$V_{\alpha} \leftarrow \psi_1^T C \gamma_{\alpha} (1 \mp \gamma^5) \phi \psi_2$
F124 F142 F214:	$\phi \leftarrow \psi_1^T C V (1 \mp \gamma^5) \psi_2$
F413 F431 F341:	$V_{\alpha} \leftarrow \psi_2^T C \gamma_{\alpha} (-(1 \pm \gamma^5) \phi \psi_1)$
F412 F421 F241:	$\phi \leftarrow \psi_2^T C V(-(1 \pm \gamma^5)\psi_1)$

Table 9.25: Vertices with two fermions (Ferm stands for Psi and Chi, but not for Grav) and two bosons (two scalars, scalar/vector, two vectors) for the BRST transformations. Part II

GBG (Gravbar, POT, Psi): $\bar{\psi}_{\mu}S\gamma^{\mu}\psi$			
F12:	$\psi \leftarrow -\gamma^{\mu}\psi_{\mu}S$	F21:	$\psi \leftarrow -S\gamma^{\mu}\psi_{\mu}$
F13:	$S \leftarrow \psi_{\mu}^T \mathbf{C} \gamma^{\mu} \psi$	F31:	$S \leftarrow \psi^T \mathbf{C}(-\gamma^\mu) \psi_\mu$
F23:	$\psi_{\mu} \leftarrow S \gamma_{\mu} \psi$	F32:	$\psi_{\mu} \leftarrow \gamma_{\mu} \psi S$
GBG (Gra	$vbar, S, Psi$): $\bar{\psi}_{\mu} k_S S \gamma^{\mu} \psi$		
F12:	$\psi \leftarrow \gamma^{\mu} k_S \psi_{\mu} S$	F21:	$\psi \leftarrow S \gamma^{\mu} k_S \psi_{\mu}$
F13:	$S \leftarrow \psi_{\mu}^T \mathbf{C} k_S \gamma^{\mu} \psi$	F31:	$S \leftarrow \psi^T C \gamma^\mu k_S \psi_\mu$
F23:	$\psi_{\mu} \leftarrow S k_S \gamma_{\mu} \psi$	F32:	$\psi_{\mu} \leftarrow k_S \gamma_{\mu} \psi S$
GBG (Gra	$vbar, P, Psi$): $\bar{\psi}_{\mu} k_P P \gamma^{\mu} \gamma_5$	ψ	
F12:	$\psi \leftarrow \gamma^{\mu} k_{P} \gamma_{5} \psi_{\mu} P$	F21:	$\psi \leftarrow P \gamma^{\mu} k_P \gamma_5 \psi_{\mu}$
F13:	$P \leftarrow \psi_{\mu}^T \mathbf{C} k_P \gamma^{\mu} \gamma_5 \psi$	F31:	$P \leftarrow \psi^T C \gamma^\mu k_P \gamma_5 \psi_\mu$
F23:	$\psi_{\mu} \leftarrow P k_P \gamma_{\mu} \gamma_5 \psi$	F32:	$\psi_{\mu} \leftarrow k_P \gamma_{\mu} \gamma_5 \psi P$
GBG (Gravbar, V, Psi): $\bar{\psi}_{\mu}[\not k_V, \not V] \gamma^{\mu} \gamma^5 \psi$			
F12:	$\psi \leftarrow \gamma^5 \gamma^\mu [k_V, \gamma^\alpha] \psi_\mu V_\alpha$	F21:	$\psi \leftarrow \gamma^5 \gamma^\mu [\rlap/k_V, \rlap/V] \psi_\mu$
F13:	$V_{\mu} \leftarrow \psi_{\rho}^{T} \mathbf{C}[k_{V}, \gamma_{\mu}] \gamma^{\rho} \gamma^{5} \psi$	F31:	$V_{\mu} \leftarrow \psi^T C \gamma^5 \gamma^{\rho} [k_V, \gamma_{\mu}] \psi_{\rho}$
F23:	$\psi_{\mu} \leftarrow [\rlap/k_V, \rlap/V] \gamma_{\mu} \gamma^5 \psi$	F32:	$\psi_{\mu} \leftarrow [\rlap/k_V, \gamma^{\alpha}] \gamma_{\mu} \gamma^5 \psi V_{\alpha}$

Table 9.26: Dimension-5 trilinear couplings including one Dirac, one Gravitino fermion and one additional particle. The option POT is for the coupling of the supersymmetric current to the derivative of the quadratic terms in the superpotential.

$GBG\ (Psibar,\ POT,\ Grav):\ \bar{\psi}\gamma^{\mu}S\psi_{\mu}$			
F12:	$\psi_{\mu} \leftarrow -\gamma_{\mu} \psi S$	F21:	$\psi_{\mu} \leftarrow -S\gamma_{\mu}\psi$
F13:	$S \leftarrow \psi^T \mathbf{C} \gamma^\mu \psi_\mu$	F31:	$S \leftarrow \psi_{\mu}^{T} \mathbf{C}(-\gamma^{\mu}) \psi$
F23:	$\psi \leftarrow S \gamma^{\mu} \psi_{\mu}$	F32:	$\psi \leftarrow \gamma^{\mu} \psi_{\mu} S$
GBG (Psi	bar, S , $Grav$): $\bar{\psi}\gamma^{\mu}k_{S}S\psi_{\mu}$		
F12:	$\psi_{\mu} \leftarrow k_S \gamma_{\mu} \psi S$	F21:	$\psi_{\mu} \leftarrow S k_S \gamma_{\mu} \psi$
F13:	$S \leftarrow \psi^T C \gamma^\mu k_S \psi_\mu$	F31:	$S \leftarrow \psi_{\mu}^T \mathbf{C} k_S \gamma^{\mu} \psi$
F23:	$\psi \leftarrow S \gamma^{\mu} k_S \psi_{\mu}$	F32:	$\psi \leftarrow \gamma^{\mu} k_S \psi_{\mu} S$
GBG (Psi	bar, P , $Grav$): $\bar{\psi}\gamma^{\mu}\gamma^{5}Pk_{P}\psi$	μ	
F12:	$\psi_{\mu} \leftarrow -k_P \gamma_{\mu} \gamma^5 \psi P$	F21:	$\psi_{\mu} \leftarrow -P k_P \gamma_{\mu} \gamma^5 \psi$
F13:	$P \leftarrow \psi^T C \gamma^\mu \gamma^5 k_P \psi_\mu$	F31:	$P \leftarrow -\psi_{\mu}^{T} \mathbf{C} k_{P} \gamma^{\mu} \gamma_{5} \psi$
F23:	$\psi \leftarrow P \gamma^{\mu} \gamma^5 k_P \psi_{\mu}$	F32:	$\psi \leftarrow \gamma^{\mu} \gamma^5 k_P \psi_{\mu} P$
$GBG \ (Psibar, \ V, \ Grav): \ \bar{\psi}\gamma^5\gamma^{\mu}[\rlap/k_V, \rlap/V]\psi_{\mu}$			
F12:	$\psi_{\mu} \leftarrow [k_V, \gamma^{\alpha}] \gamma_{\mu} \gamma^5 \psi V_{\alpha}$	F21:	$\psi_{\mu} \leftarrow [\not k_V, \not V] \gamma_{\mu} \gamma^5 \psi$
F13:	$V_{\mu} \leftarrow \psi^T C \gamma^5 \gamma^{\rho} [k_V, \gamma_{\mu}] \psi_{\rho}$	F31:	$V_{\mu} \leftarrow \psi_{\rho}^{T} \mathbf{C}[k_{V}, \gamma_{\mu}] \gamma^{\rho} \gamma^{5} \psi$
F23:	$\psi \leftarrow \gamma^5 \gamma^\mu [k_V, V] \psi_\mu$	F32:	$\psi \leftarrow \gamma^5 \gamma^\mu [k_V, \gamma^\alpha] \psi_\mu V_\alpha$

Table 9.27: Dimension-5 trilinear couplings including one conjugated Dirac, one Gravitino fermion and one additional particle.

GBG (Gravbar, POT, Chi): $\bar{\psi}_{\mu}S\gamma^{\mu}\chi$			
F12:	$\chi \leftarrow -\gamma^{\mu}\psi_{\mu}S$	F21:	$\chi \leftarrow -S\gamma^{\mu}\psi_{\mu}$
F13:	$S \leftarrow \psi_{\mu}^T \mathbf{C} \gamma^{\mu} \chi$	F31:	$S \leftarrow \chi^T \mathbf{C}(-\gamma^\mu) \psi_\mu$
F23:	$\psi_{\mu} \leftarrow S \gamma_{\mu} \chi$	F32:	$\psi_{\mu} \leftarrow \gamma_{\mu} \chi S$
GBG (Gra	$vbar, S, Chi): \bar{\psi}_{\mu} k_S S \gamma^{\mu} \chi$		
F12:	$\chi \leftarrow \gamma^{\mu} k_S \psi_{\mu} S$	F21:	$\chi \leftarrow S \gamma^{\mu} k_S \psi_{\mu}$
F13:	$S \leftarrow \psi_{\mu}^T \mathbf{C} k_S \gamma^{\mu} \chi$	F31:	$S \leftarrow \chi^T \mathbf{C} \gamma^\mu / \!\! k_S \psi_\mu$
F23:	$\psi_{\mu} \leftarrow S k_S \gamma_{\mu} \chi$	F32:	$\psi_{\mu} \leftarrow k_S \gamma_{\mu} \chi S$
GBG (Gra	$vbar, P, Chi$): $\bar{\psi}_{\mu} k_P P \gamma^{\mu} \gamma$	5χ	
F12:	$\chi \leftarrow \gamma^{\mu} k_P \gamma_5 \psi_{\mu} P$	F21:	$\chi \leftarrow P \gamma^\mu k_P \gamma_5 \psi_\mu$
F13:	$P \leftarrow \psi_{\mu}^{T} \mathbf{C} k_{P} \gamma^{\mu} \gamma_{5} \chi$	F31:	$P \leftarrow \chi^T C \gamma^\mu k_P \gamma_5 \psi_\mu$
F23:	$\psi_{\mu} \leftarrow P k_P \gamma_{\mu} \gamma_5 \chi$	F32:	$\psi_{\mu} \leftarrow k_P \gamma_{\mu} \gamma_5 \chi P$
GBG (Gravbar, V, Chi): $\bar{\psi}_{\mu}[k_V, V]\gamma^{\mu}\gamma^5\chi$			
F12:	$\chi \leftarrow \gamma^5 \gamma^\mu [k_V, \gamma^\alpha] \psi_\mu V_\alpha$	F21:	$\chi \leftarrow \gamma^5 \gamma^\mu [k_V, V] \psi_\mu$
F13:	$V_{\mu} \leftarrow \psi_{\rho}^{T} \mathbf{C}[k_{V}, \gamma_{\mu}] \gamma^{\rho} \gamma^{5} \chi$	F31:	$V_{\mu} \leftarrow \chi^T C \gamma^5 \gamma^{\rho} [k_V, \gamma_{\mu}] \psi_{\rho}$
F23:	$\psi_{\mu} \leftarrow [\not k_V, \not V] \gamma_{\mu} \gamma^5 \chi$	F32:	$\psi_{\mu} \leftarrow [k_V, \gamma^{\alpha}] \gamma_{\mu} \gamma^5 \chi V_{\alpha}$

Table 9.28: Dimension-5 trilinear couplings including one Majorana, one Gravitino fermion and one additional particle. The table is essentially the same as the one with the Dirac fermion and only written for the sake of completeness.

GBG (Chi	bar, POT, Grav): $\bar{\chi}\gamma^{\mu}S\psi_{\mu}$		
F12:	$\psi_{\mu} \leftarrow -\gamma_{\mu} \chi S$	F21:	$\psi_{\mu} \leftarrow -S\gamma_{\mu}\chi$
F13:	$S \leftarrow \chi^T \mathbf{C} \gamma^\mu \psi_\mu$	F31:	$S \leftarrow \psi_{\mu}^T \mathbf{C}(-\gamma^{\mu}) \chi$
F23:	$\chi \leftarrow S \gamma^{\mu} \psi_{\mu}$	F32:	$\chi \leftarrow \gamma^{\mu} \psi_{\mu} S$
GBG (Chi	$bar, S, Grav): \bar{\chi}\gamma^{\mu} k_S S\psi_{\mu}$		
F12:	$\psi_{\mu} \leftarrow k_S \gamma_{\mu} \chi S$	F21:	$\psi_{\mu} \leftarrow S k_S \gamma_{\mu} \chi$
F13:	$S \leftarrow \chi^T C \gamma^\mu k_S \psi_\mu$	F31:	$S \leftarrow \psi_{\mu}^T \mathbf{C} k_S \gamma^{\mu} \chi$
F23:	$\chi \leftarrow S \gamma^{\mu} k_S \psi_{\mu}$	F32:	$\chi \leftarrow \gamma^{\mu} k_S \psi_{\mu} S$
GBG (Chi	$bar, P, Grav$): $\bar{\chi}\gamma^{\mu}\gamma^5 P k_P \psi$	'μ	
F12:	$\psi_{\mu} \leftarrow -k_P \gamma_{\mu} \gamma^5 \chi P$	F21:	$\psi_{\mu} \leftarrow -P k_P \gamma_{\mu} \gamma^5 \chi$
F13:	$P \leftarrow \chi^T C \gamma^\mu \gamma^5 k_P \psi_\mu$	F31:	$P \leftarrow -\psi_{\mu}^T \mathbf{C} k_P \gamma^{\mu} \gamma_5 \chi$
F23:	$\chi \leftarrow P \gamma^{\mu} \gamma^5 k_P \psi_{\mu}$	F32:	$\chi \leftarrow \gamma^{\mu} \gamma^5 k_P \psi_{\mu} P$
GBG (Chibar, V , Grav): $\bar{\chi}\gamma^5\gamma^{\mu}[k_V,V]\psi_{\mu}$			
F12:	$\psi_{\mu} \leftarrow [k_V, \gamma^{\alpha}] \gamma_{\mu} \gamma^5 \chi V_{\alpha}$	F21:	$\psi_{\mu} \leftarrow [k_V, V] \gamma_{\mu} \gamma^5 \chi$
F13:	$V_{\mu} \leftarrow \chi^T C \gamma^5 \gamma^{\rho} [k_V, \gamma_{\mu}] \psi_{\rho}$	F31:	$V_{\mu} \leftarrow \psi_{\rho}^{T} \mathbf{C}[k_{V}, \gamma_{\mu}] \gamma^{\rho} \gamma^{5} \chi$
F23:	$\chi \leftarrow \gamma^5 \gamma^\mu [k_V, V] \psi_\mu$	F32:	$\chi \leftarrow \gamma^5 \gamma^\mu [k_V, \gamma^\alpha] \psi_\mu V_\alpha$

Table 9.29: Dimension-5 trilinear couplings including one conjugated Majorana, one Gravitino fermion and one additional particle. This table is not only the same as the one with the conjugated Dirac fermion but also the same part of the Lagrangian density as the one with the Majorana particle on the right of the gravitino.

GBBG (Gravbar, S2, Psi): $\bar{\psi}_{\mu}S_1S_2\gamma^{\mu}\psi$	
F123 F213 F132 F231 F312 F321:	$\psi \leftarrow -\gamma^{\mu} S_1 S_2 \psi_{\mu}$
F423 F243 F432 F234 F342 F324:	$\psi_{\mu} \leftarrow \gamma_{\mu} S_1 S_2 \psi$
F134 F143 F314:	$S_1 \leftarrow \psi_\mu^T C S_2 \gamma^\mu \psi$
F124 F142 F214:	$S_2 \leftarrow \psi_\mu^T C S_1 \gamma^\mu \psi$
F413 F431 F341:	$S_1 \leftarrow -\psi^T C S_2 \gamma^\mu \psi_\mu$
F412 F421 F241:	$S_2 \leftarrow -\psi^T C S_1 \gamma^\mu \psi_\mu$
GBBG (Gravbar, SV, Psi): $\bar{\psi}_{\mu}SV\gamma^{\mu}\gamma^5\psi$	
F123 F213 F132 F231 F312 F321:	$\psi \leftarrow \gamma^5 \gamma^\mu S V \psi_\mu$
F423 F243 F432 F234 F342 F324:	$\psi_{\mu} \leftarrow V S \gamma_{\mu} \gamma^5 \psi$
F134 F143 F314:	$S \leftarrow \psi_{\mu}^T C V \gamma^{\mu} \gamma^5 \psi$
F124 F142 F214:	$V_{\mu} \leftarrow \psi_{\rho}^{T} C S \gamma_{\mu} \gamma^{\rho} \gamma^{5} \psi$
F413 F431 F341:	$S \leftarrow \psi^T C \gamma^5 \gamma^\mu V \psi_\mu$
F412 F421 F241:	$V_{\mu} \leftarrow \psi^T C S \gamma^5 \gamma^{\rho} \gamma_{\mu} \psi_{\rho}$
GBBG (Gravbar, PV, Psi): $\bar{\psi}_{\mu}PV\gamma^{\mu}\psi$	
F123 F213 F132 F231 F312 F321:	$\psi \leftarrow \gamma^{\mu} P V \psi_{\mu}$
F423 F243 F432 F234 F342 F324:	$\psi_{\mu} \leftarrow V P \gamma_{\mu} \psi$
F134 F143 F314:	$P \leftarrow \psi_{\mu}^T C V \gamma^{\mu} \psi$
F124 F142 F214:	$V_{\mu} \leftarrow \psi_{\rho}^T C P \gamma_{\mu} \gamma^{\rho} \psi$
F413 F431 F341:	$P \leftarrow \psi^T C \gamma^\mu V \psi_\mu$
F412 F421 F241:	$V_{\mu} \leftarrow \psi^T C P \gamma^{\rho} \gamma_{\mu} \psi_{\rho}$
GBBG (Gravbar, V2, Psi): $\bar{\psi}_{\mu}f_{abc}[V^a, V^b]$	$^{b}]\gamma^{\mu}\gamma^{5}\psi$
F123 F213 F132 F231 F312 F321:	$\psi \leftarrow f_{abc} \gamma^5 \gamma^{\mu} [V^a, V^b] \psi_{\mu}$
F423 F243 F432 F234 F342 F324:	$\psi_{\mu} \leftarrow f_{abc}[V^a, V^b] \gamma_{\mu} \gamma^5 \psi$
F134 F143 F314 F124 F142 F214:	$V_{\mu}^{a} \leftarrow \psi_{\rho}^{T} C f_{abc}[\gamma_{\mu}, V^{b}] \gamma^{\rho} \gamma^{5} \psi$
F413 F431 F341 F412 F421 F241:	$V_{\mu}^{a} \leftarrow \overline{\psi^{T} C f_{abc} \gamma^{5} \gamma^{\rho} [\gamma_{\mu}, V^{b}] \psi_{\rho}}$

Table 9.30: Dimension-5 trilinear couplings including one Dirac, one Gravitino fermion and two additional bosons. In each lines we list the fusion possibilities with the same order of the fermions, but the order of the bosons is arbitrary (of course, one has to take care of this order in the mapping of the wave functions in *fusion*).

GBBG (Psibar, S2, Grav): $\bar{\psi}$	$S_1 S_2 \gamma \mu_2 / 1$	
	•	
F123 F213 F132 F231 F		$\psi_{\mu} \leftarrow -\gamma_{\mu} S_1 S_2 \psi$
F423 F243 F432 F234 F		$\psi \leftarrow \gamma^{\mu} S_1 S_2 \psi_{\mu}$
F134 F	143 F314:	$S_1 \leftarrow \psi^T C S_2 \gamma^\mu \psi_\mu$
F124 F	142 F214:	$S_2 \leftarrow \psi^T C S_1 \gamma^\mu \psi_\mu$
F413 F.	431 F341:	$S_1 \leftarrow -\psi_\mu^T C S_2 \gamma^\mu \psi$
F412 F	421 F241:	$S_2 \leftarrow -\psi_\mu^T C S_1 \gamma^\mu \psi$
GBBG (Psibar, SV, Grav): v	$\bar{\psi}S\gamma^{\mu}\gamma^5V\psi_{\mu}$	
F123 F213 F132 F231 F	312 F321:	$\psi_{\mu} \leftarrow V S \gamma^5 \gamma^{\mu} \psi$
F423 F243 F432 F234 F	342 F324:	$\psi \leftarrow \gamma^{\mu} \gamma^5 S V \psi_{\mu}$
F134 F	143 F314:	$S \leftarrow \psi^T C \gamma^\mu \gamma^5 V \psi$
F124 F	142 F214:	$V_{\mu} \leftarrow \psi^T C \gamma^{\rho} \gamma^5 S \gamma_{\mu} \psi_{\rho}$
F413 F	431 F341:	$S \leftarrow \psi_{\mu}^T C V \gamma^5 \gamma^{\mu} \psi$
F412 F	421 F241:	$V_{\mu} \leftarrow \psi_{\rho}^{T} C S \gamma_{\mu} \gamma^{5} \gamma^{\rho} \psi$
GBBG (Psibar, PV, Grav): v	$ar{\psi} P \gamma^{\mu} V \psi_{\mu}$	
F123 F213 F132 F231 F	312 F321:	$\psi_{\mu} \leftarrow V \gamma_{\mu} P \psi$
F423 F243 F432 F234 F	342 F324:	$\psi \leftarrow \gamma^{\mu} V P \psi_{\mu}$
F134 F	143 F314:	$P \leftarrow \psi^T C \gamma^\mu V \psi_\mu$
F124 F	142 F214:	$V_{\mu} \leftarrow \psi^T C P \gamma^{\rho} \gamma_{\mu} \psi_{\rho}$
F413 F	431 F341:	$P \leftarrow \psi_{\mu}^T C V \gamma^{\mu} \psi$
F412 F	421 F241:	$V_{\mu} \leftarrow \psi_{\rho}^T C P \gamma_{\mu} \gamma^{\rho} \psi$
$GBBG \ (Psibar, \ V2, \ Grav): \bar{\psi}$	$\bar{b}f_{abc}\gamma^5\gamma^\mu [V^a]$	$[\psi,V^b]\overline{\psi_\mu}$
F123 F213 F132 F231 F	312 F321:	$\psi_{\mu} \leftarrow f_{abc}[V^a, V^b] \gamma_{\mu} \gamma^5 \psi$
F423 F243 F432 F234 F	342 F324:	$\psi \leftarrow f_{abc} \gamma^5 \gamma^{\mu} [V^a, V^b] \psi_{\mu}$
F134 F143 F314 F124 F	142 F214:	$V_{\mu}^{a} \leftarrow \psi^{T} C f_{abc} \gamma^{5} \gamma^{\rho} [\gamma_{\mu}, V^{b}] \psi_{\rho}$
F413 F431 F341 F412 F	421 F241:	$V_{\mu}^{a} \leftarrow \psi_{\rho}^{T} C f_{abc}[\gamma_{\mu}, V^{b}] \gamma^{\rho} \gamma^{5} \psi$

Table 9.31: Dimension-5 trilinear couplings including one conjugated Dirac, one Gravitino fermion and two additional bosons. The couplings of Majorana fermions to the gravitino and two bosons are essentially the same as for Dirac fermions and they are omitted here.

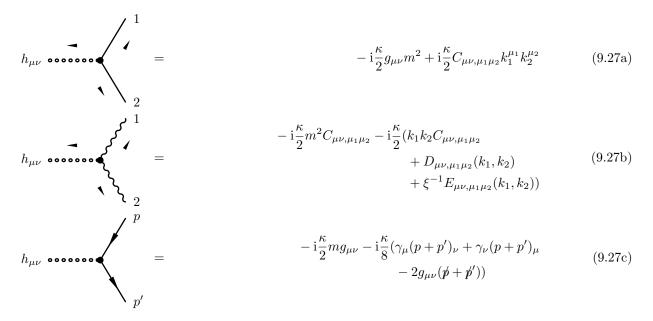


Figure 9.3: Three-point graviton couplings.

9.1.7 Perturbative Quantum Gravity and Kaluza-Klein Interactions

The gravitational coupling constant and the relative strength of the dilaton coupling are abbreviated as

$$\kappa = \sqrt{16\pi G_N} \tag{9.25a}$$

$$\omega = \sqrt{\frac{2}{3(n+2)}} = \sqrt{\frac{2}{3(d-2)}},$$
(9.25b)

where n = d - 4 is the number of extra space dimensions. In (9.27-9.34), we use the notation of [13]:

$$C_{\mu\nu,\rho\sigma} = g_{\mu\rho}g_{\nu\sigma} + g_{\mu\sigma}g_{\nu\rho} - g_{\mu\nu}g_{\rho\sigma} \tag{9.26a}$$

$$D_{\mu\nu,\rho\sigma}(k_1, k_2) = g_{\mu\nu} k_{1,\sigma} k_{2,\rho} - (g_{\mu\sigma} k_{1,\nu} k_{2,\rho} + g_{\mu\rho} k_{1,\sigma} k_{2,\nu} - g_{\rho\sigma} k_{1,\mu} k_{2,\nu} + (\mu \leftrightarrow \nu))$$
(9.26b)

$$E_{\mu\nu,\rho\sigma}(k_1,k_2) = g_{\mu\nu}(k_{1,\rho}k_{1,\sigma} + k_{2,\rho}k_{2,\sigma} + k_{1,\rho}k_{2,\sigma}) - (g_{\nu\sigma}k_{1,\mu}k_{1,\rho} + g_{\nu\rho}k_{2,\mu}k_{2,\sigma} + (\mu \leftrightarrow \nu)) \quad (9.26c)$$

$$F_{\mu\nu,\rho\sigma\lambda}(k_1, k_2, k_3) = g_{\mu\rho}g_{\sigma\lambda}(k_2 - k_3)_{\nu} + g_{\mu\sigma}g_{\lambda\rho}(k_3 - k_1)_{\nu} + g_{\mu\lambda}g_{\rho\sigma}(k_1 - k_2)_{\nu} + (\mu \leftrightarrow \nu) \quad (9.26d)$$

$$G_{\mu\nu,\rho\sigma\lambda\delta} = g_{\mu\nu}(g_{\rho\sigma}g_{\lambda\delta} - g_{\rho\delta}g_{\lambda\sigma}) + (g_{\mu\rho}g_{\nu\delta}g_{\lambda\sigma} + g_{\mu\lambda}g_{\nu\sigma}g_{\rho\delta} - g_{\mu\rho}g_{\nu\sigma}g_{\lambda\delta} - g_{\mu\lambda}g_{\nu\delta}g_{\rho\sigma} + (\mu \leftrightarrow \nu))$$
(9.26e)

Derivation of (9.27a)

$$L = \frac{1}{2} (\partial_{\mu} \phi)(\partial^{\mu} \phi) - \frac{m^2}{2} \phi^2$$
(9.28a)

$$(\partial_{\mu}\phi)\frac{\partial L}{\partial(\partial^{\nu}\phi)} = (\partial_{\mu}\phi)(\partial_{\nu}\phi) \tag{9.28b}$$

$$T_{\mu\nu} = -g_{\mu\nu}L + (\partial_{\mu}\phi)\frac{\partial L}{\partial(\partial^{\nu}\phi)} +$$
 (9.28c)

Graviton_Scalar	$Scalar: h_{\mu\nu}C_0^{\mu\nu}(k_1,k_2)\phi_1\phi_2$
F12 F21:	$\phi_2 \leftarrow \mathbf{i} \cdot h_{\mu\nu} C_0^{\mu\nu} (k_1, -k - k_1) \phi_1$
F13 F31:	$\phi_1 \leftarrow \mathbf{i} \cdot h_{\mu\nu} C_0^{\mu\nu} (-k - k_2, k_2) \phi_2$
F23 F32:	$h^{\mu\nu} \leftarrow i \cdot C_0^{\mu\nu}(k_1, k_2) \phi_1 \phi_2$
$Graviton_Vector$	_ Vector: $h_{\mu\nu}C_1^{\mu\nu,\mu_1\mu_2}(k_1,k_2,\xi)V_{\mu_1}V_{\mu_2}$
F12 F21:	$V_2^{\mu} \leftarrow i \cdot h_{\kappa\lambda} C_1^{\kappa\lambda,\mu\nu} (-k - k_1, k_1 \xi) V_{1,\nu}$
F13 F31:	$V_1^{\mu} \leftarrow i \cdot h_{\kappa\lambda} C_1^{\kappa\lambda,\mu\nu} (-k - k_2, k_2, \xi) V_{2,\nu}$
F23 F32:	$h^{\mu\nu} \leftarrow i \cdot C_1^{\mu\nu,\mu_1\mu_2}(k_1,k_2,\xi)V_{1,\mu_1}V_{2,\mu_2}$
Graviton_Spinor	$_Spinor: h_{\mu\nu}\bar{\psi}_1 C^{\mu\nu}_{\frac{1}{2}}(k_1, k_2)\psi_2$
F12:	$\bar{\psi}_2 \leftarrow i \cdot h_{\mu\nu} \bar{\psi}_1 C_{\frac{1}{2}}^{\mu\nu} (k_1, -k - k_1)$
F21:	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot \dots$
F13:	$\psi_1 \leftarrow \mathbf{i} \cdot h_{\mu\nu} C_{\frac{1}{2}}^{\mu\nu} (-k - k_2, k_2) \psi_2$
F31:	$\psi_1 \leftarrow i \cdot \dots$
F23:	$h^{\mu\nu} \leftarrow \mathbf{i} \cdot \bar{\psi}_1 C_{\frac{1}{2}}^{\mu\nu}(k_1, k_2) \psi_2$
F32:	$h^{\mu\nu} \leftarrow \mathrm{i} \cdot \dots$

Table 9.32: ...

$$C_0^{\mu\nu}(k_1, k_2) = C^{\mu\nu, \mu_1 \mu_2} k_{1, \mu_1} k_{2, \mu_2}$$
(9.29a)

$$C_1^{\mu\nu,\mu_1\mu_2}(k_1,k_2,\xi) = k_1 k_2 C^{\mu\nu,\mu_1\mu_2} + D^{\mu\nu,\mu_1\mu_2}(k_1,k_2) + \xi^{-1} E^{\mu\nu,\mu_1\mu_2}(k_1,k_2)$$
(9.29b)

$$C^{\mu\nu}_{\frac{1}{2},\alpha\beta}(p,p') = \gamma^{\mu}_{\alpha\beta}(p+p')^{\nu} + \gamma^{\nu}_{\alpha\beta}(p+p')^{\mu} - 2g^{\mu\nu}(\not p + \not p')_{\alpha\beta}$$
(9.29c)

9.1.8 Dependent Parameters

This is a simple abstract syntax for parameter dependencies. Later, there will be a parser for a convenient concrete syntax as a part of a concrete syntax for models. There is no intention to do *any* symbolic manipulation with this. The expressions will be translated directly by *Targets* to the target language.

```
type \alpha \ expr =
      I
       Integer of int
       Float of float
       Atom \ {\rm of} \ \alpha
       Sum \ {\it of} \ \alpha \ expr \ list
       Diff of \alpha \ expr \times \alpha \ expr
       Neg 	ext{ of } \alpha 	ext{ } expr
       Prod of \alpha expr list
       Quot of \alpha \ expr \times \alpha \ expr
       Rec of \alpha expr
       Pow 	ext{ of } \alpha 	ext{ } expr 	ext{ } 	ext{ } int
       PowX of \alpha \ expr \times \alpha \ expr
       Sqrt of \alpha expr
       Sin \ {
m of} \ \alpha \ expr
       Cos of \alpha expr
       Tan 	ext{ of } \alpha 	ext{ } expr
```

$$\phi(k) \cdots \qquad \qquad = -i\omega\kappa 2m^2 - i\omega\kappa k_1 k_2 \qquad (9.30a)$$

$$\phi(k) \cdots \qquad \qquad = -i\omega\kappa g_{\mu_1\mu_2} m^2 - i\omega\kappa \xi^{-1}(k_{1,\mu_1}k_{\mu_2} + k_{2,\mu_2}k_{\mu_1}) \qquad (9.30b)$$

$$\phi(k) \cdots \qquad \qquad p' \qquad \qquad (9.30c)$$

Figure 9.4: Three-point dilaton couplings.

$Dilaton_Scalar_S$	Scalar: $\phi \dots k_1 k_2 \phi_1 \phi_2$
F12 F21:	$\phi_2 \leftarrow \mathbf{i} \cdot k_1(-k-k_1)\phi\phi_1$
$F13 \mid F31$:	$\phi_1 \leftarrow \mathbf{i} \cdot (-k - k_2) k_2 \phi \phi_2$
F23 F32:	$\phi \leftarrow \mathbf{i} \cdot k_1 k_2 \phi_1 \phi_2$
$Dilaton_Vector_$	$Vector: \phi \dots$
F12:	$V_{2,\mu} \leftarrow \mathrm{i} \cdot \dots$
F21:	$V_{2,\mu} \leftarrow \mathrm{i} \cdot \dots$
F13:	$V_{1,\mu} \leftarrow \mathrm{i} \cdot \dots$
F31:	$V_{1,\mu} \leftarrow \mathrm{i} \cdot \dots$
F23:	$\phi \leftarrow i \cdot \dots$
F32:	$\phi \leftarrow i \cdot \dots$
$Dilaton_Spinor_$	$Spinor: \phi \dots$
F12:	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot \dots$
F21:	$\bar{\psi}_2 \leftarrow \mathrm{i} \cdot \dots$
F13:	$\psi_1 \leftarrow i \cdot \dots$
F31:	$\psi_1 \leftarrow i \cdot \dots$
F23:	$\phi \leftarrow i \cdot \dots$
F32:	$\phi \leftarrow i \cdot \dots$

Table 9.33: ...

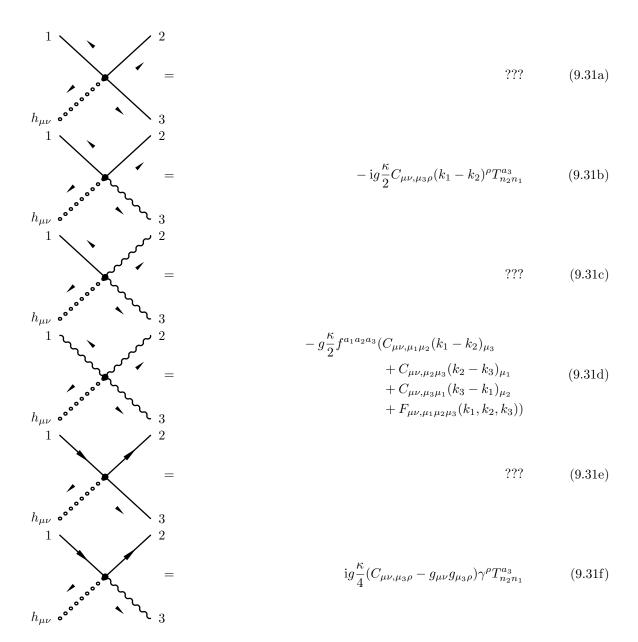


Figure 9.5: Four-point graviton couplings. (9.31a), (9.31c), and (?? are missing in [13], but should be generated by standard model Higgs selfcouplings, Higgs-gaugeboson couplings, and Yukawa couplings.

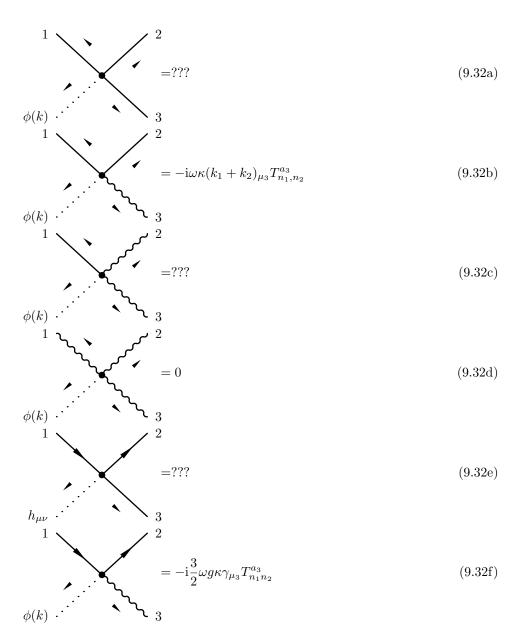


Figure 9.6: Four-point dilaton couplings. (9.32a), (9.32c) and (9.32e) are missing in [13], but could be generated by standard model Higgs selfcouplings, Higgs-gaugeboson couplings, and Yukawa couplings.

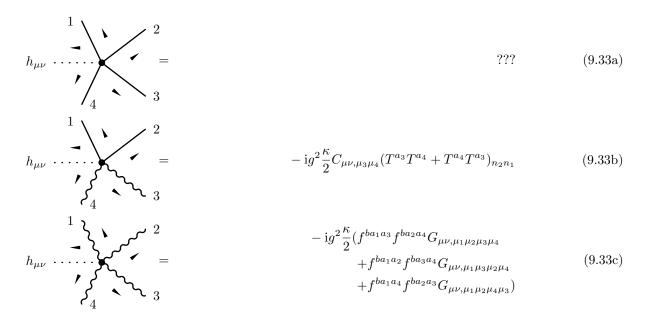


Figure 9.7: Five-point graviton couplings. (9.33a) is missing in [13], but should be generated by standard model Higgs selfcouplings.

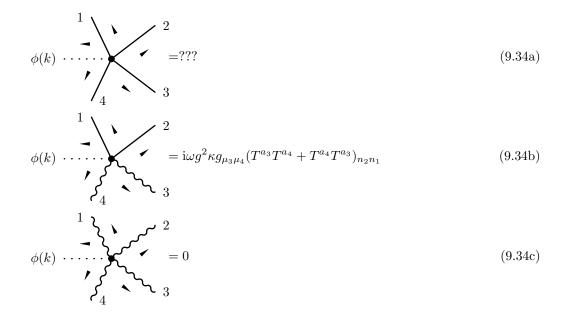


Figure 9.8: Five-point dilaton couplings. (9.34a) is missing in [13], but could be generated by standard model Higgs selfcouplings.

$Dim5_Scalar_Vector_Vector_T: \mathcal{L}_I = g\phi(i\partial_{\mu}V_1^{\nu})(i\partial_{\nu}V_2^{\mu})$		
F23:	$\phi(k_2 + k_3) \leftarrow i \cdot g k_3^{\mu} V_{1,\mu}(k_2) k_2^{\nu} V_{2,\nu}(k_3)$	
F32:	$\phi(k_2 + k_3) \leftarrow i \cdot g k_2^{\mu} V_{2,\mu}(k_3) k_3^{\nu} V_{1,\nu}(k_2)$	
F12:	$V_2^{\mu}(k_1 + k_2) \leftarrow i \cdot g k_2^{\mu} \phi(k_1) (-k_1^{\nu} - k_2^{\nu}) V_{1,\nu}(k_2)$	
F21:	$V_2^{\mu}(k_1 + k_2) \leftarrow i \cdot gk_2^{\mu}(-k_1^{\nu} - k_2^{\nu})V_{1,\nu}(k_2)\phi(k_1)$	
F13:	$V_1^{\mu}(k_1 + k_3) \leftarrow i \cdot g k_3^{\mu} \phi(k_1) (-k_1^{\nu} - k_3^{\nu}) V_{2,\nu}(k_3)$	
F31:	$V_1^{\mu}(k_1 + k_3) \leftarrow i \cdot gk_3^{\mu}(-k_1^{\nu} - k_3^{\nu})V_{2,\nu}(k_3)\phi(k_1)$	

Table 9.34: ...

```
Dim6\_Vector\_Vector\_Vector\_T: \mathcal{L}_I = gV_1^{\mu}((i\partial_{\nu}V_2^{\rho})i\overleftarrow{\partial_{\mu}}(i\partial_{\rho}V_3^{\nu}))
F23: V_1^{\mu}(k_2 + k_3) \leftarrow i \cdot g(k_2^{\mu} - k_3^{\mu})k_3^{\nu}V_{2,\nu}(k_2)k_2^{\rho}V_{3,\rho}(k_3)
F32: V_1^{\mu}(k_2 + k_3) \leftarrow i \cdot g(k_2^{\mu} - k_3^{\mu})k_2^{\nu}V_{3,\nu}(k_3)k_3^{\rho}V_{2,\rho}(k_2)
F12: V_3^{\mu}(k_1 + k_2) \leftarrow i \cdot gk_2^{\mu}(k_1^{\nu} + 2k_2^{\nu})V_{1,\nu}(k_1)(-k_1^{\rho} - k_2^{\rho})V_{2,\rho}(k_2)
F21: V_3^{\mu}(k_1 + k_2) \leftarrow i \cdot gk_2^{\mu}(-k_1^{\rho} - k_2^{\rho})V_{2,\rho}(k_2)(k_1^{\nu} + 2k_2^{\nu})V_{1,\nu}(k_1)
F13: V_2^{\mu}(k_1 + k_3) \leftarrow i \cdot gk_3^{\mu}(k_1^{\nu} + 2k_3^{\nu})V_{1,\nu}(k_1)(-k_1^{\rho} - k_3^{\rho})V_{3,\rho}(k_3)
F31: V_2^{\mu}(k_1 + k_3) \leftarrow i \cdot gk_3^{\mu}(-k_1^{\rho} - k_3^{\rho})V_{3,\rho}(k_3)(k_1^{\nu} + 2k_3^{\nu})V_{1,\nu}(k_1)
```

Table 9.35: ...

```
Cot \ {\it of} \ \alpha \ expr
      Asin of \alpha expr
      A\cos of \alpha expr
      Atan of \alpha expr
      Atan2 of \alpha \ expr \times \alpha \ expr
      Sinh of \alpha expr
      Cosh of \alpha expr
      Tanh 	ext{ of } \alpha 	ext{ } expr
      \mathit{Exp}\ \mathsf{of}\ \alpha\ \mathit{expr}
      Log 	ext{ of } \alpha 	ext{ } expr
      Log10 of \alpha expr
      Conj of \alpha expr
     Abs of \alpha expr
type \alpha variable = Real of <math>\alpha \mid Complex of \alpha
type \alpha variable\_array = Real\_Array of \alpha \mid Complex\_Array of \alpha
type \alpha parameters =
      \{ input : (\alpha \times float) \ list; \}
         derived : (\alpha variable \times \alpha expr) list;
         derived\_arrays : (\alpha \ variable\_array \times \alpha \ expr \ list) \ list \}
                                                              More Exotic Couplings
                                                             Interface of Model
                                                    9.2
                                           9.2.1 General Quantum Field Theories
```

 ${\it flavor}$ abstractly encodes all quantum numbers. ${\it type} \ {\it flavor}$

module type T =

sig

Tensor_2_Vector_Vector: $\mathcal{L}_I = gT^{\mu\nu}(V_{1,\mu}V_{2,\nu} + V_{1,\nu}V_{2,\mu})$		
F23:	$T^{\mu\nu}(k_2 + k_3) \leftarrow i \cdot g(V_{1,\mu}(k_2)V_{2,\nu}(k_3) + V_{1,\nu}(k_2)V_{2,\mu}(k_3))$	
F32:	$T^{\mu\nu}(k_2 + k_3) \leftarrow i \cdot g(V_{2,\nu}(k_3)V_{1,\mu}(k_2) + V_{2,\mu}(k_3)V_{1,\nu}(k_2))$	
F12:	$V_2^{\mu}(k_1 + k_2) \leftarrow i \cdot g(T^{\mu\nu}(k_1) + T^{\nu\mu}(k_1))V_{1,\nu}(k_2)$	
F21:	$V_2^{\mu}(k_1 + k_2) \leftarrow i \cdot gV_{1,\nu}(k_2)(T^{\mu\nu}(k_1) + T^{\nu\mu}(k_1))$	
F13:	$V_1^{\mu}(k_1 + k_3) \leftarrow i \cdot g(T^{\mu\nu}(k_1) + T^{\nu\mu}(k_1))V_{2,\nu}(k_3)$	
F31:	$V_1^{\mu}(k_1 + k_3) \leftarrow i \cdot gV_{2,\nu}(k_3)(T^{\mu\nu}(k_1) + T^{\nu\mu}(k_1))$	

Table 9.36: ...

$Dim5_Tensor_2_Vector_Vector_1: \mathcal{L}_I = gT^{\alpha\beta}(V_1^{\mu} i \overleftrightarrow{\partial}_{\alpha} i \overleftrightarrow{\partial}_{\beta} V_{2,\mu})$		
F23:	$T^{\alpha\beta}(k_2+k_3) \leftarrow i \cdot g(k_2^{\alpha}-k_3^{\alpha})(k_2^{\beta}-k_3^{\beta})V_1^{\mu}(k_2)V_{2,\mu}(k_3)$	
F32:	$T^{\alpha\beta}(k_2+k_3) \leftarrow i \cdot g(k_2^{\alpha}-k_3^{\alpha})(k_2^{\beta}-k_3^{\beta})V_{2,\mu}(k_3)V_1^{\mu}(k_2)$	
F12:	$V_2^{\mu}(k_1 + k_2) \leftarrow i \cdot g(k_1^{\alpha} + 2k_2^{\alpha})(k_1^{\beta} + 2k_2^{\beta})T_{\alpha\beta}(k_1)V_1^{\mu}(k_2)$	
F21:	$V_2^{\mu}(k_1 + k_2) \leftarrow i \cdot g(k_1^{\alpha} + 2k_2^{\alpha})(k_1^{\beta} + 2k_2^{\beta})V_1^{\mu}(k_2)T_{\alpha\beta}(k_1)$	
F13:	$V_1^{\mu}(k_1 + k_3) \leftarrow i \cdot g(k_1^{\alpha} + 2k_3^{\alpha})(k_1^{\beta} + 2k_3^{\beta})T_{\alpha\beta}(k_1)V_2^{\mu}(k_3)$	
F31:	$V_1^{\mu}(k_1 + k_3) \leftarrow i \cdot g(k_1^{\alpha} + 2k_3^{\alpha})(k_1^{\beta} + 2k_3^{\beta})V_2^{\mu}(k_3)T_{\alpha\beta}(k_1)$	

Table 9.37: \dots

$Dim5_Tensor_2_Vector_Vector_2: \mathcal{L}_I = gT^{\alpha\beta}(V_1^{\mu}i\overleftrightarrow{\partial}_{\beta}(i\partial_{\mu}V_{2,\alpha}) + V_1^{\mu}i\overleftrightarrow{\partial}_{\alpha}(i\partial_{\mu}V_{2,\beta}))$		
F23:	$T^{\alpha\beta}(k_2 + k_3) \leftarrow i \cdot g(k_3^{\beta} - k_2^{\beta})k_3^{\mu}V_{1,\mu}(k_2)V_2^{\alpha}(k_3) + (\alpha \leftrightarrow \beta)$	
F32:	$T^{\alpha\beta}(k_2 + k_3) \leftarrow i \cdot g(k_3^{\beta} - k_2^{\beta}) V_2^{\alpha}(k_3) k_3^{\mu} V_{1,\mu}(k_2) + (\alpha \leftrightarrow \beta)$	
F12:	$V_2^{\alpha}(k_1 + k_2) \leftarrow i \cdot g(k_1^{\beta} + 2k_2^{\beta})(T^{\alpha\beta}(k_1) + T^{\beta\alpha}(k_1))(k_1^{\mu} + k_2^{\mu})V_{1,\mu}(k_2)$	
F21:	$V_2^{\alpha}(k_1 + k_2) \leftarrow i \cdot g(k_1^{\mu} + k_2^{\mu}) V_{1,\mu}(k_2) (k_1^{\beta} + 2k_2^{\beta}) (T^{\alpha\beta}(k_1) + T^{\beta\alpha}(k_1))$	
F13:	$V_1^{\alpha}(k_1 + k_3) \leftarrow i \cdot g(k_1^{\beta} + 2k_3^{\beta})(T^{\alpha\beta}(k_1) + T^{\beta\alpha}(k_1))(k_1^{\mu} + k_3^{\mu})V_{2,\mu}(k_3)$	
F31:	$V_1^{\alpha}(k_1 + k_3) \leftarrow i \cdot g(k_1^{\mu} + k_3^{\mu}) V_{2,\mu}(k_3) (k_1^{\beta} + 2k_3^{\beta}) (T^{\alpha\beta}(k_1) + T^{\beta\alpha}(k_1))$	

Table 9.38: ...

$Dim 7_Tensor_2_Vector_Vector_T: \ \mathcal{L}_I = gT^{\alpha\beta}((\mathrm{i}\partial^\mu V_1^\nu)\mathrm{i} \stackrel{\longleftrightarrow}{\partial}_\alpha \mathrm{i} \stackrel{\longleftrightarrow}{\partial}_\beta (\mathrm{i}\partial_\nu V_{2,\mu}))$		
F23:	$T^{\alpha\beta}(k_2+k_3) \leftarrow i \cdot g(k_2^{\alpha}-k_3^{\alpha})(k_2^{\beta}-k_3^{\beta})k_3^{\mu}V_{1,\mu}(k_2)k_2^{\nu}V_{2,\nu}(k_3)$	
F32:	$T^{\alpha\beta}(k_2+k_3) \leftarrow i \cdot g(k_2^{\alpha}-k_3^{\alpha})(k_2^{\beta}-k_3^{\beta})k_2^{\nu}V_{2,\nu}(k_3)k_3^{\mu}V_{1,\mu}(k_2)$	
F12:	$V_2^{\mu}(k_1 + k_2) \leftarrow i \cdot gk_2^{\mu}(k_1^{\alpha} + 2k_2^{\alpha})(k_1^{\beta} + 2k_2^{\beta})T_{\alpha\beta}(k_1)(-k_1^{\nu} - k_2^{\nu})V_{1,\nu}(k_2)$	
F21:	$V_2^{\mu}(k_1 + k_2) \leftarrow i \cdot gk_2^{\mu}(-k_1^{\nu} - k_2^{\nu})V_{1,\nu}(k_2)(k_1^{\alpha} + 2k_2^{\alpha})(k_1^{\beta} + 2k_2^{\beta})T_{\alpha\beta}(k_1)$	
F13:	$V_1^{\mu}(k_1 + k_3) \leftarrow i \cdot g k_3^{\mu}(k_1^{\alpha} + 2k_3^{\alpha})(k_1^{\beta} + 2k_3^{\beta}) T_{\alpha\beta}(k_1)(-k_1^{\nu} - k_3^{\nu}) V_{2,\nu}(k_3)$	
F31:	$V_1^{\mu}(k_1 + k_3) \leftarrow i \cdot gk_3^{\mu}(-k_1^{\nu} - k_3^{\nu})V_{2,\nu}(k_3)(k_1^{\alpha} + 2k_3^{\alpha})(k_1^{\beta} + 2k_3^{\beta})T_{\alpha\beta}(k_1)$	

Table 9.39: ...

Color.t encodes the (SU(N)) color representation.

```
val\ color\ :\ flavor\ 	o\ Color.t
\mathsf{val}\ nc\ :\ unit \to\ int
```

The set of conserved charges.

```
module Ch : Charges.T
val\ charges: flavor \rightarrow Ch.t
```

The PDG particle code for interfacing with Monte Carlos.

```
\mathsf{val}\ pdg\ :\ flavor\ \to\ int
```

The Lorentz representation of the particle.

```
val\ lorentz\ :\ flavor\ 	o\ Coupling.lorentz
```

The propagator for the particle, which *can* depend on a gauge parameter.

```
type gauge
```

```
val\ propagator\ :\ flavor\ 	o\ gauge\ Coupling.propagator
```

Not the symbol for the numerical value, but the scheme or strategy.

```
val \ width : flavor \rightarrow Coupling.width
```

Charge conjugation, with and without color.

```
val\ conjugate\ :\ flavor\ 	o\ flavor
```

Returns 1 for fermions, -1 for anti-fermions, 2 for Majoranas and 0 otherwise.

```
val\ fermion: flavor \rightarrow int
```

The Feynman rules. vertices and (fuse2, fuse3, fusen) are redundant, of course. However, vertices is required for building functors for models and vertices can be recovered from (fuse2, fuse3, fusen) only at great cost.



Nevertheless: vertices is a candidate for removal, b/c we can build a smarter Colorize functor acting on (fuse2, fuse3, fusen). It can support an arbitrary numer of color lines. But we have to test whether it is efficient enough. And we have to make sure that this wouldn't break the UFO interface.

type constant

Later: type orders to count orders of couplings

```
val\ max\_degree : unit \rightarrow int
val\ vertices\ :\ unit 
ightarrow
   (((flavor \times flavor \times flavor) \times constant Coupling.vertex3 \times constant) list)
       \times (((flavor \times flavor \times flavor \times flavor \times flavor) \times constant Coupling.vertex4 \times constant) \ list)
       \times (((flavor\ list) \times constant\ Coupling.vertexn \times constant)\ list))
val\ fuse2: flavor 
ightarrow flavor 
ightarrow (flavor 	imes constant\ Coupling.t)\ list
val fuse3: flavor \rightarrow flavor \rightarrow flavor \rightarrow (flavor \times constant\ Coupling.t)\ list
val\ fuse\ :\ flavor\ list\ 
ightarrow\ (flavor\ 	imes\ constant\ Coupling.t)\ list
```

Later: val orders: $constant \rightarrow orders$ counting orders of couplings The list of all known flavors.

```
val\ flavors\ :\ unit 
ightarrow\ flavor\ list
```

The flavors that can appear in incoming or outgoing states, grouped in a way that is useful for user interfaces.

```
val\ external\_flavors\ :\ unit \rightarrow\ (string \times flavor\ list)\ list
```

The Goldstone bosons corresponding to a gauge field, if any.

```
val\ goldstone\ :\ flavor\ 	o\ (flavor\ 	imes\ constant\ Coupling.expr)\ option
```

The dependent parameters.

```
val\ parameters\ :\ unit 
ightarrow\ constant\ Coupling.parameters
```

Translate from and to convenient textual representations of flavors.

```
val\ flavor\_of\_string : string \rightarrow flavor
val\ flavor\_to\_string : flavor 
ightarrow string
```

T_FX and L^AT_FX

```
val\ flavor\_to\_TeX : flavor \rightarrow string
```

The following must return unique symbols that are acceptable as symbols in all programming languages under consideration as targets. Strings of alphanumeric characters (starting with a letter) should be safe. Underscores are also usable, but would violate strict Fortran77.

```
val\ flavor\_symbol\ :\ flavor\ 	o\ string
val\ gauge\_symbol:\ gauge 
ightarrow string
val\ mass\_symbol\ :\ flavor\ 	o\ string
val\ width\_symbol\ :\ flavor\ 	o\ string
val\ constant\_symbol\ :\ constant\ 	o\ string
```

Model specific options.

```
val options : Options.t
```

Not ready for prime time or other warnings to be written to the source files for the amplitudes.

```
val\ caveats:\ unit \rightarrow\ string\ list
end
```

In addition to hardcoded models, we can have models that are initialized at run time.

9.2.2 Mutable Quantum Field Theories

```
module type Mutable =
  sig
     include T
     val\ init : unit \rightarrow unit
```

Export only one big initialization function to discourage partial initializations. Labels make this usable.

```
val\ setup:
         color: (flavor \rightarrow Color.t) \rightarrow
         nc:(unit \rightarrow int) \rightarrow
         pdg:(flavor \rightarrow int) \rightarrow
         lorentz: (flavor \rightarrow Coupling.lorentz) \rightarrow
         propagator: (flavor \rightarrow gauge\ Coupling.propagator) \rightarrow
         width: (flavor \rightarrow Coupling.width) \rightarrow
         goldstone: (flavor \rightarrow (flavor \times constant\ Coupling.expr)\ option) \rightarrow
         conjugate: (flavor \rightarrow flavor) \rightarrow
         fermion: (flavor \rightarrow int) \rightarrow
         vertices:
            (unit \rightarrow
              ((((flavor \times flavor \times flavor) \times constant\ Coupling.vertex3 \times constant)\ list)
                \times (((flavor \times flavor \times flavor \times flavor \times flavor) \times constant Coupling.vertex4 \times constant) \ list)
                \times (((flavor\ list)\ \times\ constant\ Coupling.vertexn\ \times\ constant)\ list)))\ 	o
         flavors: ((string \times flavor\ list)\ list) \rightarrow
         parameters: (unit \rightarrow constant\ Coupling.parameters) \rightarrow
         flavor\_of\_string:(string \rightarrow flavor) \rightarrow
         flavor\_to\_string:(flavor \rightarrow string) \rightarrow
         flavor\_to\_TeX:(flavor \rightarrow string) \rightarrow
         flavor\_symbol: (flavor \rightarrow string) \rightarrow
         gauge\_symbol:(gauge \rightarrow string) \rightarrow
         mass\_symbol: (flavor \rightarrow string) \rightarrow
          width\_symbol: (flavor \rightarrow string) \rightarrow
          constant\_symbol:(constant \rightarrow string) \rightarrow
          unit
end
```

9.2.3 Gauge Field Theories

The following signatures are used only for model building. The diagrammatics and numerics is supposed to be completely ignorant about the detail of the models and expected to rely on the interface T exclusively.



end

 $rac{}{}$ In the end, we might have functors $(M:T) \rightarrow Gauge$, but we will need to add the quantum numbers to T.

```
module type Gauge =
  sig
    include T
```

Matter field carry conserved quantum numbers and can be replicated in generations without changing the gauge sector.

```
type matter\_field
Gauge bosons proper.
    type gauge\_boson
Higgses, Goldstones and all the rest:
    type other
We can query the kind of field
    type field =
         Matter of matter_field
         Gauge\_boson
         Other \ {\rm of} \ other
    val field : flavor \rightarrow field
and we can build new fields of a given kind:
    val\ matter\_field: matter\_field 
ightarrow flavor
    val\ gauge\_boson\ :\ gauge\_boson\ 	o\ flavor
    val\ other\ :\ other\ 	o\ flavor
```

Gauge Field Theories with Broken Gauge Symmetries

Both are carefully crafted as subtypes of Gauge so that they can be used in place of Gauge and T everywhere:

```
module type Broken\_Gauge =
  sig
    include Gauge
    type massless
    type massive
    type \ goldstone
     \mathsf{type} \; kind \; = \;
         Massless of massless
          Massive of massive
          Goldstone of goldstone
    val \ kind : gauge\_boson \rightarrow kind
    \verb|val|| massless : massive \rightarrow gauge\_boson
    val\ massive\ :\ massive\ 	o\ gauge\_boson
    val\ goldstone\ :\ goldstone\ 	o\ gauge\_boson
module type Unitarity\_Gauge =
  sig
    include Gauge
    type massless
     type \ massive
```

```
type kind =
          Massless of massless
          Massive of massive
     \mathsf{val}\ kind\ :\ gauge\_boson\ \to\ kind
     val\ massless\ :\ massive\ 	o\ gauge\_boson
     val\ massive\ :\ massive\ 	o\ gauge\_boson
  end
module type Colorized =
  sig
     include T
     type flavor\_sans\_color
     \verb|val| flavor\_sans\_color|: flavor\_sans\_color|
     val\ conjugate\_sans\_color\ :\ flavor\_sans\_color\ 	o\ flavor\_sans\_color
     val\ amplitude: flavor\_sans\_color\ list 
ightarrow\ flavor\_sans\_color\ list 
ightarrow
        (flavor\ list \times flavor\ list)\ list
     \mathsf{val}\ \mathit{flow}\ :\ \mathit{flavor}\ \mathit{list}\ \rightarrow\ \mathit{flavor}\ \mathit{list}\ \rightarrow\ \mathit{Color}.\mathit{Flow}.t
  end
module type Colorized\_Gauge =
  sig
     include Gauge
     type flavor_sans_color
     \verb|val| flavor\_sans\_color|: flavor\_ + flavor\_sans\_color|
     val\ conjugate\_sans\_color\ :\ flavor\_sans\_color\ 	o\ flavor\_sans\_color
     \verb|val|| amplitude : flavor\_sans\_color| list \rightarrow flavor\_sans\_color| list \rightarrow
        (flavor\ list \times flavor\ list)\ list
     \mathsf{val}\ flow: flavor\ list 	o flavor\ list 	o\ Color.Flow.t
  end
                                                       Interface of Dirac
                                              9.3
                                                 9.3.1 Dirac \gamma-matrices
module type T =
  sig
Matrices with complex rational entries.
     type qc = Algebra.QC.t
     type t = qc \ array \ array
Complex rational constants.
     val zero : qc
     val one : qc
     {\tt val}\ minus\_one\ :\ qc
     \mathsf{val}\;i\;:\;qc
     \mathsf{val}\ minus\_i\ :\ qc
Basic \gamma-matrices.
     val unit: t
     val null : t
     val\ gamma0\ :\ t
     val\ gamma1: t
     val\ gamma2: t
     val\ gamma3 : t
     val\ gamma5 : t
```

```
(\gamma_0, \gamma_1, \gamma_2, \gamma_3)
     val\ gamma\ :\ t\ array
Charge conjugation
     \mathsf{val}\ cc\ :\ t
Algebraic operations on \gamma-matrices
     \mathsf{val}\ neg\ :\ t\ \to\ t
     \mathsf{val}\ add\ :\ t\ \to\ t\ \to\ t
     \mathsf{val}\ sub\ :\ t\ \to\ t\ \to\ t
     \mathsf{val}\ mul\ :\ t\ \to\ t\ \to\ t
     \mathsf{val}\ times\ :\ qc\ \to\ t\ \to\ t
     \mathsf{val}\ transpose\ :\ t\ \to\ t
     \mathsf{val}\ adjoint\ :\ t\ \to\ t
     \mathsf{val}\ \mathit{conj}\ :\ t\ \to\ t
     val\ product : t\ list 
ightarrow t
Toplevel
     \mathsf{val}\ pp\ :\ Format.formatter\ 	o\ t\ 	o\ unit
Unit tests
     val\ test\_suite\ :\ OUnit.test
  end
module\ Chiral\ :\ T
module\ Dirac\ :\ T
module Majorana : T
                                             9.4 Implementation of Dirac
                                                       9.4.1 Dirac \gamma-matrices
module type T =
  sig
     type qc = Algebra.QC.t
     \mathsf{type}\ t\ =\ qc\ array\ array
     val\ zero\ :\ qc
     val one : qc
     val minus\_one : qc
     \mathsf{val}\;i\;:\;qc
     val minus_i : qc
     val\ unit:t
     val null : t
     val\ qamma0 : t
     val\ gamma1: t
     val\ gamma2 : t
     val\ gamma3: t
     val\ gamma5 : t
     val\ gamma\ :\ t\ array
     \mathsf{val}\ \mathit{cc}\ :\ \mathit{t}
     \mathsf{val}\ neq\ :\ t\ \to\ t
     \mathsf{val}\ add\ :\ t\ \to\ t\ \to\ t
     \mathsf{val}\ sub\ :\ t\ \to\ t\ \to\ t
     \mathsf{val}\ mul\ :\ t\ \to\ t\ \to\ t
     val times: qc \rightarrow t \rightarrow t
     \mathsf{val}\ transpose\ :\ t\ \to\ t
     \mathsf{val}\ adjoint\ :\ t\ \to\ t
     val\ conj\ :\ t\ 	o\ t
     val\ product : t\ list \rightarrow t
     \mathsf{val}\ pp\ :\ Format.formatter\ \to\ t\ \to\ unit
```

```
val test_suite : OUnit.test
end
```

Matrices with complex rational entries

```
module Q = Algebra.Q
module QC = Algebra.QC
type complex\_rational = QC.t
\mathsf{let}\ \mathit{zero}\ =\ \mathit{QC.null}
let one = QC.unit
\mathsf{let}\ minus\_one\ =\ QC.neg\ one
\mathsf{let}\ i\ =\ QC.make\ Q.null\ Q.unit
let minus_i = QC.conj i
type matrix = complex\_rational \ array \ array
                                                   Dirac \ \gamma-matrices
module type R =
  sig
    type qc = complex\_rational
    type t = matrix
    val\ gamma\theta\ :\ t
    val\ gamma1\ :\ t
    val\ gamma2\ :\ t
    val\ gamma3 : t
    val\ gamma5 : t
    \mathsf{val}\ \mathit{cc}\ :\ \mathit{t}
    val\ cc\_is\_i\_gamma2\_gamma\_0\ :\ bool
  end
module Make (R : R) : T =
  struct
     \mathsf{type}\ qc\ =\ complex\_rational
    type t = matrix
    let zero = zero
    \mathsf{let} \ one \ = \ one
    let minus\_one = minus\_one
    let i = i
    let minus_i = minus_i
    \mathsf{let}\ null\ =
       [| [| zero; zero; zero; zero |];
           [| zero; zero; zero; zero |];
           [| zero; zero; zero; zero |];
           [| zero; zero; zero; zero |] |]
    let unit =
       [| [| one; zero; zero; zero |];
           [| zero; one; zero; zero |];
           [| zero; zero; one; zero |];
           [|zero; zero; zero; one|]|
    \mathsf{let} \ gamma\theta \ = \ R.gamma\theta
    let gamma1 = R.gamma1
    let gamma2 = R.gamma2
    let gamma3 = R.gamma3
    let gamma5 = R.gamma5
    let gamma = [|gamma0; gamma1; gamma2; gamma3|]
```

let cc = R.cc

```
let neg g =
   let g' = Array.make\_matrix \ 4 \ 4 \ zero in
   \quad \text{for } i \ = \ 0 \ \text{to} \ 3 \ \text{do}
      for j = 0 to 3 do
         g'.(i).(j) \leftarrow QC.neg g.(i).(j)
      done
   done;
   g'
let add \ g1 \ g2 =
   \mathsf{let}\ g12\ =\ \mathit{Array.make\_matrix}\ 4\ 4\ \mathit{zero}\ \mathsf{in}
   \quad \text{for } i \ = \ 0 \ \text{to} \ 3 \ \text{do}
      for j = 0 to 3 do
         g12.(i).(j) \leftarrow QC.add \ g1.(i).(j) \ g2.(i).(j)
      done
   done;
   g12
\mathsf{let}\ \mathit{sub}\ \mathit{g1}\ \mathit{g2}\ =
   \mathsf{let}\ g12\ =\ Array.make\_matrix\ 4\ 4\ zero\ \mathsf{in}
   \quad \text{for } i \ = \ 0 \ \text{to} \ 3 \ \text{do}
      for i = 0 to 3 do
         g12.(i).(j) \leftarrow QC.sub \ g1.(i).(j) \ g2.(i).(j)
      done
   done;
   g12
let mul\ g1\ g2\ =
   let g12 = Array.make\_matrix 4 4 zero in
   for i = 0 to 3 do
      for k = 0 to 3 do
         for j = 0 to 3 do
            g12.(i).(k) \leftarrow QC.add \ g12.(i).(k) \ (QC.mul \ g1.(i).(j) \ g2.(j).(k))
      done
   done;
   g12
\mathsf{let}\ \mathit{times}\ q\ g\ =
   let g' = Array.make\_matrix 4 4 zero in
   \quad \text{for } i \ = \ 0 \ \text{to} \ 3 \ \text{do}
      for j = 0 to 3 do
         g'.(i).(j) \leftarrow QC.mul \ q \ g.(i).(j)
      done
   done;
   g'
\mathsf{let}\ transpose\ g\ =
   let g' = Array.make\_matrix 4 4 zero in
   for i = 0 to 3 do
      for j = 0 to 3 do
         g'.(i).(j) \leftarrow g.(j).(i)
      done
   done;
   g'
\mathsf{let} \ \mathit{adjoint} \ g \ = \\
   let g' = Array.make\_matrix \ 4 \ 4 \ zero in
   for i = 0 to 3 do
      for j = 0 to 3 do
         g'.(i).(j) \leftarrow QC.conj \ g.(j).(i)
      done
   done;
   g'
```

```
let conj g =
  let g' = Array.make\_matrix \ 4 \ 4 \ zero in
  \quad \text{for } i \ = \ 0 \ \text{to} \ 3 \ \text{do}
    for j = 0 to 3 do
       g'.(i).(j) \leftarrow QC.conj \ g.(i).(j)
  done;
  g'
let product glist =
  List.fold_right mul glist unit
let pp fmt g =
  let pp\_row i =
    for j = 0 to 3 do
       done in
  Format.fprintf\ fmt\ "\n_{\sqcup}/";
  pp\_row 0;
  Format.fprintf\ fmt\ "$\sqcup \\n"$;
  \quad \text{for } i \ = \ 1 \ \text{to} \ 2 \ \text{do}
     pp\_row i;
     Format.fprintf\ fmt\ "\lu|\n"
  done:
  Format.fprintf\ fmt\ "{}_{\sqcup}\verb|\\|`";
  pp\_row 3;
  Format.fprintf fmt "□/\n"
open OUnit
\mathsf{let}\ two\ =\ QC.make\ (Q.make\ 2\ 1)\ Q.null
let half = QC.make (Q.make 1 2) Q.null
let \ two\_unit = times \ two \ unit
let ac\_lhs mu nu =
  add \ (mul \ gamma.(mu) \ gamma.(nu)) \ (mul \ gamma.(nu) \ gamma.(mu))
let ac\_rhs mu nu =
  if mu = nu then
    if mu = 0 then
       two\_unit
    else
       neg two\_unit
  else
     null
let test\_ac mu nu =
  (ac\_lhs \ mu \ nu) = (ac\_rhs \ mu \ nu)
let ac_lhs_all =
  let lhs = Array.make\_matrix 4 4 null in
  for mu = 0 to 3 do
     for nu = 0 to 3 do
       lhs.(mu).(nu) \leftarrow ac\_lhs mu nu
     done
  done;
  lhs
let ac\_rhs\_all =
  let \ rhs = Array.make\_matrix \ 4 \ 4 \ null \ in
  for mu = 0 to 3 do
    for nu = 0 to 3 do
       rhs.(mu).(nu) \leftarrow ac\_rhs mu nu
     done
  done;
```

```
rhs
let dump2 lhs rhs =
        \quad \text{for } i \ = \ 0 \ \text{to} \ 3 \ \text{do}
                for j = 0 to 3 do
                         Printf.printf
                                  "_{ \cup \cup \cup} i_{ \cup} =_{ \cup } \ 'd,_{ \cup} j_{ \cup} = \ 'd:_{ \cup } \ 's_{ \cup} +_{ \cup } \ 's*I_{ \cup} |_{ \cup } \ 's_{ \cup} +_{ \cup } \ 's*I_{ \cap} |_{ \cup } \ 's_{ \cup } +_{ \cup } \ 's*I_{ \cup } |_{ \cup } \ 's_{ \cup } +_{ \cup } \ 's_{ \cup
                                 i j
                                 (Q.to\_string\ (QC.real\ lhs.(i).(j)))
                                 (Q.to\_string\ (QC.imag\ lhs.(i).(j)))
                                 (Q.to\_string\ (QC.real\ rhs.(i).(j)))
                                 (Q.to\_string\ (QC.imag\ rhs.(i).(j)))
                 done
        done
let \ dump2\_all \ lhs \ rhs \ =
        \quad \text{for } mu \ = \ 0 \ \text{to} \ 3 \ \text{do}
                for nu = 0 to 3 do
                         Printf.printf \;"\mathtt{mu} \sqcup = \sqcup \% \mathtt{d} \;, \sqcup \mathtt{nu} \sqcup = \% \mathtt{d} \;: \sqcup \backslash \mathtt{n} " \; mu \; nu \;;
                         dump2 \ lhs.(mu).(nu) \ rhs.(mu).(nu)
                done
        done
let anticommute =
         "anticommutation_{\sqcup}relations" >::
                 (fun () \rightarrow
                         assert\_bool
                                 11 11
                                 (if ac\_lhs\_all = ac\_rhs\_all then
                                     else
                                             begin
                                                     dump2\_all\ ac\_lhs\_all\ ac\_rhs\_all;
                                             end))
let equal\_or\_dump2 lhs rhs =
        if lhs = rhs then
                true
        else
                begin
                         dump2 lhs rhs;
                        false
                end
let gamma5\_def =
         "gamma5" >::
                 (fun () \rightarrow
                         assert\_bool
                                 "definition"
                                 (equal\_or\_dump2
                                             gamma5
                                             (times i (product [gamma0; gamma1; gamma2; gamma3]))))
let self_adjoint =
         "(anti)selfadjointness" >:::
                 [ "gamma0" >::
                                 (fun () \rightarrow
                                         assert_bool "self" (equal_or_dump2 gamma0 (adjoint gamma0)));
                          "gamma1" >::
                                 (fun () \rightarrow
                                         assert_bool "anti" (equal_or_dump2 gamma1 (neg (adjoint gamma1))));
                          "gamma2" >::
                                 (\mathsf{fun}\ ()\ \to
```

```
assert_bool "anti" (equal_or_dump2 gamma2 (neg (adjoint gamma2))));
            "gamma3" >::
               (fun () \rightarrow
                 assert_bool "anti" (equal_or_dump2 gamma3 (neg (adjoint gamma3))));
            "gamma5" >::
              (fun () \rightarrow
                 assert_bool "self" (equal_or_dump2 gamma5 (adjoint gamma5)))]
C^2 = -1 is not true in all realizations, but we assume it at several points in UFO_Lorentz. Therefore we must
test it here for all realizations that are implemented.
    let cc_inv = neg cc
Verify that \Gamma^T = -C\Gamma C^{-1} using the actual matrix transpose:
    let cc\_qamma\ q\ =
       equal\_or\_dump2 \ (neg \ (transpose \ g)) \ (product \ [cc; \ g; \ cc\_inv])
Of course, C = i\gamma^2\gamma^0 is also not true in all realizations. But it is true in the chiral representation used here
and we can test it.
    let charge\_conjugation =
       "charge_conjugation" >:::
          ["inverse" >::
              (fun () \rightarrow
                 assert_bool "" (equal_or_dump2 (mul cc cc_inv) unit));
            "gamma0" >:: (fun () \rightarrow assert\_bool "" (cc\_gamma\ gamma\theta));
            "gamma1" >:: (fun () \rightarrow assert_bool "" (cc_gamma gamma1));
            "gamma2" >:: (fun () \rightarrow assert_bool "" (cc_gamma gamma2));
            "gamma3" >:: (fun () \rightarrow assert_bool "" (cc_gamma gamma3));
            "gamma5" >::
              (\mathsf{fun}\ ()\ \to
                 assert\_bool "" (equal\_or\_dump2 (transpose gamma5)
                                                      (product [cc; gamma5; cc\_inv]));
            "=i*g2*g0" >::
              (fun () \rightarrow
                 skip\_if (\neg R.cc\_is\_i\_gamma2\_gamma\_0)
                   "representation_dependence";
                 assert\_bool "" (equal\_or\_dump2 \ cc \ (times \ i \ (mul \ gamma2 \ gamma0))))
    let test\_suite =
       "Dirac Matrices" >:::
         [anticommute;
           gamma5\_def;
           self\_adjoint;
           charge\_conjugation
  end
module \ Chiral\_R : R =
  struct
    \mathsf{type}\ qc\ =\ complex\_rational
    type t = matrix
    let qamma\theta =
       [| [| zero; zero; one; zero |];
           [| zero; zero; zero; one |];
           [| one; zero; zero; zero |];
          [| zero; one; zero; zero |] |]
    let gamma1 =
       [| [| zero; zero; zero; one |];
           [| zero; zero; one; zero |];
           [| zero; minus_one; zero; zero |];
```

```
[ minus_one; zero; zero; zero | | |
    let gamma2 =
       [| [| zero; zero; zero; minus_i |];
           [|zero; zero; i; zero|];
           [|zero; i; zero; zero|];
           [| minus_i; zero; zero; zero |] |]
    let gamma3 =
       [| [| zero; zero; one; zero |];
           [| zero; zero; zero; minus_one |];
           [| minus_one; zero; zero; zero |];
           [| zero; one; zero; zero |] |]
    let gamma5 =
       [| [| minus_one; zero; zero; zero |];
           [| zero; minus_one; zero; zero |];
           [| zero; zero; one; zero |];
           [| zero; zero; zero; one |] |]
    let cc =
       [| [| zero; one; zero; zero |];
           [| minus_one; zero; zero; zero |];
           [| zero; zero; zero; minus_one |];
           [| zero; zero; one; zero |] |]
    let cc\_is\_i\_gamma2\_gamma\_0 = true
  end
module Dirac_R : R =
  struct
    type qc = complex\_rational
    \mathsf{type}\ t\ =\ \mathit{matrix}
    let qamma\theta =
       [| [| one; zero; zero; zero |];
           [| zero; one; zero; zero |];
           | zero; zero; minus_one; zero | ;
           [| zero; zero; zero; minus_one |] |]
    let gamma1 = Chiral\_R.gamma1
    let gamma2 = Chiral\_R.gamma2
    let gamma3 = Chiral\_R.gamma3
    \mathsf{let} \ gamma5 \ =
       [| [| zero; zero; one; zero |];
           [| zero; zero; zero; one |];
           [| one; zero; zero; zero |];
           [\mid zero; one; zero; zero \mid] \mid]
    let cc =
       [| [| zero; zero; zero; minus_one |];
           [| zero; zero; one; zero |];
           [| zero; minus_one; zero; zero |];
           [| one; zero; zero; zero |] |]
    let cc\_is\_i\_gamma2\_gamma\_0 = true
  end
module Majorana_R : R =
  struct
    type qc = complex\_rational
    \mathsf{type}\ t\ =\ \mathit{matrix}
    \mathsf{let} \ gamma\theta \ = \\
       [| [| zero; zero; zero; minus_i |];
```

```
[|zero; zero; i; zero|];
          [| zero; minus_i; zero; zero |];
          [|i; zero; zero; zero|]|
    let gamma1 =
       [| [| i; zero; zero; zero |];
          [| zero; minus_i; zero; zero |];
          [|zero; zero; i; zero|];
          [| zero; zero; zero; minus_i |]
    let qamma2 =
       [| [| zero; zero; zero; i |];
          [| zero; zero; minus_i; zero |];
          [| zero; minus_i; zero; zero |];
          [|i; zero; zero; zero|]|
    let gamma3 =
       [| [| zero; minus_i; zero; zero |];
          [| minus_i; zero; zero; zero |];
          [| zero; zero; zero; minus_i |];
          [| zero; zero; minus_i; zero |] |]
    let gamma5 =
       [| [| zero; minus_i; zero; zero |];
          [|i; zero; zero; zero|];
          [\mid zero; zero; zero; i \mid];
          [|zero; zero; minus_i; zero|]|]
    let cc =
       [| [| zero; zero; zero; minus_one |];
          [| zero; zero; one; zero |];
          [| zero; minus_one; zero; zero |];
          [| one; zero; zero; zero |] |]
    let cc\_is\_i\_gamma2\_gamma\_0 = false
  end
module Chiral = Make (Chiral_R)
module \ Dirac = Make \ (Dirac\_R)
module Majorana = Make (Majorana_R)
```

9.5 Interface of Vertex

```
\begin{array}{lll} \text{val } parse\_string \ : \ string \rightarrow \ Vertex\_syntax.File.t \\ \text{val } parse\_file \ : \ string \rightarrow \ Vertex\_syntax.File.t \\ \\ \text{module type } Test \ = \\ & \text{sig} \\ & \text{val } example \ : \ unit \rightarrow \ unit \\ & \text{val } suite \ : \ OUnit.test \\ & \text{end} \\ \\ \text{module } Test \ (M \ : \ Model.T) \ : \ Test \\ \\ \text{module } Parser\_Test \ : \ Test \\ \\ \text{module } Modelfile\_Test \ : \ Test \\ \end{array}
```

9.6 Implementation of Vertex

Avoid referring to *Pervasives.compare*, because *Pervasives* will become *Stdlib.Pervasives* in O'Caml 4.07 and *Stdlib* in O'Caml 4.08.

```
\begin{array}{lll} {\sf let} \ pcompare &= \ compare \\ {\sf module} \ {\sf type} \ Test &= \end{array}
```

```
sig
     val\ example\ :\ unit 
ightarrow\ unit
     val suite : OUnit.test
  end
                                  9.6.1 New Implementation: Next Version
let \ error\_in\_string \ text \ start\_pos \ end\_pos =
  let i = start\_pos.Lexing.pos\_cnum
  and j = end_pos.Lexing.pos_cnum in
  String.sub\ text\ i\ (j\ -\ i)
let \ error\_in\_file \ name \ start\_pos \ end\_pos =
  Printf.sprintf
     "%s:%d.%d-%d.%d"
     name
     start_pos.Lexing.pos_lnum
     (start\_pos.Lexing.pos\_cnum - start\_pos.Lexing.pos\_bol)
     end\_pos.Lexing.pos\_lnum
     (end\_pos.Lexing.pos\_cnum - end\_pos.Lexing.pos\_bol)
let parse\_string text =
   Vertex\_syntax.File.expand\_includes
     (\mathsf{fun}\ \mathit{file}\ \rightarrow\ \mathit{invalid\_arg}\ (\texttt{"parse\_string:} \sqcup \mathsf{found} \sqcup \mathsf{include} \sqcup \texttt{`"}\ \hat{\ }\mathit{file}\ \hat{\ } \texttt{""}"))
         Vertex_parser.file
            Vertex\_lexer.token
           (Vertex_lexer.init_position "" (Lexing.from_string text))
      with
      |Vertex\_syntax\_Error\ (msg,\ start\_pos,\ end\_pos)| \rightarrow
         invalid_arg (Printf.sprintf "syntax_error_(%s)_at:_'%s'"
                            msg (error_in_string text start_pos end_pos))
      \mid Parsing.Parse\_error \rightarrow
         invalid_arg ("parse_error: " ^ text))
let parse\_file name =
  let parse\_file\_tree name =
     let ic = open\_in name in
     let file_tree =
       begin try
          Vertex\_parser.file
             Vertex\_lexer.token
             (Vertex_lexer.init_position name (Lexing.from_channel ic))
       with
          Vertex\_syntax.Syntax\_Error\ (msg,\ start\_pos,\ end\_pos) \rightarrow
             close\_in\ ic;
             invalid_arg (Printf.sprintf
                                \space{1.5}"%s:\syntax\error\(\%s\)"
                                (error_in_file name start_pos end_pos) msg)
        | Parsing.Parse\_error \rightarrow
          begin
             close\_in\ ic;
             invalid_arg ("parse_error: " ^ name)
          end
       end in
     close\_in\ ic;
     file_tree in
   Vertex_syntax.File.expand_includes parse_file_tree (parse_file_tree name)
let dump_file pfx f =
```

```
List.iter
    (\text{fun } s \rightarrow print\_endline (pfx ^ ": " ^ s))
    (Vertex\_syntax.File.to\_strings\ f)
module Parser\_Test : Test =
  struct
    let example () =
      ()
    open OUnit
    let compare s_out s_in () =
       assert\_equal \ \tilde{} printer : (String.concat \ " \sqcup ")
         [s\_out] (Vertex\_syntax.File.to\_strings (parse\_string s\_in))
    let parse\_error\ error\ s\ () =
       assert\_raises \ (Invalid\_argument \ error) \ (fun \ () \rightarrow parse\_string \ s)
    let syntax\_error (msq, error) s () =
      parse\_error \ ("syntax\_error\_(" \hat msg ^ ")\_at:\_``" ^ error ^ "'") \ s \ ()
    let (=>) s_in s_out =
       let (? >) s =
      s => s
    \mathsf{let} \ (=>!!!) \ s \ error \ =
       "_{\square}" \hat{} s > :: parse\_error error s
    let (=>!) s error =
       " \_ " \hat{\ } s > :: syntax\_error \ error \ s
    let empty =
       "empty" >::
         (fun () \rightarrow assert\_equal [] (parse\_string ""))
    let expr =
       "expr" >:::
         ["\vertex[2_{\sqcup}*_{\sqcup}(17_{\sqcup}+_{\sqcup}4)]{}" => "\vertex[42]{{}}";
           "\\vertex[2_{\cup}*_{\cup}17_{\cup}+_{\cup}4]{}" => "\\vertex[38]{{}}";
           "\\vertex[2" =>! ("missing_\']', "[2");
           "\\vertex]{}" =>! ("expected_', [', or_', ', "\\vertex]");
           "\\vertex2]{}" =>! ("expected_'['_\or_'{', "\\vertex2"});
           "\\vertex}{}" =>! ("expected_'(''_or_'(', "\\vertex}");
           "\\vertex2}{}" =>! ("expected_'[', or_'{', "\\vertex2"});
           "\\vertex[(2){}" =>! ("expected_\')', \_found_\'\}', "(2\}");
           "\\vertex[(2]{}" =>! ("expected_\')', \_found_\']', "(2]");
           "\\vertex{2]{}" =>! ("syntax_error", "2");
           "\\vertex[2]{}" =>! ("expected_\']', \_found_\'}', ", "[2]");
           "\\vertex[2{}" =>! ("syntax_error", "2");
           "\\vertex[2*]{}" =>! ("syntax_error", "2")]
    let index =
       "index" >:::
         ["\vertex{{a}_{-}{1}^{2}}" => "\vertex{a^2_1}";
           "\\vertex\{a_{11}^2\}" => "\\vertex\{a^2_{11}\}";
           "\\vertex\{a_{1_1}^2\}" => "\\vertex\{a^2_{1_1}\}"
    let \ electron1 =
       "electron1" >:::
         [? > "\\charged{e^-}{e^+}";
           "\\charged\{\{e^-\}\}\{\{e^+\}\}" => "\\charged\{e^-\}\{e^+\}" ]
    let \ electron2 =
       "electron2" >:::
         ["\charged{e^-}{e^+}\charged{e^-} =>
           "\\charged{e^-}{e^+}\\fortran{{ele}}}";
```

```
"\\charged{e^-}{e^+}\\fortran{electron}\\fortran{ele}" =>
      \label{eq:charged} $$ \operatorname{e^-}(e^+)\operatorname{charged}(e^-)^{e^+}} \
      \label{eq:charged} $$ \charged{e^-}{e^+}\\lambda[e^2]\times[e^2]^{=>} $$
      \label{eq:charged} $$ \operatorname{e^-}\{e^+}\leq \{e1\}} \alias{\{e2\}}";
      \label{eq:chargeder} $$ \c - {e^+} \operatorname{ele} \operatorname{lonti} \operatorname{pos} = > $$
      "\\charged{e^-}{e^+}\\fortran{{ele}}\\anti\\fortran{{pos}}" ]
let particles =
  "particles" >:::
    [electron1;
     electron2
let parameters =
  "parameters" >:::
    [ ? > "\\parameter{\\alpha}{1/137}";
      ?> "\derived{\\lambda_s}{1/\\ln{\\frac{\\mu_s}}}";
      ("invalid_parameter_attribute", "\\anti")]
let indices =
  "indices" >:::
    [? > "\\index{a}\\color{8}";
      let tensors =
  "tensors" >:::
    [ "\tensor{T}\color{3}" => "\tensor{T}\color{3}"]
let \ vertices =
  "vertex" >:::
    [ "\vertex{\\hoar\psi\gamma_\mu\psi_A_\mu}" => 
      "\vertex{{{\\bar\\psi\\gamma_\mu\\psi_A_\mu}}}" ]
module T = Vertex\_syntax.Token
let parse\_token s =
  match parse\_string ("\\vertex{" \hat{s} "}") with
  | [Vertex\_syntax.File.Vertex (\_, v)] \rightarrow v
  | _ → invalid_arg "only_vertex"
let print\_token pfx t =
  print\_endline\ (pfx \ ^ ": \_ " \ ^ T.to\_string\ t)
let test\_stem \ s\_out \ s\_in \ () =
  assert_equal ~printer: T.to_string
    (parse\_token \ s\_out)
    (T.stem (parse\_token s\_in))
let (=>>) s_i n s_o ut =
  "stem_{\sqcup}" ^ s_{-}in >:: test\_stem\ s_{-}out\ s_{-}in
let tokens =
  "tokens" >:::
    [ "\\vertex{a'}" => "\\vertex{a^\\prime}";
      "\\vertex\{a''\}" => "\\vertex\{a^{\langle \cdot \rangle}";
      "\\bar\\psi'',_{i,\\alpha}" =>> "\\psi";
      "\\phi^\\dagger_{i'}" =>> "\\phi";
      "\\bar{\\phi\\psi}''_{i,\\alpha}" =>> "\\psi";
      "\\vertex{\\phi}" => "\\vertex{\\phi}";
      "\\vertex{\\phi_1}" => "\\vertex{\\phi_1}";
      "\vertex{{\{\nhi\}'\}}" => "\vertex{\nhi^\prime}";}
      "\\text{\hat}{\hat}_1" => "\\text{\hat}\psi_1}";
      "\\vertex\{a_b\}_{cd}\}" => "\\vertex\{a_{bcd}\}";
      "\\vertex{{\\phi_1}_2}" => "\\vertex{\\phi_{12}}\";
      "\sqrt{12}_{34}" => "\sqrt{\frac{1234}}";
      "\\vertex{\{\\hat{12}}^{34}\}" => "\\vertex{\\hat{34}_{12}}";
```

```
let suite =
       "Vertex_Parser" >:::
         [empty;
          index;
          expr;
          particles;
          parameters;
          indices;
          tensors;
          vertices;
          tokens
  end
                                                Symbol Tables
module type Symbol =
  sig
    type file = Vertex\_syntax.File.t
    type t = Vertex\_syntax.Token.t
Tensors and their indices are representations of color, flavor or Lorentz groups. In the end it might turn out to
be unnecessary to distinguish Color from Flavor.
    type \ space =
      Color of Vertex_syntax.Lie.t
      Flavor of t list 	imes t list
    | Lorentz of t list
A symbol (i.e. a Symbol t = Vertex_syntax. Token.t) can refer either to particles, to parameters (derived and
input) or to tensors and indices.
    type kind =
      Neutral
      Charged
      Anti
      Parameter
      Derived
      Index of space
     | Tensor of space
    type table
    \mathsf{val}\ load\ :\ file\ \to\ table
    \mathsf{val}\ dump\ :\ out\_channel\ \to\ table\ \to\ unit
Look up the kind of a symbol.
    val \ kind\_of\_symbol : table \rightarrow t \rightarrow kind \ option
Look up the kind of a symbol's stem.
    val\ kind\_of\_stem\ :\ table\ 	o\ t\ 	o\ kind\ option
Look up the kind of a symbol and fall back to the kind of the symbol's stem, if necessary.
    \verb|val|| kind\_of\_symbol\_or\_stem|: table \rightarrow t \rightarrow kind option|
A table to look up all symbols with the same stem.
    \mathsf{val}\ common\_stem\ :\ table\ \to\ t\ \to\ t\ list
    exception Missing\_Space of t
    exception Conflicting\_Space of t
  end
module Symbol : Symbol =
```

```
struct
  module T = Vertex\_syntax.Token
  module F = Vertex\_syntax.File
  P = Vertex\_syntax.Particle
  module I = Vertex\_syntax.Index
  module L = Vertex\_syntax.Lie
  \mathsf{module}\ Q\ =\ \mathit{Vertex\_syntax}.\mathit{Parameter}
  \mathsf{module}\ X = \mathit{Vertex\_syntax}.\mathit{Tensor}
  type file = F.t
  type t = T.t
  type \ space =
    Color of L.t
    Flavor of t list \times t list
    Lorentz of t list
  let space\_to\_string = function
     \mid Color(g, r) \rightarrow
        "color:" ^ L.group\_to\_string g ^ ":" ^ L.rep\_to\_string r
       Flavor(\_, \_) \rightarrow "flavor"
      Lorentz \_ \rightarrow "Lorentz"
  type kind =
    Neutral
     Charged
    Anti
    Parameter
    Derived
     Index of space
    Tensor of space
  let kind\_to\_string = function
       Neutral \rightarrow "neutral_particle"
       Charged \rightarrow "charged \sqcup particle"
       Anti \rightarrow "charged_anti_particle"
       Parameter \rightarrow "input_{\sqcup}parameter"
       Derived \rightarrow "derived \square parameter"
       Index\ space\ 	o\ space\_to\_string\ space\ ^ " {\it \sqcup} index"
       Tensor\ space\ 	o\ space\_to\_string\ space\ ^ "_{\sqcup}tensor"
  module ST = Map.Make (T)
  module SS = Set.Make (T)
  type \ table =
       \{ symbol\_kinds : kind ST.t; \}
          stem\_kinds: kind ST.t;
          common_stems : SS.t ST.t }
  let empty =
       \{ symbol\_kinds = ST.empty; \}
          stem\_kinds = ST.empty;
          common\_stems = ST.empty  }
  let kind\_of\_symbol table token =
     try Some (ST.find \ token \ table.symbol\_kinds) with Not\_found \rightarrow None
  let kind\_of\_stem table token =
       Some (ST.find (T.stem token) table.stem\_kinds)
     with
     | Not\_found \rightarrow None
  let kind_of_symbol_or_stem symbol_table token =
     match kind\_of\_symbol\_table\ token with
     \mid Some \_ as kind \rightarrow kind
```

```
None \rightarrow kind\_of\_stem\ symbol\_table\ token
     let common_stem table token =
           SS.elements (ST.find (T.stem token) table.common_stems)
        with
        | Not\_found \rightarrow []
     let \ add\_symbol\_kind \ table \ token \ kind \ =
          let \ old\_kind = ST.find \ token \ table \ in
          if kind = old\_kind then
             table
          else
             invalid_arg ("conflicting_symbol_kind:_" ^
                                  T.to\_string\ token \ ^ " \_ - > \_ " \ ^
                                    kind\_to\_string\ kind ^ "uvsu" ^
                                      kind\_to\_string\ old\_kind)
        with
        \mid Not\_found \rightarrow ST.add \ token \ kind \ table
     let \ add\_stem\_kind \ table \ token \ kind =
        let stem = T.stem token in
        try
          \mathsf{let}\ \mathit{old\_kind}\ =\ \mathit{ST.find}\ \mathit{stem}\ \mathit{table}\ \mathsf{in}
          if kind = old\_kind then
             table
          else begin
                match kind, old\_kind with
                  Charged, \ Anti \ 	o \ ST.add \ stem \ Charged \ table
                  Anti, Charged \rightarrow table
                 \overline{\phantom{a}} \rightarrow
                    invalid\_arg ("conflicting_stem_kind:_" ^
                                        T.to\_string\ token \ ^ " \_ - > \_ " \ ^
                                           T.to\_string\ stem\ ^{\circ}\ "_{\sqcup}->_{\sqcup}" ^
                                             kind\_to\_string\ kind\ ^ "\_vs\_"\ ^
                                                kind_to_string old_kind)
             end
        with
        \mid Not\_found \rightarrow ST.add stem kind table
     let \ add\_kind \ table \ token \ kind =
        { table with
           symbol\_kinds = add\_symbol\_kind \ table.symbol\_kinds \ token \ kind;
          stem\_kinds = add\_stem\_kind table.stem\_kinds token kind 
     let \ add\_stem \ table \ token =
        let stem = T.stem token in
        \mathsf{let} \ \mathit{set} \ =
             ST.find\ stem\ table.common\_stems
          with
           | Not\_found \rightarrow SS.empty in
        { table with
           common\_stems = ST.add stem (SS.add token set) table.common\_stems 
Go through the list of attributes, make sure that the space is declared and unique. Return the space.
     exception Missing\_Space of t
     exception Conflicting\_Space of t
     let \ group\_rep\_of\_tokens \ group \ rep \ =
        let group =
          match group with
          [] \rightarrow L.default\_group
```

```
| group \rightarrow L.group\_of\_string (T.list\_to\_string group) in
           Color (group, L.rep\_of\_string group (T.list\_to\_string rep))
     let index\_space index =
       let spaces =
          List.fold\_left
             (fun acc \rightarrow function
              I.Color (group, rep) \rightarrow group\_rep\_of\_tokens group rep :: acc
               I.Flavor\ (group,\ rep) \rightarrow Flavor\ (rep,\ group) :: acc
               I.Lorentz \ t \rightarrow Lorentz \ t :: acc)
             [] index.I.attr in
       match ThoList.uniq (List.sort compare spaces) with
          [space] \rightarrow space
         [] \rightarrow raise (Missing\_Space index.I.name)
        \_ \rightarrow raise (Conflicting\_Space index.I.name)
     let tensor\_space tensor =
       let spaces =
          List.fold\_left
             (fun acc \rightarrow function
               X.Color (group, rep) \rightarrow group\_rep\_of\_tokens rep group :: acc
               X.Flavor\ (group,\ rep) \rightarrow Flavor\ (rep,\ group) :: acc
               X.Lorentz \ t \rightarrow Lorentz \ t :: acc)
             [] tensor.X.attr in
       match ThoList.uniq (List.sort compare spaces) with
        | [space] \rightarrow space
        [] \rightarrow raise (Missing\_Space tensor.X.name)
        | \_ \rightarrow raise (Conflicting\_Space tensor.X.name)|
NB: if P. Charged (name, name) below, only the Charged will survive, Anti will be shadowed.
     \mathsf{let}\ insert\_kind\ table\ =\ \mathsf{function}
        \mid F.Particle p \rightarrow
          begin match p.P.name with
          \mid P.Neutral \ name \rightarrow add\_kind \ table \ name \ Neutral
          \mid P.Charged (name, anti) \rightarrow
             add_kind (add_kind table anti Anti) name Charged
          end
          F.Index i \rightarrow add\_kind table i.I.name (Index (index\_space i))
          F.Tensor t \rightarrow add\_kind table t.X.name (Tensor (tensor\_space t))
        \mid F.Parameter p \rightarrow
          begin match p with
          |\ Q.Parameter\ name\ 
ightarrow\ add\_kind\ table\ name.Q.name\ Parameter
          Q.Derived\ name\ 	o\ add\_kind\ table\ name.Q.name\ Derived
          end
       \mid F.Vertex \_ \rightarrow table
     \mathsf{let}\ insert\_stem\ table\ =\ \mathsf{function}
        \mid F.Particle p \rightarrow
          begin match p.P.name with
           \mid P.Neutral \ name \rightarrow add\_stem \ table \ name
          \mid P.Charged (name, anti) \rightarrow add\_stem (add\_stem table name) anti
          end
          F.Index i \rightarrow add\_stem table i.I.name
          F.Tensor\ t \rightarrow add\_stem\ table\ t.X.name
        \mid F.Parameter p \rightarrow
          begin match p with
          | Q.Parameter name
          | Q.Derived name \rightarrow add\_stem table name.Q.name
          end
        \mid F.Vertex \rightarrow table
     let insert table token =
        insert_stem (insert_kind table token) token
```

```
lot load decls =
        List.fold_left insert empty decls
     \mathsf{let}\ \mathit{dump}\ \mathit{oc}\ \mathit{table}\ =
        Printf.fprintf\ oc\ "<<< \sqcup Symbol \sqcup Table: \sqcup>>> \n";
        ST.iter
           (fun s k \rightarrow
            Printf.fprintf\ oc\ "%s_{\sqcup}->_{\sqcup}%s_{\square}"\ (T.to\_string\ s)\ (kind\_to\_string\ k))
           table.symbol\_kinds;
        Printf.fprintf\ oc\ "<<< \sqcup Stem \sqcup Table: \sqcup >>> \n";
        ST.iter
           (fun s k \rightarrow
            Printf.fprintf\ oc\ "%s_{\sqcup}->_{\sqcup}%s_{\square}"\ (T.to\_string\ s)\ (kind\_to\_string\ k))
           table.stem\_kinds;
        Printf.fprintf oc "<<<ul>Common_Stems:_n";
        ST.iter
           (fun stem \ symbols \rightarrow
            Printf.fprintf
               oc "%s_{\square}->_{\square}%s_{\square}"
               (T.to\_string\ stem)
               (String.concat
                   ",\square" (List.map T.to_string (SS.elements symbols))))
           table.common\_stems
  end
                                                            Declarations
module type Declaration =
  sig
     type t
     val\ of\_string : string \rightarrow t\ list
     val to\_string: t list 	o string
For testing and debugging
     val\ of\_string\_and\_back\ :\ string 
ightarrow\ string
     val\ count\_indices\ :\ t\ 	o\ (int\ 	imes\ Symbol.t)\ list
     val\ indices\_ok : t \rightarrow unit
  end
module\ Declaration\ :\ Declaration\ =
  struct
     \mathsf{module}\ S\ =\ Symbol
     module T = Vertex\_syntax.Token
     \mathsf{type}\; factor \; = \;
        \{ stem : T.t;
           prefix : T.prefix list;
           particle: T.t list;
           color: T.t \ list;
           flavor : T.t \ list;
           lorentz : T.t \ list;
           other : T.t \ list \}
     type t = factor \ list
     let factor\_stem token =
        \{ stem = token.T.stem; 
           prefix = token.T.prefix;
           particle = [];
           color = [];
```

```
flavor = [];
     lorentz = [];
     other = [] 
let rev factor =
   \{ stem = factor.stem; \}
     prefix = List.rev factor.prefix;
     particle = List.rev factor.particle;
     color = List.rev factor.color;
     flavor = List.rev factor.flavor;
     lorentz = List.rev factor.lorentz;
     other = List.rev factor.other }
let factor_add_prefix factor token =
   \{ factor with prefix = T.prefix\_of\_string token :: factor.prefix \}
let factor\_add\_particle factor token =
   \{ factor \ with \ particle = token :: factor.particle \}
let factor_add_color_index t factor token =
   \{ factor \ with \ color = token :: factor.color \}
let factor_add_lorentz_index t factor token =
   (* diagnostics: Printf.eprintf "[L:[%s]]\n" (T.to_string token); *)
   \{ factor \ with \ lorentz = token :: factor.lorentz \}
let factor_add_flavor_index t factor token =
   \{ factor with flavor = token :: factor.flavor \}
{\tt let} \; factor\_add\_other\_index \; factor \; token \; = \;
   \{ factor \ with \ other = token :: factor.other \}
let factor\_add\_kind\ factor\ token\ =\ function
    S.Neutral \mid S.Charged \mid S.Anti \rightarrow factor\_add\_particle factor token
    S.Index\ (S.Color\ (rep,\ group)) \rightarrow
      factor_add_color_index (rep, group) factor token
    S.Index\ (S.Flavor\ (rep,\ group)) \rightarrow
      factor\_add\_flavor\_index\ (rep,\ group)\ factor\ token
    S.Index\ (S.Lorentz\ t) \rightarrow factor\_add\_lorentz\_index\ t\ factor\ token
    S.Tensor \_ \rightarrow invalid\_arg "factor_add_index:\_\setminus \tensor"
    S.Parameter \rightarrow invalid\_arg "factor_add_index:_\\parameter"
    S.Derived \rightarrow invalid\_arg "factor\_add\_index:_\derived"
let factor\_add\_index \ symbol\_table \ factor = function
     T.Token "," \rightarrow factor
     T.Token ("*" | "\setminus ast" as star) \rightarrow factor\_add\_prefix factor star
    token \rightarrow
      begin
         match \ S.kind\_of\_symbol\_or\_stem \ symbol\_table \ token \ with
         | Some kind \rightarrow factor_add_kind factor token kind
           None \rightarrow factor\_add\_other\_index\ factor\ token
      end
let factor_of_token symbol_table token =
  let \ token \ = \ T.wrap\_scripted \ token \ in
  rev (List.fold_left
           (factor\_add\_index\ symbol\_table)
           (factor_stem token)
           (token.T.super @ token.T.sub))
let list\_to\_string \ tag = function
  | [] \rightarrow ""
  | l \rightarrow "; \square" \hat{tag} = "`String.concat", "(List.map T.to_string l)
let factor\_to\_string factor =
    "[" ^{^{\circ}} T.to\_string\ factor.stem
      (match factor.prefix with
```

```
| [] \rightarrow ""
                \rightarrow "; prefix=" \hat{}
                       String.concat "," (List.map T.prefix_to_string l)) ^
              list\_to\_string "particle" factor.particle
              list_to_string "color" factor.color `
              list\_to\_string "flavor" factor.flavor
              list\_to\_string "lorentz" factor.lorentz ^
              list\_to\_string "other" factor.other ^ "]"
     let count_indices factors =
        ThoList.classify
          (ThoList.flatmap (fun f \rightarrow f.color @ f.flavor @ f.lorentz) factors)
     let format_mismatch (n, index) =
        Printf.sprintf "index_\%s_appears_\%d_\times" (T.to\_string\ index) n
     let indices\_ok factors =
       match List.filter (fun (n, -) \rightarrow n \neq 2) (count_indices factors) with
         [] \rightarrow ()
        \mid mismatches \rightarrow
           invalid_arg (String.concat ", " (List.map format_mismatch mismatches))
     let of\_string s =
       let \ decls = parse\_string \ s \ in
       let symbol\_table = Symbol.load decls in
       (* diagnostics: Symbol.dump stderr symbol_table; *)
       let tokens =
          List.fold\_left
             (\text{fun } acc \ \rightarrow \ \text{function}
               Vertex\_syntax.File.Vertex(\_, v) \rightarrow T.wrap\_list v :: acc
              \rightarrow acc
            [] decls in
       let \ vlist = List.map \ (List.map \ (factor\_of\_token \ symbol\_table)) \ tokens \ in
       List.iter indices_ok vlist;
       vlist
     let to\_string\ decls\ =
       String.concat "; "
          (List.map
              (\text{fun } v \rightarrow String.concat "$\sqcup *$\sqcup " (List.map factor\_to\_string v))
              decls)
     let of\_string\_and\_back \ s =
       to\_string\ (of\_string\ s)
     type field =
        \{ name : T.t \ list \}
  end
                                                     Complete Models
module Modelfile =
  struct
  end
module Modelfile\_Test =
  struct
     let example () =
     open OUnit
     let index\_mismatches =
        "index_{\sqcup}mismatches" >:::
```

```
[ "1" >::
                                                (fun () \rightarrow
                                                      assert\_raises
                                                                 (Invalid\_argument "index\_a\_1\_appears\_1\_times,\_)
\sqcup \sqcup index \sqcup a - 2 \sqcup appears \sqcup 1 \sqcup times"
                                                                 (fun () \rightarrow Declaration.of\_string\_and\_back)
                                                                                                                                        \ \\index{a}\\color{3}\
"3" >::
                                                (fun () \rightarrow
                                                      assert\_raises
                                                                 (Invalid\_argument "index\_a\_appears\_3\_times")
                                                                 (fun () \rightarrow Declaration.of\_string\_and\_back)
                                                                                                                                       "\index{a}\\color{3}\
let kind\_conflicts =
                 "kind conflictings" >:::
                           [ "lorentz_ / _color" >::
                                                 (fun () \rightarrow
                                                       assert\_raises
                                                                 (Invalid\_argument
                                                                                  "conflicting_{\sqcup}stem_{\sqcup}kind:_{\sqcup}a_{-}2_{\sqcup}->_{\sqcup}a_{\sqcup}->_{\sqcup}\backslash
___Lorentz_index_vs_color:SU(3):3_index")
                                                                 (fun () \rightarrow Declaration.of\_string\_and\_back)
                                                                                                                                       "color<sub>□</sub>/<sub>□</sub>color" >::
                                                 (fun () \rightarrow
                                                      assert\_raises
                                                                 (Invalid\_argument
                                                                                 "conflicting\_stem\_kind:\_a\_2_{\sqcup}->_{\sqcup}a_{\sqcup}->_{\sqcup}\backslash
___color:SU(3):8_index_vs_color:SU(3):3_index")
                                                                 (fun () \rightarrow Declaration.of\_string\_and\_back
                                                                                                                                        "neutral_/_charged" >::
                                                 (fun () \rightarrow
                                                      assert\_raises
                                                                 (Invalid\_argument
                                                                                  "conflicting_stem_kind:_{\square}H^{-}_{\square}->_{\square}H_{\square}->_{\square}\setminus
uuuchargeduantiuparticleuvsuneutraluparticle")
                                                                 (fun () \rightarrow Declaration.of\_string\_and\_back)
                                                                                                                                       "\\
\Box\Box\Box\Box\Box\\charged{H^+}{H^-}"))
    let suite =
                 "Modelfile_Test" >:::
                           [ "ok" >::
                                                (fun () \rightarrow
                                                            assert\_equal \ \tilde{} printer : (fun \ s \ \rightarrow \ s)
                                                                       "[\\psi;_prefix=\\bar;_\
\verb|uuuuuuuuuuuuuparticle=e;|| color=a;|| lorentz= \verb|\alpha_1|| | u*|| | u*|| | lorentz= \verb|\alpha_1|| | u*|| |
uuuuuuuuuu [\\gamma;ulorentz=\\mu,\\alpha_1,\\alpha_2]u*u\
uuuuuuuuuu[\\psi;uparticle=e;ucolor=a;ulorentz=\\alpha_2]u*u\
uuuuuuuuuuuuu [A;ulorentz=\\mu]"
                                                                       (Declaration.of\_string\_and\_back
                                                                                      "\\charged\{e^-\}\{e^+\}\
\verb| uuuuuuuuuuu| \texttt{X} \\ \\ | u \\ |
\verb| uuuuuuuuuuu| \land \verb| index{\| alpha} \land \verb| lorentz{X}| \land \verb| alpha| \land alpha
```

```
index\_mismatches;
       kind\_conflicts;
       "QCD.omf" >::
        (fun () \rightarrow
          dump_file "QCD" (parse_file "QCD.omf"));
       "SM.omf" >::
        (fun () \rightarrow
          dump_file "SM" (parse_file "SM.omf"));
       "SM-error.omf" >::
        (fun () \rightarrow
         assert\_raises
           (Invalid\_argument
              "SM-error.omf:32.22-32.27: \_syntax\_error\_(syntax\_error)")
           (fun () \rightarrow parse\_file "SM-error.omf"));
       "cyclic.omf" >::
        (\mathsf{fun}\ ()\ \to
         assert\_raises
           (Invalid_argument "cyclic_\\include{cyclic.omf}")
           (fun () \rightarrow parse\_file "cyclic.omf"))]
end
```

9.6.2 New Implementation: Obsolete Version 1

Start of version 1 of the new implementation. The old syntax will not be used in the real implementation, but the library for dealing with indices and permutations will remail important.

Note that $arity = length\ lorentz_reps = length\ color_reps$. Do we need to enforce this by an abstract type constructor?

A cleaner approach would be type context = (Coupling.lorentz, Color.t) array, but it would also require more tedious deconstruction of the pairs. Well, an abstract type with accessors might be the way to go after all . . .

An abstract type that allows us to distinguish offsets in the field array from color and Lorentz indices in different representations.

```
\begin{array}{ll} \text{module type } Index &= \\ \text{sig} \\ \text{type } t \\ \text{val } of\_int \ : \ int \rightarrow \ t \\ \text{val } to\_int \ : \ t \ \rightarrow \ int \\ \text{end} \end{array}
```

While the number of allowed indices is unlimited, the allowed offsets into the field arrays are of course restricted to the fields in the current *context*.

```
module type Field = sig
```

```
type t
     exception Out\_of\_range of int
     val\ of\_int: context \rightarrow int \rightarrow t
     val to_int : t \rightarrow int
     val get: \alpha \ array \rightarrow t \rightarrow \alpha
module Field : Field =
  struct
     type t = int
     exception Out\_of\_range of int
     let of_int \ context \ i =
       if 0 \le i \land i < context.arity then
       else
          raise (Out\_of\_range i)
     let to_int i = 0
     let get = Array.get
  end
type field = Field.t
module type Lorentz =
  sig
```

We combine indices I and offsets F into the field array into a single type so that we can unify vectors with vector components.

```
type index = I of int \mid F of field

type vector = Vector of index

type spinor = Spinor of index

type conjspinor = ConjSpinor of index
```

These are all the primitive ways to construct Lorentz tensors, a. k. a. objects with Lorentz indices, from momenta, other Lorentz tensors and Dirac spinors:

```
 \begin{array}{l} \text{type } primitive \ = \\ \mid G \text{ of } vector \ \times \ vector \ (* \ g_{\mu_1\mu_2} \ *) \\ \mid E \text{ of } vector \ \times \ vector \ \times \ vector \ (* \ \epsilon_{\mu_1\mu_2\mu_3\mu_4} \ *) \\ \mid K \text{ of } vector \ \times \ field \ (* \ k_2^{\mu_1} \ *) \\ \mid S \text{ of } conjspinor \ \times \ spinor \ (* \ \bar{\psi}_1\psi_2 \ *) \\ \mid V \text{ of } vector \ \times \ conjspinor \ \times \ spinor \ (* \ \bar{\psi}_1\gamma_{\mu_2}\psi_3 \ *) \\ \mid T \text{ of } vector \ \times \ vector \ \times \ conjspinor \ \times \ spinor \ (* \ \bar{\psi}_1\gamma_{\mu_2}\gamma_5\psi_3 \ *) \\ \mid A \text{ of } vector \ \times \ conjspinor \ \times \ spinor \ (* \ \bar{\psi}_1\gamma_{\mu_2}\gamma_5\psi_3 \ *) \\ \mid P \text{ of } conjspinor \ \times \ spinor \ (* \ \bar{\psi}_1\gamma_5\psi_2 \ *) \\ \end{aligned}   \text{type } tensor \ = \ int \ \times \ primitive \ list
```

Below, we will need to permute fields. For this purpose, we introduce the function $map_primitive\ v_idx\ v_fld\ s_idx\ s_fld\ c_idx$ that returns a structurally identical tensor, with $v_idx\ :\ int\ \to\ int$ applied to all vector indices, $v_fld\ :\ field\ \to\ field$ to all vector fields, s_idx and c_idx to all (conj)spinor indices and s_fld and c_fld to all (conj)spinor fields.

Note we must treat spinors and vectors differently, even for simple permuations, in order to handle the statistics properly.

```
val map\_tensor: (int \rightarrow int) \rightarrow (field \rightarrow field) \rightarrow (int \rightarrow int) \rightarrow (field \rightarrow field) \rightarrow (int \rightarrow int) \rightarrow (field \rightarrow field) \rightarrow tensor \rightarrow tensor
```

Check whether the tensor is well formed in the context.

```
val\ tensor\_ok\ :\ context\ 	o\ tensor\ 	o\ bool
```

The lattice $\mathbf{N}+\mathrm{i}\mathbf{N}\subset\mathbf{C}$, which suffices for representing the matrix elements of Dirac matrices. We hope to be able to avoid the lattice $\mathbf{Q}+\mathrm{i}\mathbf{Q}\subset\mathbf{C}$ or \mathbf{C} itself down the road.

```
module Complex :
```

```
\begin{array}{l} \text{sig} \\ \text{type } t = int \times int \\ \text{type } t' = \\ \mid Z (* 0 *) \\ \mid O (* 1 *) \\ \mid M (* - 1 *) \\ \mid I (* i *) \\ \mid J (* - i *) \\ \mid C \text{ of } int \times int \ (* \ x + iy \ *) \\ \text{val } to\_fortran \ : \ t' \rightarrow string \\ \text{end} \end{array}
```

Sparse Dirac matrices as maps from Lorentz and Spinor indices to complex numbers. This is supposed to be independent of the representation.

```
\begin{array}{l} \text{module type } \textit{Dirac} = \\ \text{sig} \\ \text{val } \textit{scalar} : \textit{int} \rightarrow \textit{int} \rightarrow \textit{Complex.t'} \\ \text{val } \textit{vector} : \textit{int} \rightarrow \textit{int} \rightarrow \textit{int} \rightarrow \textit{Complex.t'} \\ \text{val } \textit{tensor} : \textit{int} \rightarrow \textit{int} \rightarrow \textit{int} \rightarrow \textit{int} \rightarrow \textit{Complex.t'} \\ \text{val } \textit{axial} : \textit{int} \rightarrow \textit{int} \rightarrow \textit{int} \rightarrow \textit{Complex.t'} \\ \text{val } \textit{pseudo} : \textit{int} \rightarrow \textit{int} \rightarrow \textit{Complex.t'} \\ \text{end} \end{array}
```

Dirac matrices as tables of nonzero entries. There will be one concrete Module per realization.

```
\begin{array}{lll} \text{module type } \textit{Dirac\_Matrices} &= \\ \text{sig} \\ \text{type } t &= (int \times int \times \textit{Complex.t'}) \; \textit{list} \\ \text{val } \textit{scalar} \; : \; t \\ \text{val } \textit{vector} \; : \; (int \times t) \; \textit{list} \\ \text{val } \textit{tensor} \; : \; (int \times int \times t) \; \textit{list} \\ \text{val } \textit{axial} \; : \; (int \times t) \; \textit{list} \\ \text{val } \textit{pseudo} \; : \; t \\ \text{end} \end{array}
```

E.g. the chiral representation:

```
module\ Chiral\ :\ Dirac\_Matrices
```

Here's the functor to create the maps corresponding to a given realization.

```
\begin{array}{lll} \operatorname{module} \ Dirac : \ \operatorname{functor} \ (M : Dirac\_Matrices) \ \rightarrow \ Dirac \\ & \operatorname{end} \\ \\ \operatorname{module} \ Lorentz : \ Lorentz = \\ & \operatorname{struct} \\ \\ \operatorname{type} \ index = \\ & | \ I \ \operatorname{of} \ int \ (* \ \mu_0, \mu_1, \ldots, \ \operatorname{not} \ 0, 1, 2, 3 \ *) \\ & | \ F \ \operatorname{of} \ field \\ \\ \operatorname{let} \ map\_index \ fi \ ff = \ \operatorname{function} \\ & | \ I \ i \ \rightarrow \ I \ (fi \ i) \\ & | \ F \ i \ \rightarrow \ F \ (ff \ i) \\ \\ \operatorname{let} \ indices = \ \operatorname{function} \\ & | \ I \ i \ \rightarrow \ [i] \\ & | \ F \ \_ \ \rightarrow \ [] \end{array}
```

Is the following level of type checks useful or redundant?

```
TODO: should we also support a tensor like F_{\mu_1\mu_2}?
```

```
type vector = Vector of index

type spinor = Spinor of index

type conjspinor = ConjSpinor of index

let map\_vector fi ff (Vector\ i) = Vector\ (map\_index\ fi\ ff\ i)
```

```
let map\_spinor\ fi\ ff\ (Spinor\ i)\ =\ Spinor\ (map\_index\ fi\ ff\ i)
     let map\_conjspinor\ fi\ ff\ (ConjSpinor\ i)\ =\ ConjSpinor\ (map\_index\ fi\ ff\ i)
     \mathsf{let}\ \mathit{vector\_ok}\ \mathit{context}\ =\ \mathsf{function}
       \mid Vector(I_{-}) \rightarrow
          (* we could perform additional checks! *)
          true
        \mid Vector(F i) \rightarrow
             begin
                match Field.get context.lorentz_reps i with
                  Coupling.Vector \rightarrow true
                  Coupling.Vectorspinor \rightarrow
                     failwith "Lorentz.vector_ok: incomplete"
             end
     let spinor_ok context = function
        \mid Spinor (I \_) \rightarrow
          (* we could perfrom additional checks! *)
        \mid Spinor(F i) \rightarrow
             begin
               match Field.get context.lorentz_reps i with
                  Coupling.Spinor \rightarrow true
                  Coupling.Vectorspinor \mid Coupling.Majorana \rightarrow
                     failwith "Lorentz.spinor_ok:_incomplete"
                  _{-} \rightarrow false
             end
     let conjspinor\_ok context = function
        | ConjSpinor (I_{-}) \rightarrow
          (* we could perform additional checks! *)
          ConjSpinor(F i) \rightarrow
             begin
               match Field.get context.lorentz_reps i with
                  Coupling.ConjSpinor \rightarrow true
                 Coupling.Vectorspinor \mid Coupling.Majorana \rightarrow
                     failwith "Lorentz.conjspinor_ok:\sqcupincomplete"
                  _{-} 
ightarrow false
             end
the same slot. This is however not the case for Weyl and Majorana spinors.
     let spinor\_sandwitch\_ok context i j =
```

Note that distinct i j is automatically guaranteed for Dirac spinors, because the $\bar{\psi}$ and ψ can not appear in

```
conjspinor\_ok context i \land spinor\_ok context j
type primitive =
     G 	ext{ of } vector 	imes vector
     E 	ext{ of } vector 	imes vector 	imes vector 	imes vector
     K 	ext{ of } vector 	imes field
     S of conjspinor \times spinor
      V of vector \times conjspinor \times spinor
     T of vector \times vector \times conjspinor \times spinor
     A of vector \times conjspinor \times spinor
     P of conjspinor \times spinor
{\tt let} \ \mathit{map\_primitive} \ \mathit{fvi} \ \mathit{fvi} \ \mathit{fsi} \ \mathit{fsf} \ \mathit{fci} \ \mathit{fcf} \ = \ {\tt function}
   \mid G(mu, nu) \rightarrow
         G (map_vector fvi fvf mu, map_vector fvi fvf nu)
   \mid E (mu, nu, rho, sigma) \rightarrow
         E (map\_vector fvi fvf mu,
             map\_vector fvi fvf nu,
             map_vector fvi fvf rho,
```

```
map_vector fvi fvf sigma)
  \mid K(mu, i) \rightarrow
        K (map_vector fvi fvf mu, fvf i)
    S(i, j) \rightarrow
        S (map\_conjspinor fci fcf i, map\_spinor fsi fsf j)
    V (mu, i, j) \rightarrow
        V (map\_vector fvi fvf mu,
            map\_conjspinor\ fci\ fcf\ i,
            map\_spinor\ fsi\ fsf\ j)
   \mid T (mu, nu, i, j) \rightarrow
        T (map_vector fvi fvf mu,
            map_vector fvi fvf nu,
            map\_conjspinor\ fci\ fcf\ i,
            map\_spinor\ fsi\ fsf\ j)
  A (mu, i, j) \rightarrow
        A (map\_vector fvi fvf mu,
            map\_conjspinor\ fci\ fcf\ i,
            map\_spinor\ fsi\ fsf\ j)
  \mid P(i, j) \rightarrow
        P (map\_conjspinor fci fcf i, map\_spinor fsi fsf j)
let primitive_ok context =
  function
     \mid G(mu, nu) \rightarrow
           distinct2 \ mu \ nu \ \land
           vector\_ok context mu \land vector\_ok context nu
     \mid E (mu, nu, rho, sigma) \rightarrow
          let i = [mu; nu; rho; sigma] in
           distinct \ i \ \land \ List.for\_all \ (vector\_ok \ context) \ i
     \mid K(mu, i) \rightarrow
           vector\_ok\ context\ mu
     \mid S(i, j) \mid P(i, j) \rightarrow
           spinor_sandwitch_ok context i j
     V(mu, i, j) \mid A(mu, i, j) \rightarrow
           vector\_ok\ context\ mu\ \land\ spinor\_sandwitch\_ok\ context\ i\ j
     \mid T (mu, nu, i, j) \rightarrow
           vector\_ok\ context\ mu\ \land\ vector\_ok\ context\ nu\ \land
           spinor_sandwitch_ok context i j
let primitive\_vector\_indices = function
   | G (Vector mu, Vector nu) | T (Vector mu, Vector nu, \_, \_) \rightarrow
        indices\ mu\ @\ indices\ nu
   \mid E \text{ (Vector } mu, \text{ Vector } nu, \text{ Vector } rho, \text{ Vector } sigma) \rightarrow
        indices mu @ indices nu @ indices rho @ indices sigma
     K (Vector mu, _{-})
     V (Vector mu, \_, \_)
     A \ (Vector \ mu, \ \_, \ \_) \rightarrow indices \ mu
   \mid S(\_,\_) \mid P(\_,\_) \rightarrow []
let vector\_indices p =
   ThoList.flatmap primitive_vector_indices p
\mathsf{let}\ \mathit{primitive\_spinor\_indices}\ =\ \mathsf{function}
   \mid G(\_,\_) \mid E(\_,\_,\_,\_) \mid K(\_,\_) \rightarrow []
     S (_, Spinor\ alpha) | V (_, _, Spinor\ alpha)
     T (\_, \_, \_, Spinor\ alpha)
    A (_, _, Spinor\ alpha) | P (_, Spinor\ alpha) \rightarrow\ indices\ alpha
let spinor\_indices p =
   ThoList.flatmap primitive_spinor_indices p
let primitive\_conjspinor\_indices = function
    G(-, -) \mid E(-, -, -, -) \mid K(-, -) \rightarrow []
   \mid S (ConjSpinor alpha, \_) \mid V (\_, ConjSpinor alpha, \_)
```

```
T (\_, \_, ConjSpinor alpha, \_)
  A (\_, ConjSpinor alpha, \_) \mid P (ConjSpinor alpha, \_) \rightarrow indices alpha
let \ conjspinor\_indices \ p =
   Tho List. flat map\ primitive\_conj spinor\_indices\ p
let \ vector\_contraction\_ok \ p = 1
  let c = ThoList.classify (vector\_indices p) in
  print\_endline
     (String.concat ", "
          (List.map
              (\text{fun }(n, i) \rightarrow string\_of\_int n ^ "_{\sqcup}*_{\sqcup}" ^ string\_of\_int i)
  flush stdout;
  let res = List.for\_all 	ext{ (fun } (n, \_) \rightarrow n = 2) c 	ext{ in}
  res
let two\_of\_each indices p =
   List.for\_all \text{ (fun } (n, \_) \rightarrow n = 2) \text{ (} ThoList.classify (indices p)\text{)}
let vector_contraction_ok = two_of_each vector_indices
{\tt let}\ spinor\_contraction\_ok\ =\ two\_of\_each\ spinor\_indices
let\ conjspinor\_contraction\_ok\ =\ two\_of\_each\ conjspinor\_indices
let contraction\_ok p =
   vector\_contraction\_ok p \land
  spinor\_contraction\_ok\ p\ \land\ conjspinor\_contraction\_ok\ p
type tensor = int \times primitive \ list
let map_tensor fvi fvf fsi fsf fci fcf (factor, primitives) =
  (factor, List.map (map_primitive fvi fvf fsi fsf fci fcf ) primitives)
let \ tensor\_ok \ context \ (\_, \ primitives) =
   List.for\_all\ (primitive\_ok\ context)\ primitives\ \land
   contraction\_ok primitives
module\ Complex\ =
  struct
     type t = int \times int
     type t' = Z \mid O \mid M \mid I \mid J \mid C of int \times int
     \mathsf{let}\ to\_fortran\ =\ \mathsf{function}
        |Z \rightarrow "(0,0)"
          O \rightarrow "(1,0)"
          M \rightarrow "(-1,0)"
         \mid I \rightarrow "(0,1)"
          J \rightarrow "(0,-1)"
         C(r, i) \rightarrow "(" \hat{string\_of\_int} r ", " \hat{string\_of\_int} i ")"
  end
module type Dirac =
     val\ scalar\ :\ int \rightarrow\ int \rightarrow\ Complex.t'
     val\ vector: int \rightarrow int \rightarrow int \rightarrow Complex.t'
     \mathsf{val}\ tensor\ :\ int \to\ int \to\ int \to\ int \to\ Complex.t'
     val axial : int \rightarrow int \rightarrow int \rightarrow Complex.t'
     val\ pseudo: int \rightarrow int \rightarrow Complex.t'
module type Dirac\_Matrices =
     type t = (int \times int \times Complex.t') list
     val\ scalar\ :\ t
     val\ vector\ :\ (int \times t)\ list
     val\ tensor\ :\ (int \times int \times t)\ list
```

```
val \ axial : (int \times t) \ list
    val pseudo : t
module\ Chiral\ :\ Dirac\_Matrices\ =
  struct
    type t = (int \times int \times Complex.t') list
    let scalar =
       [(1, 1, Complex. O);
         (2, 2, Complex.O);
         (3, 3, Complex.O);
         (4, 4, Complex.O)
    let \ vector =
       [ (0, [ (1, 4, Complex. O);
                (4, 1, Complex. O);
                (2, 3, Complex.M);
                (3, 2, Complex.M)]);
         (1, [(1, 3, Complex. O);
                (3, 1, Complex. O);
                (2, 4, Complex.M);
                (4, 2, Complex.M) ]);
         (2, [(1, 3, Complex.I);
                (3, 1, Complex.I);
                (2, 4, Complex.I);
                (4, 2, Complex.I);
         (3, [(1, 4, Complex.M);
                (4, 1, Complex.M);
                (2, 3, Complex.M);
                (3, 2, Complex.M)])
    let \ tensor =
       [ (* TODO!!! *) ]
    let axial =
       [\ (0,\ [\ (1,\ 4,\ Complex.M);
                (4, 1, Complex. O);
                (2, 3, Complex. O);
                (3, 2, Complex.M)]);
         (1, [(1, 3, Complex.M);
                (3, 1, Complex. O);
                (2, 4, Complex. O);
                (4, 2, Complex.M) ]);
         (2, [(1, 3, Complex.J);
                (3, 1, Complex.I);
                (2, 4, Complex.J);
                (4, 2, Complex.I) ]);
         (3, [(1, 4, Complex. O);
                (4, 1, Complex.M);
                (2, 3, Complex. O);
                (3, 2, Complex.M)])]
    \mathsf{let}\ pseudo\ =
       [(1, 1, Complex.M);
         (2, 2, Complex.M);
         (3, 3, Complex.O);
         (4, 4, Complex.O)
  end
module\ Dirac\ (M\ :\ Dirac\_Matrices)\ :\ Dirac\ =
  struct
    module Map2 =
```

```
Map.Make
     (struct
       \mathsf{type}\ t\ =\ int\ \times\ int
       let \ compare = pcompare
     end)
let init2 triples =
  List.fold\_left
     (fun acc\ (i,\ j,\ e)\ \rightarrow\ Map2.add\ (i,\ j)\ e\ acc)
     Map2.empty triples
let bounds\_check2 i j =
  if i < 1 \lor i > 4 \lor j < 0 \lor j > 4 then
     invalid_arg "Chiral.bounds_check2"
let lookup2 \ map \ i \ j =
  bounds\_check2 \ i \ j;
  try Map2.find (i, j) map with Not\_found \rightarrow Complex.Z
module Map3 =
  Map.Make
     (struct
       type t = int \times (int \times int)
       let compare = pcompare
     end)
let \ init 3 \ quadruples \ =
  List.fold\_left
     (fun acc\ (mu,\ gamma)\ 	o
      List.fold\_right
        (\text{fun } (i, j, e) \rightarrow Map3.add (mu, (i, j)) e)
        gamma acc)
     Map3.empty quadruples
let bounds\_check3 mu i j =
  bounds\_check2 \ i \ j;
  if mu < 0 \lor mu > 3 then
     invalid\_arg "Chiral.bounds_check3"
let lookup3 \ map \ mu \ i \ j =
  bounds_check3 mu i j;
  try Map3.find\ (mu,\ (i,\ j))\ map\ with\ Not\_found\ 	o\ Complex.Z
module Map4 =
  Map.Make
     (struct
       \mathsf{type}\ t\ =\ int\ \times\ int\ \times\ (int\ \times\ int)
       let compare = pcompare
     end)
let init4 quadruples =
  List.fold\_left
     (fun \ acc \ (mu, \ nu, \ gamma) \rightarrow
      List.fold\_right
        (\text{fun }(i, j, e) \rightarrow Map4.add (mu, nu, (i, j)) e)
        gamma acc)
     Map4.empty quadruples
let bounds\_check4 mu nu i j =
  bounds\_check3 nu i j;
  if mu < 0 \lor mu > 3 then
     invalid_arg "Chiral.bounds_check4"
let lookup4 map mu nu i j =
  bounds_check4 mu nu i j;
  try Map4.find\ (mu,\ nu,\ (i,\ j))\ map\ with\ Not\_found\ 	o\ Complex.Z
```

```
let scalar map = init2 M.scalar
           let \ vector\_map = init3 \ M.vector
           let tensor\_map = init4 M.tensor
           let axial\_map = init3 M.axial
           let pseudo\_map = init2 M.pseudo
           let scalar = lookup2 scalar\_map
           let \ vector = lookup3 \ vector\_map
           \mathsf{let}\ tensor\ mu\ nu\ i\ j\ =
              lookup4 tensor_map mu nu i j
           let tensor mu nu i j =
              failwith "tensor: uincomplete"
           let axial = lookup3 axial\_map
           let pseudo = lookup2 pseudo\_map
  end
module type Color =
  sig
     module\ Index\ :\ Index
     type index = Index.t
     type color\_rep = F of field \mid C of field \mid A of field
     type primitive =
           D of field \times field
           E 	ext{ of } field 	imes field 	imes field (* only for <math>SU(3) *)
           T of field \times field \times field
           F 	ext{ of } field 	imes field 	imes field 	imes field
     val map\_primitive : (field \rightarrow field) \rightarrow primitive \rightarrow primitive
     \verb|val|| primitive\_indices|: primitive| \rightarrow field| list|
     \mathsf{val}\ indices\ :\ primitive\ list \to\ field\ list
     type \ tensor = int \times primitive \ list
     val\ map\_tensor :
        (field \rightarrow field) \rightarrow \alpha \times primitive \ list \rightarrow \alpha \times primitive \ list
     val tensor\_ok : context \rightarrow \alpha \times primitive \ list \rightarrow \ bool
module\ Color\ :\ Color\ =
  struct
     module\ Index\ :\ Index\ =
           \mathsf{type}\ t\ =\ int
           let of_int i = i
           let to\_int i = i
a_0, a_1, \ldots, \text{ not } 0, 1, \ldots
     type index = Index.t
     \mathsf{type}\ color\_rep\ =
           F of field
           C of field
         | A of field
     {\sf type} \,\, primitive \,\, = \,\,
         \mid D \text{ of } field \times field
           E 	ext{ of } field 	imes field 	imes field 	imes field
           T 	ext{ of } field 	imes field 	imes field 	imes field
        \mid F \text{ of } field \times field \times field
     let map\_primitive f = function
         | D(i, j) \rightarrow D(f i, f j)|
           E(i, j, k) \rightarrow E(f i, f j, f k)
        T(a, i, j) \rightarrow T(f a, f i, f j)
```

```
\mid F(a, b, c) \rightarrow F(f a, f b, f c)
     let primitive\_ok \ ctx =
       function
          \mid D(i, j) \rightarrow
                distinct2 \ i \ j \ \land
               (match Field.get ctx.color_reps i, Field.get ctx.color_reps j with
               | Color.SUN (n1), Color.SUN (n2) \rightarrow
                     n1 = -n2 \wedge n2 > 0
                | -, - \rightarrow false)
          \mid \ E\ (i,\ j,\ k)\ \rightarrow
               distinct3 \ i \ j \ k \ \land
               (match Field.get ctx.color_reps i,
                  Field.get\ ctx.color\_reps\ j,\ Field.get\ ctx.color\_reps\ k with
               | Color.SUN (n1), Color.SUN (n2), Color.SUN (n3) \rightarrow
                     n1 = 3 \wedge n2 = 3 \wedge n3 = 3 \vee
                    n1 = -3 \land n2 = -3 \land n3 = -3
                  | -, -, - \rightarrow false)
          \mid T(a, i, j) \rightarrow
               distinct3 \ a \ i \ j \ \land
               (match Field.get ctx.color_reps a,
                  Field.get ctx.color_reps i, Field.get ctx.color_reps j with
               | Color.AdjSUN(n1), Color.SUN(n2), Color.SUN(n3) \rightarrow
                    n1 = n3 \wedge n2 = -n3 \wedge n3 > 0
               | -, -, - \rightarrow false)
          \mid F(a, b, c) \rightarrow
               distinct3 a b c \wedge
               (match Field.get ctx.color_reps a,
                  Field.get ctx.color_reps b, Field.get ctx.color_reps c with
                 Color.AdjSUN(n1), \ Color.AdjSUN(n2), \ Color.AdjSUN(n3) \rightarrow
                     n1 = n2 \wedge n2 = n3 \wedge n1 > 0
               | \ \_, \ \_, \ \_ \rightarrow \ \mathsf{false})
     let primitive\_indices = function
        \mid D (\_, \_) \rightarrow []
          E (\_, \_, \_) \rightarrow []
          T (a, -, -) \rightarrow [a]
        | F(a, b, c) \rightarrow [a; b; c]
     let indices p =
        ThoList.flatmap primitive_indices p
     let contraction\_ok p =
        List.for\_all
          (\text{fun }(n, \_) \rightarrow n = 2)
          (ThoList.classify\ (indices\ p))
     type \ tensor = int \times primitive \ list
     let map\_tensor f (factor, primitives) =
        (factor, List.map (map\_primitive f) primitives)
     let \ tensor\_ok \ context \ (\_, \ primitives) =
        List.for_all (primitive_ok context) primitives
  end
type t =
     { fields : string array;
       lorentz : Lorentz.tensor list;
        color : Color.tensor list }
module \ Test \ (M : Model.T) : Test =
  struct
     module Permutation = Permutation. Default
```

```
let context_of_flavors flavors =
  \{ arity = Array.length flavors; \}
     lorentz\_reps = Array.map M.lorentz flavors;
     color\_reps = Array.map \ M.color \ flavors \ \}
let context_of_flavor_names names =
   context\_of\_flavors\ (Array.map\ M.flavor\_of\_string\ names)
let context\_of\_vertex v =
   context_of_flavor_names v.fields
let ok \ v =
  \mathsf{let}\ context\ =\ context\_of\_vertex\ v\ \mathsf{in}
  List.for\_all\ (Lorentz.tensor\_ok\ context)\ v.lorentz\ \land
     List.for_all (Color.tensor_ok context) v.color
module PM =
   Partial.Make (struct type t = field let compare = compare end)
\mathsf{let}\ permute\ v\ p\ =
  let \ context = \ context\_of\_vertex \ v \ in
  let sorted =
     List.map
        (Field.of_int context)
        (ThoList.range\ 0\ (Array.length\ v.fields\ -\ 1)) in
     PM.apply (PM.of_lists sorted (List.map (Field.of_int context) p)) in
   \{ fields = Permutation.array (Permutation.of\_list p) v.fields; \}
     lorentz = List.map
        (Lorentz.map_tensor id permute id permute id permute) v.lorentz;
     color = List.map (Color.map\_tensor permute) v.color 
let permutations v =
   List.map\ (permute\ v)
     (Combinatorics.permute\ (ThoList.range\ 0\ (Array.length\ v.fields\ -\ 1)))
let wf_declaration flavor =
  match M.lorentz (M.flavor\_of\_string \ flavor) with
     Coupling.Vector \rightarrow "vector"
     Coupling.Spinor 
ightarrow "spinor"
     Coupling.ConjSpinor \rightarrow "conjspinor"
     \_ \rightarrow failwith "wf_declaration:_incomplete"
module Chiral = Lorentz.Dirac(Lorentz.Chiral)
let write\_fusion \ v =
  match Array.to_list v.fields with
   lhs :: rhs \rightarrow
        let name = lhs \ ^ \ "\_of\_" \ ^ \ String.concat \ "\_" \ rhs in
       \mathsf{let}\ momenta\ =\ List.map\ (\mathsf{fun}\ n\ \to\ \verb"k_"\ \hat{\ } n)\ \mathit{rhs}\ \mathsf{in}
        Printf.printf "pure_function_%s_(%s)_result_(%s)\n"
          name\ (String.concat\ ", \_"
                     (List.flatten
                        (List.map2 (fun \ wf \ p \rightarrow [wf; \ p]) \ rhs \ momenta)))
        Printf.printf "_{\sqcup\sqcup} type(%s)_{\sqcup} : _{\sqcup}%s\n" (wf\_declaration lhs) lhs;
        List.iter
          (fun wf \rightarrow
             Printf.printf "_{\sqcup\sqcup} type(%s),_{\sqcup} intent(in)_{\sqcup}:_{\sqcup} %s\n"
                (wf\_declaration \ wf) \ wf)
          rhs;
        List.iter
          (Printf.printf "_{\sqcup\sqcup} type(momentum),_{\sqcup} intent(in)_{\sqcup}:_{\sqcup}%s\n")
          momenta;
```

```
\mathsf{let}\ \mathit{rhs1}\ =\ \mathit{List.hd}\ \mathit{rhs}
             and rhs2 = List.hd (List.tl rhs) in
             begin match M.lorentz (M.flavor\_of\_string lhs) with
               Coupling.Vector \rightarrow
                  begin
                    for mu = 0 to 3 do
                       Printf.printf "_{\sqcup \sqcup} %s(%d)_{\sqcup} = " lhs mu;
                       \quad \text{for } i \ = \ 1 \ \text{to} \ 4 \ \text{do}
                          for j = 1 to 4 do
                            match Chiral.vector mu i j with
                              Lorentz.Complex.Z \rightarrow ()
                              c \rightarrow
                                  Printf.printf "_+_%s*%s(%d)*%s(%d)"
                                    (Lorentz.Complex.to_fortran c) rhs1 i rhs2 j
                          done
                       done;
                       Printf.printf "\n"
                    done
                  end:
             | Coupling.Spinor | Coupling.ConjSpinor \rightarrow
                  begin
                    for i = 1 to 4 do
                       Printf.printf "_{\sqcup \sqcup} \%s (\%d)_{\sqcup} = " lhs i;
                       for mu = 0 to 3 do
                          for j = 1 to 4 do
                            match Chiral.vector mu i j with
                              Lorentz.Complex.Z \rightarrow ()
                              c \rightarrow
                                  Printf.printf "⊔+⊔%s*%s(%d)*%s(%d)"
                                    (Lorentz.Complex.to_fortran c) rhs1 mu rhs2 j
                          done
                       done:
                       Printf.printf "\n"
                    done
                  end:
             \bot \rightarrow failwith "write_fusion:\Boxincomplete"
             Printf.printf "end_function_\%s\n" name;
             ()
       | [] \rightarrow ()
     let write\_fusions \ v =
        List.iter\ write\_fusion\ (permutations\ v)
Testing:
     let \ vector\_field \ context \ i =
       Lorentz. Vector (Lorentz. F (Field. of _int context i))
     let spinor_field context i =
        Lorentz.Spinor\ (Lorentz.F\ (Field.of\_int\ context\ i))
     let conjspinor\_field context i =
        Lorentz.ConjSpinor\ (Lorentz.F\ (Field.of\_int\ context\ i))
     let mu = Lorentz.Vector (Lorentz.I 0)
     and nu = Lorentz.Vector (Lorentz.I 1)
     let tbar_gl_t = [|"tbar"; "gl"; "t"|]
     let \ context = \ context\_of\_flavor\_names \ tbar\_gl\_t
     let \ vector\_current\_ok =
        \{ fields = tbar\_gl\_t; 
          lorentz = [(1, [Lorentz.V (vector\_field context 1,
                                               conjspinor_field context 0,
```

```
spinor_field context 2)]) ];
    color = [(1, [Color.T (Field.of\_int context 1,
                                 Field.of_int context 0,
                                 Field.of\_int\ context\ 2)])]
let\ vector\_current\_vector\_misplaced\ =
  \{ fields = tbar\_gl\_t; \}
    lorentz = [(1, [Lorentz.V (vector\_field context 2,
                                      conjspinor_field context 0,
                                      spinor_field context 2)]) ];
    color = [(1, [Color.T (Field.of\_int context 1,
                                 Field.of\_int\ context\ 0,
                                 Field.of\_int\ context\ 2)])]
{\tt let} \ vector\_current\_spinor\_misplaced \ =
  \{ fields = tbar\_gl\_t; 
    lorentz = [(1, [Lorentz.V (vector\_field context 1,
                                      conjspinor_field context 0,
                                      spinor_field context 1)]) ];
    color = [(1, [Color.T (Field.of\_int context 1,
                                 Field.of_int context 0,
                                 Field.of\_int\ context\ 2)])]
let \ vector\_current\_conjspinor\_misplaced =
  \{ fields = tbar\_gl\_t; 
    lorentz = [(1, [Lorentz.V (vector\_field context 1,
                                      conjspinor_field context 1,
                                      spinor_field context 2)]) ];
    color = [(1, [Color.T (Field.of\_int context 1,
                                 Field.of\_int\ context\ 0,
                                 Field.of\_int\ context\ 2)])]
let vector\_current\_out\_of\_bounds () =
  \{ fields = tbar\_gl\_t; \}
    lorentz = [(1, [Lorentz.V (mu,
                                      conjspinor_field context 3,
                                      spinor_field context 2)]) ];
    color = [(1, [Color.T (Field.of\_int context 1,
                                 Field.of_int context 0,
                                 Field.of\_int\ context\ 2)])]\ \}
let vector_current_color_mismatch =
  let names = [| "t"; "gl"; "t" |] in
  let context = context\_of\_flavor\_names names in
  \{ fields = names; \}
    lorentz = [(1, [Lorentz.V (mu,
                                      conjspinor_field context 0,
                                      spinor_field context 2)]) ];
    color = [(1, [Color.T (Field.of\_int context 1,
                                 Field.of\_int\ context\ 0,
                                 Field.of_int context 2)])] }
let wwzz = [| "W+"; "W-"; "Z"; "Z" |]
let context = context_of_flavor_names wwzz
let anomalous\_couplings =
  \{ fields = wwzz; 
    lorentz = [(1, [Lorentz.K (mu, Field.of\_int context 0);
                          Lorentz.K (mu, Field.of_int context 1) ]) ];
    color = []
let anomalous_couplings_index_mismatch =
  \{ fields = wwzz; 
    lorentz = [(1, [Lorentz.K (mu, Field.of\_int context 0);
                          Lorentz.K (nu, Field.of_int context 1)])];
```

```
color = [] 
    exception Inconsistent\_vertex
    let example () =
       if \neg (ok \ vector\_current\_ok) then begin
         raise\ Inconsistent\_vertex
       end:
       write\_fusions\ vector\_current\_ok
    open OUnit
    let \ vertex\_indices\_ok =
       "indices/ok" >::
         (fun () \rightarrow
            List.iter
              (fun v \rightarrow
                 assert_bool "vector_current" (ok v))
              (permutations\ vector\_current\_ok))
    let vertex_indices_broken =
       "indices/broken" >::
         (fun () \rightarrow
            assert_bool "vector_misplaced"
              (\neg (ok \ vector\_current\_vector\_misplaced));
            assert\_bool "conjugate\_spinor\_misplaced"
              (\neg (ok\ vector\_current\_spinor\_misplaced));
            assert\_bool "conjugate_spinor_misplaced"
              (\neg (ok\ vector\_current\_conjspinor\_misplaced));
            assert_raises (Field.Out_of_range 3)
              vector_current_out_of_bounds;
            assert\_bool "color\sqcupmismatch"
              (\neg (ok\ vector\_current\_color\_mismatch)))
    let anomalous\_couplings\_ok =
       "anomalous_couplings/ok" >::
         (fun () \rightarrow
            assert\_bool "anomalous_couplings"
              (ok\ anomalous\_couplings))
    let anomalous_couplings_broken =
       "anomalous_couplings/broken" >::
         (fun () \rightarrow
            assert\_bool "anomalous\_couplings"
              (\neg (ok \ anomalous\_couplings\_index\_mismatch)))
    \mathsf{let} \ \mathit{suite} \ =
       "Vertex" >:::
         [vertex_indices_ok;
          vertex\_indices\_broken;
          anomalous_couplings_ok;
          anomalous\_couplings\_broken
  end
                                        9.7 Interface of Target
module type T =
  sig
    type amplitudes
    val\ options : Options.t
    type diagnostic = All | Arguments | Momenta | Gauge
Format the amplitudes as a sequence of strings.
```

—10—

Conserved Quantum Numbers

10.1 Interface of Charges

10.1.1 Abstract Type

 $\begin{array}{rl} \mathsf{module} \ \mathsf{type} \ T \ = \\ \mathsf{sig} \end{array}$

The abstract type of the set of conserved charges or additive quantum numbers.

type t

Add the quantum numbers of a pair or a list of particles.

 $\begin{array}{l} \mathsf{val} \ add \ : \ t \ \to \ t \ \to \ t \\ \mathsf{val} \ sum \ : \ t \ list \to \ t \end{array}$

Test the charge conservation.

 $\mathsf{val}\ is_null\ :\ t\ \to\ bool$

end

10.1.2 Trivial Realisation

 $\mathsf{module}\ \mathit{Null}\ :\ \mathit{T}\ \mathsf{with}\ \mathsf{type}\ \mathit{t}\ =\ \mathit{unit}$

10.1.3 Nontrivial Realisations

 \mathbf{Z}

 $\mathsf{module}\ Z\ :\ T\ \mathsf{with}\ \mathsf{type}\ t\ =\ int$

 $\mathbf{Z}\times\mathbf{Z}\times\cdots\times\mathbf{Z}$

module ZZ: T with type $t = int \ list$

 \mathbf{Q}

 $\mathsf{module}\ Q\ :\ T\ \mathsf{with}\ \mathsf{type}\ t\ =\ Algebra.Small_Rational.t$

 $\mathbf{Q}\times\mathbf{Q}\times\cdots\times\mathbf{Q}$

 $\mathsf{module}\ QQ\ :\ T\ \mathsf{with}\ \mathsf{type}\ t\ =\ Algebra.Small_Rational.t\ list$

10.2 Implementation of Charges

```
\mathsf{module}\ \mathsf{type}\ T\ =
   sig
      type t
      \mathsf{val}\ add\ :\ t\ \to\ t\ \to\ t
      \mathsf{val}\ \mathit{sum}\ :\ t\ \mathit{list} \to\ t
      val is\_null : t \rightarrow bool
\mathsf{module}\ \mathit{Null}\ :\ \mathit{T}\ \mathsf{with}\ \mathsf{type}\ \mathit{t}\ =\ \mathit{unit}\ =
   struct
      \mathsf{type}\ t\ =\ unit
      let add () () = ()
      let sum _{-} = ()
      \mathsf{let}\ is\_null\ \_\ =\ \mathsf{true}
   end
module Z: T with type t = int =
   struct
      type t = int
      let add = (+)
      \mathsf{let} \ \mathit{sum} \ = \ \mathit{List.fold\_left} \ \mathit{add} \ 0
      let is\_null n = (n = 0)
   end
\mathsf{module}\ ZZ\ :\ T\ \mathsf{with}\ \mathsf{type}\ t\ =\ int\ list\ =
   struct
      type t = int list
      let \ add = List.map2 \ ( \ + \ )
      \mathsf{let}\ \mathit{sum}\ =\ \mathsf{function}
         | [] \rightarrow []
            [charges] \rightarrow charges
          | charges :: rest \rightarrow List.fold\_left add charges rest
      let is\_null = List.for\_all \text{ (fun } n \rightarrow n = 0)
module \ Rat = Algebra.Small\_Rational
module Q: T with type t = Rat.t =
   struct
      type t = Rat.t
      let add = Rat.add
      \mathsf{let} \ sum \ = \ List.fold\_left \ Rat.add \ Rat.null
      \mathsf{let}\ is\_null\ =\ Rat.is\_null
module QQ: T with type t = Rat.t \ list =
   struct
      \mathsf{type}\ t\ =\ Rat.t\ \mathit{list}
      let \ add = List.map2 \ Rat.add
      \mathsf{let}\ sum\ =\ \mathsf{function}
         | [] \rightarrow []
          | [charges] \rightarrow charges
         | charges :: rest \rightarrow List.fold\_left add charges rest
      let is\_null = List.for\_all Rat.is\_null
   end
```

—11— COLORIZATION

11.1 Interface of Colorize

11.1.1 ...

11.2 Implementation of Colorize

11.2.1 Auxiliary functions

Exceptions

```
let incomplete s =
  failwith ("Colorize." ^ s ^ "_not_done_yet!")
let invalid s =
   invalid\_arg ("Colorize." \hat{s} \hat{s} "\_must\_not\_be\_evaluated!")
   invalid\_arg ("Colorize." ^ s ^ "_{\sqcup}can't_{\sqcup}happen!_{\sqcup}(but_{\sqcup}just_{\sqcup}did_{\sqcup}...)")
\mathsf{let}\ mismatch\ s\ =\ 
  invalid\_arg ("Colorize." \hat{s} \hat{s} "\_mismatch\_of\_representations!")
let su\theta s =
   invalid\_arg ("Colorize." \hat{s} \hat{s} ": \Boxfound\BoxSU(0)!")
let colored\_vertex s =
   invalid_arg ("Colorize." ^ s ^ ": \( \) colored \( \) vertex!")
let baryonic\_vertex s =
  invalid\_arg ("Colorize." \hat{\ }s \hat{\ }
                       ":_baryonic_(i.e._eps_ijk)_vertices_not_supported_yet!")
{\tt let} \ color\_flow\_ambiguous \ s \ = \\
   invalid\_arg ("Colorize." \hat{s} \hat{s} ":_\_ambiguous_\_color_\_flow!")
let color\_flow\_of\_string s =
  let c = int\_of\_string s in
  \quad \text{if } c \ < \ 1 \ \text{then} \\
     invalid\_arg ("Colorize." \hat{s} \hat{s} ":\Boxcolor\Boxflow\Box#\Box<\Box1!")
  else
     c
```

Multiplying Vertices by a Constant Factor

```
module Q = Algebra.Q
module QC = Algebra.QC
let of_int n =
   QC.make \ (Q.make \ n \ 1) \ Q.null
let integer z =
  if Q.is\_null\ (QC.imag\ z) then
     let x = QC.real z in
        Some (Q.to\_integer x)
     with
     |  \rightarrow None
  else
     None
let mult\_vertex3 \ x \ v =
  let open Coupling in
  match v with
    FBF(c, fb, coup, f) \rightarrow
      FBF ((x \times c), fb, coup, f)
    PBP(c, fb, coup, f) \rightarrow
      PBP ((x \times c), fb, coup, f)
  \mid BBB \ (c, fb, coup, f) \rightarrow
      BBB ((x \times c), fb, coup, f)
  \mid GBG(c, fb, coup, f) \rightarrow
       GBG ((x \times c), fb, coup, f)
    Gauge\_Gauge\_Gauge\ c\ 	o
      Gauge\_Gauge\_Gauge\ (x \times c)
  I\_Gauge\_Gauge\_Gauge\ c\ 
ightarrow
      I\_Gauge\_Gauge\_Gauge\ (x \times c)
    Aux\_Gauge\_Gauge\ c\ 	o
      Aux\_Gauge\_Gauge\ (x \times c)
    Scalar\_Vector\_Vector c \rightarrow
      Scalar\_Vector\_Vector(x \times c)
    Aux\_Vector\_Vector\ c\ 	o
      Aux\_Vector\_Vector(x \times c)
    Aux\_Scalar\_Vector\ c\ 	o
      Aux\_Scalar\_Vector(x \times c)
    Scalar\_Scalar\_Scalar \ c \rightarrow
      Scalar\_Scalar\_Scalar\ (x \times c)
    Aux\_Scalar\_Scalar \ c \rightarrow
      Aux\_Scalar\_Scalar\ (x \times c)
     Vector\_Scalar\_Scalar \ c \ \rightarrow
       Vector\_Scalar\_Scalar\ (x \times c)
     Graviton\_Scalar\_Scalar \ c \rightarrow
      Graviton\_Scalar\_Scalar\ (x \times c)
     Graviton\_Vector\_Vector\ c\ 	o
      Graviton\_Vector\_Vector(x \times c)
     Graviton\_Spinor\_Spinor\ c\ 	o
       Graviton\_Spinor\_Spinor\ (x \times c)
    Dim4\_Vector\_Vector\_Vector\_T c \rightarrow
      Dim4\_Vector\_Vector\_Vector\_T (x \times c)
    Dim4\_Vector\_Vector\_Vector\_L \ c \rightarrow
      Dim4\_Vector\_Vector\_Vector\_L(x \times c)
  | Dim4\_Vector\_Vector\_Vector\_T5 c \rightarrow
      Dim4\_Vector\_Vector\_Vector\_T5 (x \times c)
    Dim4\_Vector\_Vector\_Vector\_L5\ c\ 
ightarrow
      Dim4\_Vector\_Vector\_Vector\_L5 (x \times c)
    Dim6\_Gauge\_Gauge\_Gauge\_c \rightarrow
      Dim6\_Gauge\_Gauge\_Gauge\ (x \times c)
```

```
Dim6\_Gauge\_Gauge\_Gauge\_5\ c\ 
ightarrow
  Dim6\_Gauge\_Gauge\_Gauge\_5 (x \times c)
Aux\_DScalar\_DScalar\ c\ 	o
  Aux\_DScalar\_DScalar\ (x \times c)
Aux\_Vector\_DScalar\ c\ 	o
  Aux\_Vector\_DScalar\ (x \times c)
Dim5\_Scalar\_Gauge2 \ c \rightarrow
  Dim5\_Scalar\_Gauge2 (x \times c)
Dim5\_Scalar\_Gauge2\_Skew \ c \rightarrow
  Dim5\_Scalar\_Gauge2\_Skew\ (x \times c)
Dim5\_Scalar\_Vector\_Vector\_T \ c \rightarrow
  Dim5\_Scalar\_Vector\_Vector\_T (x \times c)
Dim5\_Scalar\_Vector\_Vector\_U c \rightarrow
  Dim5\_Scalar\_Vector\_Vector\_U (x \times c)
Dim5\_Scalar\_Vector\_Vector\_TU \ c \rightarrow
  Dim5\_Scalar\_Vector\_Vector\_TU (x \times c)
Dim5\_Scalar\_Scalar2\ c\ 	o
  Dim5\_Scalar\_Scalar2 (x \times c)
Scalar\_Vector\_Vector\_t c \rightarrow
  Scalar\_Vector\_Vector\_t (x \times c)
Dim6\_Vector\_Vector\_Vector\_T c \rightarrow
  Dim6\_Vector\_Vector\_Vector\_T (x \times c)
Tensor_2\_Vector\_Vector\ c\ 	o
  Tensor_2 - Vector_Vector(x \times c)
Tensor_2 \_Vector_Vector_cf \ c \rightarrow
  Tensor_2 - Vector_Vector_cf(x \times c)
Tensor\_2\_Scalar\_Scalar \ c \rightarrow
  Tensor_2\_Scalar\_Scalar\ (x \times c)
Tensor\_2\_Scalar\_Scalar\_cf \ c \rightarrow
  Tensor_2\_Scalar\_Scalar\_cf (x \times c)
Tensor_2 - Vector_1 - Vector_1 c \rightarrow
  Tensor_2 - Vector_1 - Vector_1 (x \times c)
Tensor_2\_Vector\_Vector\_t \ c \rightarrow
  Tensor_2 - Vector_Vector_t (x \times c)
Dim5\_Tensor\_2\_Vector\_Vector\_1 \ c \rightarrow
  Dim5\_Tensor\_2\_Vector\_Vector\_1 (x \times c)
Dim5\_Tensor\_2\_Vector\_Vector\_2 \ c \rightarrow
  Dim5\_Tensor\_2\_Vector\_Vector\_2 (x \times c)
TensorVector\_Vector\_Vector\ c\ 	o
  TensorVector\_Vector\_Vector(x \times c)
TensorVector\_Vector\_Vector\_cf \ c \rightarrow
  TensorVector\_Vector\_Vector\_cf (x \times c)
TensorVector\_Scalar\_Scalar \ c \rightarrow
  TensorVector\_Scalar\_Scalar\ (x \times c)
TensorVector\_Scalar\_Scalar\_cf \ c \rightarrow
  TensorVector\_Scalar\_Scalar\_cf (x \times c)
TensorScalar\_Vector\_Vector c \rightarrow
  TensorScalar\_Vector\_Vector\ (x \times c)
TensorScalar\_Vector\_Vector\_cf \ c \rightarrow
  TensorScalar\_Vector\_Vector\_cf (x \times c)
TensorScalar\_Scalar\_Scalar\_c \rightarrow
  TensorScalar\_Scalar\_Scalar\ (x \times c)
TensorScalar\_Scalar\_Scalar\_cf \ c \rightarrow
  TensorScalar\_Scalar\_Scalar\_cf (x \times c)
Dim7\_Tensor\_2\_Vector\_Vector\_T c \rightarrow
  Dim7\_Tensor\_2\_Vector\_Vector\_T (x \times c)
Dim6\_Scalar\_Vector\_Vector\_D c \rightarrow
  Dim6\_Scalar\_Vector\_Vector\_D (x \times c)
Dim6\_Scalar\_Vector\_Vector\_DP \ c \rightarrow
  Dim6\_Scalar\_Vector\_Vector\_DP (x \times c)
Dim6\_HAZ\_D c \rightarrow
```

```
Dim6\_HAZ\_D (x \times c)
     Dim6\_HAZ\_DP \ c \rightarrow
       Dim6\_HAZ\_DP (x \times c)
     Gauge\_Gauge\_i \ c \rightarrow
       Gauge\_Gauge\_i (x \times c)
     Dim6\_GGG \ c \rightarrow
       Dim6\_GGG(x \times c)
     Dim6\_AWW\_DP \ c \rightarrow
       Dim6\_AWW\_DP(x \times c)
    Dim6\_AWW\_DW \ c \rightarrow
       Dim6\_AWW\_DW (x \times c)
    Dim6\_Gauge\_Gauge\_i \ c \rightarrow
       Dim6\_Gauge\_Gauge\_i\ (x \times c)
     Dim6\_HHH c \rightarrow
       Dim6\_HHH (x \times c)
     Dim6\_WWZ\_DPWDW \ c \rightarrow
       Dim6\_WWZ\_DPWDW (x \times c)
     Dim6\_WWZ\_DW c \rightarrow
       Dim6\_WWZ\_DW (x \times c)
     Dim6\_WWZ\_D c \rightarrow
       Dim6 - WWZ - D (x \times c)
let cmult_vertex3 z v =
  match integer z with
     None \rightarrow invalid\_arg "cmult_vertex3"
    Some \ x \rightarrow mult\_vertex3 \ x \ v
let mult_vertex 4 x v =
  let open Coupling in
  \mathsf{match}\ v\ \mathsf{with}
  \mid Scalar4 c \rightarrow
       Scalar4 (x \times c)
     Scalar2 - Vector2 c \rightarrow
       Scalar2 \_ Vector2 (x \times c)
     Vector 4 ic 4 list \rightarrow
       Vector4 (List.map (fun (c, icl) \rightarrow (x \times c, icl)) ic4\_list)
     DScalar 4 ic 4 \_list \rightarrow
       DScalar4 (List.map (fun (c, icl) \rightarrow (x \times c, icl)) ic4\_list)
     DScalar2\_Vector2\ ic4\_list \rightarrow
       DScalar2\_Vector2\ (List.map\ (fun\ (c,\ icl)\ 	o\ (x\ 	imes\ c,\ icl))\ ic4\_list)
     GBBG(c, fb, b2, f) \rightarrow
       GBBG ((x \times c), fb, b2, f)
     Vector4\_K\_Matrix\_tho\ (c,\ ic4\_list) \rightarrow
       Vector 4 \_K \_Matrix \_tho ((x \times c), ic4 \_list)
     Vector4\_K\_Matrix\_jr\ (c,\ ch2\_list) \rightarrow
       Vector 4 \_K \_Matrix \_jr ((x \times c), ch2 \_list)
     Vector4\_K\_Matrix\_cf\_t0 \ (c, \ ch2\_list) \rightarrow
       Vector 4 \_K \_Matrix \_cf \_t0 ((x \times c), ch2 \_list)
     Vector 4 K_Matrix_c f_t 1 (c, ch2\_list) \rightarrow
       Vector 4 \_K \_Matrix \_cf \_t1 \ ((x \times c), ch2 \_list)
     Vector4\_K\_Matrix\_cf\_t2\ (c,\ ch2\_list) \rightarrow
       Vector \angle K_Matrix_c f_t = t2 ((x \times c), ch2_list)
     Vector4\_K\_Matrix\_cf\_t\_rsi\ (c,\ ch2\_list) \rightarrow
       Vector \angle K_Matrix_c f_t rsi((x \times c), ch2\_list)
     Vector4\_K\_Matrix\_cf\_m0 \ (c, ch2\_list) \rightarrow
       Vector 4 \_K \_Matrix \_cf \_m0 \ ((x \times c), ch2 \_list)
     Vector4\_K\_Matrix\_cf\_m1 \ (c, ch2\_list) \rightarrow
       Vector \angle K_Matrix\_cf_m 1 ((x \times c), ch2\_list)
     Vector4\_K\_Matrix\_cf\_m7\ (c,\ ch2\_list) \rightarrow
       Vector 4 \_K \_Matrix \_cf \_m7 ((x \times c), ch2 \_list)
    DScalar2\_Vector2\_K\_Matrix\_ms\ (c,\ ch2\_list) \rightarrow
       DScalar2\_Vector2\_K\_Matrix\_ms\ ((x \times c),\ ch2\_list)
```

```
DScalar2\_Vector2\_m\_0\_K\_Matrix\_cf\ (c,\ ch2\_list) \rightarrow
    DScalar2\_Vector2\_m\_0\_K\_Matrix\_cf ((x \times c), ch2\_list)
  DScalar2\_Vector2\_m\_1\_K\_Matrix\_cf (c, ch2\_list) \rightarrow
    DScalar2\_Vector2\_m\_1\_K\_Matrix\_cf ((x \times c), ch2\_list)
  DScalar2\_Vector2\_m\_7\_K\_Matrix\_cf\ (c,\ ch2\_list) \rightarrow
    DScalar2\_Vector2\_m\_7\_K\_Matrix\_cf ((x × c), ch2\_list)
  DScalar 4 \_K \_Matrix \_ms (c, ch2 \_list) \rightarrow
    DScalar4\_K\_Matrix\_ms ((x \times c), ch2\_list)
  Dim8\_Scalar2\_Vector2\_1 \ c \rightarrow
    Dim8\_Scalar2\_Vector2\_1 (x \times c)
 Dim8\_Scalar2\_Vector2\_2 \ c \rightarrow
    Dim8\_Scalar2\_Vector2\_1 (x \times c)
  Dim8\_Scalar2\_Vector2\_m\_0 \ c \rightarrow
    Dim8\_Scalar2\_Vector2\_m\_0 (x \times c)
  Dim8\_Scalar2\_Vector2\_m\_1 \ c \rightarrow
    Dim8\_Scalar2\_Vector2\_m\_1 (x \times c)
  Dim8\_Scalar2\_Vector2\_m\_7 \ c \rightarrow
    Dim8\_Scalar2\_Vector2\_m\_7 (x \times c)
  Dim8\_Scalar4\ c\ 
ightarrow
    Dim8\_Scalar4 (x \times c)
  Dim8\_Vector4\_t\_0 ic4\_list \rightarrow
    Dim8 - Vector4 - t - 0 (List.map (fun (c, icl) \rightarrow (x \times c, icl)) ic4 - list)
  Dim8\_Vector4\_t\_1 ic4\_list \rightarrow
    Dim8\_Vector4\_t\_1 (List.map (fun (c, icl) \rightarrow (x \times c, icl)) ic4\_list)
  Dim8\_Vector4\_t\_2\ ic4\_list \rightarrow
    Dim8 - Vector4 - t - 2 \ (List.map \ (fun \ (c, icl) \rightarrow (x \times c, icl)) \ ic4 - list)
  Dim8\_Vector4\_m\_0\ ic4\_list \rightarrow
    Dim8\_Vector4\_m\_0 (List.map (fun (c, icl) \rightarrow (x \times c, icl)) ic4\_list)
  Dim8\_Vector4\_m\_1\ ic4\_list \rightarrow
    Dim8\_Vector4\_m\_1\ (List.map\ (\mathsf{fun}\ (c,\ icl)\ \to\ (x\ \times\ c,\ icl))\ ic4\_list)
  Dim8\_Vector4\_m\_7\ ic4\_list \rightarrow
    Dim8 - Vector4 - m - 7 (List.map (fun (c, icl) \rightarrow (x \times c, icl)) ic4 - list)
  Dim6_H_4_P_2 c \rightarrow
    Dim6_-H4_-P2 (x \times c)
  Dim6\_AHWW\_DPB\ c\ 
ightarrow
    Dim6\_AHWW\_DPB (x \times c)
  Dim6\_AHWW\_DPW c \rightarrow
    Dim6\_AHWW\_DPW (x \times c)
  Dim6\_AHWW\_DW \ c \rightarrow
    Dim6\_AHWW\_DW (x \times c)
 Dim6\_Vector4\_DW \ c \rightarrow
    Dim6\_Vector4\_DW (x \times c)
 Dim6\_Vector4\_W \ c \rightarrow
    Dim6\_Vector4\_W (x \times c)
  Dim6\_Scalar2\_Vector2\_PB \ c \rightarrow
    Dim6\_Scalar2\_Vector2\_PB (x \times c)
  Dim6\_Scalar2\_Vector2\_D \ c \rightarrow
    Dim6\_Scalar2\_Vector2\_D (x \times c)
  Dim6\_Scalar2\_Vector2\_DP \ c \rightarrow
    Dim6\_Scalar2\_Vector2\_DP (x \times c)
  Dim6\_HHZZ\_T c \rightarrow
    Dim6\_HHZZ\_T (x \times c)
  Dim6\_HWWZ\_DW \ c \rightarrow
    Dim6\_HWWZ\_DW (x \times c)
  Dim6\_HWWZ\_DPB \ c \rightarrow
    Dim6\_HWWZ\_DPB (x \times c)
| Dim6\_HWWZ\_DDPW c \rightarrow
    Dim6\_HWWZ\_DDPW (x \times c)
  Dim6\_HWWZ\_DPW \ c \rightarrow
    Dim6\_HWWZ\_DPW (x \times c)
  Dim6\_AHHZ\_D \ c \rightarrow
```

```
Dim6\_AHHZ\_D (x \times c)
     Dim6\_AHHZ\_DP \ c \rightarrow
      Dim6\_AHHZ\_DP (x \times c)
     Dim6\_AHHZ\_PB\ c\ 
ightarrow
      Dim6\_AHHZ\_PB (x \times c)
let \ cmult\_vertex4 \ z \ v =
  match integer\ z with
     None \rightarrow invalid\_arg "cmult_vertex4"
    Some \ x \rightarrow mult\_vertex 4 \ x \ v
\mathsf{let}\ \mathit{mult\_vertexn}\ x\ =\ \mathsf{function}
  | _ → incomplete "mult_vertexn"
let \ cmult\_vertexn \ z \ v =
  let open Coupling in
  \mathsf{match}\ v\ \mathsf{with}
  \mid UFO(c, v, s, fl, col) \rightarrow
       UFO(QC.mul\ z\ c,\ v,\ s,\ fl,\ col)
let mult\_vertex \ x \ v =
  let open Coupling in
  match v with
     V3 (v, fuse, c) \rightarrow V3 (mult\_vertex3 \ x \ v, fuse, c)
     V4\ (v,\ fuse,\ c)\ 	o\ V4\ (mult\_vertex4\ x\ v,\ fuse,\ c)
     Vn (v, fuse, c) \rightarrow Vn (mult\_vertexn x v, fuse, c)
let \ cmult\_vertex \ z \ v \ =
  let open Coupling in
  \mathsf{match}\ v\ \mathsf{with}
     V3\ (v, fuse, c) \rightarrow V3\ (cmult\_vertex3\ z\ v, fuse, c)
     V4\ (v,\ fuse,\ c)\ 	o\ V4\ (cmult\_vertex4\ z\ v,\ fuse,\ c)
    Vn (v, fuse, c) \rightarrow Vn (cmult\_vertexn z v, fuse, c)
                                               Flavors Adorned with Colorflows
module\ Flavor\ (M\ :\ Model.\ T)\ =
  struct
     type cf_in = int
     type cf\_out = int
     type t =
           White of M.flavor
          CF\_in 	ext{ of } M.flavor 	imes cf\_in
          CF\_out of M.flavor \times cf\_out
          CF\_io\ of\ M.flavor\ 	imes\ cf\_in\ 	imes\ cf\_out
          CF\_aux of M.flavor
     let flavor\_sans\_color = function
          White f \rightarrow f
          CF_{-}in (f, \_) \rightarrow f
          CF\_out\ (f,\ \_)\ 	o\ f
          CF_{-}io\ (f, -, -) \rightarrow f
          CF\_aux\ f \rightarrow f
     let pullback f arg1 =
        f (flavor\_sans\_color arg1)
```

11.2.3 The Legacy Implementation

We have to keep this legacy implementation around, because it infers the color flows from the SU(3) representations of a particle in vertices with three and four legs (except for four triplets, where the connections

are ambiguous). The new implementation is already used for UFO models exclusively, since they don't use Coupling. V2 and Coupling. V3 at all.

```
\begin{array}{lll} \operatorname{module}\ Legacy\_Implementation\ (M\ :\ Model.T) \ = \\ \operatorname{struct} \\ & \operatorname{module}\ C \ = \ Color \\ & \operatorname{module}\ Colored\_Flavor \ = \ Flavor(M) \\ \operatorname{open}\ Colored\_Flavor \\ & \operatorname{open}\ Coupling \\ & \operatorname{let}\ nc \ = \ M.nc \end{array}
```

Auxiliary functions

Below, we will need to permute Lorentz structures. The following permutes the three possible contractions of four vectors. We permute the first three indices, as they correspond to the particles entering the fusion.

```
{\sf type} \ permutation {\it 4} \ = \\
      P123 | P231 | P312
    | P213 | P321 | P132
let permute\_contract4 = function
   \mid P123 \rightarrow
          begin function
                C_{-}12_{-}34 \rightarrow C_{-}12_{-}34
                C_{-}13_{-}42 \rightarrow C_{-}13_{-}42
                C_{-}14_{-}23 \rightarrow C_{-}14_{-}23
          end
   \mid P231 \rightarrow
          begin function
                C_{-}12_{-}34 \rightarrow C_{-}14_{-}23
                C_{-}13_{-}42 \rightarrow C_{-}12_{-}34
                C_{-}14_{-}23 \rightarrow C_{-}13_{-}42
          end
   \mid P312 \rightarrow
          begin function
                C_{-}12_{-}34 \rightarrow C_{-}13_{-}42
                C_{-}13_{-}42 \rightarrow C_{-}14_{-}23
                C_{-}14_{-}23 \rightarrow C_{-}12_{-}34
          end
   P213 \rightarrow
          begin function
                C_{-}12_{-}34 \rightarrow C_{-}12_{-}34
                C_{-}13_{-}42 \rightarrow C_{-}14_{-}23
                C_{-}14_{-}23 \rightarrow C_{-}13_{-}42
          end
   \mid P321 \rightarrow
          begin function
                C\_12\_34 \rightarrow C\_14\_23
                C_{-}13_{-}42 \rightarrow C_{-}13_{-}42
                C_{-}14_{-}23 \rightarrow C_{-}12_{-}34
          end
   \mid P132 \rightarrow
          begin function
                C_{-}12_{-}34 \rightarrow C_{-}13_{-}42
                C_{-}13_{-}42 \rightarrow C_{-}12_{-}34
                C_{-}14_{-}23 \rightarrow C_{-}14_{-}23
          end
{\tt let} \ permute\_contract \textit{4\_list} \ perm \ ic \textit{4\_list} \ =
   List.map (fun (i, c4) \rightarrow (i, permute\_contract4 perm c4)) <math>ic4\_list
```

```
let permute\_vertex4' perm = function
   Scalar4 c \rightarrow
       Scalar4 c
     Vector 4 ic 4 list \rightarrow
        Vector4 (permute_contract4_list perm ic4_list)
     Vector4\_K\_Matrix\_jr\ (c,\ ic4\_list) \rightarrow
        Vector \c 4\_K\_Matrix\_jr\ (c,\ permute\_contract \c 4\_list\ perm\ ic \c 4\_list)
     Vector4\_K\_Matrix\_cf\_t0\ (c,\ ic4\_list) \rightarrow
        Vector \angle A = Matrix = cf = t0 (c, permute = contract \angle A = list)
     Vector 4 \_K \_Matrix \_cf \_t1 \ (c, ic4 \_list) \rightarrow
        Vector4\_K\_Matrix\_cf\_t1 (c, permute\_contract4\_list perm ic4\_list)
     Vector 4 K_Matrix_c f_t 2 (c, ic4_list) \rightarrow
        Vector4\_K\_Matrix\_cf\_t2 (c, permute\_contract4\_list perm ic4\_list)
    Vector4\_K\_Matrix\_cf\_t\_rsi\ (c,\ ic4\_list) \rightarrow
        Vector4\_K\_Matrix\_cf\_t\_rsi\ (c,\ permute\_contract4\_list\ perm\ ic4\_list)
     Vector 4 \_K \_Matrix \_cf \_m0 \ (c, ic4 \_list) \rightarrow
        Vector 4\_K\_Matrix\_cf\_m0 (c, permute\_contract 4\_list perm\ ic 4\_list)
     Vector4\_K\_Matrix\_cf\_m1 (c, ic4\_list) \rightarrow
        Vector4\_K\_Matrix\_cf\_m1 (c, permute\_contract4\_list perm ic4\_list)
     Vector4\_K\_Matrix\_cf\_m7 (c, ic4\_list) \rightarrow
        Vector 4 \_K \_Matrix \_cf \_m7 (c, permute \_contract 4 \_list perm ic 4 \_list)
    DScalar2\_Vector2\_K\_Matrix\_ms\ (c,\ ic4\_list) \rightarrow
        DScalar2\_Vector2\_K\_Matrix\_ms (c, permute\_contract4\_list perm ic4\_list)
    DScalar2\_Vector2\_m\_0\_K\_Matrix\_cf\ (c,\ ic4\_list) \rightarrow
        DScalar2\_Vector2\_m\_0\_K\_Matrix\_cf (c, permute\_contract4\_list perm ic4_list)
   DScalar2\_Vector2\_m\_1\_K\_Matrix\_cf\ (c,\ ic4\_list) \rightarrow
        DScalar2\_Vector2\_m\_1\_K\_Matrix\_cf (c, permute_contract4_list perm ic4_list)
  DScalar2\_Vector2\_m\_7\_K\_Matrix\_cf\ (c,\ ic4\_list) \rightarrow
        DScalar2\_Vector2\_m\_7\_K\_Matrix\_cf (c, permute\_contract4\_list perm ic4\_list)
  | DScalar_4\_K\_Matrix\_ms (c, ic_4\_list) \rightarrow
        DScalar4\_K\_Matrix\_ms (c, permute\_contract4\_list perm ic4\_list)
    Scalar2\_Vector2 \ c \rightarrow
        incomplete "permute_vertex4'uScalar2_Vector2"
    DScalar 4\ ic 4\_list \rightarrow
        incomplete "permute_vertex4'_{\sqcup}DScalar4"
    DScalar2\_Vector2\ ic4\_list \rightarrow
        incomplete "permute_vertex4'\u00cdDScalar2_Vector2"
    GBBG(c, fb, b2, f) \rightarrow
        incomplete "permute_vertex4'uGBBG"
     Vector 4 \_K \_Matrix \_tho (c, ch2 \_list) \rightarrow
        incomplete "permute_vertex4'\_Vector4_K_Matrix_tho"
  | Dim8\_Scalar2\_Vector2\_1 ic4\_list \rightarrow
        incomplete "permute_vertex4'\_Dim8_Scalar2_Vector2_1"
  Dim8\_Scalar2\_Vector2\_2 ic4\_list \rightarrow
        incomplete "permute_vertex4'\sqcupDim8_Scalar2_Vector2_2"
  | Dim8\_Scalar2\_Vector2\_m\_0 \ ic4\_list \rightarrow
        incomplete "permute_vertex4'_{\sqcup}Dim8_{	extsf{S}}Scalar2_{	extsf{V}}Vector2_{	extsf{m}}_{	extsf{O}}"
    Dim8\_Scalar2\_Vector2\_m\_1\ ic4\_list \rightarrow
        incomplete "permute_vertex4'\sqcupDim8_Scalar2_Vector2\_m\_1"
    Dim8\_Scalar2\_Vector2\_m\_7\ ic4\_list \rightarrow
        incomplete "permute_vertex4'\sqcupDim8_Scalar2_Vector2\_m\_7"
    Dim8\_Scalar4\ ic4\_list \rightarrow
        incomplete "permute_vertex4'

Dim8_Scalar4"
    Dim8\_Vector4\_t\_0 ic4\_list \rightarrow
        incomplete "permute_vertex4'∟Dim8_Vector4_t_0"
    Dim8\_Vector4\_t\_1 ic4\_list \rightarrow
        incomplete "permute_vertex4'\_Dim8_Vector4_t_1"
  Dim8\_Vector4\_t\_2\ ic4\_list \rightarrow
        incomplete "permute_vertex4'∟Dim8_Vector4_t_2"
   Dim8\_Vector4\_m\_0\ ic4\_list \rightarrow
        incomplete "permute_vertex4'_{\square}Dim8_Vector4_{\underline{m}}0"
```

```
Dim8\_Vector4\_m\_1 ic4\_list \rightarrow
       incomplete "permute_vertex4',Dim8_Vector4_m_1"
    Dim8\_Vector4\_m\_7\ ic4\_list\ 	o
       incomplete "permute_vertex4'∟Dim8_Vector4_m_7"
    Dim6\_H4\_P2\ ic4\_list \rightarrow
       incomplete "permute_vertex4'_{\sqcup}Dim6_H4_P2"
    Dim6\_AHWW\_DPB\ ic4\_list\ 	o
       incomplete "permute_vertex4'\_Dim6_AHWW_DPB"
    Dim6\_AHWW\_DPW ic4\_list \rightarrow
       incomplete "permute_vertex4'\_Dim6_AHWW_DPW"
  | Dim6\_AHWW\_DW ic4\_list \rightarrow
       incomplete "permute_vertex4' ∟Dim6_AHWW_DW"
  Dim6\_Vector4\_DW\ ic4\_list \rightarrow
       incomplete "permute_vertex4'\sqcupDim6_Vector4_DW"
  | Dim6\_Vector4\_W ic4\_list \rightarrow
       incomplete "permute_vertex4'\sqcupDim6_Vector4\_W"
    Dim6\_Scalar2\_Vector2\_D ic4\_list \rightarrow
       incomplete "permute_vertex4'\_Dim6_Scalar2_Vector2_D"
    Dim6\_Scalar2\_Vector2\_DP\ ic4\_list\ 	o
       incomplete "permute_vertex4'\sqcupDim6\_Scalar2\_Vector2\_DP"
    Dim6\_Scalar2\_Vector2\_PB\ ic4\_list\ 
ightarrow
       incomplete "permute_vertex4'\sqcupDim6_Scalar2_Vector2_PB"
    Dim6\_HHZZ\_T ic4\_list \rightarrow
       incomplete "permute_vertex4'_{\sqcup}Dim6_HHZZ_T"
    Dim6\_HWWZ\_DW ic4\_list \rightarrow
       incomplete "permute_vertex4' ∟Dim6_HWWZ_DW"
  | Dim6\_HWWZ\_DPB ic4\_list \rightarrow
       incomplete "permute_vertex4'\sqcupDim6_HWWZ_DPB"
  | Dim6\_HWWZ\_DDPW ic4\_list \rightarrow
       incomplete "permute_vertex4'\_Dim6_HWWZ_DDPW"
  | Dim6\_HWWZ\_DPW ic4\_list \rightarrow
       incomplete "permute_vertex4'_Dim6_HWWZ_DPW"
    Dim6\_AHHZ\_D ic4\_list \rightarrow
       incomplete "permute_vertex4'_Dim6_AHHZ_D"
    Dim6\_AHHZ\_DP\ ic4\_list\ 	o
       incomplete "permute_vertex4'_{lue}Dim6_AHHZ_DP"
    Dim6\_AHHZ\_PB\ ic4\_list\ 
ightarrow
       incomplete "permute_vertex4'uDim6_AHHZ_PB"
let permute\_vertex4 \ perm = function
    V3 (v, fuse, c) \rightarrow V3 (v, fuse, c)
    V4\ (v, fuse, c) \rightarrow V4\ (permute\_vertex4'\ perm\ v, fuse, c)
    Vn (v, fuse, c) \rightarrow Vn (v, fuse, c)
```

Cubic Vertices

\$

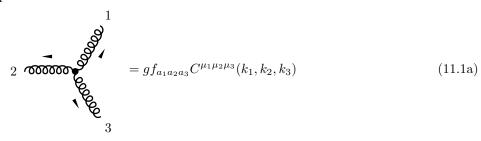
The following pattern matches could eventually become quite long. The O'Caml compiler will (hopefully) optimize them aggressively (http://pauillac.inria.fr/~maranget/papers/opat/).

```
\begin{split} & \text{let } colorize\_fusion2 \; f1 \; f2 \; (f, \; v) \; = \\ & \text{match } M.color \; f \; \text{with} \\ & | \; C.Singlet \; \rightarrow \\ & \; \text{begin match } f1, \; f2 \; \text{with} \\ & | \; White \; \_, \; White \; \_ \; \rightarrow \\ & \; [White \; f, \; v] \\ & | \; CF\_in \; (\_, \; c1), \; CF\_out \; (\_, \; c2') \; \rightarrow \\ & \; \text{if } \; c1 \; = \; c2' \; \text{then} \\ & \; [White \; f, \; v] \end{split}
```

```
else
           |CF_{-io}(f1, c1, c1'), CF_{-io}(f2, c2, c2')| \rightarrow
         if c1 = c2' \wedge c2 = c1' then
           [White f, v]
         else
           | CF_aux f1, CF_aux f2 \rightarrow
         [White f, mult\_vertex (- (nc ())) v]
    \mid CF\_aux\_, CF\_io\_ \mid CF\_io\_, CF\_aux\_ \rightarrow
         (CF\_in\_ \mid CF\_out\_ \mid CF\_io\_ \mid CF\_aux\_), White\_
      White \_, (CF\_in \_ \mid CF\_out \_ \mid CF\_io \_ \mid CF\_aux \_)
     (CF\_io\_ \mid CF\_aux\_), (CF\_in\_ \mid CF\_out\_)
    | (CF_in - | CF_out -), (CF_io - | CF_aux -) |
      CF_{-in} _, CF_{-in} _ | CF_{-out} _, CF_{-out} _ \rightarrow
         colored_vertex "colorize_fusion2"
    end
\mid C.SUN \ nc1 \rightarrow
    begin match f1, f2 with
    | CF_in(_-, c1), (White_- | CF_aux_-)
    | (White \_ | CF\_aux \_), CF\_in (\_, c1) \rightarrow
         if nc1 > 0 then
           [CF\_in\ (f,\ c1),\ v]
         else
            colored_vertex "colorize_fusion2"
    CF_{out}(\_, c1'), (White \_ | CF_{aux} \_)
    | (White \_ | CF\_aux \_), CF\_out (\_, c1') \rightarrow
         if nc1 < 0 then
           [CF\_out\ (f,\ c1'),\ v]
         else
            colored_vertex "colorize_fusion2"
    | CF_{-}in (-, c1), CF_{-}io (-, c2, c2') |
    CF_{-io}(-, c2, c2'), CF_{-in}(-, c1) \rightarrow
         if nc1 > 0 then begin
           if c1 = c2' then
              [CF\_in (f, c2), v]
           else
         end else
           colored_vertex "colorize_fusion2"
      CF_{-out}(-, c1'), CF_{-io}(-, c2, c2')
      CF\_io(\_, c2, c2'), CF\_out(\_, c1') \rightarrow
         if nc1 < 0 then begin
           if c1' = c2 then
              [CF\_out\ (f,\ c2'),\ v]
           else
         end else
           colored_vertex "colorize_fusion2"
    | CF_in_{-}, CF_in_{-} \rightarrow
         if nc1 > 0 then
            baryonic\_vertex "colorize_fusion2"
         else
            colored_vertex "colorize_fusion2"
```

```
| CF\_out\_, CF\_out\_ \rightarrow
          if nc1 < 0 then
             baryonic_vertex "colorize_fusion2"
           else
             colored_vertex "colorize_fusion2"
       CF_{-in} -, CF_{-out} - | CF_{-out} -, CF_{-in} -
     | \ (\mathit{White} \ \_ \ | \ \mathit{CF\_io} \ \_ \ | \ \mathit{CF\_aux} \ \_),
             (White \_ | CF\_io \_ | CF\_aux \_) \rightarrow
           colored_vertex "colorize_fusion2"
     end
\mid C.AdjSUN \_ \rightarrow
     begin match f1, f2 with
     | White \_, CF\_io (\_, c1, c2') | CF\_io (\_, c1, c2'), White \_ \rightarrow
          [CF\_io\ (f,\ c1,\ c2'),\ v]
       White \_, CF_{aux} \_ | CF_{aux} \_, White \_ \rightarrow
          [CF\_aux\ f,\ mult\_vertex\ (-\ (nc\ ()))\ v]
     CF_in(-, c1), CF_out(-, c2')
     | CF\_out(\_, c2'), CF\_in(\_, c1) \rightarrow
          if c1 \neq c2' then
            [CF\_io\ (f,\ c1,\ c2'),\ v]
             [CF\_aux\ f,\ v]
```

In the adjoint representation



with

$$C^{\mu_1\mu_2\mu_3}(k_1, k_2, k_3) = (g^{\mu_1\mu_2}(k_1^{\mu_3} - k_2^{\mu_3}) + g^{\mu_2\mu_3}(k_2^{\mu_1} - k_3^{\mu_1}) + g^{\mu_3\mu_1}(k_3^{\mu_2} - k_1^{\mu_2})) \quad (11.1b)$$

while in the color flow basis find from

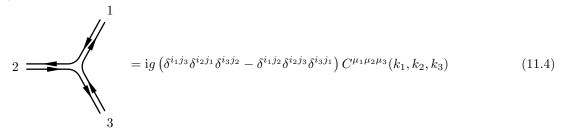
$$if_{a_1 a_2 a_3} = tr(T_{a_1}[T_{a_2}, T_{a_3}]) = tr(T_{a_1} T_{a_2} T_{a_3}) - tr(T_{a_1} T_{a_3} T_{a_2})$$
 (11.2)

the decomposition

$$if_{a_1 a_2 a_3} T_{a_1}^{i_1 j_1} T_{a_2}^{i_2 j_2} T_{a_3}^{i_3 j_3} = \delta^{i_1 j_2} \delta^{i_2 j_3} \delta^{i_3 j_1} - \delta^{i_1 j_3} \delta^{i_3 j_2} \delta^{i_2 j_1}.$$

$$(11.3)$$

The resulting Feynman rule is



\$

We have to generalize this for cases of three particles in the adjoint that are not all gluons (gluinos, scalar octets):

- scalar-scalar-scalar
- scalar-scalar-vector
- scalar-vector-vector

- scalar-fermion-fermion
- vector-fermion-fermion



We could use a better understanding of the signs for the gaugino-gaugino-gaugeboson couplings!!!

```
CF_{-io} (f1, c1, c1'), CF_{-io} (f2, c2, c2') \rightarrow
     let phase =
        begin match v with
          V3 (Gauge\_Gauge\_Gauge\_, \_, \_)
          V3 (I\_Gauge\_Gauge\_Gauge\_, \_, \_)
          V3\ (Aux\_Gauge\_Gauge\_, \ \_, \ \_)\ \rightarrow\ of\_int\ 1
         V3 \ (FBF \ (\_, \_, \_, \_), \ fuse2, \_) \rightarrow
             begin match fuse2 with
               F12 \rightarrow of_int \ 1 \ (* works, needs underpinning *)
               F21 \rightarrow of_{-}int (-1) (* dto. *)
               F31 \rightarrow of_{-}int \ 1 \ (* dto. *)
               F32 \rightarrow of_int(-1) (* transposition of F12 *)
               F23 \rightarrow of_int \ 1 \ (* transposition of F21 *)
              F13 \rightarrow of_int (-1) (* transposition of F12 *)
             end
          V3 \rightarrow incomplete "colorize_fusion2\sqcup(V3\sqcup-)"
        V4 \rightarrow impossible "colorize_fusion2\sqcup(V4\sqcup-)"
        |Vn_{-} \rightarrow impossible "colorize_fusion2_{\sqcup}(Vn_{\sqcup})"
       end in
     if c1' = c2 then
       [CF\_io\ (f,\ c1,\ c2'),\ cmult\_vertex\ (QC.neg\ phase)\ v]
     else if c2' = c1 then
       [CF\_io\ (f,\ c2,\ c1'),\ cmult\_vertex\ (\ phase)\ v]
     else
       CF\_aux\_, CF\_io\_
  CF\_io\_ , CF\_aux\_
  CF\_aux\_, CF\_aux\_ \rightarrow
     White _, White _
  (White \_ \mid CF\_io \_ \mid CF\_aux \_), (CF\_in \_ \mid CF\_out \_)
  (CF_in_i \mid CF_out_i), (White_i \mid CF_io_i \mid CF_oux_i)
  CF\_in\_, CF\_in\_ \mid CF\_out\_, CF\_out\_ \rightarrow
     colored_vertex "colorize_fusion2"
end
```

Quartic Vertices

```
 \begin{array}{l} \text{let } colorize\_fusion3 \; f1 \; f2 \; f3 \; (f, \; v) \; = \\ \text{match } M.color \; f \; \text{ with} \\ \\ | \; C.Singlet \; \to \\ \text{begin match } f1, \; f2, \; f3 \; \text{ with} \\ \\ | \; White \; _-, \; White \; _-, \; White \; _- \to \\ [White \; f, \; v] \\ \\ | \; (White \; _- \mid \; CF\_aux \; _-), \; CF\_in \; (_-, \; c1), \; CF\_out \; (_-, \; c2') \\ | \; (White \; _- \mid \; CF\_aux \; _-), \; CF\_out \; (_-, \; c1), \; CF\_in \; (_-, \; c2') \\ | \; CF\_in \; (_-, \; c1), \; (White \; _- \mid \; CF\_aux \; _-), \; CF\_out \; (_-, \; c2') \\ | \; CF\_out \; (_-, \; c1), \; (White \; _- \mid \; CF\_aux \; _-), \; CF\_in \; (_-, \; c2') \\ | \; CF\_out \; (_-, \; c1), \; CF\_out \; (_-, \; c2'), \; (White \; _- \mid \; CF\_aux \; _-) \\ | \; CF\_out \; (_-, \; c1), \; CF\_in \; (_-, \; c2'), \; (White \; _- \mid \; CF\_aux \; _-) \\ | \; CF\_out \; (_-, \; c1), \; CF\_in \; (_-, \; c2'), \; (White \; _- \mid \; CF\_aux \; _-) \\ | \; if \; c1 \; = \; c2' \; \text{then} \\ [White \; f, \; v] \end{array}
```

```
else
      White \_, CF_{-io}(\_, c1, c1'), CF_{-io}(\_, c2, c2')
  CF_{-io} (_, c1, c1'), White_{-}, CF_{-io} (_, c2, c2')
CF_{-io}(-, c1, c1'), CF_{-io}(-, c2, c2'), White \rightarrow
    if c1 = c2' \wedge c2 = c1' then
      [White f, v]
    else
      White _, CF_aux _, CF_aux _
  CF_aux _, White _, CF_aux _
 CF_{-}aux_{-}, CF_{-}aux_{-}, White_{-} \rightarrow
    [White f, mult\_vertex (-(nc())) v]
  White \_, CF_io_j, CF_aux_j
  White \_, CF\_aux <math>\_, CF\_io <math>\_
  CF_io_, White_, CF_aux_
  CF_aux _, White _, CF_io _
  CF_{-io} _, CF_{-aux} _, White _
  CF\_aux\_, CF\_io\_, White\_\rightarrow
  CF_io (-, c1, c1'), CF_in (-, c2), CF_out (-, c3')
  CF_{-io}(-, c1, c1'), CF_{-out}(-, c3'), CF_{-in}(-, c2)
  CF_{-in}(-, c2), CF_{-io}(-, c1, c1'), CF_{-out}(-, c3')
  CF_{-}out(-, c3'), CF_{-}io(-, c1, c1'), CF_{-}in(-, c2)
  CF_{-in}(-, c2), CF_{-out}(-, c3'), CF_{-io}(-, c1, c1')
  CF\_out(\_, c3'), CF\_in(\_, c2), CF\_io(\_, c1, c1') \rightarrow
    if c1 = c3' \wedge c1' = c2 then
       [White f, v]
    else
      | CF_io (_, c1, c1'), CF_io (_, c2, c2'), CF_io (_, c3, c3') \rightarrow
    if c1' = c2 \wedge c2' = c3 \wedge c3' = c1 then
      [White f, mult\_vertex (-1) v]
    else if c1' = c3 \wedge c2' = c1 \wedge c3' = c2 then
      [White f, mult\_vertex (1) v]
    else
      CF\_io\_, CF\_io\_, CF\_aux\_
  CF\_io\_, CF\_aux\_, CF\_io\_
  CF\_aux\_, CF\_io\_, CF\_io\_
  CF\_io\_, CF\_aux\_, CF\_aux\_
  CF\_aux\_, CF\_io\_, CF\_aux\_
  CF_{-}aux_{-}, CF_{-}aux_{-}, CF_{-}io_{-}
  CF\_aux\_, CF\_aux\_, CF\_aux\_ 	o
  CF_{-}in_{-}, CF_{-}in_{-}, CF_{-}in_{-}
 CF\_out\_, CF\_out\_, CF\_out\_ \rightarrow
    baryonic_vertex "colorize_fusion3"
  CF_{-in} _, CF_{-in} _, CF_{-out} _
  CF_{-in} _, CF_{-out} _, CF_{-in} _
  CF\_out\_, CF\_in\_, CF\_in\_
  CF_in _, CF_out _, CF_out _
  CF\_out\_, CF\_in\_, CF\_out\_
  CF\_out\_, CF\_out\_, CF\_in\_
  White \_, White \_, (CF\_io\_ | CF\_aux\_)
  White \_, (CF\_io\_ | CF\_aux\_), White \_
  (CF\_io\_ \mid CF\_aux\_), White\_, White\_
```

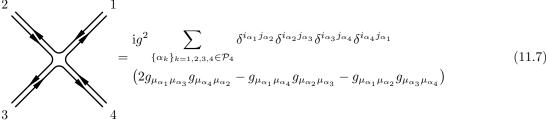
```
(White \_ | CF\_io \_ | CF\_aux \_), CF\_in \_, CF\_in \_
       CF_{-in} , (White _{-} | CF_{-io} _{-} | CF_{-aux} _{-}), CF_{-in} _{-}
      CF_{-in} -, CF_{-in} -, (White_{-} \mid CF_{-io} - \mid CF_{-aux} -)
       (White _ | CF_io _ | CF_aux _), CF_out _, CF_out _
       CF\_out\_, (White \_ | CF\_io\_ | CF\_aux\_), CF\_out\_
     | CF\_out\_, CF\_out\_, (White\_| CF\_io\_| CF\_aux\_)
     | (CF_in_i | CF_out_i),
         (White \_ | CF\_io \_ | CF\_aux \_),
          (White \_ | CF\_io \_ | CF\_aux \_)
    | (White \_ | CF\_io \_ | CF\_aux \_),
         (CF_in_i | CF_out_i),
          (White \_ | CF\_io \_ | CF\_aux \_)
    | \ (\mathit{White} \ \_ \ | \ \mathit{CF\_io} \ \_ \ | \ \mathit{CF\_aux} \ \_),
         (White \_ | CF\_io \_ | CF\_aux \_),
          (CF_in_{-} \mid CF_out_{-}) \rightarrow
            colored_vertex "colorize_fusion3"
    end
\mid C.SUN \ nc1 \rightarrow
    begin match f1, f2, f3 with
       CF_{-in}(-, c1), CF_{-io}(-, c2, c2'), CF_{-io}(-, c3, c3')
       CF\_io(\_, c2, c2'), CF\_in(\_, c1), CF\_io(\_, c3, c3')
       CF\_io (_, c2, c2'), CF\_io (_, c3, c3'), CF\_in (_, c1) \rightarrow
         if nc1 > 0 then
            if c1 = c2' \wedge c2 = c3' then
              [CF_in(f, c3), v]
            else if c1 = c3' \wedge c3 = c2' then
              [CF\_in (f, c2), v]
            else
         else
            colored_vertex "colorize_fusion3"
       CF_{-}out(-, c1'), CF_{-}io(-, c2, c2'), CF_{-}io(-, c3, c3')
       CF_{-io}(-, c2, c2'), CF_{-out}(-, c1'), CF_{-io}(-, c3, c3')
      CF\_io (_, c2, c2'), CF\_io (_, c3, c3'), CF\_out (_, c1') \rightarrow
         \quad \text{if } nc1 \ < \ 0 \ \text{then} \\
            if c1' = c2 \wedge c2' = c3 then
              [CF\_out\ (f,\ c3'),\ v]
            else if c1' = c3 \wedge c3' = c2 then
              [CF\_out\ (f,\ c2'),\ v]
            else
         else
            colored\_vertex "colorize_fusion3"
       CF_{-}aux_{-}, CF_{-}in(_{-}, c1), CF_{-}io(_{-}, c2, c2')
       CF_{-}aux_{-}, CF_{-}io_{-}(c2, c2'), CF_{-}in_{-}(c1)
       CF_{-in}(-, c1), CF_{-aux}_{-}, CF_{-io}(-, c2, c2')
       CF_{-io}(-, c2, c2'), CF_{-aux}_{-}, CF_{-in}(-, c1)
       CF_{in}(-, c1), CF_{io}(-, c2, c2'), CF_{aux}
      CF\_io(\_, c2, c2'), CF\_in(\_, c1), CF\_aux\_ \rightarrow
         if nc1 > 0 then
            if c1 = c2' then
              [CF\_in\ (f,\ c2),\ mult\_vertex\ (2)\ v]
            else
         else
            colored_vertex "colorize_fusion3"
    | CF_aux _, CF_out (_, c1'), CF_io (_, c2, c2')
```

```
CF_{-}aux_{-}, CF_{-}io(_{-}, c2, c2'), CF_{-}out(_{-}, c1')
  CF_{-out} (_, c1'), CF_{-aux} _, CF_{-io} (_, c2, c2')
  CF_{-io}(-, c2, c2'), CF_{-aux}_{-}, CF_{-out}(-, c1')
  CF_out (_, c1'), CF_io (_, c2, c2'), CF_aux _
  CF\_io (_, c2, c2'), CF\_out (_, c1'), CF\_aux _ \rightarrow
    if nc1 < 0 then
       if c1' = c2 then
         [CF\_out\ (f,\ c2'),\ mult\_vertex\ (\ 2)\ v]
       else
    else
       colored_vertex "colorize_fusion3"
  White _, CF_in (_, c1), CF_io (_, c2, c2')
  White \_, CF_io(\_, c2, c2'), CF_in(\_, c1)
  CF_{-in}(-, c1), White -, CF_{-io}(-, c2, c2')
  CF_{-io} (_, c2, c2'), White_{-}, CF_{-in} (_, c1)
  CF_{-in} (-, c1), CF_{-io} (-, c2, c2'), White -
  CF_{-io} (_, c2, c2'), CF_{-in} (_, c1), White_{-} \rightarrow
    if nc1 > 0 then
       if c1 = c2' then
         [CF\_in (f, c2), v]
    else
       colored_vertex "colorize_fusion3"
  White \_, CF\_out (\_, c1'), CF\_io (\_, c2, c2')
  White \_, CF_{-io} (\_, c2, c2'), CF_{-out} (\_, c1')
  CF\_out(\_, c1'), White\_, CF\_io(\_, c2, c2')
  CF_{-io} (_, c2, c2'), White_{-}, CF_{-out} (_, c1')
  CF_{-out} (_, c1'), CF_{-io} (_, c2, c2'), White_{-}
 CF_{-io} (_, c2, c2'), CF_{-out} (_, c1'), White_{-} \rightarrow
    \quad \text{if } nc1 \ < \ 0 \ \text{then} \\
       if c2 = c1' then
         [CF\_out\ (f,\ c2'),\ v]
       else
    else
       colored_vertex "colorize_fusion3"
  CF_{-in} (_, c1), CF_{-aux} _, CF_{-aux} _
  CF_{aux} -, CF_{in} (-, c1), CF_{aux} -
|CF\_aux\_, CF\_aux\_, CF\_in(\_, c1)| \rightarrow
    if nc1 > 0 then
       [CF_in(f, c1), mult_vertex(2)v]
    else
       colored_vertex "colorize_fusion3"
  CF_{-in} (_, c1), CF_{-aux} _, White _
  CF_{-in}(-, c1), White -, CF_{-aux}
  CF\_in (\_, c1), White \_, White \_
  CF\_aux\_, CF\_in(\_, c1), White\_
  White \_, CF_in(\_, c1), CF_aux
  White \_, CF_in(\_, c1), White \_
  CF_{aux}, White_{a}, CF_{in} (a, c1)
  White \_, CF\_aux <math>\_, CF\_in(\_, c1)
  White \_, White \_, CF_in(\_, c1) \rightarrow
    if nc1 > 0 then
       [CF_in(f, c1), v]
    else
       colored_vertex "colorize_fusion3"
CF_{-}out(-, c1'), CF_{-}aux_{-}, CF_{-}aux_{-}
```

```
CF_{aux}, CF_{out} (_, c1'), CF_{aux}
      CF\_aux\_, CF\_aux\_, CF\_out(\_, c1') \rightarrow
         if nc1 < 0 then
           [CF\_out\ (f,\ c1'),\ mult\_vertex\ (\ 2)\ v]
         else
           colored_vertex "colorize_fusion3"
      CF\_out(\_, c1'), CF\_aux\_, White\_
      CF\_out(\_, c1'), White\_, CF\_aux\_
      CF_{-}out(-, c1'), White_{-}, White_{-}
      CF_{aux}, CF_{out} (_, c1'), White
      White \_, CF\_out (\_, c1'), CF\_aux \_
       White \_, CF\_out(\_, c1'), White \_
      CF_{aux}, White_{-}, CF_{out} (-, c1')
       White \_, CF\_aux <math>\_, CF\_out (\_, c1')
      White \_, White \_, CF\_out(\_, c1') \rightarrow
         if nc1 < 0 then
           [CF\_out\ (f,\ c1'),\ v]
         else
           colored_vertex "colorize_fusion3"
      CF_{-in} -, CF_{-in} -, CF_{-out} -
      CF_{-in} _, CF_{-out} _, CF_{-in} _
      CF\_out\_, CF\_in\_, CF\_in\_ \rightarrow
         if nc1 > 0 then
           color_flow_ambiguous "colorize_fusion3"
         else
           colored_vertex "colorize_fusion3"
      CF_{-in} -, CF_{-out} -, CF_{-out} -
      CF\_out\_, CF\_in\_, CF\_out\_
      CF\_out\_, CF\_out\_, CF\_in\_ \rightarrow
         if nc1 < 0 then
           color_flow_ambiguous "colorize_fusion3"
         else
           colored_vertex "colorize_fusion3"
      CF_{in} , CF_{in} , CF_{in}
      CF\_out\_, CF\_out\_, CF\_out\_
    | (White \_ | CF\_io \_ | CF\_aux \_),
       White \_ \mid CF\_io \_ \mid CF\_aux \_),
       (White \_ | CF\_io \_ | CF\_aux \_)
    | (CF_in_i | CF_out_i),
         (CF_in_i | CF_out_i),
         (White \_ | CF\_io \_ | CF\_aux \_)
    | (CF_in_i | CF_out_i),
         (White \_ | CF\_io \_ | CF\_aux \_),
         (CF\_in \_ | CF\_out \_)
    | (White \_ | CF\_io \_ | CF\_aux \_),
         (CF_in_i | CF_out_i),
         (CF\_in \_ \mid CF\_out \_) \rightarrow
         colored_vertex "colorize_fusion3"
    end
\mid C.AdjSUN \ nc \rightarrow
    begin match f1, f2, f3 with
      CF_{-in}(-, c1), CF_{-out}(-, c1'), White_{-}
      CF\_out(\_, c1'), CF\_in(\_, c1), White\_
      CF_{in}(-, c1), White -, CF_{out}(-, c1')
      CF\_out(\_, c1'), White\_, CF\_in(\_, c1)
      White \_, CF_in(\_, c1), CF_out(\_, c1')
      White \_, CF\_out(\_, c1'), CF\_in(\_, c1) \rightarrow
```

```
if c1 \neq c1' then
    [CF\_io\ (f,\ c1,\ c1'),\ v]
  else
     [CF\_aux\ f,\ v]
CF_{-in}(-, c1), CF_{-out}(-, c1'), CF_{-aux}
CF\_out(\_, c1'), CF\_in(\_, c1), CF\_aux\_
CF_{in}(\_, c1), CF_{aux}\_, CF_{out}(\_, c1')
CF\_out(\_, c1'), CF\_aux\_, CF\_in(\_, c1)
CF_{aux}, CF_{in} (_, c1), CF_{out} (_, c1')
CF\_aux\_, CF\_out(\_, c1'), CF\_in(\_, c1) \rightarrow
  if c1 \neq c1' then
    [CF\_io\ (f,\ c1,\ c1'),\ mult\_vertex\ (\ 2)\ v]
  else
    [CF\_aux\ f,\ mult\_vertex\ (\ 2)\ v]
CF_{-in}(-, c1), CF_{-out}(-, c1'), CF_{-io}(-, c2, c2')
CF_{-}out(-, c1'), CF_{-}in(-, c1), CF_{-}io(-, c2, c2')
CF_{-in}(-, c1), CF_{-io}(-, c2, c2'), CF_{-out}(-, c1')
CF_out (_, c1'), CF_io (_, c2, c2'), CF_in (_, c1)
CF_{-io}(-, c2, c2'), CF_{-in}(-, c1), CF_{-out}(-, c1')
CF\_io (_, c2, c2'), CF\_out (_, c1'), CF\_in (_, c1) \rightarrow
  if c1 = c2' \wedge c2 = c1' then
    [CF\_aux\ f,\ mult\_vertex\ (\ 2)\ v]
  else if c1 = c2' then
    [CF\_io\ (f,\ c2,\ c1'),\ v]
  else if c2 = c1' then
    [CF\_io\ (f,\ c1,\ c2'),\ v]
  else
```

as the set of permutations of $\{1, 2, 3, 4\}$ with the cyclic permutations factored out, we have:



The different color connections correspond to permutations of the particles entering the fusion and have to be matched by a corresponding permutation of the Lorentz structure:



We have to generalize this for cases of four particles in the adjoint that are not all gluons:

- scalar-scalar-scalar
- scalar-scalar-vector-vector

and even ones including fermions (gluinos) if higher dimensional operators are involved.

|
$$CF_io$$
 (_, $c1$, $c1'$), CF_io (_, $c2$, $c2'$), CF_io (_, $c3$, $c3'$) \rightarrow if $c1'$ = $c2$ \land $c2'$ = $c3$ then

```
[CF\_io\ (f,\ c1,\ c3'),\ permute\_vertex4\ P123\ v]
    else if c1' = c3 \wedge c3' = c2 then
       [CF\_io\ (f,\ c1,\ c2'),\ permute\_vertex4\ P132\ v]
     else if c2' = c3 \wedge c3' = c1 then
       [CF\_io\ (f,\ c2,\ c1'),\ permute\_vertex4\ P231\ v]
     else if c2' = c1 \wedge c1' = c3 then
       [CF\_io\ (f,\ c2,\ c3'),\ permute\_vertex4\ P213\ v]
     else if c3' = c1 \wedge c1' = c2 then
       [CF\_io\ (f,\ c3,\ c2'),\ permute\_vertex4\ P312\ v]
     else if c3' = c2 \wedge c2' = c1 then
       [CF\_io\ (f,\ c3,\ c1'),\ permute\_vertex4\ P321\ v]
     else
  CF\_io\_, CF\_io\_, CF\_aux\_
  CF\_io _, CF\_aux _, CF\_io _
  CF\_aux\_, CF\_io\_, CF\_io\_
  CF\_io\_, CF\_aux\_, CF\_aux\_
  CF\_aux\_, CF\_aux\_, CF\_io\_
  CF_aux _, CF_io _, CF_aux _
  CF\_aux\_, CF\_aux\_, CF\_aux\_ \rightarrow
    CF_{-io} (_, c1, c1'), CF_{-io} (_, c2, c2'), White_{-}
  CF\_io (_, c1, c1'), White _, CF\_io (_, c2, c2')
 White \_, CF\_io (\_, c1, c1'), CF\_io (\_, c2, c2') \rightarrow
    if c1' = c2 then
      [CF\_io\ (f,\ c1,\ c2'),\ mult\_vertex\ (-1)\ v]
    else if c2' = c1 then
       [CF\_io\ (f,\ c2,\ c1'),\ mult\_vertex\ (\ 1)\ v]
     else
       CF_{-io} (_, c1, c1'), CF_{-aux} _, White _
  CF\_aux\_, CF\_io(\_, c1, c1'), White\_
  CF_{-io} (_, c1, c1'), White_{-}, CF_{-aux_{-}}
  CF-aux -, White -, CF-io (-, c1, c1')
  White \_, CF_{-io} (\_, c1, c1'), CF_{-aux}
  White \_, CF\_aux <math>\_, CF\_io(\_, c1, c1') \rightarrow
    CF_{aux} , CF_{aux} , White
  CF\_aux _, White _, CF\_aux _
  White \_, CF_aux <math>\_, CF_aux <math>\_ \rightarrow
    White \_, White \_, CF_{-io}(\_, c1, c1')
  White \_, CF_{-io}(\_, c1, c1'), White \_
  CF_{-io} (-, c1, c1'), White -, White \rightarrow
    [CF\_io\ (f,\ c1,\ c1'),\ v]
  White _, White _, CF_aux _
  White _, CF_aux _, White _
  CF\_aux\_, White\_, White\_ \rightarrow
 White _, White _, White _
| (White \_ | CF\_io \_ | CF\_aux \_),
     (White \_ | CF\_io \_ | CF\_aux \_),
     (CF\_in \_ | CF\_out \_)
| (White \_ | CF\_io \_ | CF\_aux \_),
    (CF_in_{-} | CF_out_{-}),
     (White \_ | CF\_io \_ | CF\_aux \_)
| (CF_in_i | CF_out_i),
     (White \_ | CF\_io \_ | CF\_aux \_),
```

```
(White \_ | CF\_io \_ | CF\_aux \_)
              CF_{-in} -, CF_{-in} -, (White_{-} \mid CF_{-io} - \mid CF_{-aux} -)
              CF_in_, (White_ | CF_io_ | CF_aux_), CF_in_
            (White _ | CF_io _ | CF_aux _), CF_in _, CF_in _
              CF\_out\_,\ CF\_out\_,\ (White\_\mid\ CF\_io\_\mid\ CF\_aux\_)
              CF\_out\_, (White \_ | CF\_io\_ | CF\_aux\_), CF\_out\_
            | (White \_ | CF\_io \_ | CF\_aux \_), CF\_out \_, CF\_out \_
            | (CF_in_i | CF_out_i),
                (CF_in \_ | CF_out \_),
                (CF_in_{-} \mid CF_out_{-}) \rightarrow
                 colored_vertex "colorize_fusion3"
           end
                                           Quintic and Higher Vertices
    let is\_white = function
         White \_ \rightarrow true
         _{-} 
ightarrow false
    let colorize\_fusionn\ flist\ (f,\ v)\ =
       let incomplete\_match () =
         incomplete
            ("colorize_fusionn<sub>□</sub>{<sub>□</sub>" ^
               String.concat ",\Box" (List.map (pullback M.flavor_to_string) flist) \hat{}
             \sqcup_\_->_\" ^ M.flavor\_to\_string f) in
       match M.color f with
         C.Singlet \rightarrow
            if List.for\_all\ is\_white\ flist then
              [White f, v]
            else
              incomplete\_match ()
       \mid C.SUN \_ \rightarrow
           if List.for_all is_white flist then
              colored_vertex "colorize_fusionn"
              incomplete_match ()
       \mid C.AdjSUN \_ \rightarrow
            if List.for_all is_white flist then
              colored_vertex "colorize_fusionn"
           else
              incomplete\_match ()
                                 11.2.4
                                           Colorizing a Monochrome Model
module\ It\ (M\ :\ Model.\ T)\ =
    open Coupling
    module C = Color
    module\ Colored\_Flavor\ =\ Flavor(M)
    type flavor = Colored\_Flavor.t
    type flavor\_sans\_color = M.flavor
    let flavor\_sans\_color = Colored\_Flavor\_flavor\_sans\_color
    type \ gauge = M.gauge
    type constant = M.constant
```

struct

```
\begin{array}{lll} \text{let } options &= M.options \\ \text{let } caveats &= M.caveats \\ \\ \text{open } Colored\_Flavor \\ \\ \text{let } color &= pullback \ M.color \\ \\ \text{let } nc &= M.nc \\ \\ \text{let } pdg &= pullback \ M.pdg \\ \\ \text{let } lorentz &= pullback \ M.lorentz \\ \\ \text{module } Ch &= M.Ch \\ \\ \text{let } charges &= pullback \ M.charges \\ \end{array}
```

For the propagator we cannot use pullback because we have to add the case of the color singlet propagator by hand.

```
let cf_aux_propagator = function
     Prop\_Scalar \rightarrow Prop\_Col\_Scalar (* Spin 0 octets. *)
     Prop\_Majorana \rightarrow Prop\_Col\_Majorana (* Spin 1/2 octets. *)
     Prop\_Feynman \rightarrow Prop\_Col\_Feynman (* Spin 1 states, massless. *)
     Prop\_Unitarity \rightarrow Prop\_Col\_Unitarity (* Spin 1 states, massive. *)
     Aux\_Scalar \rightarrow Aux\_Col\_Scalar (* constant colored scalar propagator *)
     Aux\_Vector \rightarrow Aux\_Col\_Vector (* constant colored vector propagator *)
     Aux\_Tensor\_1 \rightarrow Aux\_Col\_Tensor\_1  (* constant colored tensor propagator *)
     Prop\_Col\_Scalar \mid Prop\_Col\_Feynman
     Prop\_Col\_Majorana \mid Prop\_Col\_Unitarity
     Aux\_Col\_Scalar \mid Aux\_Col\_Vector \mid Aux\_Col\_Tensor\_1

ightarrow \ failwith ("Colorize.It().colorize_propagator:_\perparallelarrow already_colored_particle!")
     _ → failwith ("Colorize.It().colorize_propagator: impossible!")
let propagator = function
     CF\_aux\ f \rightarrow cf\_aux\_propagator\ (M.propagator\ f)
     White f \rightarrow M.propagator f
     CF\_in\ (f,\ \_)\ \to\ M.propagator\ f
     CF\_out(f, \_) \rightarrow M.propagatorf
     CF\_io\ (f, \_, \_) \rightarrow M.propagator\ f
let \ width = pullback \ M.width
let \ goldstone = function
  | White f \rightarrow
       begin match M.goldstone\ f with
        | None \rightarrow None |
       | Some (f', g) \rightarrow Some (White f', g)
       end
  |CF_in(f, c)| \rightarrow
       begin match M.goldstone\ f with
          None \rightarrow None
         Some (f', g) \rightarrow Some (CF_in (f', c), g)
       end
  | CF\_out(f, c) \rightarrow
       begin match M.goldstone\ f with
        | None \rightarrow None |
         Some (f', g) \rightarrow Some (CF\_out (f', c), g)
       end
  | CF_io(f, c1, c2) \rightarrow
       begin match M.goldstone\ f with
         None \rightarrow None
        | Some (f', g) \rightarrow Some (CF_io (f', c1, c2), g) |
       end
  \mid CF_{-}aux f \rightarrow
       begin match M.goldstone\ f with
        | None \rightarrow None |
        | Some (f', g) \rightarrow Some (CF\_aux f', g) |
       end
```

```
let conjugate = function
     White f \rightarrow White (M.conjugate f)
     CF_{-in}(f, c) \rightarrow CF_{-out}(M.conjugate f, c)
     CF\_out\ (f,\ c)\ 	o\ CF\_in\ (M.conjugate\ f,\ c)
     CF\_io\ (f,\ c1,\ c2)\ 	o\ CF\_io\ (M.conjugate\ f,\ c2,\ c1)
     CF\_aux\ f \rightarrow CF\_aux\ (M.conjugate\ f)
let conjugate\_sans\_color = M.conjugate
\mathsf{let}\ \mathit{fermion}\ =\ \mathit{pullback}\ \mathit{M.fermion}
let max\_degree = M.max\_degree
let flavors() =
   invalid "flavors"
let external\_flavors() =
   invalid "external_flavors"
let parameters = M.parameters
let split\_color\_string s =
  try
     \mathsf{let}\ i1\ =\ \mathit{String}.\mathit{index}\ s\ \texttt{'/'}\ \mathsf{in}
     let i2 = String.index\_from \ s \ (succ \ i1) '/' in
     let sf = String.sub s 0 i1
     and sc1 = String.sub \ s \ (succ \ i1) \ (i2 - i1 - 1)
     and sc2 = String.sub \ s \ (succ \ i2) \ (String.length \ s \ - \ i2 \ - \ 1) in
     (sf, sc1, sc2)
  with
  | Not\_found \rightarrow (s, "", "")
let flavor\_of\_string \ s =
  try
     let sf, sc1, sc2 = split\_color\_string s in
     let f = M.flavor\_of\_string \ sf in
     match M.color f with
     \mid C.Singlet \rightarrow White f
     \mid C.SUN \ nc \rightarrow
          if nc > 0 then
             CF_in (f, color\_flow\_of\_string sc1)
             CF\_out\ (f,\ color\_flow\_of\_string\ sc2)
     \mid C.AdjSUN \_ \rightarrow
          begin match sc1, sc2 with
           | "", "" \rightarrow CF\_aux f
           | \_, \_ \rightarrow CF\_io (f, color\_flow\_of\_string sc1, color\_flow\_of\_string sc2)
          end
  with
   \mid Failure \ s \rightarrow
       if s = "int_of_string" then
         invalid_arg "Colorize().flavor_of_string:_expecting_integer"
      else
         failwith \ ("Colorize().flavor_of_string: \ unexpected \ Failure(" ^ s ^ ")")
let flavor\_to\_string = function
   | White f \rightarrow
        M.flavor\_to\_string f
   \mid CF_{-}in(f, c) \rightarrow
        M.flavor\_to\_string\ f\ ^ "/"\ ^ string\_of\_int\ c\ ^ "/"
  | CF\_out(f, c) \rightarrow
        M.flavor\_to\_string\ f\ ^"//"\ ^string\_of\_int\ c
   | CF_{-}io(f, c1, c2) \rightarrow
        M.flavor\_to\_string \ f \ ^ "/" \ ^ string\_of\_int \ c1 \ ^ "/" \ ^ string\_of\_int \ c2
  \mid CF\_aux f \rightarrow
        M.flavor\_to\_string\ f\ ^ "//"
```

```
let flavor_to_TeX = function
         White f \rightarrow
             M.flavor\_to\_TeX f
          CF_{-}in (f, c) \rightarrow
             "\{" ^{n}M.flavor\_to\_TeX\ f ^{n}\}_{\infty}"}" \{^{n}M.flavor\_to\_TeX\ f ^{n}\}_{\infty}"
        | CF\_out(f, c) \rightarrow
             "{" ^{n}M.flavor\_to\_TeX\ f ^{n}}_{\infty}_{\infty} 
             string\_of\_int\ c\ ^"}""
       | CF_io(f, c1, c2) \rightarrow
             "\{" ^{\land} M.flavor\_to\_TeX f ^{\land}"\}_{\
             string_of_int c1 ^ "\\overline{" ^ string_of_int c2 ^ "}}"
       \mid CF_{-}aux f \rightarrow
             "\{" ^ M.flavor_to_TeX f ^ "\}_{\{\mbox{$\setminus$ mathstrut}_{\square}0\}}"
     let flavor\_symbol = function
        | White f \rightarrow
             M.flavor\_symbol\ f
          CF_{-}in(f, c) \rightarrow
             M.flavor\_symbol\ f\ ^ "_"\ ^ string\_of\_int\ c\ ^ "_"
       | CF\_out(f, c) \rightarrow
             M.flavor\_symbol\ f ^ "\_" ^ string\_of\_int\ c
        CF_{-}io(f, c1, c2) \rightarrow
             M.flavor\_symbol\ f\ ``"\_"\ `string\_of\_int\ c1\ ``"\_"\ `string\_of\_int\ c2
       | CF_aux f \rightarrow
             M.flavor\_symbol\ f\ ^ "\_\_"
     let gauge\_symbol = M.gauge\_symbol
Masses and widths must not depend on the colors anyway!
     let \ mass\_symbol = pullback \ M.mass\_symbol
     let \ width\_symbol = pullback \ M.width\_symbol
     let constant\_symbol = M.constant\_symbol
```

Vertices

vertices are only used by functor applications and for indexing a cache of precomputed fusion rules, which is not used for colorized models.

```
\begin{split} &\text{let } vertices\;() = \\ & invalid\; "\texttt{vertices}" \\ &\text{module } Legacy = Legacy\_Implementation\;(M) \\ &\text{let } colorize\_fusion2\;f1\;f2\;(f,\;v) = \\ & \mathsf{match}\;v\;\mathsf{with} \\ &\mid V3\_ \to Legacy.colorize\_fusion2\;f1\;f2\;(f,\;v) \\ &\mid \_ \to [] \\ &\text{let } colorize\_fusion3\;f1\;f2\;f3\;(f,\;v) = \\ & \mathsf{match}\;v\;\mathsf{with} \\ &\mid V4\_ \to Legacy.colorize\_fusion3\;f1\;f2\;f3\;(f,\;v) \\ &\mid \_ \to [] \\ \end{split}
```

In order to match the *correct* positions of the fields in the vertices, we have to undo the permutation effected by the fusion according to *Coupling.fusen*.

```
module \ PosMap = Partial.Make \ (struct \ type \ t = int \ let \ compare = compare \ end)
```

Note that due to the *inverse*, the list l' can be interpreted here as a map reshuffling the indices. E.g., inverse (Permutation.Default.list [2;0;1]) applied to [1;2;3] gives [3;1;2].

```
let partial\_map\_redoing\_permutation\ l\ l'= let module\ P=Permutation.Default\ in let p=P.inverse\ (P.of\_list\ (List.map\ pred\ l'))\ in
```

```
PosMap.of\_lists\ l\ (P.list\ p\ l)
```

Note that, the list l' can not be interpreted as a map reshuffling the indices, but gives the new order of the argument. E. g., Permutation.Default.list [2;0;1] applied to [1;2;3] gives [2;3;1].

```
let \ partial\_map\_undoing\_permutation \ l \ l' \ =
  let module P = Permutation.Default in
  let p = P.of\_list (List.map pred l') in
  PosMap.of\_lists\ l\ (P.list\ p\ l)
module CA = Color.Arrow
\mathsf{module}\ CV\ =\ Color.Vertex
module CP = Color.Propagator
let color\_sans\_flavor = function
     White \_ \rightarrow CP.W
     CF_{-in}(-, cfi) \rightarrow CP.I cfi
     CF\_out\ (\_,\ cfo)\ 	o\ CP.O\ cfo
     CF\_io (\_, cfi, cfo) \rightarrow CP.IO (cfi, cfo)
     CF\_aux \_ \rightarrow CP.G
let color\_with\_flavor f = function
    CP.W \rightarrow White f
     CP.I \ cfi \rightarrow CF\_in \ (f, \ cfi)
     CP.O \ cfo \rightarrow CF\_out \ (f, \ cfo)
     CP.IO\ (cfi,\ cfo) \rightarrow CF\_io\ (f,\ cfi,\ cfo)
    CP.G \rightarrow CF\_aux f
let colorize \ vertex\_list \ flavors \ f \ v =
   List.map
     (fun (coef, cf) \rightarrow (color\_with\_flavor f cf, cmult\_vertex coef v))
     (CV.fuse (nc ()) vertex_list (List.map color_sans_flavor flavors))
let partial\_map\_undoing\_fusen fusen =
  partial\_map\_undoing\_permutation
     (ThoList.range\ 1\ (List.length\ fusen))
     fusen
let undo_permutation_of_fusen fusen =
   PosMap.apply\_with\_fallback
     (fun _ → invalid_arg "permutation_of_fusen")
     (partial_map_undoing_fusen fusen)
let\ colorize\_fusionn\_ufo\ flist\ f\ c\ v\ spins\ flines\ color\ fuse\ xtra\ =
  let v = Vn (UFO (c, v, spins, flines, Color.Vertex.one), fuse, xtra) in
  let p = undo\_permutation\_of\_fusen fuse in
   colorize (CV.relocate p color) flist f v
let colorize\_fusionn\ flist\ (f,\ v)\ =
  \mathsf{match}\ v\ \mathsf{with}
   | Vn (UFO (c, v, spins, flines, color), fuse, xtra) \rightarrow
      colorize_fusionn_ufo flist f c v spins flines color fuse xtra
  -\rightarrow
let fuse\_list flist =
   ThoList.flatmap
     (colorize_fusionn flist)
     (M.fuse\ (List.map\ flavor\_sans\_color\ flist))
let fuse2 f1 f2 =
   List.rev\_append
     (fuse\_list [f1; f2])
     (ThoList.flatmap
         (colorize_fusion2 f1 f2)
         (M.fuse2)
             (flavor\_sans\_color\ f1)
             (flavor\_sans\_color\ f2)))
```

```
let fuse3 f1 f2 f3 =
        List.rev_append
          (fuse\_list [f1; f2; f3])
          (ThoList.flatmap
              (colorize_fusion3 f1 f2 f3)
              (M.fuse3)
                  (flavor\_sans\_color\ f1)
                  (flavor\_sans\_color f2)
                  (flavor\_sans\_color\ f3)))
     let fuse = function
         [] \mid [\_] \rightarrow invalid\_arg "Colorize.It().fuse"
         [f1; f2] \rightarrow fuse2 f1 f2
          [f1; f2; f3] \rightarrow fuse3 f1 f2 f3
         flist \rightarrow fuse\_list flist
     let max\_degree = M.max\_degree
                                           Adding Color to External Particles
     let count\_color\_strings f\_list =
       let rec count\_color\_strings' n\_in n\_out n\_glue = function
          | f :: rest \rightarrow
               begin match M.color\ f with
                  C.Singlet \rightarrow count\_color\_strings' n\_in n\_out n\_glue rest
                 C.SUN \ nc \rightarrow
                    if nc > 0 then
                       count\_color\_strings' (succ n\_in) n\_out n\_glue rest
                    else if nc < 0 then
                       count\_color\_strings' n\_in (succ n\_out) n\_glue rest
                       su\theta "count_color_strings"
               \mid C.AdjSUN \_ \rightarrow
                     count\_color\_strings' (succ n\_in) (succ n\_out) (succ n\_glue) rest
          | \ | \ | \ \rightarrow \ (n_-in, \ n_-out, \ n_-glue)
       in
        count_color_strings' 0 0 0 f_list
     let external\_color\_flows f\_list =
       let n_i, n_j out, n_j lue = count_i color_strings f_i list in
       if n_{-}in \neq n_{-}out then
          else
          let color\_strings = ThoList.range \ 1 \ n\_in in
          List.rev\_map
             (\text{fun } permutation \rightarrow (color\_strings, permutation))
             (Combinatorics.permute\ color\_strings)
If there are only adjoints and there are no couplings of adjoints to singlets, we can ignore the U(1)-ghosts.
     let pure\_adjoints\ f\_list\ =
        List.for\_all (fun f \rightarrow match \ M.color \ f with C.AdjSUN \ \_ \rightarrow true \ | \ \_ \rightarrow false) \ f\_list
     let two\_adjoints\_couple\_to\_singlets() =
       let vertices3, vertices4, verticesn = M.vertices() in
       List.exists (fun ((f1, f2, f3), _, _) \rightarrow
          match M.color f1, M.color f2, M.color f3 with
            C.AdjSUN _, C.AdjSUN _, C.Singlet
            C.AdjSUN _, C.Singlet, C.AdjSUN _
            C.Singlet, C.AdjSUN \_, C.AdjSUN \_ \rightarrow true
          \mid \_ \rightarrow \mathsf{false}) \ vertices3 \lor
       List.exists (fun ((f1, f2, f3, f4), _, _) \rightarrow
```

We use List.hd and List.tl instead of pattern matching, because we consume ecf_in and ecf_out at a different pace.

```
let tail\_opt = function
   [] \rightarrow []
  - :: tail \rightarrow tail
let head\_req = function
  | [] \rightarrow
        invalid\_arg "Colorize_It().colorize\_crossed_amplitude1:_{\sqcup}insufficient_{\sqcup}flows"
\label{let_rec_colorize_crossed_amplitude1} \ \ ghosts \ \ acc \ f\_list \ (ecf\_in, \ ecf\_out) \ = \\
  match f_list, ecf_in, ecf_out with
   [], [], [] \rightarrow [List.rev acc]
  | [], -, - \rightarrow
        invalid_arg "Colorize.It().colorize_crossed_amplitude1: _leftover_flows"
  | f :: rest, \_, \_ \rightarrow
        begin match M.color\ f with
        \mid C.Singlet \rightarrow
             colorize_crossed_amplitude1 ghosts
                (White f :: acc)
                rest (ecf_in, ecf_out)
        \mid C.SUN \ nc \rightarrow
             if nc > 0 then
                colorize\_crossed\_amplitude1\ ghosts
                   (CF_in (f, head_req ecf_in) :: acc)
                  rest (tail_opt ecf_in, ecf_out)
             else if nc < 0 then
                colorize\_crossed\_amplitude1 ghosts
                   (CF\_out\ (f,\ head\_req\ ecf\_out)\ ::\ acc)
                  rest (ecf_in, tail_opt ecf_out)
             else
                su\theta "colorize_flavor"
        \mid C.AdjSUN \_ \rightarrow
             \mathsf{let}\ \mathit{ecf\_in'}\ =\ \mathit{head\_req}\ \mathit{ecf\_in}
             and ecf\_out' = head\_req \ ecf\_out in
             if ecf_in' = ecf_out' then begin
               if ghosts then
                  colorize\_crossed\_amplitude1 ghosts
                     (CF\_aux\ f\ ::\ acc)
                     rest\ (tail\_opt\ ecf\_in,\ tail\_opt\ ecf\_out)
               else
             end else
                colorize\_crossed\_amplitude1 ghosts
                  (CF\_io\ (f,\ ecf\_in',\ ecf\_out')\ ::\ acc)
                  rest (tail_opt ecf_in, tail_opt ecf_out)
```

```
let\ colorize\_crossed\_amplitude1\ ghosts\ f\_list\ (ecf\_in,\ ecf\_out)\ =
              colorize_crossed_amplitude1 ghosts [] f_list (ecf_in, ecf_out)
         let colorize\_crossed\_amplitude\ f\_list\ =
              ThoList.rev_flatmap
                   (colorize\_crossed\_amplitude1\ (external\_ghosts\ f\_list)\ f\_list)
                   (external\_color\_flows f\_list)
         let cross\_uncolored p\_in p\_out =
              (List.map M.conjugate \ p\_in) @ p\_out
         let uncross\_colored\ n\_in\ p\_lists\_colorized\ =
              let p_i = n_i - out_i = list_i = list
              List.map
                   (fun (p\_in\_colored, p\_out\_colored) \rightarrow
                        (List.map\ conjugate\ p\_in\_colored,\ p\_out\_colored))
                   p\_in\_out\_colorized
         let amplitude \ p_in \ p_out =
              uncross\_colored
                   (List.length \ p_in)
                   (colorize\_crossed\_amplitude\ (cross\_uncolored\ p\_in\ p\_out))
The —-sign in the second component is redundant, but a Whizard convention.
         let indices = function
                  White \_ \rightarrow Color.Flow.of\_list [0; 0]
                  CF_{-in}(-, c) \rightarrow Color.Flow.of_{-list}[c; 0]
                  CF\_out(\_, c) \rightarrow Color.Flow.of\_list[0; -c]
                  CF\_io(\_, c1, c2) \rightarrow Color.Flow.of\_list[c1; -c2]
                  CF\_aux\ f \rightarrow Color.Flow.ghost\ ()
         let flow p_in p_out =
              (List.map\ indices\ p\_in,\ List.map\ indices\ p\_out)
    end
                                                                             Colorizing a Monochrome Gauge Model
module \ Gauge \ (M : Model. Gauge) =
    struct
         module CM = It(M)
         type flavor = CM.flavor
         type flavor\_sans\_color = CM.flavor\_sans\_color
         \mathsf{type}\ \mathit{gauge}\ =\ \mathit{CM}.\mathit{gauge}
         type constant = CM.constant
         module Ch = CM.Ch
         let charges = CM.charges
         let flavor\_sans\_color = CM.flavor\_sans\_color
         let color = CM.color
         let pdg = CM.pdg
         let lorentz = CM.lorentz
         let propagator = CM.propagator
         let width = CM.width
         let conjugate = CM.conjugate
         let conjugate_sans_color = CM.conjugate_sans_color
         let fermion = CM.fermion
         let max\_degree = CM.max\_degree
         let \ vertices = CM.vertices
         let fuse2 = CM.fuse2
         let fuse 3 = CM.fuse 3
         let fuse = CM.fuse
```

```
let flavors = CM.flavors
let nc = CM.nc
let\ external\_flavors\ =\ CM.external\_flavors
\mathsf{let}\ goldstone\ =\ CM.goldstone
let parameters = CM.parameters
let flavor\_of\_string = CM.flavor\_of\_string
\mathsf{let}\ \mathit{flavor\_to\_string}\ =\ \mathit{CM.flavor\_to\_string}
\mathsf{let}\ \mathit{flavor\_to\_TeX}\ =\ \mathit{CM.flavor\_to\_TeX}
let flavor\_symbol = CM.flavor\_symbol
let gauge\_symbol = CM.gauge\_symbol
\mathsf{let}\ mass\_symbol\ =\ CM.mass\_symbol
let \ width\_symbol = CM.width\_symbol
let constant\_symbol = CM.constant\_symbol
let options = CM.options
\mathsf{let}\ \mathit{caveats}\ =\ \mathit{CM.caveats}
let incomplete s =
  failwith \ (\verb"Colorize.Gauge()." ^ s ^ " \_not \_done \_yet!")
\mathsf{type}\ \mathit{matter\_field}\ =\ \mathit{M.matter\_field}
type gauge\_boson = M.gauge\_boson
type other = M.other
\mathsf{type} \ \mathit{field} \ =
     Matter\ {
m of}\ matter\_field
     Gauge\ {\it of}\ gauge\_boson
     Other of other
let field f =
   incomplete "field"
let matter\_field f =
  incomplete "matter_field"
let gauge\_boson f =
  incomplete "gauge_boson"
\mathsf{let}\ \mathit{other}\ f\ =
   incomplete "other"
let \ amplitude = CM.amplitude
let flow = CM.flow
```

-12-

Processes

12.1 Interface of Process

module type T =sig type flavor



Eventually this should become an abstract type:

```
type t = flavor\ list \times flavor\ list
val\ incoming\ :\ t\ 	o\ flavor\ list
val outgoing: t \rightarrow flavor\ list
```

 $parse_decay$ s decodes a decay description " $a_{\sqcup}->_{\sqcup}b_{\sqcup}c_{\sqcup}...$ ", where each word is split into a bag of flavors separated by ':'s.

```
type decay
val\ parse\_decay : string \rightarrow decay
val\ expand\_decays:\ decay\ list 
ightarrow \ t\ list
```

 $parse_scattering \ s \ decodes \ a \ scattering \ description \ "a_{\sqcup}b_{\sqcup}->_{\sqcup}c_{\sqcup}d_{\sqcup}\dots$ ", where each word is split into a bag of flavors separated by ':'s.

```
type scattering
\verb|val|| parse\_scattering : string \rightarrow scattering|
val expand\_scatterings: scattering\ list \rightarrow\ t\ list
```

parse_process s decodes process descriptions

"a b c d"
$$\Rightarrow Any [a; b; c; d]$$
 (12.1a)

"a -> b c d"
$$\Rightarrow$$
 $Decay (a, [b; c; d])$ (12.1b)

"a b
$$\rightarrow$$
 c d" \Rightarrow Scattering $(a, b, [c; d])$ (12.1c)

where each word is split into a bag of flavors separated by ': 's.

```
type any
type process = Any of any | Decay of decay | Scattering of scattering
val parse\_process: string \rightarrow process
```

remove_duplicate_final_states partition processes removes duplicates from processes, which differ only by a permutation of final state particles. The permutation must respect the partitioning given by the offset 1 integers in partition.

```
val remove\_duplicate\_final\_states: int\ list\ list 
ightarrow t\ list 
ightarrow t\ list
```

diff set1 set2 returns the processes in set1 with the processes in set2 removed. set2 does not need to be a subset of set1.

$$\mathsf{val}\ \mathit{diff}\ :\ t\ \mathit{list}\,\rightarrow\ t\ \mathit{list}\,\rightarrow\ t\ \mathit{list}$$



Not functional yet. Interface subject to change. Should be moved to Fusion. Multi, because we will want to cross colored matrix elements.

```
Factor amplitudes that are related by crossing symmetry.
      val\ crossing\ :\ t\ list 
ightarrow\ (flavor\ list 	imes\ int\ list 	imes\ t)\ list
  end
module\ Make\ (M\ :\ Model.T)\ :\ T\ with\ type\ flavor\ =\ M.flavor
                                                       Implementation of Process
module type T =
  sig
      type flavor
     type t = flavor \ list \times flavor \ list
     val\ incoming\ :\ t\ 	o\ flavor\ list
     \mathsf{val}\ outgoing\ :\ t\ \to\ \mathit{flavor}\ \mathit{list}
     type decay
     \mathsf{val}\ parse\_decay\ :\ string \to\ decay
     val\ expand\_decays: decay\ list \rightarrow\ t\ list
      type scattering
     val parse\_scattering : string \rightarrow scattering
     val expand\_scatterings: scattering\ list \rightarrow t\ list
     type any
     type process = Any of any | Decay of decay | Scattering of scattering
     val\ parse\_process : string \rightarrow process
     val\ remove\_duplicate\_final\_states\ :\ int\ list\ list\ 
ightarrow\ t\ list\ 
ightarrow\ t\ list
     \mathsf{val}\ \mathit{diff}\ :\ \mathit{t}\ \mathit{list} \to\ \mathit{t}\ \mathit{list} \to\ \mathit{t}\ \mathit{list}
     val\ crossing:\ t\ list 
ightarrow\ (flavor\ list 	imes\ int\ list 	imes\ t)\ list
module\ Make\ (M\ :\ Model.\ T)\ =
  struct
     type flavor = M.flavor
     type t = flavor list \times flavor list
     let incoming (fin, \_) = fin
     let outgoing( \_, fout) = fout
                                       12.2.1 Select Charge Conserving Processes
     let allowed (fin, fout) =
         M.Ch.is_null (M.Ch.sum (List.map M.charges (List.map M.conjugate fin @ fout)))
                                             12.2.2 Parsing Process Descriptions
     \mathsf{type} \,\, \alpha \,\, \mathit{bag} \,\, = \,\, \alpha \,\, \mathit{list}
      \mathsf{type} \ any \ = \ \mathit{flavor} \ \mathit{bag} \ \mathit{list}
      type decay = flavor bag \times flavor bag list
      \mathsf{type} \ \mathit{scattering} \ = \ \mathit{flavor} \ \mathit{bag} \ \times \ \mathit{flavor} \ \mathit{bag} \ \times \ \mathit{flavor} \ \mathit{bag} \ \mathit{ist}
     type process =
           Any of any
           Decay of decay
         | Scattering of scattering
     let unique\_flavors\ f\_bags\ =
         List.for\_all \text{ (function } [f] \rightarrow \text{true } | \_ \rightarrow \text{ false) } f\_bags
     let unique\_final\_state = function
           Any \ fs \rightarrow unique\_flavors \ fs
         | Decay(_-, fs) \rightarrow unique\_flavors fs
```

```
| Scattering(_-, _-, fs) \rightarrow unique\_flavors fs
let parse_process process =
  let \ last = String.length \ process \ - \ 1
  and flavor off len = M.flavor_of_string (String.sub process off len) in
  let add_flavors flavors = function
       Any \ l \rightarrow Any \ (List.rev \ flavors :: l)
        Decay(i, f) \rightarrow Decay(i, List.rev flavors :: f)
     | Scattering (i1, i2, f) \rightarrow Scattering (i1, i2, List.rev flavors :: f) in
  let rec scan_list so_far n =
     if n > last then
        so\_far
     else
        \mathsf{let}\ n'\ =\ \mathit{succ}\ n\ \mathsf{in}
        match process.[n] with
        | ', ', | '\n' \rightarrow scan\_list so\_far n'
        , -, \rightarrow scan\_qtr so\_far n'
        c \rightarrow scan\_flavors so\_far [] n n'
  and scan_flavors so_far flavors w n =
     if n > last then
        add_{-}flavors (flavor w (last - w + 1) :: flavors) so_{-}far
     else
        \mathsf{let}\ n'\ =\ \mathit{succ}\ n\ \mathsf{in}
        match process.[n] with
        \mid '', \mid '\n' \rightarrow
              scan\_list (add\_flavors (flavor w (n - w) :: flavors) so\_far) n'
        |  ':' \rightarrow  scan\_flavors \ so\_far \ (flavor \ w \ (n \ - \ w) \ :: \ flavors) \ n' \ n'
        \mid \_ \rightarrow scan\_flavors so\_far flavors w n'
  and scan\_gtr \ so\_far \ n =
     if n > last then
        invalid_arg "expecting<sub>□</sub>'>'"
     else
        \mathsf{let}\ n'\ =\ \mathit{succ}\ n\ \mathsf{in}
        match process.[n] with
        \mid '>' \rightarrow
              begin match so\_far with
              |Any[i] \rightarrow scan\_list(Decay(i, [])) n'
                Any [i2; i1] \rightarrow scan\_list (Scattering (i1, i2, [])) n'
              \mid Any \_ \rightarrow invalid\_arg  "only\_1\_or\_2\_particles\_in\_|in>"
              | _ → invalid_arg "too_many_'.->'s"
              \quad \text{end} \quad
        \mid \_ \rightarrow invalid\_arg "expecting_{\sqcup}'>'" in
  \mathsf{match}\ \mathit{scan\_list}\ (\mathit{Any}\ [\,])\ 0\ \mathsf{with}
     Any \ l \rightarrow Any \ (List.rev \ l)
     Decay(i, f) \rightarrow Decay(i, List.rev f)
     Scattering (i1, i2, f) \rightarrow Scattering (i1, i2, List.rev f)
let parse\_decay process =
  match parse_process process with
   |Any(i :: f) \rightarrow
        prerr_endline "missing_'`->'uin_process_description,_assuming_decay.";
        (i, f)
   | Decay (i, f) \rightarrow (i, f)
   |\hspace{.06cm} \_ \hspace{.06cm} 
ightarrow \hspace{.06cm} invalid\_arg "expecting\_decay\_description:\_got\_scattering"
let parse_scattering process =
  match parse_process process with
   Any (i1 :: i2 :: f) \rightarrow
        prerr\_endline \ \verb"missing$$\_$`->,$$$ $\_in$$$\_process$$\_description,$$$$\_assuming$$\_scattering.";
        (i1, i2, f)
```

```
Scattering (i1, i2, f) \rightarrow (i1, i2, f)
        | \_ \rightarrow invalid\_arg "expecting_{\sqcup}scattering_{\sqcup}description:_{\sqcup}got_{\sqcup}decay"
     let\ expand\_scatterings\ scatterings\ =
        Tho List. flat map
           (function (fin1, fin2, fout) \rightarrow
             Product.fold
                (fun flist \ acc \rightarrow
                   match flist with
                   | fin1' :: fin2' :: fout' \rightarrow
                        let fin\_fout' = ([fin1'; fin2'], fout') in
                        if allowed \ fin\_fout' then
                           fin\_fout' :: acc
                        else
                           acc
                   [\_] \ [] \ 	o \ failwith "Omega.expand_scatterings:\_can't_happen")
                (fin1 :: fin2 :: fout) []) scatterings
     let expand\_decays decays =
        Tho List. flat map \\
           (function (fin, fout) \rightarrow
             Product.fold
                (fun flist \ acc \rightarrow
                   match flist with
                   \mid fin' :: fout' \rightarrow
                        \mathsf{let}\ \mathit{fin\_fout'}\ =\ ([\mathit{fin'}],\ \mathit{fout'})\ \mathsf{in}
                        if allowed \ fin\_fout' then
                           fin\_fout' :: acc
                        else
                   | ~[] ~\to ~ \it{failwith} ~" \tt Omega.expand\_decays: \_ can't \_ happen")
                (fin :: fout) []) decays
                                      12.2.3 Remove Duplicate Final States
Test if all final states are the same. Identical to ThoList.homogeneous \circ (List.map\ snd).
     let rec homogeneous\_final\_state = function
        | [] | [_{-}] \rightarrow \mathsf{true}
        | (\_, fs1) :: ((\_, fs2) :: \_ as rest) \rightarrow
             if fs1 \neq fs2 then
                false
             else
                homogeneous\_final\_state\ rest
     let by\_color f1 f2 =
        let c = Color.compare (M.color f1) (M.color f2) in
        if c \neq 0 then
           c
        else
           compare f1 f2
     module Pre\_Bundle =
        struct
          \mathsf{type}\ elt\ =\ t
           type base = elt
          let compare\_elt (fin1, fout1) (fin2, fout2) =
             let c = ThoList.compare \ \ cmp : by\_color \ fin1 \ fin2 \ in
             if c \neq 0 then
                c
             else
                ThoList.compare ~cmp: by_color fout1 fout2
```

```
let \ compare\_base \ b1 \ b2 \ = \ compare\_elt \ b2 \ b1
     module \ Process\_Bundle = Bundle.Dyn \ (Pre\_Bundle)
     let to\_string\ (fin,\ fout) =
        String.concat "" (List.map M.flavor_to_string fin)
         " \_ -> \_ " \hat{String.concat} " \_ " (List.map\ M.flavor\_to\_string\ fout)
     let fiber_to_string (base, fiber) =
        (to\_string\ base) ^ "_{\sqcup} ->_{\sqcup} ["
        (String.concat ", " (List.map to\_string fiber)) ^ "] "
     let bundle\_to\_strings\ list =
        List.map fiber_to_string list
Subtract n+1 from each element in index\_set and drop all negative numbers from the result.
     let shift\_left\_pred' \ n \ index\_set =
        List.fold\_right
           (fun i \ acc \rightarrow \text{let } i' = i - n - 1 \text{ in if } i' < 0 \text{ then } acc \text{ else } i' :: acc)
           index\_set
Convert 1-based indices for initial and final state to 0-based indices for the final state only. (NB: ThoList.partitioned_sort
expects 0-based indices.)
     let \ shift\_left\_pred \ fin \ index\_sets \ =
        let n~=~{\rm match}~{\it fin}~{\rm with}~[\_]~\rightarrow~1~|~[\_;\_]~\rightarrow~2~|~\_~\rightarrow~0~{\rm in}
        List.fold\_right
          (fun iset \ acc \rightarrow
             \mathsf{match}\ \mathit{shift\_left\_pred'}\ \mathit{n}\ \mathit{iset}\ \mathsf{with}
              | [] \rightarrow acc
              |iset' \rightarrow iset' :: acc)
           index_sets []
     module FSet = Set.Make (struct type t = flavor let compare = compare end)
Take a list of final states and return a list of sets of flavors appearing in each slot.
     let flavors = function
        | [] \rightarrow []
        | fs :: fs\_list \rightarrow
             List.fold_right (List.map2 FSet.add) fs_list (List.map FSet.singleton fs)
     let flavor_sums flavor_sets =
        let _{-}, result =
          List.fold\_left
             (fun (n, acc) flavors \rightarrow
                if FSet.cardinal\ flavors = 1 then
                   (succ \ n, \ acc)
                else
                   (succ\ n,\ (n,\ flavors)\ ::\ acc))
             (0, []) flavor_sets in
        List.rev result
     let overlapping s1 s2 =
        \neg (FSet.is\_empty (FSet.inter s1 s2))
     let rec merge\_overlapping (n, flavors) = function
        [] \rightarrow [([n], flavors)]
        \mid (n\_list, flavor\_set) :: rest \rightarrow
             if overlapping flavors flavor_set then
                (n :: n\_list, FSet.union flavors flavor\_set) :: rest
             else
                (n\_list, flavor\_set) :: merge\_overlapping (n, flavors) rest
     let overlapping_flavor_sums flavor_sums =
        List.rev\_map
```

```
(\text{fun } (n\_list, flavor\_set) \rightarrow (n\_list, FSet.elements flavor\_set))
      (List.fold_right merge_overlapping flavor_sums [])
let integer\_range \ n1 \ n2 =
   let rec integer\_range' acc n' =
     if n' < n1 then
         acc
      else
         integer\_range' (Sets.Int.add n' acc) (pred n') in
   integer_range' Sets.Int.empty n2
let coarsest\_partition = function
   [] 
ightarrow invalid\_arg "coarsest_partition:\squareempty\squareprocess\squarelist"
   ((\_, fs) :: \_) as proc\_list \rightarrow
        let fs\_list = List.map snd proc\_list in
        let \ overlaps =
           List.map fst (overlapping_flavor_sums (flavor_sums (flavors fs_list))) in
        let singletons =
           Sets.Int.elements
              (List.fold\_right\ Sets.Int.remove
                 (List.concat overlaps) (integer_range 0 (pred (List.length fs)))) in
         List.map (fun n \rightarrow [n]) singletons @ overlaps
module IPowSet =
   PowSet.Make (struct type t = int let compare = compare let to\_string = string\_of\_int end)
let merge\_partitions p\_list =
   IPowSet.to\_lists (IPowSet.basis (IPowSet.union (List.map IPowSet.of\_lists p\_list)))
let \ remove\_duplicate\_final\_states \ cascade\_partition = function
     [] \rightarrow []
     [process] \rightarrow [process]
     list \rightarrow
        if homogeneous_final_state list then
        else
           let partition = coarsest\_partition \ list \ in
           let pi (fin, fout) =
             let partition' =
                merge_partitions [partition; shift_left_pred fin cascade_partition] in
              (fin, ThoList.partitioned_sort by_color partition' fout) in
           Process_Bundle.base (Process_Bundle.of_list pi list)
\mathsf{type}\ t'\ =\ t
module PSet = Set.Make (struct type t = t' let compare = compare end)
let set \ list =
   List.fold_right PSet.add list PSet.empty
let diff\ list1\ list2\ =
   PSet.elements (PSet.diff (set list1) (set list2))
Not functional yet.
module Crossing\_Projection =
   struct
      type elt = t
     \mathsf{type}\ base\ =\ \mathit{flavor}\ \mathit{list}\ \times\ \mathit{int}\ \mathit{list}\ \times\ \mathit{t}
     let compare\_elt\ (fin1,\ fout1)\ (fin2,\ fout2)\ =
        let c = ThoList.compare \ \ \ \ cmp : by\_color \ fin1 \ fin2 \ in
        if c \neq 0 then
           c
        else
           ThoList.compare ~cmp: by_color fout1 fout2
```

```
 \begin{array}{l} \text{let } compare\_base\ (f1,\ \_,\ \_)\ (f2,\ \_,\ \_)\ = \\ ThoList.compare\ \~cmp:by\_color\ f1\ f2 \\ \\ \text{let } pi\ (fin,\ fout\ as\ process)\ = \\ \text{let } flist,\ indices\ = \\ ThoList.ariadne\_sort\ \~cmp:by\_color\ (List.map\ M.conjugate\ fin\ @\ fout)\ in\ (flist,\ indices,\ process) \\ \text{end} \\ \\ \text{module } Crossing\_Bundle\ =\ Bundle.Make\ (Crossing\_Projection) \\ \\ \text{let } crossing\ processes\ = \\ List.map\ (fun\ (fin,\ fout\ as\ process)\ \to \\ (List.map\ M.conjugate\ fin\ @\ fout,\ [\ ],\ process)) \\ \\ processes \\ \\ \text{end} \\ \\ \end{array}
```

—13— Model Files

13.1 Interface of Vertex_syntax

The concrete syntax described below is modelled on LATEX and correct model descriptions should be correct LATEX-input (provided a few simple macros have been loaded.

13.1.1 Abstract Syntax

exception $Syntax_Error$ of $string \times Lexing.position \times Lexing.position$

Tokens

Tokenization follows TEX's rules.

```
egin{array}{c} \mathsf{module} \ \mathit{Token} \ : \ \mathsf{sig} \end{array}
```

Single-character tokens other than digits are stored as one character strings. Multi-character tokens like ψ are stored as a string including the leading $\$. Since a_12 is interpreted by T_EX as $\{a_1\}_2$, we can not use the lexer to construct integers, but interpret them as lists of digits. Below, in Expr, the parser can interpret then as integers.

```
\begin{array}{lll} \mbox{type} \ t &= \mbox{ private} \\ | \ \textit{Digit} \ \mbox{of} \ int \\ | \ \textit{Token} \ \mbox{of} \ string \\ | \ \textit{Scripted} \ \mbox{of} \ scripted \\ | \ \textit{List} \ \mbox{of} \ t \ \textit{list} \end{array}
```

TODO: investigate if it is possible to introduce stem as a separate type to allow more fine-grained compile-time checks

In addition to super- and subscripts, there are prefixes such as \bar, \hat, etc.

```
and scripted = private { stem : t; prefix : prefix list; super : t list; sub : t list } and prefix = | Bar \mid Hat \mid Tilde | Dagger \mid Star | Prime val prefix\_of\_string : string <math>\rightarrow prefix val prefix\_to\_string : prefix <math>\rightarrow string
```

Smart constructors that avoid redundant nestings of lists and scripted tokens with empty scripts.

```
\begin{array}{lll} \mathsf{val} \ digit \ : \ int \to \ t \\ \mathsf{val} \ token \ : \ string \to \ t \\ \mathsf{val} \ scripted \ : \ string \ list \to \ t \ \to \ t \ option \times t \ option \to \ t \\ \mathsf{val} \ list \ : \ t \ list \to \ t \end{array}
```

If it's Scripted, return unchanged, else as a scripted token with empty prefix, super- and subscripts.

```
val\ wrap\_scripted: t \rightarrow scripted
```

If it's a List, return the list itself, otherwise a singleton list.

```
val wrap\_list : t \rightarrow t \ list
```

Recursively strip all prefixes, super- and subscripts and return only the LAST token in a list. I.e. stem "\\bar\\psi_i" and stem "\\bar{\\psi}'" both yield "\\psi".

```
\mathsf{val}\ stem\ :\ t\ \to\ t
```

Unparse the abstract syntax. Since the smart constructors perform some normalization and minimize nested braces, the result is not guaranteed to be identical to the string that has been parsed, just equivalent.

```
\begin{array}{l} \text{val } to\_string \ : \ t \ \rightarrow \ string \\ \text{val } scripted\_to\_string \ : \ scripted \ \rightarrow \ string \\ \text{val } list\_to\_string \ : \ t \ list \ \rightarrow \ string \\ \text{val } compare \ : \ t \ \rightarrow \ t \ \rightarrow \ int \\ \text{end} \end{array}
```

Expressions

A straightforward type for recursive expressions. Note that values (a. k. a. variables) are represented as functions with an empty argument list.

```
\mathsf{module}\ \mathit{Expr}\ :
```

```
\begin{array}{llll} & \text{type } t = & & & & \\ & | \ \textit{Integer} \ \text{of } \ \textit{int} \\ & | \ \textit{Sum of } t \ \textit{list} \ | \ \textit{Diff of } t \ \times \ t \\ & | \ \textit{Product of } t \ \textit{list} \ | \ \textit{Ratio of } t \ \times \ t \\ & | \ \textit{Function of } \ \textit{Token.} t \ \times \ t \ \textit{list} \\ & \text{val } \ \textit{integer} \ : \ \textit{int} \ \rightarrow \ t \\ & \text{val } \ \textit{add} \ : \ t \ \rightarrow \ t \ \rightarrow \ t \\ & \text{val } \ \textit{sub} \ : \ t \ \rightarrow \ t \ \rightarrow \ t \\ & \text{val } \ \textit{mult} \ : \ t \ \rightarrow \ t \ \rightarrow \ t \\ & \text{val } \ \textit{div} \ : \ t \ \rightarrow \ t \ \rightarrow \ t \\ & \text{val } \ \textit{apply} \ : \ \textit{Token.} t \ \rightarrow \ t \ \textit{list} \ \rightarrow \ t \\ & \text{val } \ \textit{to\_string} \ : \ t \ \rightarrow \ \textit{string} \\ & \text{end} \\ \end{array}
```

Particle Declarations

module Particle:

sig

Neutral particles are known by a single name, charged particles also by the name of the anti-particle, ...

```
\begin{array}{ll} \text{type } name &= \\ \mid \ Neutral \ \text{of} \ Token.t \\ \mid \ Charged \ \text{of} \ Token.t \ \times \ Token.t \end{array}
```

 \dots and a list of attributes: aliases, external representations for LATEX and Fortran, quantum numbers and symbols for mass and width.

```
\mathsf{type}\ t\ =
        \{ name : name;
           attr : attr list }
Unparsing:
     \mathsf{val}\ to\_string\ :\ t\ \to\ string
  end
                                                    Parameter Declarations
module Parameter :
  sig
     \mathsf{type}\ \mathit{attr}\ =
       TeX of Token.t list
       Alias of Token.t list
     | Fortran of Token.t list
     type t' =
        \{ name : Token.t; 
          value : Expr.t;
           attr : attr list
     \mathsf{type}\ t\ =
     | Parameter of t'
     \mid Derived \text{ of } t'
     val to\_string : t \rightarrow string
  end
                                                   Lie Groups and Algebras
\mathsf{module}\ \mathit{Lie}\ :
  sig
The full list SU of int \mid U of int \mid SO of int \mid O of int \mid Sp of int \mid E6 \mid E7 \mid E8 \mid F4 \mid G2 is not
realistic. In practice, we will concentrate on SU(3) for now.
     type group
     val\ default\_group : group (* SU(3), of course *)
     val\ group\_of\_string : string 	o group
     val\ group\_to\_string : group \rightarrow string
For now, we only support the \mathbf{3}, \mathbf{\bar{3}} and \mathbf{8} of SU(3).
     type rep
     \mathsf{val}\ rep\_of\_string\ :\ group\ \to\ string\ \to\ rep
     val\ rep\_to\_string : rep \rightarrow string
     \mathsf{type}\ t\ =\ group\ \times\ rep
  end
                                                   Lorentz\ Representations
module\ Lorentz:
  sig
     type rep =
     | \ Scalar \ | \ Vector
       Dirac | ConjDirac | Majorana
       Weyl | ConjWeyl
  end
```

Indices

```
module\ Index:
  sig
     type attr =
       Color of Token.t list \times Token.t list
       Flavor of Token.t list \times Token.t list
      Lorentz of Token.t list
    type t =
       \{ name : Token.t;
          attr : attr list }
    val to\_string : t \rightarrow string
  end
                                                         Tensors
module \ Tensor :
  sig
     type attr =
       Color of Token.t list 	imes Token.t list
       Flavor of Token.t list \times Token.t list
     | Lorentz of Token.t list
    type t =
       \{ name : Token.t; 
          attr : attr list }
    val to\_string : t \rightarrow string
  end
```

Files

The abstract representation of a file, immediately after lexical and syntactical analysis and before any type checking or semantic analysis, is a list of declarations.

There is one version with unexpanded \include statements.

```
module File\_Tree :
  sig
     type \ declaration =
       Particle of Particle.t
       Parameter of Parameter.t
       Index of Index.t
       Tensor of Tensor.t
       Vertex 	ext{ of } Expr.t 	imes Token.t
       Include of string
    type t = declaration list
    \mathsf{val}\ empty\ :\ t
  end
A linear file, just like File_Tree, but with all the \include statements expanded.
\mathsf{module}\ \mathit{File}\ :
  sig
     type declaration =
       Particle\ of\ Particle.t
       Parameter of Parameter.t
      Index of Index.t
```

```
| Tensor of Tensor.t
| Vertex of Expr.t \times Token.t

type t = declaration \ list
| val empty : t
| expand\_includes parser file\_tree recursively expands all include statemens in file\_tree, using parser to map a filename to a File\_Tree.t.
```

```
\mbox{val } expand\_includes \ : \ (string \to File\_Tree.t) \to File\_Tree.t \to t \mbox{val } to\_strings \ : \ t \to string \ list end
```

13.2 Implementation of Vertex_syntax

Avoid referring to *Pervasives.compare*, because *Pervasives* will become *Stdlib.Pervasives* in O'Caml 4.07 and *Stdlib* in O'Caml 4.08.

 $\mathsf{let}\ pcompare\ =\ compare$

13.2.1 Abstract Syntax

```
exception Syntax\_Error of string \times Lexing.position \times Lexing.position
module Token =
  struct
     type t =
       Digit of int
       Token of string
       Scripted of scripted
     | List of t list
     and scripted =
       \{ stem : t; 
          \textit{prefix} \; : \; \textit{prefix list};
          super: t list;
          sub : t list 
     and prefix =
       Bar | Hat | Tilde
       Dagger | Star
       Prime
     let prefix\_of\_string = function
          "\\bar" | "\\overline" 
ightarrow Bar
          "\\hat" | "\\widehat" \rightarrow Hat
          "\\tilde" | "\\widetilde" 
ightarrow Tilde
          \verb"\dagger"} \to \textit{Dagger}
          "*" | "\\ast" 	o Star
          \verb"\prime" \to \mathit{Prime}
          \_ \rightarrow invalid\_arg "Vertex_Syntax.Token.string_to_prefix"
     let prefix\_to\_string = function
         Bar \rightarrow "\\bar"
          Hat \rightarrow "\\hat"
          Tilde \rightarrow "\\tilde"
          Dagger \rightarrow " \dagger"
          Star \rightarrow "*"
          Prime \rightarrow "\prime"
     let wrap\_scripted = function
         Scripted\ st\ 	o\ st
       t \rightarrow \{ stem = t; prefix = []; super = []; sub = [] \}
```

```
let wrap\_list = function
   | List tl \rightarrow tl
  | as t \rightarrow [t]
let digit i =
  if i~\geq~0~\wedge~i~\leq~9 then
     Digit i
  else
     invalid_arg ("Vertex_Syntax.Token.digit:⊔" ^ string_of_int i)
\mathsf{let}\ token\ s\ =
   Token s
let list = function
   [] \rightarrow List[]
     [Scripted {stem = t; prefix = []; super = []; sub = []}] \rightarrow t
     [t] \rightarrow t
    tl \rightarrow List \ tl
let optional = function
    None \rightarrow []
    Some \ t \rightarrow wrap\_list \ t
let scripted prefix token (super, sub) =
   match token, prefix, super, sub with
     \_, [], None, None \rightarrow token
   | \ (\textit{Digit} \ \_ \ | \ \textit{Token} \ \_ \ | \ \textit{List} \ \_) \ \mathsf{as} \ t, \ \_, \ \_, \ \_ \ \to
     Scripted \{ stem = t; 
                     prefix = List.map\ prefix\_of\_string\ prefix;
                     super = optional super;
                     sub = optional sub 
  \mid Scripted st, \_, \_, \_ \rightarrow
     Scripted \{ stem = st.stem; 
                     prefix = List.map prefix_of_string prefix @ st.prefix;
                     super = st.super @ optional super;
                     sub = st.sub @ optional sub \}
let rec stem = function
     Digit \_ \mid Token \_ as t \rightarrow t
     Scripted \{ stem = t \} \rightarrow stem t
     List \ tl \rightarrow
     begin match List.rev tl with
     | [] \rightarrow List[]
     \mid t :: \_ \rightarrow stem t
```

Strip superfluous List and Scripted constructors.

NB: This might be unnecessary, if we used smart constructors.

```
let rec strip = function
    Digit \_ \mid Token \_ as t \rightarrow t
    Scripted { stem = t; prefix = []; super = []; sub = [] } \rightarrow strip t
    Scripted { stem = t; prefix = prefix; super = super; sub = sub } \rightarrow
     Scripted \{ stem = strip t; 
                   prefix = prefix;
                   super = List.map \ strip \ super;
                   sub = List.map \ strip \ sub \}
  | List \ tl \rightarrow
     begin match List.map\ strip\ tl with
     | [] \rightarrow List[]
     | [t] \rightarrow t
     | tl \rightarrow List tl
```

Recursively merge nested List and Scripted constructors.

NB: This might be unnecessary, if we used smart constructors.

```
let rec flatten = function
   | Digit \_ | Token \_ as t \rightarrow t
     List\ tl\ 	o\ flatten\_list\ tl
     Scripted\ st\ 	o\ flatten\_scripted\ st
and flatten\_list\ tl\ =
  match List.map flatten tl with
   [] \rightarrow List[]
    [t] \rightarrow t
  |tl \rightarrow List tl
and flatten\_scripted = function
    \{ stem = t; prefix = []; super = []; sub = [] \} \rightarrow t
     \{ stem = t; prefix = prefix; super = super; sub = sub \} \rightarrow
     \mathsf{let}\ super\ =\ List.map\ \mathit{flatten}\ super
     and sub = List.map \ flatten \ sub in
     begin match flatten t with
     | Digit \_ | Token \_ | List \_ as t \rightarrow
        Scripted \{ stem = t; 
                       prefix = prefix;
                       super = super;
                       sub = sub  }
     \mid Scripted \ st \rightarrow
        Scripted \{ stem = st.stem; 
                       prefix = prefix @ st.prefix;
                       super = st.super @ super;
                       sub = st.sub @ sub }
     end
let \ ascii\_A = Char.code 'A'
let ascii_Z = Char.code 'Z'
let \ ascii_a = Char.code 'a'
let \ ascii_z = Char.code 'z'
let is\_char c =
  let a = Char.code c in
  (ascii\_A \leq a \land a \leq ascii\_Z) \lor (ascii\_a \leq a \land a \leq ascii\_z)
let is\_backslash c =
   c = ' \ ' \ '
let first\_char s =
  s.[0]
let last\_char s =
  s.[String.length \ s - 1]
let rec to_string = function
    Digit \ i \rightarrow string\_of\_int \ i
     Token \ s \rightarrow s
     Scripted\ t \rightarrow scripted\_to\_string\ t
     List\ tl \rightarrow "\{" \cap list\_to\_string\ tl \cap "\}"
and list\_to\_string = function
  | [] \rightarrow ""
     [Scripted \ \{ \ stem \ = \ t; \ super \ = \ []; \ sub \ = \ [] \ \}] \ \rightarrow \ to\_string \ t
     [Scripted \_ as t] \rightarrow "{" ^to\_string t ^ "}"
   [t] \rightarrow to\_string t
  |tl \rightarrow "{" ^concat\_tokens tl ^ "}"
and scripted\_to\_string t =
  let super =
     match t.super with
     | [] \rightarrow ""
     \mid tl \rightarrow \text{"`"} \hat{\ } list\_to\_string \ tl
  and sub =
```

```
match t.sub with
          | [] \rightarrow ""
          \mid tl \rightarrow "_" \hat{} list\_to\_string \ tl in
        String.concat \verb""" (List.map prefix\_to\_string t.prefix) ^ \\
          to_string t.stem ^ super ^ sub
     and required\_space\ t1\ t2\ =
        let required\_space' s1 s2 =
          if is\_backslash (first\_char s2) then
             else if is\_backslash (first\_char\ s1) \land\ is\_char\ (last\_char\ s1) then
             [Token " "]
          else
             [] in
        match t1, t2 with
          Token s1, Token s2 \rightarrow required_space' s1 s2
          Scripted s1, Token s2 \rightarrow required_space' (scripted_to_string s1) s2
          Token s1, Scripted s2 \rightarrow required_space' s1 (scripted_to_string s2)
          Scripted s1, Scripted s2 \rightarrow
          required_space' (scripted_to_string s1) (scripted_to_string s2)
        | List_{-}, - | -, List_{-} | -, Digit_{-} | Digit_{-}, - \rightarrow []
     and interleave\_spaces \ tl =
        Tho List.interleave\_nearest\ required\_space\ tl
     and concat\_tokens\ tl\ =
        String.concat "" (List.map to_string (interleave_spaces tl))
     \mathsf{let}\ compare\ t1\ t2\ =
        pcompare t1 t2
  end
module Expr =
  struct
     type t =
      Integer of int
       Sum 	ext{ of } t 	ext{ } list 	ext{ } l 	ext{ } Diff 	ext{ of } t 	ext{ } 	ext{ } t
       Product of t list | Ratio of t \times t
      | Function of Token.t \times t list
     let integer i = Integer i
     let rec add \ a \ b =
        match a, b with
          Integer a, Integer b \rightarrow Integer (a + b)
          Sum \ a, \ Sum \ b \rightarrow Sum \ (a @ b)
          Sum \ a, \ b \rightarrow Sum \ (a @ [b])
          a, Sum b \rightarrow Sum (a :: b)
        | a, b \rightarrow Sum([a; b])
(a1 - a2) - (b1 - b2) = (a1 + b2) - (a2 + b1)
(a1 - a2) - b = a1 - (a2 + b)
a - (b1 - b2) = (a + b2) - b1
     and sub \ a \ b =
        match a, b with
          Integer a, Integer b \rightarrow Integer (a - b)
          Diff(a1, a2), Diff(b1, b2) \rightarrow Diff(add a1 b2, add a2 b1)
          Diff(a1, a2), b \rightarrow Diff(a1, add a2 b)
          a, Diff (b1, b2) \rightarrow Diff (add a b2, b1)
          a, b \rightarrow Diff(a, b)
     and mult \ a \ b \ =
        match a, b with
        | Integer a, Integer b \rightarrow Integer (a \times b)
```

```
Product \ a, \ Product \ b \rightarrow Product \ (a @ b)
          Product \ a, \ b \rightarrow Product \ (a @ [b])
          a, Product b \rightarrow Product (a :: b)
          a, b \rightarrow Product([a; b])
     and div \ a \ b =
        match a, b with
          Ratio (a1, a2), Ratio (b1, b2) \rightarrow Ratio (mult a1 b2, mult a2 b1)
          Ratio (a1, a2), b \rightarrow Ratio (a1, mult a2 b)
          a, Ratio (b1, b2) \rightarrow Ratio (mult a b2, b1)
        | a, b \rightarrow Ratio(a, b)|
     let apply f args =
        Function (f, args)
     let rec to\_string = function
          Integer i \rightarrow string\_of\_int i
          Sum \ ts \rightarrow String.concat "+" (List.map to_string ts)
           Diff(t1, t2) \rightarrow to\_string t1 ^ "-" ^ to\_string t2
          Product \ ts \rightarrow String.concat "*" (List.map to_string ts)
          Ratio (t1, t2) \rightarrow to_string t1 ^{"}" ^{"} to_string t2
          Function (f, args) \rightarrow
           Token.to\_string f
             String.concat ""
             (List.map (fun arg \rightarrow "\{" \hat to\_string arg \hat "\}") args)
  end
module Particle =
  struct
     type name =
      | Neutral of Token.t
     | Charged of Token.t \times Token.t
     type attr =
       TeX of Token.t list | TeX_Anti of Token.t list
       Alias 	ext{ of } Token.t \ list \mid \ Alias\_Anti 	ext{ of } Token.t \ list
       Fortran\  \, 	ext{of}\  \, Token.t\  \, list\  \, |\  \, Fortran\_Anti\  \, 	ext{of}\  \, Token.t\  \, list
        Spin 	ext{ of } Expr.t \mid Charge 	ext{ of } Expr.t
        Color of Token.t list \times Token.t list
      | Mass of Token.t list | Width of Token.t list
     type t =
        \{ name : name;
           attr : attr list }
     let name\_to\_string = function
        \mid Neutral \ p \rightarrow
            "\\ \ " \ Token.to\_string \ p \ ^ " \\ "
          Charged (p, ap) \rightarrow
           "\\charged{" ^ Token.to_string p ^ "}{" ^ Token.to_string ap ^ "}"
     let attr\_to\_string = function
           TeX \ tl \rightarrow " \setminus tex{" ^ Token.list_to_string \ tl ^ "}"
           TeX\_Anti\ tl\ 	o\ "\arti\tex{"^ } Token.list\_to\_string\ tl^ "}"
           Alias tl \rightarrow \text{"}\alias{"^ Token.list\_to\_string } tl^"}"
           Alias\_Anti\ tl \rightarrow "\alias{" ^ Token.list\_to\_string\ tl ^ "}"
           Fortran\ tl \rightarrow " \setminus fortran \{ " \cap Token.list\_to\_string\ tl \cap " \} "
          Fortran\_Anti\ tl \rightarrow "\anti\fortran{" ^ Token.list\_to\_string\ tl ^ "}"
          Spin \ e \rightarrow "\spin{" ^ Expr. to_string e ^ "}"
          Color([], rep) \rightarrow "\color{" ^ Token.list_to_string rep ^ "}"
        | Color (group, rep) \rightarrow |
            "\\color[" ^ Token.list_to_string group ^ "]{" ^
        \begin{tabular}{lll} Token.list\_to\_string & rep & """ \\ Charge & e & -- " \charge & Expr.to\_string & e & """ \\ \end{tabular}
```

```
Mass\ tl\ 	o\ "\mbox{\mass}{"\ ^ \ } Token.list\_to\_string\ tl\ ^ \ "}"
         Width\ tl\ 	o\ "\width{"\ ^\ }" \ Token.list\_to\_string\ tl\ ^\ "}"
    let to\_string p =
       name_to_string p.name ^
          String.concat "" (List.map attr_to_string (List.sort compare p.attr))
  end
module Parameter =
  struct
    type attr =
      TeX of Token.t list
       Alias of Token.t list
     | Fortran of Token.t list
    type t' =
       \{ name : Token.t;
          value : Expr.t;
          attr: attr list
     type t =
     | Parameter of t'
     \mid Derived \text{ of } t'
    let attr\_to\_string = function
         TeX \ tl \rightarrow \text{"} \text{tex} \{\text{"`Token.list\_to\_string } tl \text{`"} \}\text{"}
         Alias\ tl\ 	o\ "\alias{" ^ Token.list_to\_string\ tl\ ^ "}"
       | Fortran tl → "\\fortran{" ^ Token.list_to_string tl ^ "}"
    let to\_string' p =
        "{" ^ Token.to_string p.name ^ "}{" ^ Expr.to_string p.value ^ "}" ^
          String.concat "" (List.map attr_to_string p.attr)
    let to\_string = function
        ig| Parameter \ p \ 	o \ "ackstring' \ p
        | Derived p \rightarrow "\\derived" \hat{} to\_string' p
  end
\mathsf{module}\ \mathit{Lie}\ =
  struct
     type group =
      SU of int \mid U of int
       SO of int \mid O of int
       Sp of int
     | E6 | E7 | E8 | F4 | G2
    module T = Token
    let default\_group = SU 3
    let invalid\_group s =
        invalid\_arg ("Vertex.Lie.group_of_string:_{\sqcup}" \hat{\ } s)
    let series s name n =
       match name, n with
          "SU", n when n > 1 \rightarrow SU n
          "U", n when n \geq 1 \rightarrow U n
         "SO", n when n > 1 \rightarrow SO n
         "O", n when n \geq 1 \rightarrow O n
          "Sp", n when n \geq 2 \rightarrow Sp \ n
         \_ \rightarrow invalid\_group s
     let exceptional\ s\ name\ n\ =
       match name, n with
         "E", 6 \rightarrow E6
         "E", 7 \rightarrow E7
```

```
"E", 8 \rightarrow E8
           "F", 4 \rightarrow F4
           "G", 2 \rightarrow G2
           \_ \rightarrow invalid\_group \ s
     let group\_of\_string \ s =
        try
           Scanf.sscanf \ s \ "\%_[{]\%[SUOp](\%d)\%_[}]\%! " (series \ s)
        with
        |  \rightarrow
            try
               Scanf.sscanf\ s\ \verb"%_[{]\%[EFG]_%d\%_[}] \%! \verb" (exceptional\ s)
            with
             \mid \_ \rightarrow invalid\_group \ s
     let \ group\_to\_string \ = \ function
          SU \ n \rightarrow "SU(" \hat string\_of\_int n \hat ")"
           U n \rightarrow "U(" \hat{string} - of \_int n \hat{"})"
           SO \ n \rightarrow "SO(" \hat string\_of\_int n \hat ")"
           O\ n\ 	o\ "O("\ \hat{\ }\ string\_of\_int\ n\ \hat{\ }\ ")"
           Sp \ n \rightarrow "Sp(" \hat{\ } string\_of\_int \ n \hat{\ } ")"
           E6 \rightarrow "E6"
           E7 \rightarrow "E7"
           E8 \rightarrow "E8"
           F4 \rightarrow "F4"
           \it G2 \rightarrow "G2"
     type rep = int
     let rep_of_string group rep =
        match group with
        \mid SU   3 \rightarrow
            begin
               match rep with
                 "3" \rightarrow 3
                  \verb"\bar_$\sqcup 3" \to \ -3
                  "8" \rightarrow 8
                    invalid_arg ("Vertex.Lie.rep_of_string:" ^
                                         "_unsupported_representation_" ^{\hat{}} rep ^{\hat{}}
                                         " lof log " \cap group\_to\_string group)
        | _ → invalid_arg ("Vertex.Lie.rep_of_string:" ^
                                        "unsupported_group_" ^ group_to_string group)
     let rep\_to\_string r =
        string\_of\_int r
     \mathsf{type}\ t\ =\ group\ \times\ rep
  end
module Lorentz =
  struct
     type rep =
       Scalar | Vector
        Dirac \mid ConjDirac \mid Majorana
       Weyl \mid ConjWeyl
  end
module Index =
  struct
     type \ attr =
     | Color of Token.t list \times Token.t list
```

```
Flavor of Token.t list \times Token.t list
      Lorentz of Token.t list
    \mathsf{type}\ t\ =
       \{ name : Token.t; 
          attr : attr list }
    let attr\_to\_string = function
         Color([], rep) \rightarrow "\color{" ^ Token.list_to_string rep ^ "}"
       \mid Color (group, rep) \rightarrow
           "\\color[" ^ Token.list_to_string group ^ "]{" ^
              Token.list\_to\_string\ rep\ ^ "}"
         Flavor([], rep) \rightarrow "\flavor(" ^ Token.list_to_string rep ^ "}"
       \mid Flavor (group, rep) \rightarrow
           \verb|"\flavor[" ^ Token.list_to_string group ^ "]{|" ^ Token.list_to_string group ^ "]}|
              Token.list_to_string rep ^ "}"
       | Lorentz\ tl\ 	o "\\lorentz{" ^ Token.list\_to\_string\ tl\ ^ "}"
    let to\_string i =
       "\\index{" ^ Token.to_string i.name ^ "}" ^
          String.concat "" (List.map attr_to_string i.attr)
  end
module Tensor =
  struct
     type attr =
      Color of Token.t list \times Token.t list
       Flavor of Token.t list \times Token.t list
     | Lorentz of Token.t list
    type t =
       \{ name : Token.t; 
          attr : attr list }
    let attr\_to\_string = function
         Color([], rep) \rightarrow "\color{" ^ Token.list_to_string rep ^ "}"
       \mid Color (group, rep) \rightarrow
           "\\color[" ^ Token.list_to_string group ^ "]{" ^
              Token.list_to_string rep ^ "}"
         Flavor([], rep) \rightarrow "\flavor{" ^ Token.list_to_string rep ^ "}"
       | Flavor (group, rep) \rightarrow |
           "\\flavor[" ^ Token.list_to_string group ^ "]{" ^
              Token.list_to_string rep ^ "}"
       | Lorentz\ tl\ 	o "\\lorentz{" ^ Token.list\_to\_string\ tl\ ^ "}"
    let to\_string t =
       "\\tensor{" ^ Token.to\_string\ t.name\ ^ "}" ^ 
          String.concat "" (List.map attr_to_string t.attr)
  end
module File\_Tree =
  struct
     type \ declaration =
      Particle of Particle.t
       Parameter of Parameter.t
       Index of Index.t
       Tensor of Tensor.t
       Vertex 	ext{ of } Expr.t 	imes Token.t
     | Include of string
     type t = declaration list
    let empty = []
  end
```

_ Lexer

```
module \ File =
  struct
     type \ declaration =
       Particle of Particle.t
       Parameter of Parameter.t
       Index of Index.t
       Tensor of Tensor.t
       Vertex 	ext{ of } Expr.t 	imes Token.t
     type t = declaration list
    let empty = []
We allow to include a file more than once, but we don't optimize by memoization, because we assume that this
will be rare. However to avoid infinite loops when including a child, we make sure that it has not yet been
included as a parent.
    let expand\_includes parser unexpanded =
       let rec expand_includes' parents unexpanded expanded =
          List.fold\_right (fun decl decls \rightarrow
            match decl with
              File\_Tree.Particle\ p\ 	o\ Particle\ p\ ::\ decls
              File\_Tree.Parameter p \rightarrow Parameter p :: decls
              File\_Tree.Index\ i\ \to\ Index\ i\ ::\ decls
              File\_Tree.Tensor\ t\ 	o\ Tensor\ t\ ::\ decls
              File\_Tree.Vertex\ (e,\ v) \rightarrow Vertex\ (e,\ v) :: decls
              File\_Tree.Include\ f\ \rightarrow
                if List.mem\ f\ parents then
                  invalid\_arg \; ("cyclic_{\sqcup} \land include{" ^ f ^ "}")
                   expand\_includes' (f :: parents) (parser f) decls)
            unexpanded expanded in
       expand_includes' [] unexpanded []
    let to\_strings \ decls =
       List.map
          (function
            Particle \ p \rightarrow Particle.to\_string \ p
            Parameter p \rightarrow Parameter.to\_string p
            Index \ i \ \rightarrow \ Index.to\_string \ i
            Tensor \ t \rightarrow Tensor.to\_string \ t
            Vertex\ (Expr.Integer\ 1,\ t)\ \rightarrow
            "\vertex{" ^ Token.to\_string t ^ "}"
            Vertex (e, t) \rightarrow
             "\\vertex[" ^ Expr.to_string e ^ "]{" ^
               Token.to\_string\ t\ ^"""")
          decls
  end
                                                    13.3 Lexer
open Lexing
open Vertex\_parser
let string\_of\_char c =
  String.make\ 1\ c
let int\_of\_char c =
  int_of_string (string_of_char c)
let \ init\_position \ fname \ lexbuf \ =
```

 $let \ curr_p \ = \ lexbuf.lex_curr_p \ in$

```
lexbuf.lex\_curr\_p \leftarrow
    \{ curr_p \text{ with }
      pos\_fname = fname;
      pos\_lnum = 1;
      pos\_bol = curr\_p.pos\_cnum };
  lexbuf
}
let digit = [,0,-,9,]
let upper = ['A'-'Z']
let \ lower = ['a'-'z']
let char = upper \mid lower
let white = [', ', '\t']
let pfx = '\'
let env_arg0 = "align" | "center" | "omftable"
let env\_arg1 = "tabular"
rule \ token = parse
    white { token lexbuf } (* skip blanks *)
    '%' [^'\n']* { token\ lexbuf } (* skip comments *)
    '\n' { new_line lexbuf; token lexbuf }
   '\\'([','';'] | 'q'? "quad")
                        \{\ token\ lexbuf\ \}\ (*\ skip\ LaTeX\ white\ space\ *)
    "\\endinput" { token lexbuf } (* continue reading *)
    '\\' ( "chapter" | "sub"* "section" ) '*'? '{' [^'}']* '}'
                        { token lexbuf } (* skip sectioning FIXME!!! *)
    '\\' ( "begin" | "end" ) '{' env_arg0 '*'? '}'
    "\begin" '{' env_arg1 '*'? '}' '{' [^'}']* '}'
    "\\end" '{' env_arg1 '*'? '}'
                        { token lexbuf } (* skip environment delimiters *)
    "\\\" { token lexbuf } (* skip table line breaks *)
    '&' { token lexbuf } (* skip tabulators *)
    '\\' ( "left" | "right" | ['B''b'] "ig" 'g'? ['l''r'] )
                        { token lexbuf } (* skip parenthesis hints *)
    '=' \{ EQUAL \}
    ', ', { SUPER }
    '_' { SUB }
    '\'' { PRIME }
    '\\' ( "bar" | "overline" | "wide"? "hat" | "wide"? "tilde") as pfx
                        \{ PREFIX pfx \}
    '*' { TIMES }
    ',' { DIV }
    '+' { PLUS }
    '-' { MINUS }
    ',' { COMMA }
    '(' { LPAREN }
    ')' { RPAREN }
    '{' { LBRACE }
    '}' { RBRACE }
    '[' \{ LBRACKET \}]
    '] ' { RBRACKET }
   pfx "include{" ([^',}']+ as name) "}"
                       { INCLUDE name }
   pfx "charged" { CHARGED }
   pfx "neutral" { NEUTRAL }
   pfx "anti" { ANTI }
   pfx "tex" { TEX }
   pfx "fortran" { FORTRAN }
   pfx "alias" { \grave{A}LIAS }
   pfx "spin" { SPIN }
  \mid pfx "color" \{ COLOR \}
```

```
pfx "charge" { CHARGE }
 pfx "mass" { MASS }
 pfx "width" { WIDTH }
 \begin{array}{c} pfx \; \texttt{"vertex"} \; \{ \; \; VERTEX \; \} \\ pfx \; \texttt{"index"} \; \{ \; \; INDEX \; \} \end{array}
 pfx "tensor" { TENSOR }
 pfx "lorentz" { LORENTZ }
 pfx "flavor" { FLAVOR }
 pfx "parameter" { PARAMETER }
 pfx "derived" { DERIVED }
 digit as i \in DIGIT (int\_of\_char i) \}
  char as c { CHAR (string_of_char c) }
\mid (') \setminus (- \mid char^+) as s
                          \{ TOKEN s \}
\bot as c { failwith ("invalid_character_at_'" ^{\circ}
                                            string_of_char c ^ "',") }
\mid eof \{END\}
```

13.4 Parser

Right recursion is more convenient for constructing the value. Since the lists will always be short, there is no performace or stack size reason for prefering left recursion.

Header

Token declarations

```
\begin{tabular}{ll} \% token &< $int > DIGIT \\ \% token &< $string > PREFIX TOKEN \\ \% token &< $SUPER SUB PRIME LBRACE RBRACE LBRACKET RBRACKET \\ \% token & $LPAREN RPAREN \\ \% token & $COMMA \\ \% token & $COMMA \\ \% token & $PLUS MINUS TIMES DIV EQUAL \\ \% token &< $string > INCLUDE \\ \% token & $END \\ \% token & $NEUTRAL CHARGED \\ \% token & $ANTI ALIAS TEX FORTRAN SPIN COLOR CHARGE MASS WIDTH \\ \% token & $PARAMETER DERIVED \\ \% token & $TENSOR INDEX FLAVOR LORENTZ \\ \% token & $VERTEX \\ \end{tabular}
```

```
%left PLUS MINUS
%nonassoc NEG UPLUS
%left TIMES DIV
%start file
%type < Vertex_syntax.File_Tree.t > file
                                             Grammar rules
file ::=
 | declarations END { $1 }
declarations ::=
 |\{[]\}
 | declaration declarations { $1 :: $2 }
declaration ::=
  particle { F.Particle $1 }
  parameter { F.Parameter $1 }
  index \{ F.Index \$1 \}
  tensor \{ F.Tensor \$1 \}
 | vertex \{ let e, t = \$1 in \}
                          F.Vertex(e, t)
 | INCLUDE \{ F.Include \$1 \}
particle ::=
 | NEUTRAL token_arg particle_attributes
     \{ \{ P.name = P.Neutral \$2; P.attr = \$3 \} \}
 | CHARGED token_arg_pair particle_attributes
     \{ \text{ let } p, \ ap = \$2 \text{ in } \}
       \{P.name = P.Charged(p, ap); P.attr = \$3\}\}
expr\_arg ::=
 | LBRACKET expr RBRACKET { $2 }
  LBRACKET expr RBRACE { parse_error "expected_']',_found_'}'" }
 | LBRACKET expr END { parse_error "missing_';" }
token\_arg ::=
 | LBRACE scripted_token RBRACE { $2 }
| LBRACE scripted_token END { parse_error "missing_\'\'}'" }
token\_arg\_pair ::=
\mid token\_arg \ token\_arg \ \{ \ (\$1, \ \$2) \ \}
token\_list\_arg ::=
 | LBRACE token_list RBRACE { $2 }
| LBRACE token_list END { parse_error "missing_\'\'}'" }
/* This results in a reduce/reduce conflict:
    | LBRACE token_list RBRACKET { parse_error "expected '}', found ']'" } */
token\_list\_opt\_arg ::=
 | LBRACKET token_list RBRACKET { $2 }
| LBRACKET token_list END { parse_error "missing_\'\}'" }
particle\_attributes ::=
```

```
| { [ ] }
 | particle_attribute particle_attributes { $1 :: $2 }
particle\_attribute ::=
  ALIAS \ token\_list\_arg \{ P.Alias \$2 \}
  ANTI ALIAS token_list_arg { P.Alias $3 }
  TEX \ token\_list\_arg \{ P.TeX \$2 \}
  ANTI TEX token_list_arg { P.TeX_Anti $3 }
  FORTRAN token_list_arg { P.Fortran $2 }
  ANTI FORTRAN token_list_arg { P.Fortran_Anti $3 }
  SPIN \ arg \{ P.Spin \$2 \}
  COLOR\ token\_list\_arg\ \{\ P.Color\ ([],\ \$2)\ \}
  COLOR token_list_opt_arg token_list_arg { P.Color ($2, $3) }
  CHARGE arg { P.Charge $2 }
  MASS token_list_arg { P.Mass $2 }
 |WIDTH\ token\_list\_arg\ \{P.Width\ \$2\}
parameter ::=
 | PARAMETER token_arg arg parameter_attributes
     \{V.Parameter \{V.name = \$2; V.value = \$3; V.attr = \$4\}\}
 | DERIVED token_arg arg parameter_attributes
     \{ V.Derived \{ V.name = \$2; V.value = \$3; V.attr = \$4 \} \}
parameter\_attributes ::=
 | { [ ] }
 | parameter_attribute parameter_attributes { $1 :: $2 }
parameter\_attribute ::=
  ALIAS \ token\_list\_arg \{ V.Alias \$2 \}
  TEX token_list_arg { V.TeX $2 }
  FORTRAN token_list_arg { V.Fortran $2 }
  ANTI { invalid_parameter_attr () }
  SPIN { invalid_parameter_attr () }
  COLOR { invalid_parameter_attr () }
  CHARGE { invalid_parameter_attr () }
  MASS { invalid_parameter_attr () }
  WIDTH { invalid_parameter_attr () }
index ::=
 |INDEX token\_arg index\_attributes \{ \{ I.name = \$2; I.attr = \$3 \} \}
index\_attributes ::=
 | { [ ] }
 | index_attribute index_attributes { $1 :: $2 }
index\_attribute ::=
  COLOR\ token\_list\_arg\ \{\ I.Color\ ([\ ],\ \$2)\ \}
  COLOR\ token\_list\_opt\_arg\ token\_list\_arg\ \{\ I.Color\ (\$2,\ \$3)\ \}
  FLAVOR \ token\_list\_arg \{ I.Flavor ([], \$2) \}
  FLAVOR token_list_opt_arg token_list_arg { I.Flavor ($2, $3) }
 | LORENTZ token_list_arg { I.Lorentz $2 }
tensor ::=
 | TENSOR \ token\_arg \ tensor\_attributes \{ \{ X.name = \$2; X.attr = \$3 \} \}
tensor\_attributes ::=
|{[]}
```

```
| tensor_attribute tensor_attributes { $1 :: $2 }
tensor\_attribute ::=
  COLOR\ token\_list\_arg\ \{\ X.Color\ ([\ ],\ \$2)\ \}
  COLOR token_list_opt_arg token_list_arg { X.Color ($2, $3) }
  FLAVOR \ token\_list\_arg \{ X.Flavor ([], \$2) \}
  FLAVOR token_list_opt_arg token_list_arg { X.Flavor ($2, $3) }
 | LORENTZ token_list_arg { X.Lorentz $2 }
vertex ::=
  VERTEX token_list_arg { (E.integer 1, T.list $2) }
  VERTEX\ expr\_arg\ token\_list\_arg\ \{\ (\$2,\ T.list\ \$3)\ \}
  VERTEX expr_arg LBRACE RBRACE { ($2, T.list []) }
  VERTEX expr_arg LBRACE END { parse_error "missing_'; } '" }
 | VERTEX\ not\_arg\_or\_token\_list { parse\_error "expected_'['uor_'('")}
/* This results in a shift/reduce conflict:
    | VERTEX expr_arg LBRACE RBRACKET { parse_error "expected '}', found ']'" } */
expr ::=
  integer { E.integer $1 }
  LPAREN expr RPAREN { $2 }
  LPAREN expr RBRACKET { parse_error "expected_')',_found_']'" }
  LPAREN expr RBRACE { parse_error "expected_')', _found_'}'" }
  LPAREN expr END { parse_error "missing<sub>□</sub>')' }
  expr PLUS expr { E.add $1 $3 }
  expr MINUS expr { E.sub $1 $3 }
  expr TIMES expr { E.mult $1 $3 }
  expr\ DIV\ expr\ \{\ E.div\ \$1\ \$3\ \}
  bare_scripted_token arg_list { E.apply $1 $2 }
/* Making '*' optional introduces many shift/reduce and reduce/reduce conflicts:
    | expr expr { E.mult $1 $2 } */
arg\_list ::=
 | { [] }
 \mid arg \ arg \ list \{ \$1 :: \$2 \}
arg ::=
 | LBRACE expr RBRACE { $2 }
  LBRACE expr RBRACKET { parse_error "expected_'},__found_']'" }
 | LBRACE expr END { parse_error "missing_"; } " }
integer ::=
  DIGIT \{ \$1 \}
 \mid integer\ DIGIT \{\ 10 \times \$1 + \$2 \}
token ::=
  bare\_token { $1 }
  LBRACE scripted_token RBRACE { $2 }
  LBRACE scripted_token END { parse_error "missing_\'\'\'\'}," }
 | LBRACE scripted_token token_list RBRACE { T.list ($2 :: $3) }
 | LBRACE scripted_token token_list END { parse_error "missing_';}'" }
/* This results in a shift/reduce conflict because RBRACKET is a bare token:
    | LBRACE scripted_token RBRACKET
                                               { parse_error "expected '}', found ']'" } */
token\_list ::=
  scripted\_token \{ [\$1] \}
 | scripted_token_token_list { $1 :: $2 }
```

```
scripted\_token ::=
 | prefixes token optional_scripts { T.scripted $1 $2 $3 }
bare\_scripted\_token ::=
 | prefixes name optional_scripts { T.scripted $1 $2 $3 }
optional\_scripts ::=
  \{ (None, None) \}
  super \{ (\$1, None) \}
  sub { (None, $1) }
  super \ sub \ \{ \ (\$1, \ \$2) \ \}
  sub\ super\ \{\ (\$2,\ \$1)\ \}
  primes { ($1, None) }
  primes \ sub \ \{ \ (\$1, \ \$2) \ \}
 \mid sub \ primes \{ (\$2, \$1) \}
super ::=
 | SUPER token { Some $2 }
 |SUPER|RBRACE \{ parse\_error "superscript\_can't\_start\_with\_`\}'" \}
/* This results in many reduce/reduce conflicts:
     | SUPER RBRACKET { parse_error "superscript can't start with ']'" } */
sub ::=
 | SUB \ token \{ Some \$2 \}
 |SUB|RBRACE \{ parse\_error "subscript\_can't\_start\_with\_'\}'" \}
/* This results in many reduce/reduce conflicts:
     | SUB RBRACKET { parse_error "subscript can't start with ']'" } */
prefixes ::=
 |{ []}
 | PREFIX prefixes { $1 :: $2 }
primes ::=
 | prime\_list \{ Some (T.list \$1) \}
prime\_list ::=
  PRIME \{ [T.token "\prime"] \}
 | PRIME prime_list { T.token "\\prime" :: $2 }
name ::=
 \mid CHAR \mid T.token \mid 1 \rangle
 \mid TOKEN \mid T.token \mid 1 
bare\_token ::=
  DIGIT \{ T.digit \$1 \}
  CHAR \{ T.token \$1 \}
  TOKEN \{ T.token \$1 \}
  PLUS \{ T.token "+" \}
  MINUS \{ T.token "-" \}
  TIMES \{ T.token "*" \}
  DIV \{ T.token "/" \}
  COMMA \{ T.token "," \}
  LPAREN { T.token "(" }
 |RPAREN \{ T.token ")" \}
```

```
not_arg_or_token_list ::=

| DIGIT { () }

| CHAR { () }

| TOKEN { () }

| PLUS { () }

| MINUS { () }

| DIV { () }

| COMMA { () }

| RPAREN { () }

| RBRACKET { () }

| RBRACE { () }
```

13.5 Interface of Vertex

```
\begin{array}{lll} \operatorname{val}\;parse\_string\;:\;string\;\rightarrow\;Vertex\_syntax.File.t\\ \operatorname{val}\;parse\_file\;:\;string\;\rightarrow\;Vertex\_syntax.File.t\\ \operatorname{module}\;\operatorname{type}\;Test\;=\;&\\ \operatorname{sig}\;&\\ \operatorname{val}\;example\;:\;unit\;\rightarrow\;unit\\ \operatorname{val}\;suite\;:\;OUnit.test\\ \operatorname{end}\\ \operatorname{module}\;Test\;(M\;:\;Model.T)\;:\;Test\\ \operatorname{module}\;Parser\_Test\;:\;Test\\ \operatorname{module}\;Modelfile\_Test\;:\;Test\\ \end{array}
```

13.6 Implementation of Vertex

Avoid referring to *Pervasives.compare*, because *Pervasives* will become *Stdlib.Pervasives* in O'Caml 4.07 and *Stdlib* in O'Caml 4.08.

```
\begin{array}{lll} \text{let } pcompare &=& compare \\ \\ \text{module type } Test &=& \\ \text{sig} \\ \text{val } example &:& unit \rightarrow unit \\ \text{val } suite &:& OUnit.test \\ \text{end} \end{array}
```

13.6.1 New Implementation: Next Version

```
let error\_in\_string\ text\ start\_pos\ end\_pos\ =
let i=start\_pos.Lexing.pos\_cnum
and j=end\_pos.Lexing.pos\_cnum in
String.sub\ text\ i\ (j-i)
let error\_in\_file\ name\ start\_pos\ end\_pos\ =
Printf.sprintf
"%s:%d.%d-%d.%d"
name
start\_pos.Lexing.pos\_lnum
(start\_pos.Lexing.pos\_cnum\ -\ start\_pos.Lexing.pos\_bol)
end\_pos.Lexing.pos\_lnum
(end\_pos.Lexing.pos\_cnum\ -\ end\_pos.Lexing.pos\_bol)
let parse\_string\ text\ =
Vertex\_syntax.File.expand\_includes
(fun\ file\ \to\ invalid\_arg\ ("parse\_string: \ lound\ linclude\ linclude\
```

```
(try
         Vertex\_parser.file
           Vertex\_lexer.token
           (Vertex_lexer.init_position "" (Lexing.from_string text))
      with
        Vertex\_syntax.Syntax\_Error\ (msg,\ start\_pos,\ end\_pos) \rightarrow
        invalid_arg (Printf.sprintf "syntax_error_(%s)_at:_'%s'"
                          msg\ (error\_in\_string\ text\ start\_pos\ end\_pos))
      \mid Parsing.Parse\_error \rightarrow
        invalid_arg ("parse_error: " ^ text))
let parse\_file name =
  let parse_file_tree name =
    let ic = open\_in \ name \ in
    let file_tree =
       begin try
          Vertex\_parser.file
            Vertex\_lexer.token
            (Vertex_lexer.init_position name (Lexing.from_channel ic))
       with
       Vertex\_syntax.Syntax\_Error\ (msg,\ start\_pos,\ end\_pos) \rightarrow
           begin
            close\_in\ ic;
            invalid_arg (Printf.sprintf
                              "%s: usyntax uerror (%s) "
                              (error_in_file name start_pos end_pos) msg)
         end
       \mid Parsing.Parse\_error \rightarrow
         begin
            close\_in\ ic;
            invalid_arg ("parse_error: " ^ name)
         end
       end in
     close\_in\ ic;
    file_tree in
  Vertex_syntax.File.expand_includes parse_file_tree (parse_file_tree name)
let dump_file\ pfx\ f =
  List.iter
     (\text{fun } s \rightarrow print\_endline (pfx ^ ":_ " ^ s))
    (Vertex\_syntax.File.to\_strings\ f)
module Parser\_Test : Test =
  struct
    let example() =
       ()
    open OUnit
    let compare s_out s_in () =
       assert\_equal \ \tilde{} printer : (String.concat \ "\")
         [s\_out] (Vertex\_syntax.File.to\_strings (parse\_string s_in))
    let parse\_error\ error\ s\ () =
       assert\_raises \ (Invalid\_argument \ error) \ (fun \ () \rightarrow parse\_string \ s)
    let syntax\_error\ (msg,\ error)\ s\ () =
       parse\_error ("syntax\_error\_(" ^ msg ^ ")_{\bot}at:_{\bot}`" ^ error ^ "'") s ()
    let (=>) s_i n s_o ut =
       let (? >) s =
       s => s
    let (=>!!!) s error =
```

```
"\sqcup" \hat{} s >:: parse\_error\ error\ s
\mathsf{let}\ (=>!)\ s\ \mathit{error}\ =
  " \_ " \hat{\ } s \ > :: \ syntax\_error \ error \ s
let empty =
  "empty" >::
    (fun () \rightarrow assert\_equal [] (parse\_string ""))
let expr =
  "expr" >:::
    ["\vertex[2_{\sqcup}*_{\sqcup}(17_{\sqcup}+_{\sqcup}4)]{}" => "\vertex[42]{{}}";
       "\\vertex[2_{\sqcup}*_{\sqcup}17_{\sqcup}+_{\sqcup}4]{}" => "\\vertex[38]{{}}";
       "\\vertex[2" =>! ("missing_\']', "[2");
       $$ "\operatorname{zertex} {} " => ! ("expected_'['_or_'(', "\setminus vertex]"); "\vertex2] {} " => ! ("expected_'['_or_'(', "\setminus vertex2"); "); "); "
       "\vertex){}{"=>! ("expected_'['_or_'(', "\vertex)");}
       "\\vertex2}{}" =>! ("expected_'[', or_'{', "\\vertex2"});
       "\\vertex[(2){}" =>! ("expected_\')', \_found_\'\}', "(2\);
       "\\vertex[(2]{}" =>! ("expected_\')', \_found_\']', "(2]");
       "\\vertex\{2\}" =>! ("syntax_error", "2");
       "\\vertex[2]{}" =>! ("expected_\']', \_found_\'}'\, "[2]\");
       "\\vertex[2{}" =>! ("syntax_error", "2");
       "\\vertex[2*]{}" =>! ("syntax_error", "2")]
let index =
  "index" >:::
    [ "\vertex{{a}_{1}^{2}}" => "\vertex{a^2_1}";
       "\\vertex\{a_{11}^2\}" => "\\vertex\{a^2_{11}\}";
       "\\vertex\{a_{1_1}^2\}" => "\\vertex\{a^2_{1_1}\}" ]
let \ electron1 =
  "electron1" >:::
    [? > "\\charged{e^-}{e^+}";
       \c e^{-} {e^-} {e^-} = \c e^{-} {e^-} e^{-} 
let \ electron2 =
  "electron2" >:::
    [ "\charged{e^-}{e^-}\charged{e^-} => 
       \label{eq:charged} $$ \c - {e^+} \operatorname{cn}{\{ele\}}";
       \label{eq:charged} $$ \charged{e^-}{e^+} \charged{ectron} \charged{ele} " => $$
       "\\charged\{e^-\}\{e^+\}\\\fortran\{\{ele\}\}\\\fortran\{\{electron\}\}";
       \label{eq:charged} $$ \charged{e^-}{e^+}\sim {e^2}\charged{e1}" => $$
       "\\charged{e^-}{e^+}\\alias{{e1}}\\alias{{e2}}";
       \c e^{-}{e^+}\c e^-
       "\\charged{e^-}{e^+}\\fortran{{ele}}\\anti\\fortran{{pos}}" ]
let particles =
  "particles" >:::
    [electron 1;
     electron2]
let parameters =
  "parameters" >:::
    [ ? > "\\parameter{\\alpha}{1/137}";
       ?> "\derived{\alpha_s}{1/\ln{\frac{\mu}{\Lambda}}}";
       \label{lambda} $$ ''\rightarrow {1/137}\hat =>!
       ("invalid_parameter_attribute", "\\anti") ]
\mathsf{let} \ indices \ =
  "indices" >:::
    [? > "\\ index{a}\\ color{8}";
       \lceil SU(2) \rceil = \lceil SU(2) \rceil = \lceil SU(2) \rceil = \rceil
let tensors =
  "tensors" >:::
```

```
[ "\tensor{T}\color{3}" => "\tensor{T}\color{3}"]
   let \ vertices =
     "vertex" >:::
       [ "\vertex{\\hoar\psi\gamma_\mu\psi_A_\mu}" => 
         "\\vertex{{{\\bar\\psi\\gamma_\\mu\\psi_A_\\mu}}}"]
   module T = Vertex\_syntax.Token
   let parse\_token s =
     match parse\_string ("\\vertex{" \hat{s} "}") with
       [\mathit{Vertex\_syntax.File.Vertex}\ (\_,\ v)]\ \to\ v
     | _ → invalid_arg "only_vertex"
   let print\_token pfx t =
     print\_endline\ (pfx ` ": \_ " ` T.to\_string\ t)
   let test\_stem \ s\_out \ s\_in \ () =
     assert_equal ~printer : T.to_string
       (parse\_token s\_out)
       (T.stem (parse\_token s\_in))
   let (=>>) s_i n s_o ut =
     "stem_{\sqcup}" ^ s_{-}in >:: test\_stem\ s_{-}out\ s_{-}in
   let tokens =
     "tokens" >:::
       [ "\\vertex{a'}" => "\\vertex{a^\\prime}";
         "\\vertex{a','}" => "\\vertex{a^{\\prime\\prime}}";
         "\\bar\\psi'',_{i,\\alpha}" =>> "\\psi";
         "\\phi^\\dagger_{i'}" =>> "\\phi";
         "\\bar{\\phi\\psi}''_{i,\\alpha}" =>> "\\psi";
         "\\vertex{\\phi}" => "\\vertex{\\phi}";
         "\vertex{\\rho i_1}" => "\vertex{\\rho i_1}";
         "\\vertex{{{\\phi}'}}" => "\\vertex{\\phi^\\prime}";
         "\\text{\hat}\\hat{\hat}\\n^1}" => "\\text{\hat}\\n^1}";
         "\\vertex{a_b}_{cd}" => "\\vertex{a_{bcd}}";
         "\\vertex{{\\phi_1}_2}" => "\\vertex{\\phi_{12}}";
         "\\vertex{{\phi_{12}}_{34}}" => "\\vertex{\phi_{1234}}";
         "\\vertex{{\phi_{12}}^{34}}" => "\\vertex{\phi_{34}_{12}}";
         let suite =
     "Vertex_Parser" >:::
       [empty;
        index;
        expr;
        particles;
        parameters;
        indices;
        tensors;
        vertices;
        tokens ]
 end
                                       Symbol Tables
module type Symbol =
 sig
```

Tensors and their indices are representations of color, flavor or Lorentz groups. In the end it might turn out to be unnecessary to distinguish *Color* from *Flavor*.

 $type file = Vertex_syntax.File.t$ $type t = Vertex_syntax.Token.t$

```
type \ space =
       Color of Vertex_syntax.Lie.t
       Flavor of t list \times t list
       Lorentz of t list
A symbol (i.e. a Symbol.t = Vertex_syntax. Token.t) can refer either to particles, to parameters (derived and
input) or to tensors and indices.
    type kind =
       Neutral
       Charged
       Anti
       Parameter
       Derived
       Index of space
       Tensor of space
    type table
    val\ load\ :\ file\ 	o\ table
    val\ dump\ :\ out\_channel\ 	o\ table\ 	o\ unit
Look up the kind of a symbol.
    val \ kind\_of\_symbol : table \rightarrow t \rightarrow kind \ option
Look up the kind of a symbol's stem.
    \mathsf{val}\ kind\_of\_stem\ :\ table\ \to\ t\ \to\ kind\ option
Look up the kind of a symbol and fall back to the kind of the symbol's stem, if necessary.
    val\ kind\_of\_symbol\_or\_stem : table \rightarrow t \rightarrow kind\ option
A table to look up all symbols with the same stem.
    val\ common\_stem\ :\ table\ 	o\ t\ 	o\ t\ list
    exception Missing\_Space of t
    exception Conflicting\_Space of t
  end
module Symbol : Symbol =
  struct
    module T = Vertex\_syntax.Token
    module F = Vertex\_syntax.File
    \mathsf{module}\ P\ =\ \mathit{Vertex\_syntax.Particle}
    module I = Vertex\_syntax.Index
    module L = Vertex\_syntax.Lie
    module Q = Vertex\_syntax.Parameter
    module X = Vertex\_syntax.Tensor
    type file = F.t
    type t = T.t
    type \ space =
       Color of L.t
       Flavor of t list \times t list
      Lorentz of t list
    let space\_to\_string = function
       \mid Color(g, r) \rightarrow
           "color:" \hat{\ } L.group\_to\_string g \hat{\ } ":" \hat{\ } L.rep\_to\_string r
         Flavor(_-,_-) \rightarrow "flavor"
       | Lorentz → "Lorentz"
    type kind =
       Neutral
       Charged
      Anti
```

```
Parameter
  Derived
  Index of space
  Tensor of space
let kind\_to\_string = function
    Neutral \rightarrow "neutral_particle"
    Charged \rightarrow "charged_particle"
    Anti \rightarrow "charged_anti_particle"
    Parameter \rightarrow "input_{\sqcup}parameter"
    Derived \rightarrow "derived\_parameter"
    Tensor\ space\ 	o\ space\_to\_string\ space\ ^ "\_tensor"
module ST = Map.Make (T)
module SS = Set.Make (T)
type table =
     \{ symbol\_kinds : kind ST.t; \}
       stem\_kinds: kind ST.t;
       common\_stems : SS.t ST.t }
let empty =
    \{ symbol\_kinds = ST.empty; \}
       stem\_kinds = ST.empty;
       common\_stems = ST.empty }
let kind\_of\_symbol table token =
  try Some (ST.find \ token \ table.symbol\_kinds) with Not\_found \rightarrow None
let kind\_of\_stem table token =
     Some (ST.find (T.stem token) table.stem_kinds)
  with
  | Not\_found \rightarrow None
let kind_of_symbol_or_stem symbol_table token =
  {\sf match}\ \mathit{kind\_of\_symbol\_table}\ \mathit{token}\ {\sf with}
    Some \ \_ as kind \ 	o \ kind
   None \rightarrow kind\_of\_stem symbol\_table token
let common\_stem table token =
  try
     SS.elements (ST.find (T.stem token) table.common\_stems)
  with
  | Not\_found \rightarrow []
let \ add\_symbol\_kind \ table \ token \ kind =
  try
    let old\_kind = ST.find token table in
    if kind = old\_kind then
       table
    else
       invalid\_arg ("conflicting_symbol_kind:_\" ^
                         T.to\_string\ token ``"_{\sqcup} ->_{\sqcup}"`
                           kind_to_string kind ^ "uvsu" ^
                             kind_to_string old_kind)
  with
  \mid Not\_found \rightarrow ST.add token kind table
let \ add\_stem\_kind \ table \ token \ kind =
  let stem = T.stem token in
  try
    let old\_kind = ST.find stem table in
    if kind = old\_kind then
       table
```

```
else begin
               match kind, old_kind with
                 Charged, Anti \rightarrow ST.add stem Charged table
                 Anti, Charged \rightarrow table
                   invalid\_arg ("conflicting_stem_kind:_" ^
                                      T.to\_string\ token\ ^{\circ}\ "_{\sqcup}->_{\sqcup}"\ ^{\circ}
                                         T.to\_string\ stem\ ^{\circ}\ "_{\sqcup}->_{\sqcup}"\ ^{\circ}
                                           kind_to_string kind ^ "uvsu" ^
                                              kind_to_string old_kind)
            end
       with
       \mid Not\_found \rightarrow ST.add stem kind table
     let \ add\_kind \ table \ token \ kind =
       { table with
          symbol\_kinds = add\_symbol\_kind \ table.symbol\_kinds \ token \ kind;
          stem\_kinds = add\_stem\_kind table.stem\_kinds token kind 
    let \ add\_stem \ table \ token =
       let stem = T.stem token in
       let set =
            ST.find stem table.common_stems
          with
          | Not\_found \rightarrow SS.empty in
       { table with
          common\_stems = ST.add stem (SS.add token set) table.common\_stems 
Go through the list of attributes, make sure that the space is declared and unique. Return the space.
     exception Missing\_Space of t
     exception Conflicting\_Space of t
     let group_rep_of_tokens group rep =
       let group =
          match group with
          [] \rightarrow L.default\_group
          | group \rightarrow L.group\_of\_string (T.list\_to\_string group) in
          Color (group, L.rep\_of\_string group (T.list\_to\_string rep))
    let index\_space index =
       let spaces =
          List.fold\_left
            (fun acc \rightarrow function
              I.Color\ (group,\ rep)\ \rightarrow\ group\_rep\_of\_tokens\ group\ rep\ ::\ acc
              I.Flavor\ (group,\ rep) \rightarrow Flavor\ (rep,\ group) :: acc
             | I.Lorentz \ t \rightarrow Lorentz \ t :: acc)
            [] index.I.attr in
       match ThoList.uniq (List.sort compare spaces) with
       |[space] \rightarrow space
       [] \rightarrow raise (Missing\_Space index.I.name)
       | \_ \rightarrow raise (Conflicting\_Space index.I.name)
    let tensor_space tensor =
       let spaces =
          List.fold_left
            (fun acc \rightarrow function
               X.Color\ (group,\ rep) \rightarrow group\_rep\_of\_tokens\ rep\ group\ ::\ acc
               X.Flavor\ (group,\ rep) \rightarrow Flavor\ (rep,\ group) :: acc
               X.Lorentz\ t\ 	o\ Lorentz\ t\ ::\ acc)
             [] tensor.X.attr in
       match ThoList.uniq (List.sort compare spaces) with
         [space] \rightarrow space
       [] \rightarrow raise (Missing\_Space tensor.X.name)
```

```
\downarrow \rightarrow raise (Conflicting_Space tensor.X.name)
NB: if P. Charged (name, name) below, only the Charged will survive, Anti will be shadowed.
     let insert\_kind \ table = function
        \mid F.Particle p \rightarrow
          begin match p.P.name with
            P.Neutral\ name\ 	o\ add\_kind\ table\ name\ Neutral
          \mid P.Charged (name, anti) \rightarrow
             add_kind (add_kind table anti Anti) name Charged
          F.Index \ i \rightarrow add\_kind \ table \ i.I.name \ (Index \ (index\_space \ i))
          F.Tensor\ t \rightarrow add\_kind\ table\ t.X.name\ (Tensor\ (tensor\_space\ t))
          F.Parameter p \rightarrow
          begin match p with
          | Q.Parameter \ name \rightarrow add\_kind \ table \ name.Q.name \ Parameter
          Q.Derived\ name\ 	o\ add\_kind\ table\ name.Q.name\ Derived
          end
        \mid F.Vertex \_ \rightarrow table
     let insert\_stem table = function
        \mid F.Particle p \rightarrow
          begin match p.P.name with
            P.Neutral\ name\ 	o\ add\_stem\ table\ name
           \mid P.Charged (name, anti) \rightarrow add\_stem (add\_stem table name) anti
          end
          F.Index i \rightarrow add\_stem table i.I.name
          F.Tensor\ t \rightarrow add\_stem\ table\ t.X.name
          F.Parameter p \rightarrow
          begin match p with
             Q.Parameter\ name
           | Q.Derived name \rightarrow add\_stem table name.Q.name
          end
        \mid F.Vertex \_ \rightarrow table
     let insert table token =
        insert_stem (insert_kind table token) token
     let load decls =
        List.fold_left insert empty decls
     let dump oc table =
        Printf.fprintf\ oc\ "<<< \sqcup Symbol \sqcup Table: \sqcup>>> \n";
        ST.iter
           (fun s k \rightarrow
            Printf.fprintf\ oc\ "%s_{\sqcup}->_{\sqcup}%s_{\square}"\ (T.to\_string\ s)\ (kind\_to\_string\ k))
           table.symbol\_kinds;
        Printf.fprintf\ oc\ "<<< \sqcup Stem \sqcup Table: \sqcup>>> \n";
        ST.iter
          (fun s k \rightarrow
            Printf.fprintf\ oc\ "%s_{\sqcup}->_{\sqcup}%s_{\square}"\ (T.to\_string\ s)\ (kind\_to\_string\ k))
          table.stem\_kinds;
        Printf.fprintf\ oc\ "<<< \sqcup Common \sqcup Stems: \sqcup >>> \n";
        ST.iter
           (fun stem \ symbols \rightarrow
            Printf.fprintf
              oc "%s_{\square}->_{\square}%s_{\square}"
              (T.to\_string\ stem)
               (String.concat
                   ",\square" (List.map T.to_string (SS.elements symbols))))
           table.common\_stems
  end
```

Declarations

```
{\sf module\ type\ } Declaration\ =
  sig
    type t
    val of\_string : string \rightarrow t list
    val to\_string: t list 	o string
For testing and debugging
    val\ of\_string\_and\_back\ :\ string 
ightarrow\ string
    val\ count\_indices\ :\ t\ 	o\ (int\ 	imes\ Symbol.t)\ list
    val\ indices\_ok : t \rightarrow unit
  end
module\ Declaration\ :\ Declaration\ =
     \mathsf{module}\ S\ =\ Symbol
    module T = Vertex\_syntax.Token
     type factor =
       \{ stem : T.t; 
          prefix : T.prefix list;
          particle : T.t \ list;
          color : T.t \ list;
         flavor : T.t \ list;
          lorentz : T.t list;
          other : T.t list }
    type t = factor \ list
    let factor\_stem token =
       \{ stem = token.T.stem; \}
          prefix = token.T.prefix;
         particle = [];
          color = [];
          flavor = [];
          lorentz = [];
          other = [] 
    let rev factor =
       \{ stem = factor.stem; \}
         prefix = List.rev factor.prefix;
         particle = List.rev factor.particle;
          color = List.rev\ factor.color;
          flavor = List.rev factor.flavor;
          lorentz = List.rev factor.lorentz;
          other = List.rev factor.other }
    let factor_add_prefix factor token =
       \{ factor with prefix = T.prefix\_of\_string token :: factor.prefix \}
    let factor\_add\_particle factor token =
       { factor with particle = token :: factor.particle }
    let factor_add_color_index t factor token =
       \{ factor \text{ with } color = token :: factor.color \}
    {\tt let} \ factor\_add\_lorentz\_index \ t \ factor \ token \ =
       (* diagnostics: Printf.eprintf "[L:[%s]]\n" (T.to\_string\ token); *)
       \{ factor with lorentz = token :: factor.lorentz \}
    let factor_add_flavor_index t factor token =
       \{ factor \text{ with } flavor = token :: factor.flavor \}
```

```
let factor_add_other_index factor token =
   \{ factor with other = token :: factor.other \}
let factor\_add\_kind\ factor\ token\ =\ function
     S.Neutral \mid S.Charged \mid S.Anti \rightarrow factor\_add\_particle factor token
    S.Index\ (S.Color\ (rep,\ group)) \rightarrow
      factor_add_color_index (rep, group) factor token
   | S.Index (S.Flavor (rep, group)) \rightarrow
      factor_add_flavor_index (rep, group) factor token
     S.Index\ (S.Lorentz\ t) \rightarrow factor\_add\_lorentz\_index\ t\ factor\ token
     S.Tensor \_ \rightarrow invalid\_arg "factor_add_index:_\\tensor"
     S.Parameter \rightarrow invalid\_arg "factor_add_index:_\\parameter"
     S.Derived \rightarrow invalid\_arg "factor_add_index:_\\derived"
let factor\_add\_index\ symbol\_table\ factor\ =\ function
     T.Token "," \rightarrow factor
     T.Token ("*" | "\ast" as <math>star) \rightarrow factor\_add\_prefix factor star
    token \rightarrow
      begin
         match \ S.kind\_of\_symbol\_or\_stem \ symbol\_table \ token \ with
           Some \ kind \rightarrow factor\_add\_kind \ factor \ token \ kind
           None \rightarrow factor\_add\_other\_index factor token
      end
let factor_of_token symbol_table token =
  \mathsf{let}\ token\ =\ T.wrap\_scripted\ token\ \mathsf{in}
  rev\ (List.fold\_left
            (factor\_add\_index\ symbol\_table)
            (factor_stem token)
            (token.T.super @ token.T.sub))
let list\_to\_string \ tag = function
  | [] \rightarrow ""
  | l \rightarrow "; | " \cap tag \cap " = " \cap String.concat ", " (List.map T.to_string l)
let \ factor\_to\_string \ factor \ =
    "[" \hat{T}.to\_string\ factor.stem
      (match factor.prefix with
      | [] \rightarrow ""
      \mid l 
ightarrow "; _{\sqcup}prefix=" ^{\hat{}}
                  String.concat "," (List.map T.prefix_to_string l)) ^
         list_to_string "particle" factor.particle
         list_to_string "color" factor.color
         list_to_string "flavor" factor.flavor `
         list_to_string "lorentz" factor.lorentz
         list_to_string "other" factor.other ^ "]"
let count_indices factors =
   Tho List. {\it classify}
     (\mathit{ThoList.flatmap}\ (\mathsf{fun}\ f\ \to\ f.\mathit{color}\ @\ f.\mathit{flavor}\ @\ f.\mathit{lorentz})\ \mathit{factors})
let format\_mismatch (n, index) =
   Printf.sprintf "index_\%s_appears_\%d_times" (T.to\_string\ index) n
let indices\_ok factors =
  match List.filter (fun (n, \bot) \rightarrow n \neq 2) (count_indices factors) with
   | \ | \ | \rightarrow \ ()
    mismatches \rightarrow
      invalid_arg (String.concat ", " (List.map format_mismatch mismatches))
let of string s =
  let decls = parse\_string s in
  let symbol\_table = Symbol.load decls in
  (* diagnostics: Symbol.dump stderr symbol_table; *)
  let \ tokens =
     List.fold\_left
```

```
(fun acc \rightarrow function
                                   Vertex\_syntax.File.Vertex(\_, v) \rightarrow T.wrap\_list v :: acc
                                   \rightarrow acc
                             [] decls in
                 let vlist = List.map (List.map (factor\_of\_token symbol\_table)) tokens in
                 List.iter indices_ok vlist;
                  vlist
           let to\_string\ decls\ =
                  String.concat ";"
                        (List.map
                                 (\mathsf{fun}\ v\ \to\ \mathit{String.concat}\ "{}_{\sqcup} *_{\sqcup}"\ (\mathit{List.map}\ \mathit{factor\_to\_string}\ v))
                                 decls)
           let of\_string\_and\_back \ s =
                 to\_string\ (of\_string\ s)
           type field =
                  \{ name : T.t \ list \}
     end
                                                                                                                          Complete Models
module Modelfile =
     struct
module Modelfile\_Test =
     struct
            let example() =
                 ()
           open OUnit
           let index\_mismatches =
                  "index_{\sqcup}mismatches" >:::
                       [ "1" >::
                                   (fun () \rightarrow
                                      assert\_raises
                                             (Invalid\_argument "index\_a\_1\_appears\_1\_times,\_\
        __index_a_2_appears_1_times")
                                             (fun () \rightarrow Declaration.of\_string\_and\_back
                                                                                   "\index{a}\\color{3}\
        \label{local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_local_loc
                              "3" >::
                                   (fun () \rightarrow
                                      assert\_raises
                                            (Invalid\_argument "index\_a\_appears\_3\_times")
                                             (fun () \rightarrow Declaration.of\_string\_and\_back
                                                                                   "\index{a}\\color{3}\
        let kind\_conflicts =
                  "kind\sqcupconflictings" >:::
                       [ "lorentz_/_color" >::
                                   (fun () \rightarrow
                                      assert\_raises
                                            (Invalid\_argument
                                                      "conflicting \sqcup stem \sqcup kind: \sqcup a\_2 \sqcup -> \sqcup a \sqcup -> \sqcup \setminus
        ___Lorentz_index_vs_color:SU(3):3_index")
                                             (fun () \rightarrow Declaration.of\_string\_and\_back)
                                                                                    \(a_1)\
```

```
\square\square\square\square\square\\index\{a_2\}\\lorentz\{X\}"));
       "color_{\sqcup}/_{\sqcup}color">::
         (fun () \rightarrow
          assert\_raises
            (Invalid\_argument
               "conflicting_stem_kind:_{\sqcup}a_{-}2_{\sqcup}->_{\sqcup}a_{\sqcup}->_{\sqcup}\setminus
 (fun () \rightarrow Declaration.of\_string\_and\_back
                         \verb| uuuuu| \land (a_2) \land (8)"));
       "neutral_{\square}/_{\square}charged" >::
         (fun () \rightarrow
          assert\_raises
            (Invalid\_argument
               "conflicting \sqcup stem \sqcup kind : \sqcup H^- \sqcup -> \sqcup H \sqcup -> \sqcup \setminus
 uuuchargeduantiuparticleuvsuneutraluparticle")
            (fun () \rightarrow Declaration.of\_string\_and\_back)
                         "\\neutral{H}\
 let suite =
    "Modelfile_Test" >:::
     [ "ok" >::
         (fun () \rightarrow
           assert\_equal \ \tilde{} printer : (fun \ s \ \rightarrow \ s)
             "[\\psi;_prefix=\\bar;_\
 uuuuuuuuuuuuuuu[\\gamma;ulorentz=\\mu,\\alpha_1,\\alpha_2]u*u\
 uuuuuuuuuuuu[\\psi;uparticle=e;ucolor=a;ulorentz=\\alpha_2]u*u\
 (Declaration.of\_string\_and\_back
                \c e^-{e^+}\
 \label{localization} $$\operatorname{b}\\operatorname{SU}(3)]{8}$
 uuuuuuuuuuuuu\\index{\\alpha}\\lorentz{X}\
 uuuuuuuuuuuuuuuuu\\gamma^\\mu_{\\alpha_1\\alpha_2}\
 index\_mismatches;
       kind_conflicts;
       "QCD.omf" >::
         (fun () \rightarrow
           dump_file "QCD" (parse_file "QCD.omf"));
       "SM.omf" >::
         (\mathsf{fun}\ ()\ \to
           dump_file "SM" (parse_file "SM.omf"));
       "SM-error.omf" >::
         (fun () \rightarrow
          assert\_raises
            (Invalid\_argument
               "SM-error.omf:32.22-32.27: syntax error (syntax error)")
            (fun () \rightarrow parse\_file "SM-error.omf"));
       "cyclic.omf" >::
         (fun () \rightarrow
          assert\_raises
            (Invalid_argument "cyclic_\\include{cyclic.omf}")
            (fun () \rightarrow parse\_file "cyclic.omf"))]
end
```

13.6.2 New Implementation: Obsolete Version 1

Start of version 1 of the new implementation. The old syntax will not be used in the real implementation, but the library for dealing with indices and permutations will remail important.

Note that $arity = length\ lorentz_reps = length\ color_reps$. Do we need to enforce this by an abstract type constructor?

A cleaner approach would be type context = (Coupling.lorentz, Color.t) array, but it would also require more tedious deconstruction of the pairs. Well, an abstract type with accessors might be the way to go after all ...

```
type context = \{arity : int; \\ lorentz\_reps : Coupling.lorentz array; \\ color\_reps : Color.t array \}
let \ distinct2 \ i \ j = \\ i \neq j
let \ distinct3 \ i \ j \ k = \\ i \neq j \ \land j \neq k \ \land k \neq i
let \ distinct \ ilist = \\ List.length \ (ThoList.uniq \ (List.sort \ compare \ ilist)) = \\ List.length \ ilist
```

An abstract type that allows us to distinguish offsets in the field array from color and Lorentz indices in different representations.

```
\begin{array}{ll} \text{module type } Index &= \\ \text{sig} \\ \text{type } t \\ \text{val } of\_int \ : \ int \rightarrow \ t \\ \text{val } to\_int \ : \ t \ \rightarrow \ int \\ \text{end} \end{array}
```

While the number of allowed indices is unlimited, the allowed offsets into the field arrays are of course restricted to the fields in the current *context*.

```
module type Field =
  sig
     type t
     exception Out\_of\_range of int
     val\ of\_int: context \rightarrow int \rightarrow t
     val to_int : t \rightarrow int
     val\ get: \alpha\ array \rightarrow\ t \rightarrow\ \alpha
  end
module\ Field\ :\ Field\ =
     \mathsf{type}\ t\ =\ int
     exception Out\_of\_range of int
     let of_int context i =
        if 0 \le i \land i < context.arity then
          i
        else
           raise (Out\_of\_range i)
     let to_i int i = 0
     let get = Array.get
  end
type field = Field.t
module type Lorentz =
  sig
```

We combine indices I and offsets F into the field array into a single type so that we can unify vectors with vector components.

```
type index = I of int \mid F of field

type vector = Vector of index

type spinor = Spinor of index

type conjspinor = ConjSpinor of index
```

These are all the primitive ways to construct Lorentz tensors, a. k. a. objects with Lorentz indices, from momenta, other Lorentz tensors and Dirac spinors:

```
\begin{array}{l} \text{type } primitive \ = \\ \mid G \text{ of } vector \ \times \ vector \ (* \ g_{\mu_1\mu_2} \ *) \\ \mid E \text{ of } vector \ \times \ vector \ \times \ vector \ (* \ \epsilon_{\mu_1\mu_2\mu_3\mu_4} \ *) \\ \mid K \text{ of } vector \ \times \ field \ (* \ k_2^{\mu_1} \ *) \\ \mid S \text{ of } conjspinor \ \times \ spinor \ (* \ \bar{\psi}_1\psi_2 \ *) \\ \mid V \text{ of } vector \ \times \ conjspinor \ \times \ spinor \ (* \ \bar{\psi}_1\gamma_{\mu_2}\psi_3 \ *) \\ \mid T \text{ of } vector \ \times \ vector \ \times \ conjspinor \ \times \ spinor \ (* \ \bar{\psi}_1\gamma_{\mu_2}\gamma_5\psi_3 \ *) \\ \mid A \text{ of } vector \ \times \ conjspinor \ \times \ spinor \ (* \ \bar{\psi}_1\gamma_{\mu_2}\gamma_5\psi_3 \ *) \\ \mid P \text{ of } conjspinor \ \times \ spinor \ (* \ \bar{\psi}_1\gamma_5\psi_2 \ *) \\ \text{type } tensor \ = \ int \ \times \ primitive \ list \\ \end{array}
```

Below, we will need to permute fields. For this purpose, we introduce the function $map_primitive\ v_idx\ v_fld\ s_idx\ s_fld\ c_idx$ that returns a structurally identical tensor, with $v_idx\ :\ int\ \to\ int$ applied to all vector indices, $v_fld\ :\ field\ \to\ field$ to all vector fields, s_idx and c_idx to all (conj)spinor indices and s_fld and c_fld to all (conj)spinor fields.

Note we must treat spinors and vectors differently, even for simple permuations, in order to handle the statistics properly.

```
val map\_tensor: (int \rightarrow int) \rightarrow (field \rightarrow field) \rightarrow (int \rightarrow int) \rightarrow (field \rightarrow field) \rightarrow (int \rightarrow int) \rightarrow (field \rightarrow field) \rightarrow tensor \rightarrow tensor
```

Check whether the tensor is well formed in the context.

```
val\ tensor\_ok\ :\ context\ 	o\ tensor\ 	o\ bool
```

The lattice $\mathbf{N} + i\mathbf{N} \subset \mathbf{C}$, which suffices for representing the matrix elements of Dirac matrices. We hope to be able to avoid the lattice $\mathbf{Q} + i\mathbf{Q} \subset \mathbf{C}$ or \mathbf{C} itself down the road.

Sparse Dirac matrices as maps from Lorentz and Spinor indices to complex numbers. This is supposed to be independent of the representation.

Dirac matrices as tables of nonzero entries. There will be one concrete Module per realization.

```
module type Dirac\_Matrices =
```

```
sig
           type t = (int \times int \times Complex.t') list
          val\ scalar\ :\ t
          val\ vector\ :\ (int\times t)\ list
          val\ tensor\ :\ (int \times int \times t)\ list
          val \ axial : (int \times t) \ list
           val pseudo : t
        end
E. g. the chiral representation:
     module\ Chiral\ :\ Dirac\_Matrices
Here's the functor to create the maps corresponding to a given realization.
     module\ Dirac\ :\ functor\ (M\ :\ Dirac\_Matrices)\ 	o\ Dirac
  end
module Lorentz : Lorentz =
  struct
     type index =
          I of int (* \mu_0, \mu_1, ..., \text{ not } 0, 1, 2, 3 *)
        | F of field
     let map\_index \ fi \ ff = function
        \mid I i \rightarrow I (fi i)
        \mid F i \rightarrow F (ff i)
     \mathsf{let} \ \mathit{indices} \ = \ \mathsf{function}
        \mid I \mid i \rightarrow [i]
        F_{-} \rightarrow []
Is the following level of type checks useful or redundant?
TODO: should we also support a tensor like F_{\mu_1\mu_2}?
     type vector = Vector of index
     type \ spinor = Spinor \ of \ index
     \mathsf{type}\ conjspinor\ =\ ConjSpinor\ \mathsf{of}\ index
     let map\_vector\ fi\ ff\ (Vector\ i) = Vector\ (map\_index\ fi\ ff\ i)
     let map\_spinor\ fi\ ff\ (Spinor\ i)\ =\ Spinor\ (map\_index\ fi\ ff\ i)
     let map\_conjspinor fi ff (ConjSpinor i) = ConjSpinor (map\_index fi ff i)
     let vector\_ok context = function
          Vector(I_{-}) \rightarrow
           (* we could perform additional checks! *)
           true
          Vector(F i) \rightarrow
             begin
                match\ Field.get\ context.lorentz\_reps\ i\ with
                  Coupling.Vector \rightarrow true
                  Coupling. Vectorspinor \rightarrow
                     failwith "Lorentz.vector_ok: incomplete"
                  _{-} 
ightarrow false
             end
     let spinor_ok context = function
        \mid Spinor(I_{-}) \rightarrow
           (* we could perfrom additional checks! *)
          true
        \mid Spinor(F i) \rightarrow
             begin
                match Field.get context.lorentz_reps i with
                  Coupling.Spinor \rightarrow true
                  Coupling.Vectorspinor \mid Coupling.Majorana \rightarrow
                     failwith "Lorentz.spinor_ok:\sqcupincomplete"
```

```
|\ \_ \ \to \ \mathsf{false} end |\ \mathsf{conjspinor\_ok}\ context\ =\ \mathsf{function} |\ \mathit{ConjSpinor}\ (I\ \_)\ \to \\ (*\ \mathsf{we}\ \mathsf{could}\ \mathsf{perform}\ \mathsf{additional}\ \mathsf{checks!}\ *) \mathsf{true} |\ \mathit{ConjSpinor}\ (F\ i)\ \to \\ \mathsf{begin} \mathsf{match}\ \mathit{Field}.\mathit{get}\ \mathit{context.lorentz\_reps}\ i\ \mathsf{with} |\ \mathit{Coupling}.\mathit{ConjSpinor}\ \to\ \mathsf{true} |\ \mathit{Coupling}.\mathit{ConjSpinor}\ \to\ \mathsf{true} |\ \mathit{Coupling}.\mathit{Vectorspinor}\ |\ \mathit{Coupling}.\mathit{Majorana}\ \to \\ \mathit{failwith}\ "Lorentz.conjspinor\_ok: $\sqcup$incomplete}" |\ \_\ \to\ \mathsf{false} \mathsf{end}
```

Note that distinct2 i j is automatically guaranteed for Dirac spinors, because the $\bar{\psi}$ and ψ can not appear in the same slot. This is however not the case for Weyl and Majorana spinors.

```
let spinor\_sandwitch\_ok context i j =
   conjspinor\_ok context i \land spinor\_ok context j
type primitive =
     G 	ext{ of } vector 	ext{ } 	ext{ } vector
     E 	ext{ of } vector 	imes vector 	imes vector 	imes vector
     K 	ext{ of } vector 	imes field
     S of conjspinor \times spinor
     V of vector \times conjspinor \times spinor
     T of vector \times vector \times conjspinor \times spinor
     A of vector \times conjspinor \times spinor
    P of conjspinor 	imes spinor
let map\_primitive fvi fvf fsi fsf fci fcf = function
     G(mu, nu) \rightarrow
        G (map_vector fvi fvf mu, map_vector fvi fvf nu)
   \mid E (mu, nu, rho, sigma) \rightarrow
        E (map\_vector fvi fvf mu,
           map_vector fvi fvf nu,
           map_vector fvi fvf rho,
           map_vector fvi fvf sigma)
  \mid K (mu, i) \rightarrow
        K (map_vector fvi fvf mu, fvf i)
    S(i, j) \rightarrow
        S (map_conjspinor fci fcf i, map_spinor fsi fsf j)
     V (mu, i, j) \rightarrow
        V (map\_vector fvi fvf mu,
           map\_conjspinor\ fci\ fcf\ i,
           map\_spinor fsi fsf j)
     T (mu, nu, i, j) \rightarrow
        T (map_vector fvi fvf mu,
           map_vector fvi fvf nu,
           map\_conjspinor\ fci\ fcf\ i,
           map\_spinor\ fsi\ fsf\ j)
  A (mu, i, j) \rightarrow
        A (map\_vector fvi fvf mu,
           map\_conjspinor\ fci\ fcf\ i,
           map\_spinor\ fsi\ fsf\ j)
        P (map\_conjspinor fci fcf i, map\_spinor fsi fsf j)
let primitive\_ok context =
  function
     \mid G(mu, nu) \rightarrow
          distinct2 \ mu \ nu \ \land
```

```
vector\_ok context mu \land vector\_ok context nu
     \mid E (mu, nu, rho, sigma) \rightarrow
          let i = [mu; nu; rho; sigma] in
          distinct \ i \ \land \ List.for\_all \ (vector\_ok \ context) \ i
     K(mu, i) \rightarrow
          vector_ok context mu
     \mid S(i, j) \mid P(i, j) \rightarrow
          spinor\_sandwitch\_ok context i j
     \mid V(mu, i, j) \mid A(mu, i, j) \rightarrow
          vector\_ok\ context\ mu\ \land\ spinor\_sandwitch\_ok\ context\ i\ j
     \mid T (mu, nu, i, j) \rightarrow
          vector\_ok\ context\ mu\ \land\ vector\_ok\ context\ nu\ \land
          spinor_sandwitch_ok context i j
let primitive\_vector\_indices = function
  G (Vector mu, Vector nu) | T (Vector mu, Vector nu, \_, \_) \rightarrow
        indices mu @ indices nu
    E (Vector mu, Vector nu, Vector rho, Vector sigma) \rightarrow
        indices mu @ indices nu @ indices rho @ indices sigma
    K (Vector mu, \_)
     V (Vector mu, \_, \_)
     A \ (Vector \ mu, \ \_, \ \_) \rightarrow indices \ mu
  \mid S(\_,\_) \mid P(\_,\_) \rightarrow []
let vector\_indices p =
   ThoList.flatmap primitive_vector_indices p
let primitive\_spinor\_indices = function
    G(-, -) \mid E(-, -, -, -) \mid K(-, -) \rightarrow []
     S (\_, Spinor\ alpha) | V (\_, \_, Spinor\ alpha)
     T (\_, \_, Spinor\ alpha)
   \mid A (\_, \_, Spinor alpha) \mid P (\_, Spinor alpha) \rightarrow indices alpha
let spinor\_indices p =
   ThoList.flatmap primitive_spinor_indices p
let \ primitive\_conjspinor\_indices \ = \ function
    G(-, -) \mid E(-, -, -, -) \mid K(-, -) \rightarrow []
     S (ConjSpinor alpha, \_) \mid V (\_, ConjSpinor alpha, \_)
     T (\_, \_, ConjSpinor\ alpha, \_)
   | \ A \ (\_, \ ConjSpinor \ alpha, \ \_) \ | \ P \ (ConjSpinor \ alpha, \ \_) \ \rightarrow \ indices \ alpha
let conjspinor\_indices p =
   Tho List. flat map\ primitive\_conjspinor\_indices\ p
let \ vector\_contraction\_ok \ p = 1
  let c = ThoList.classify (vector\_indices p) in
  print\_endline
     (String.concat ", "
         (List.map
             (\text{fun } (n, i) \rightarrow string\_of\_int n ^ "_{\sqcup}*_{\sqcup}" ^ string\_of\_int i)
             c));
  flush stdout;
  let res = List.for\_all (fun (n, \_) \rightarrow n = 2) c in
let two\_of\_each indices p =
  List.for\_all \text{ (fun } (n, \_) \rightarrow n = 2) \text{ (} ThoList.classify \text{ (} indices \text{ } p\text{))}
let vector\_contraction\_ok = two\_of\_each \ vector\_indices
let \ spinor\_contraction\_ok = two\_of\_each \ spinor\_indices
let\ conjspinor\_contraction\_ok\ =\ two\_of\_each\ conjspinor\_indices
let contraction\_ok p =
  vector\_contraction\_ok p \land
  spinor\_contraction\_ok\ p\ \land\ conjspinor\_contraction\_ok\ p
```

```
type tensor = int \times primitive \ list
let map_tensor fvi fvf fsi fsf fci fcf (factor, primitives) =
   (factor, List.map (map_primitive fvi fvf fsi fsf fci fcf ) primitives)
let \ tensor\_ok \ context \ (\_, \ primitives) \ =
   List.for\_all\ (primitive\_ok\ context)\ primitives\ \land
   contraction\_ok\ primitives
module Complex =
  struct
     type t = int \times int
     type t' = Z \mid O \mid M \mid I \mid J \mid C of int \times int
     \mathsf{let}\ to\_fortran\ =\ \mathsf{function}
        |Z \rightarrow "(0,0)"
          O \rightarrow "(1,0)"
          M \rightarrow "(-1,0)"
          I 	o "(0,1)"
          J \rightarrow "(0,-1)"
          C(r, i) \rightarrow "(" \hat{string\_of\_int r} ", " \hat{string\_of\_int i} ")"
  end
module type Dirac =
  sig
     val\ scalar\ :\ int \rightarrow\ int \rightarrow\ Complex.t'
     val\ vector\ :\ int \rightarrow\ int \rightarrow\ int \rightarrow\ Complex.t'
     \mathsf{val}\ tensor\ :\ int \to\ int \to\ int \to\ int \to\ Complex.t'
     val \ axial : int \rightarrow int \rightarrow int \rightarrow Complex.t'
     val\ pseudo: int \rightarrow int \rightarrow Complex.t'
  end
module type Dirac_Matrices =
  sig
     type t = (int \times int \times Complex.t') list
     \mathsf{val}\ scalar\ :\ t
     \mathsf{val}\ vector\ :\ (int\ \times\ t)\ \mathit{list}
     val\ tensor\ :\ (int \times int \times t)\ list
     val \ axial : (int \times t) \ list
     val pseudo : t
  end
module\ Chiral\ :\ Dirac\_Matrices\ =
     type t = (int \times int \times Complex.t') list
     \mathsf{let}\ scalar\ =
        [(1, 1, Complex. O);
           (2, 2, Complex. O);
           (3, 3, Complex.O);
           (4, 4, Complex.O)
     let \ vector =
        [\ (0,\ [\ (1,\ 4,\ Complex.O);
                   (4, 1, Complex. O);
                   (2, 3, Complex.M);
                   (3, 2, Complex.M)]);
           (1, [(1, 3, Complex. O);
                   (3, 1, Complex. O);
                   (2, 4, Complex.M);
                   (4, 2, Complex.M)]);
           (2, [(1, 3, Complex.I);
                   (3, 1, Complex.I);
                   (2, 4, Complex.I);
```

```
(4, 2, Complex.I);
         (3, [(1, 4, Complex.M);
                 (4, 1, Complex.M);
                 (2, 3, Complex.M);
                 (3, 2, Complex.M)])
    let \ tensor =
       [ (* TODO!!! *) ]
    \mathsf{let} \ \mathit{axial} \ =
       [ (0, [ (1, 4, Complex.M);
                 (4, 1, Complex. O);
                 (2, 3, Complex. O);
                 (3, 2, Complex.M) ]);
         (1, [(1, 3, Complex.M);
                 (3, 1, Complex. O);
                 (2, 4, Complex. O);
                 (4, 2, Complex.M)]);
         (2, [(1, 3, Complex.J);
                 (3, 1, Complex.I);
                 (2, 4, Complex.J);
                 (4, 2, Complex.I);
         (3, [(1, 4, Complex. O);
                 (4, 1, Complex.M);
                 (2, 3, Complex. O);
                 (3, 2, Complex.M)])
    let pseudo =
       [(1, 1, Complex.M);
          (2, 2, Complex.M);
         (3, 3, Complex.O);
         (4, 4, Complex.O)
  end
module\ Dirac\ (M\ :\ Dirac\_Matrices)\ :\ Dirac\ =
  struct
     module Map2 =
       Map.Make
          (struct
            type t = int \times int
            let compare = pcompare
         end)
    {\sf let} \ init2 \ triples \ =
       List.fold\_left
          (fun acc\ (i,\ j,\ e)\ \rightarrow\ Map2.add\ (i,\ j)\ e\ acc)
          Map2.empty triples
    let bounds\_check2 \ i \ j =
       if i < 1 \lor i > 4 \lor j < 0 \lor j > 4 then
          invalid_arg "Chiral.bounds_check2"
    let lookup2 map i j =
       bounds\_check2 \ i \ j;
       try Map2.find (i, j) map with Not\_found \rightarrow Complex.Z
     module Map3 =
       Map.Make
          (struct
            \mathsf{type}\ t\ =\ int\ \times\ (int\ \times\ int)
            let compare = pcompare
         end)
    {\tt let} \ init \textit{3} \ quadruples \ = \\
       List.fold\_left
```

```
(fun \ acc \ (mu, \ gamma) \ \rightarrow
                List.fold\_right
                   (\text{fun } (i, j, e) \rightarrow Map3.add (mu, (i, j)) e)
                   gamma acc)
               Map3.empty quadruples
         let bounds\_check3 mu i j =
            bounds\_check2 \ i \ j;
            if mu < 0 \lor mu > 3 then
               invalid_arg "Chiral.bounds_check3"
         let lookup3 \ map \ mu \ i \ j =
            bounds_check3 mu i j;
            try Map3.find\ (mu,\ (i,\ j))\ map\ with\ Not\_found\ 	o\ Complex.Z
          module Map4 =
            Map.Make
               (struct
                 \mathsf{type}\ t\ =\ int\ \times\ int\ \times\ (int\ \times\ int)
                 let compare = pcompare
               end)
         let init4 quadruples =
            List.fold\_left
               (fun \ acc \ (mu, \ nu, \ gamma) \rightarrow
                List.fold\_right
                   (\text{fun } (i, j, e) \rightarrow Map4.add (mu, nu, (i, j)) e)
                   gamma acc)
               Map4.empty quadruples
         let bounds\_check4 mu nu i j =
            bounds\_check3 nu i j;
            if mu~<~0~\vee~mu~>~3 then
               invalid_arg "Chiral.bounds_check4"
         let lookup4 \ map \ mu \ nu \ i \ j =
            bounds_check4 mu nu i j;
            try Map4.find\ (mu,\ nu,\ (i,\ j))\ map\ with\ Not\_found\ 	o\ Complex.Z
         let scalar map = init2 M.scalar
         let \ vector\_map = init3 \ M.vector
         let \ tensor\_map \ = \ init 4 \ M.tensor
         let axial\_map = init3 M.axial
         let pseudo\_map = init2 M.pseudo
         let scalar = lookup2 scalar\_map
         \mathsf{let}\ vector\ =\ lookup3\ vector\_map
         \mathsf{let}\ tensor\ mu\ nu\ i\ j\ =
            lookup4 tensor_map mu nu i j
         let tensor mu nu i j =
            failwith "tensor: incomplete"
         let axial = lookup3 axial\_map
         let pseudo = lookup2 pseudo\_map
       end
  end
module type Color =
  sig
     module\ Index:\ Index
    type index = Index.t
     type color\_rep = F of field \mid C of field \mid A of field
     type primitive =
         D of field \times field
         E 	ext{ of } field 	imes field 	imes field (* only for <math>SU(3) *)
       \mid T \text{ of } field \times field \times field
```

```
\mid F \text{ of } field \times field \times field
     val map\_primitive : (field \rightarrow field) \rightarrow primitive \rightarrow primitive
     val\ primitive\_indices\ :\ primitive\ 	o\ field\ list
     val\ indices\ :\ primitive\ list\ 
ightarrow\ field\ list
     type tensor = int \times primitive list
     val\ map\_tensor:
        (field \rightarrow field) \rightarrow \alpha \times primitive \ list \rightarrow \alpha \times primitive \ list
     \verb|val| tensor_ok : context| \rightarrow \alpha \times primitive \ list \rightarrow \ bool
   end
module\ Color\ :\ Color\ =
  struct
     module\ Index : Index =
        struct
           type t = int
           let of_int i = i
           let to_int i = i
        end
a_0, a_1, \ldots, \text{ not } 0, 1, \ldots
     type index = Index.t
     type color\_rep =
           F of field
           C of field
         | A of field
     type primitive =
           D of field \times field
           E 	ext{ of } field 	ext{ } 	ext{ } field 	ext{ } 	ext{ } 	ext{ } field 
           T of field \times field \times field
         \mid F \text{ of } field \times field \times field
     let map\_primitive f = function
         D(i, j) \rightarrow D(f i, f j)
           E(i, j, k) \rightarrow E(f i, f j, f k)
           T(a, i, j) \rightarrow T(f a, f i, f j)
         F(a, b, c) \rightarrow F(f a, f b, f c)
     let primitive\_ok \ ctx =
        function
           \mid D(i, j) \rightarrow
                  distinct2 \ i \ j \ \land
                 (match Field.get ctx.color_reps i, Field.get ctx.color_reps j with
                 \mid Color.SUN (n1), Color.SUN (n2) \rightarrow
                       n1 = -n2 \wedge n2 > 0
                   _{-},~_{-}~
ightarrow~ false)
           \mid E(i, j, k) \rightarrow
                 distinct3 \ i \ j \ k \ \land
                 (match Field.get ctx.color_reps i,
                     Field.get\ ctx.color\_reps\ j,\ Field.get\ ctx.color\_reps\ k with
                 | Color.SUN (n1), Color.SUN (n2), Color.SUN (n3) \rightarrow
                       n1 = 3 \wedge n2 = 3 \wedge n3 = 3 \vee
                       n1 = -3 \land n2 = -3 \land n3 = -3
                    | \  \  \_, \  \  \_, \  \  \_ \  \  \, \mathsf{false})
           \mid \ T\ (a,\ i,\ j)\ \rightarrow
                 distinct3 \ a \ i \ j \ \land
                 (match Field.get ctx.color_reps a,
                    Field.get ctx.color_reps i, Field.get ctx.color_reps j with
                 | Color.AdjSUN(n1), Color.SUN(n2), Color.SUN(n3) \rightarrow
                       n1 = n3 \wedge n2 = -n3 \wedge n3 > 0
                   \mathsf{\_},\ \mathsf{\_},\ \mathsf{\_}\ \to\ \mathsf{false})
           \mid F(a, b, c) \rightarrow
```

```
distinct3 a b c \land
               (match Field.get ctx.color_reps a,
                  Field.get ctx.color_reps b, Field.get ctx.color_reps c with
                 Color.AdjSUN(n1), Color.AdjSUN(n2), Color.AdjSUN(n3) \rightarrow
                    n1 = n2 \wedge n2 = n3 \wedge n1 > 0
               | \  \  \_, \  \  \_, \  \  \_ \  \  \to \  \, \mathsf{false})
     \mathsf{let}\ \mathit{primitive\_indices}\ =\ \mathsf{function}
        | D (\_, \_) \rightarrow []
          E (\_, \_, \_) \rightarrow []
          T(a, \_, \_) \rightarrow [a]
         F(a, b, c) \rightarrow [a; b; c]
     let indices p =
        ThoList.flatmap primitive_indices p
     let contraction\_ok p =
       List.for\_all
          (\mathsf{fun}\ (n,\ \_)\ \to\ n\ =\ 2)
          (ThoList.classify\ (indices\ p))
     \mathsf{type}\ \mathit{tensor}\ =\ \mathit{int}\ \times\ \mathit{primitive}\ \mathit{list}
     let map\_tensor f (factor, primitives) =
        (factor, List.map (map\_primitive f) primitives)
     let tensor_ok context (_, primitives) =
        List.for_all (primitive_ok context) primitives
  end
type t =
     \{ fields : string \ array; \}
       lorentz : Lorentz.tensor list;
        color : Color.tensor list }
module \ Test \ (M : Model.T) : Test =
  struct
     module Permutation = Permutation. Default
     let context_of_flavors flavors =
        \{ arity = Array.length flavors; \}
          lorentz\_reps = Array.map \ M.lorentz \ flavors;
          color\_reps = Array.map\ M.color\ flavors\ \}
     let context_of_flavor_names names =
        context_of_flavors (Array.map M.flavor_of_string names)
     let context\_of\_vertex v =
        context_of_flavor_names v.fields
       let context = context\_of\_vertex \ v \ in
       List.for\_all\ (Lorentz.tensor\_ok\ context)\ v.lorentz\ \land
          List.for\_all\ (Color.tensor\_ok\ context)\ v.color
     module PM =
        Partial.Make (struct type t = field let compare = compare end)
     let id x = x
     let permute \ v \ p =
       let \ context = \ context\_of\_vertex \ v \ in
       let sorted =
          List.map
             (Field.of\_int\ context)
             (ThoList.range\ 0\ (Array.length\ v.fields\ -\ 1)) in
       let permute =
          PM.apply (PM.of_lists sorted (List.map (Field.of_int context) p)) in
```

```
\{ fields = Permutation.array (Permutation.of\_list p) v.fields; \}
     lorentz = List.map
       (Lorentz.map_tensor id permute id permute id permute) v.lorentz;
     color = List.map (Color.map\_tensor permute) v.color 
let permutations v =
   List.map (permute v)
     (Combinatorics.permute\ (ThoList.range\ 0\ (Array.length\ v.fields\ -\ 1)))
let wf_declaration flavor =
  match M.lorentz (M.flavor_of_string flavor) with
     Coupling.Vector \rightarrow "vector"
     Coupling.Spinor \rightarrow "spinor"
     Coupling.ConjSpinor \rightarrow "conjspinor"
    \_ \rightarrow failwith "wf_declaration:_incomplete"
module Chiral = Lorentz.Dirac(Lorentz.Chiral)
let write\_fusion \ v =
  match Array.to_list v.fields with
   | lhs :: rhs \rightarrow
       let name = lhs ` "\_of\_" ` String.concat "\_" rhs in
       let momenta = List.map (fun n \rightarrow "k\_" ^n) rhs in
       Printf.printf "pure_function_%s_(%s)_result_(%s)\n"
          name (String.concat ", "
                    (List.flatten
                        (List.map2 (fun \ wf \ p \rightarrow [wf; \ p]) \ rhs \ momenta)))
        Printf.printf "_{\sqcup\sqcup} type(%s)_{\sqcup} : _{\sqcup}%s\n" (wf\_declaration lhs) lhs;
        List.iter
          (fun wf \rightarrow
             Printf.printf "uutype(%s),uintent(in)u::u%s\n"
               (wf\_declaration \ wf) \ wf)
          rhs:
        List.iter
          (Printf.printf "_{\sqcup\sqcup} type (momentum),_{\sqcup} intent(in)_{\sqcup} ::_{\sqcup} s\n")
          momenta;
       let rhs1 = List.hd rhs
       and rhs2 = List.hd (List.tl rhs) in
       begin match M.lorentz (M.flavor\_of\_string lhs) with
        \mid Coupling.Vector \rightarrow
            begin
               for mu = 0 to 3 do
                  Printf.printf "_{\sqcup \sqcup} \%s (\%d)_{\sqcup} = " lhs mu;
                  \quad \text{for } i \ = \ 1 \ \text{to} \ 4 \ \text{do}
                    for j = 1 to 4 do
                       match Chiral.vector mu i j with
                         Lorentz.Complex.Z \rightarrow ()
                         c \rightarrow
                            Printf.printf "_+,%s*%s(%d)*%s(%d)"
                               (Lorentz.Complex.to_fortran c) rhs1 i rhs2 j
                    done
                  done;
                  Printf.printf "\n"
               done
             end;
        | Coupling.Spinor | Coupling.ConjSpinor \rightarrow
            begin
               for i = 1 to 4 do
                  Printf.printf "_{\sqcup \sqcup} %s(%d)_{\sqcup} = " lhs i;
                  for mu = 0 to 3 do
                    for i = 1 to 4 do
                       match Chiral.vector mu i j with
```

```
Lorentz.Complex.Z \rightarrow ()
                            c \rightarrow
                               Printf.printf "_+_%s*%s(%d)*%s(%d)"
                                  (Lorentz.Complex.to_fortran c) rhs1 mu rhs2 j
                        done
                     done;
                      Printf.printf "\n"
                   done
                 end;
            \mid _ 
ightarrow failwith "write_fusion:_incomplete"
            end:
            Printf.printf "end_function_%s\n" name;
            ()
       | [] \rightarrow ()
    let write\_fusions \ v =
       List.iter\ write\_fusion\ (permutations\ v)
Testing:
    let \ vector\_field \ context \ i =
       Lorentz.Vector\ (Lorentz.F\ (Field.of\_int\ context\ i))
    let spinor\_field context i =
       Lorentz.Spinor\ (Lorentz.F\ (Field.of\_int\ context\ i))
    let conjspinor\_field context i =
       Lorentz.ConjSpinor (Lorentz.F (Field.of_int context i))
    let mu = Lorentz.Vector (Lorentz.I 0)
    and nu = Lorentz.Vector(Lorentz.I 1)
    let tbar_gl_t = [| "tbar"; "gl"; "t" |]
    let context = context\_of\_flavor\_names tbar\_gl\_t
    let \ vector\_current\_ok =
       \{ fields = tbar\_gl\_t; 
         lorentz = [(1, [Lorentz.V (vector\_field context 1,
                                           conjspinor_field context 0,
                                           spinor_field context 2)]) ];
         color = [(1, [Color.T (Field.of\_int context 1,
                                      Field.of\_int\ context\ 0,
                                      Field.of\_int\ context\ 2)])]
    let vector_current_vector_misplaced =
       \{ fields = tbar\_gl\_t; \}
         lorentz = [(1, [Lorentz.V (vector\_field context 2,
                                           conjspinor_field context 0,
                                           spinor_field context 2)]) ];
         color = [(1, [Color.T (Field.of\_int context 1,
                                      Field.of\_int\ context\ 0,
                                      Field.of\_int\ context\ 2)])]
    let \ vector\_current\_spinor\_misplaced =
       \{ fields = tbar\_gl\_t; \}
         lorentz = [(1, [Lorentz.V (vector\_field context 1,
                                           conjspinor_field context 0,
                                           spinor_field context 1)|) |;
         color = [(1, [Color.T (Field.of\_int context 1,
                                      Field.of_int context 0,
                                      Field.of_int context 2)])] }
    let vector_current_conjspinor_misplaced =
       \{ fields = tbar\_gl\_t; 
         lorentz = [(1, [Lorentz.V (vector\_field context 1,
                                           conjspinor_field context 1,
                                           spinor_field context 2)]) ];
```

```
color = [(1, [Color.T (Field.of\_int context 1,
                                 Field. of _int context 0,
                                 Field.of\_int\ context\ 2)])]
let vector\_current\_out\_of\_bounds () =
  \{ fields = tbar\_gl\_t; \}
     lorentz = [(1, [Lorentz.V (mu,
                                      conjspinor_field context 3,
                                      spinor_field context 2)]) ];
     color = [(1, [Color.T (Field.of\_int context 1,
                                 Field.of\_int\ context\ 0,
                                 Field.of\_int\ context\ 2)])]
let \ vector\_current\_color\_mismatch =
  let names = [| "t"; "gl"; "t" |] in
  let \ context = \ context\_of\_flavor\_names \ names \ in
  \{ fields = names; \}
     lorentz = [(1, [Lorentz.V (mu,
                                      conjspinor_field context 0,
                                      spinor_field context 2)]) ];
     color = [(1, [Color.T (Field.of\_int context 1,
                                 Field. of _int context 0,
                                 Field.of\_int\ context\ 2)])]
let wwzz = [| "W+"; "W-"; "Z"; "Z" |]
let context = context_of_flavor_names wwzz
let anomalous\_couplings =
  \{ fields = wwzz; 
     lorentz = [(1, [Lorentz.K (mu, Field.of\_int context 0);
                          Lorentz.K (mu, Field.of_int context 1) ]) ];
     color = [] 
let anomalous_couplings_index_mismatch =
  \{ fields = wwzz; 
     lorentz = [(1, [Lorentz.K (mu, Field.of\_int context 0);
                          Lorentz.K (nu, Field.of_int context 1) ]) ];
     color = [] 
exception Inconsistent\_vertex
let example () =
  if \neg (ok \ vector\_current\_ok) then begin
     raise\ Inconsistent\_vertex
  end:
  write_fusions vector_current_ok
open OUnit
let \ vertex\_indices\_ok =
  "indices/ok" >::
     (fun () \rightarrow
       List.iter
         (fun v \rightarrow
            assert_bool "vector_current" (ok v))
         (permutations\ vector\_current\_ok))
let vertex_indices_broken =
  "indices/broken" >::
     (fun () \rightarrow
       assert\_bool "vector\_misplaced"
         (\neg (ok \ vector\_current\_vector\_misplaced));
       assert\_bool "conjugate_spinor_misplaced"
         (\neg (ok\ vector\_current\_spinor\_misplaced));
       assert\_bool "conjugate\_spinor\_misplaced"
         (\neg (ok\ vector\_current\_conjspinor\_misplaced));
```

```
assert_raises (Field.Out_of_range 3)
          vector_current_out_of_bounds;
       assert\_bool \ \verb"color$\_mismatch"
          (\neg \ (ok \ vector\_current\_color\_mismatch)))
{\tt let} \ anomalous\_couplings\_ok \ = \\
   "anomalous_couplings/ok" >::
     (\mathsf{fun}\ ()\ \to
        assert\_bool "anomalous\sqcupcouplings"
          (ok\ anomalous\_couplings))
{\tt let} \ anomalous\_couplings\_broken \ = \\
   "anomalous_couplings/broken" >::
     (\mathsf{fun}\ ()\ \to
       assert\_bool "anomalous\sqcupcouplings"
          (\neg (ok \ anomalous\_couplings\_index\_mismatch)))
let suite =
  "Vertex" >:::
     [vertex\_indices\_ok;
      vertex\_indices\_broken;
      anomalous\_couplings\_ok;
      anomalous\_couplings\_broken]
```

end

—14— UFO Models

14.1 Interface of UFOx_syntax

14.1.1 Abstract Syntax

exception $Syntax_Error$ of $string \times Lexing.position \times Lexing.position$

```
type expr =
     Integer of int
     Float of float
      Variable of string
     Quoted of string
     Sum 	ext{ of } expr 	ext{ } 	ext{ } expr
     Difference of expr \times expr
     Product 	ext{ of } expr 	imes expr
     Quotient of expr \times expr
     Power of expr \times expr
     Application of string \times expr list
val\ integer\ :\ int 
ightarrow\ expr
\mathsf{val}\ \mathit{float} : \mathit{float} \to \ \mathit{expr}
\mathsf{val}\ variable\ :\ string \to\ expr
val\ quoted: string \rightarrow expr
val\ add\ :\ expr\ 	o\ expr\ 	o\ expr
\mathsf{val}\ subtract\ :\ expr\ \to\ expr\ \to\ expr
\mathsf{val}\ \mathit{multiply}\ :\ \mathit{expr}\ \to\ \mathit{expr}\ \to\ \mathit{expr}
\mathsf{val}\ divide\ :\ expr\ \to\ expr\ \to\ expr
\mathsf{val}\ power\ :\ expr\ 	o\ expr\ 	o\ expr
\mathsf{val}\ apply\ :\ string \to\ expr\ list \to\ expr
Return the sets of variable and function names referenced in the expression.
val\ variables\ :\ expr\ 	o\ Sets.String\_Caseless.t
\mathsf{val}\ functions\ :\ expr\ \to\ Sets.String\_Caseless.t
```

14.2 Implementation of UFOx_syntax

14.2.1 Abstract Syntax

exception $Syntax_Error$ of $string \times Lexing.position \times Lexing.position$

```
type expr = 
| Integer of int
| Float of float
| Variable of string
| Quoted of string
| Sum of expr \times expr
| Difference of expr \times expr
| Product of expr \times expr
| Quotient of expr \times expr
```

```
Power of expr \times expr
    Application of string \times expr list
\mathsf{let}\ integer\ i\ =
  Integer i
let float x =
  Float x
\mathsf{let}\ variable\ s\ =\ 
   Variable s
let quoted s =
  Quoted s
\mathsf{let}\ add\ e1\ e2\ =
  Sum (e1, e2)
let \ subtract \ e1 \ e2 \ =
  Difference (e1, e2)
let multiply e1 e2 =
  Product (e1, e2)
let divide \ e1 \ e2 =
   Quotient (e1, e2)
\mathsf{let}\ power\ e\ p\ =
  Power(e, p)
\mathsf{let}\ \mathit{apply}\ f\ \mathit{args}\ =
  Application (f, args)
module \ CSet = Sets.String\_Caseless
let rec \ variables = function
     Integer \_ \mid Float \_ \mid Quoted \_ \rightarrow CSet.empty
     Variable \ name \rightarrow CSet.singleton \ name
     Sum (e1, e2) \mid Difference (e1, e2)
    Product (e1, e2) \mid Quotient (e1, e2)
     Power(e1, e2) \rightarrow CSet.union(variables e1)(variables e2)
    Application (\_, elist) \rightarrow
      List.fold_left CSet.union CSet.empty (List.map variables elist)
let rec functions = function
     Integer \_ \mid Float \_ \mid Variable \_ \mid Quoted \_ \rightarrow CSet.empty
     Sum (e1, e2) \mid Difference (e1, e2)
     Product (e1, e2) \mid Quotient (e1, e2)
    Power(e1, e2) \rightarrow CSet.union(functions e1)(functions e2)
    Application (f, elist) \rightarrow
      List.fold_left CSet.union (CSet.singleton f) (List.map functions elist)
                                           14.3 Expression Lexer
open Lexing
open UFOx_parser
let string\_of\_char c =
  String.make 1 c
let init\_position fname lexbuf =
  let curr_p = lexbuf.lex_curr_p in
  lexbuf.lex\_curr\_p \leftarrow
     \{ curr_p \text{ with }
       pos\_fname = fname;
       pos\_lnum = 1;
```

```
pos\_bol = curr\_p.pos\_cnum };
  lexbuf
}
let digit = [,0,-,9,]
let upper = ['A'-'Z']
let \ lower = ['a'-'z']
\mathsf{let}\ char\ =\ upper\ |\ lower
let word = char \mid digit \mid '_-'
let white = [', ', '\t', '\n']
rule \ token = parse
     white { token lexbuf } (* skip blanks *)
    (' \{ LPAREN \})
    ')' { RPAREN }
    ',' { COMMA }
    '*' '*' { POWER }
    '*' { TIMES }
    ',' { DIV }
    '+' { PLUS }
    '-' { MINUS }
    (digit^+ \text{ as } i) (".",",")?
                          \{ INT (int\_of\_string i) \}
  \mid (digit \mid digit^*, digit^+)
              | digit^+ '.' digit^* ) ( ['E''e'] '-'? digit^+ )? as x
                          \{ FLOAT (float\_of\_string x) \}
  | ' \ ' \ (char\ word^*\ as\ s) ' \ '
                          \{ QUOTED s \}
  | char \ word^* ('.' char \ word^+)? as s
                          \{ID s\}
  | '\\', '[' (word^+ \text{ as } stem) ']' (word^* \text{ as } suffix)
                          { ID (UFO_tools.mathematica_symbol stem suffix) }
  \_ as c { raise (UFO\_tools.Lexical\_Error
                                       ("invalid_character_'" ^ string_of_char c ^ "',",
                                        lexbuf.lex\_start\_p, lexbuf.lex\_curr\_p)) }
  \mid eof \{END\}
```

14.4 Expression Parser

Right recursion is more convenient for constructing the value. Since the lists will always be short, there is no performace or stack size reason for prefering left recursion.

Header

Token declarations

```
\begin{tabular}{ll} \% to ken &< int > INT \\ \% to ken &< float > FLOAT \\ \% to ken &< string > ID \ QUOTED \\ \end{tabular}
```

```
%token PLUS MINUS TIMES POWER DIV
%token LPAREN RPAREN COMMA DOT
%token END
\%left PLUS\ MINUS
%left TIMES DIV
%left POWER
%nonassoc UNARY
%start input
\text{\%type} < UFOx\_syntax.expr > input
                                                 Grammar rules
input ::=
| expr END { $1 }
 | MINUS INT \% prec UNARY \{ X.integer (- $2) \}
  MINUS\ FLOAT\ \%prec\ UNARY\{\ X.float\ (-.\ \$2)\ \}
  INT \{ X.integer \$1 \}
  FLOAT \{ X.float \$1 \}
  ID \{ X.variable \$1 \}
  QUOTED \{ X.quoted \$1 \}
  expr\ PLUS\ expr\ \{\ X.add\ \$1\ \$3\ \}
  expr\ MINUS\ expr\ \{\ X.subtract\ \$1\ \$3\ \}
  expr TIMES expr { X.multiply $1 $3 }
  expr\ DIV\ expr\ \{\ X.divide\ \$1\ \$3\ \}
  PLUS expr %prec UNARY { $2 }
  MINUS \ expr \ \%prec \ UNARY \ \{ \ X.multiply \ (X.integer \ (-1)) \ \$2 \ \}
  expr POWER expr { X.power $1 $3 }
  LPAREN expr RPAREN { $2 }
  ID LPAREN RPAREN { X.apply $1 [] }
 | ID LPAREN args RPAREN { X.apply $1 $3 }
args ::=
 | expr { [$1] }
 | expr COMMA args { $1 :: $3 }
                                        14.5 Interface of UFOx
\mathsf{module}\; \mathit{Expr}\;:
  sig
    type t
    val of string : string \rightarrow t
    val\ of\_strings\ :\ string\ list 
ightarrow\ t
    \mathsf{val}\ substitute\ :\ string\ \rightarrow\ t\ \rightarrow\ t\ \rightarrow\ t
    val rename : (string \times string) \ list \rightarrow t \rightarrow t
    val\ half\ :\ string 
ightarrow \ t
    val\ variables:\ t\ 	o\ Sets.String\_Caseless.t
    val\ functions: t \rightarrow Sets.String\_Caseless.t
module \ Value :
  sig
    val\ of\ expr:\ Expr.t\ 	o\ t
```

```
val to\_string: t \to string val to\_coupling: (string \to \beta) \to t \to \beta \ Coupling.expr end
```



UFO represents rank-2 indices (i, j) as $1000 \cdot j + i$. This should be replaced by a proper union type eventually. Unfortunately, this requires many changes in the *Atoms* in *UFOx*. Therefore, we try a quick'n'dirty proof of principle first.

```
\begin{array}{l} \text{module type } \mathit{Index} \ = \\ \text{sig} \\ \text{type } t \ = \ \mathit{int} \\ \\ \text{val } \mathit{position} \ : \ t \ \rightarrow \ \mathit{int} \\ \text{val } \mathit{factor} \ : \ t \ \rightarrow \ \mathit{int} \times \ \mathit{int} \\ \text{val } \mathit{unpack} \ : \ t \ \rightarrow \ \mathit{int} \times \ \mathit{int} \\ \text{val } \mathit{pack} \ : \ \mathit{int} \ \rightarrow \ \mathit{int} \rightarrow \ \mathit{t} \\ \text{val } \mathit{map\_position} \ : \ (\mathit{int} \ \rightarrow \ \mathit{int}) \ \rightarrow \ t \ \rightarrow \ t \\ \text{val } \mathit{to\_string} \ : \ t \ \rightarrow \ \mathit{string} \\ \text{val } \mathit{list\_to\_string} \ : \ t \ \mathit{list} \ \rightarrow \ \mathit{string} \\ \end{array}
```

Indices are represented by a pair $int \times \rho$, where ρ denotes the representation the index belongs to. free indices returns all free indices in the list indices, i. e. all positive indices.

```
\mathsf{val}\; \mathit{free}\; :\; (t\; \times\; \rho)\; \mathit{list} \; \rightarrow \; (t\; \times\; \rho)\; \mathit{list}
```

summation indices returns all summation indices in the list indices, i. e. all negative indices.

```
val summation: (t \times \rho) \ list \rightarrow (t \times \rho) \ list
val classes\_to\_string: (\rho \rightarrow string) \rightarrow (t \times \rho) \ list \rightarrow string
```

Generate summation indices, starting from -1001. TODO: check that there are no clashes with explicitely named indices.

```
\begin{array}{l} \text{val } fresh\_summation \ : \ unit \to \ t \\ \text{val } named\_summation \ : \ string \to \ unit \to \ t \\ \\ \text{end} \\ \\ \text{module } Index \ : \ Index \\ \\ \text{module type } Tensor \ = \\ \\ \text{sig} \\ \\ \text{type } atom \end{array}
```

A tensor is a linear combination of products of atoms with rational coefficients. The following could be refined by introducing scalar atoms and restricting the denominators to $(scalar\ list \times Algebra.QC.t)\ list$. At the moment, this restriction is implemented dynamically by of_expr and not statically in the type system. Polymorphic variants appear to be the right tool, either directly or as phantom types. However, this is certainly only nice-to-have and is not essential.

```
type \alpha linear = (\alpha \ list \times Algebra.QC.t) list
type t = | Linear of atom linear | Ratios of (atom \ linear \times atom \ linear) list
```

We might need to replace atoms if the syntax is not context free.

```
val\ map\_atoms : (atom \rightarrow atom) \rightarrow t \rightarrow t
```

We need to rename indices to implement permutations ...

```
val map\_indices: (int \rightarrow int) \rightarrow t \rightarrow t
```

... but in order to to clean up inconsistencies in the syntax of lorentz.py and propagators.py we also need to rename indices without touching the second argument of P, the argument of Mass etc.

```
val\ rename\_indices\ :\ (int \rightarrow\ int)\ \rightarrow\ t\ \rightarrow\ t
```

We need scale coefficients.

```
val\ map\_coeff\ :\ (Algebra.QC.t\ 	o\ Algebra.QC.t)\ 	o\ t\ 	o\ t
```

Try to contract adjacent pairs of atoms as allowed but $Atom.contract_pair$. This is not exhaustive, but helps a lot with invariant squares of momenta in applications of Lorentz.

```
val\ contract\_pairs : t \rightarrow t
```

The list of variable referenced in the tensor expression, that will need to be imported by the numerical code.

```
val\ variables\ :\ t\ 	o\ string\ list
```

Parsing and unparsing. Lists of *strings* are interpreted as sums.

```
\begin{array}{lll} \text{val } of\_expr : UFOx\_syntax.expr \rightarrow t \\ \text{val } of\_string : string \rightarrow t \\ \text{val } of\_strings : string \ list \rightarrow t \\ \text{val } to\_string : t \rightarrow string \end{array}
```

The supported representations.

```
type r val classify\_indices: t \rightarrow (int \times r) list val rep\_to\_string: r \rightarrow string val rep\_to\_string\_whizard: r \rightarrow string val rep\_of\_int: bool \rightarrow int \rightarrow r val rep\_conjugate: r \rightarrow r val rep\_trivial: r \rightarrow bool
```

There is not a 1-to-1 mapping between the representations in the model files and the representations used by O'Mega, e.g. in *Coupling.lorentz*. We might need to use heuristics.

```
\mathsf{type}\ r\_omega
      val omega: r \rightarrow r\_omega
   end
module type Atom =
   sig
      val map\_indices: (int \rightarrow int) \rightarrow t \rightarrow t
      val rename\_indices: (int \rightarrow int) \rightarrow t \rightarrow t
      val\ contract\_pair\ :\ t\ 	o\ t\ option
      \mathsf{val}\ variable\ :\ t\ \to\ string\ option
      val\ scalar\ :\ t\ 	o\ bool
      val is\_unit : t \rightarrow bool
      val\ invertible\ :\ t\ 	o\ bool
      val\ invert : t \rightarrow t
      val\ of\_expr\ :\ string 
ightarrow\ UFOx\_syntax.expr\ list 
ightarrow\ t\ list
      val to\_string : t \rightarrow string
      \mathsf{val}\ \mathit{classify\_indices}\ :\ t\ \mathit{list}\ \rightarrow\ (\mathit{int}\ \times\ r)\ \mathit{list}
      \verb|val|| disambiguate\_indices|: t| list \rightarrow t| list
      val rep\_to\_string : r \rightarrow string
      val\ rep\_to\_string\_whizard: r \rightarrow string
      val\ rep\_of\_int : bool \rightarrow int \rightarrow r
      val\ rep\_conjugate : r \rightarrow r
      val rep_trivial : r \rightarrow bool
      type r\_omega
      val omega: r \rightarrow r\_omega
   end
module type Lorentz\_Atom =
   sig
      type dirac = private
            C of int \times int
            Gamma of int \times int \times int
            Gamma5 of int \times int
```

```
Identity of int \times int
           ProjP of int \times int
           ProjM of int \times int
           Sigma 	ext{ of } int 	imes int 	imes int 	imes int
     type vector = (* private *)
           Epsilon of int \times int \times int \times int
           Metric of int \times int
          P 	ext{ of } int 	imes int
     type scalar = (* private *)
           Mass of int
           Width of int
           P2 of int
           P12 of int \times int
           Variable of string
          Coeff of Value.t
     type t = (* private *)
           Dirac of dirac
           Vector of vector
           Scalar of scalar
           Inverse of scalar
     val map\_indices\_scalar : (int \rightarrow int) \rightarrow scalar \rightarrow scalar
     val\ map\_indices\_vector: (int \rightarrow int) \rightarrow vector \rightarrow vector
     val\ rename\_indices\_vector\ :\ (int 
ightarrow\ int)\ 
ightarrow\ vector\ 
ightarrow\ vector
  end
module\ Lorentz\_Atom\ :\ Lorentz\_Atom
module\ Lorentz\ :\ Tensor
  with type atom = Lorentz\_Atom.t and type r\_omega = Coupling.lorentz
module type Color\_Atom =
     \mathsf{type}\ t\ =\ (*\ \mathsf{private}\ *)
           Identity of int \times int
           Identity8 of int \times int
           T 	ext{ of } int \times int \times int
           F of int \times int \times int
           D of int \times int \times int
           Epsilon of int \times int \times int
           EpsilonBar of int \times int \times int
           T6 	ext{ of } int \times int \times int
           K6 	ext{ of } int \times int \times int
           K6Bar of int \times int \times int
  end
module\ Color\_Atom\ :\ Color\_Atom
module\ Color\ :\ Tensor
  with type atom = Color\_Atom.t and type r\_omega = Color.t
module type Test =
  sig
     \mathsf{val}\ example\ :\ unit \to\ unit
     val\ suite\ :\ OUnit.test
  end
```

14.6 Implementation of UFOx

```
\begin{array}{lll} \text{let } error\_in\_string \ text \ start\_pos \ end\_pos = \\ & \text{let } i = max \ 0 \ start\_pos.Lexing.pos\_cnum \ \text{in} \\ & \text{let } j = min \ (String.length \ text) \ (max \ (i + 1) \ end\_pos.Lexing.pos\_cnum) \ \text{in} \end{array}
```

```
String.sub text i (j - i)
let error_in_file name start_pos end_pos =
  Printf.sprintf
     "%s:%d.%d-%d.%d"
     name
     start_pos.Lexing.pos_lnum
     (start\_pos.Lexing.pos\_cnum - start\_pos.Lexing.pos\_bol)
     end_pos.Lexing.pos_lnum
     (end\_pos.Lexing.pos\_cnum - end\_pos.Lexing.pos\_bol)
module SMap = Map.Make (struct type t = string let compare = compare end)
module Expr =
  struct
     type t = UFOx\_syntax.expr
    let of\_string\ text =
       try
          UFOx\_parser.input
            UFOx\_lexer.token
            (UFOx\_lexer.init\_position "" (Lexing.from\_string\ text))
       with
       UFO\_tools.Lexical\_Error\ (msg,\ start\_pos,\ end\_pos) \rightarrow
           invalid_arg (Printf.sprintf "lexical_error_(%s)_at:_'%s'"
                              msg (error_in_string text start_pos end_pos))
       | UFOx_syntax_Syntax_Error (msg, start_pos, end_pos) \rightarrow
           invalid\_arg\ (Printf.sprintf\ "syntax\_error\_(%s)\_at:\_`%s'"
                              msg (error_in_string text start_pos end_pos))
       \mid Parsing.Parse\_error \rightarrow
           invalid_arg ("parse_error: " ^ text)
     let of\_strings = function
       [] \rightarrow UFOx\_syntax.integer 0
       \mid string :: strings \rightarrow
           List.fold\_right
             (\text{fun } s \ acc \rightarrow UFOx\_syntax.add \ (of\_string \ s) \ acc)
             strings (of_string string)
    open UFOx_syntax
    let rec map f = function
         Integer \_ \mid Float \_ \mid Quoted \_ as e \rightarrow e
         Variable \ s \ \mathsf{as} \ e \ 	o
           begin match f s with
             Some \ value \rightarrow value
           | None \rightarrow e
           end
         Sum (e1, e2) \rightarrow Sum (map f e1, map f e2)
         Difference (e1, e2) \rightarrow Difference (map f e1, map f e2)
         Product\ (e1,\ e2) \rightarrow Product\ (map\ f\ e1,\ map\ f\ e2)
         Quotient (e1, e2) \rightarrow Quotient (map f e1, map f e2)
         Power~(e1,~e2)~\rightarrow~Power~(map~f~e1,~map~f~e2)
         Application (s, el) \rightarrow Application (s, List.map (map f) el)
    {\tt let} \ substitute \ name \ value \ expr \ =
       map 	ext{ (fun } s \rightarrow \text{ if } s = name 	ext{ then } Some 	ext{ value else } None) 	ext{ } expr
    let rename1 name_map name =
       try Some (Variable (SMap.find name name_map)) with Not_found → None
     let rename alist_names value =
       let name_map =
          List.fold\_left
            (fun acc\ (name,\ name') \rightarrow SMap.add\ name\ name'\ acc)
            SMap.empty \ alist\_names \ in
```

```
map (rename1 name_map) value
     let half name =
        Quotient (Variable name, Integer 2)
     let \ variables = UFOx\_syntax.variables
     let functions = UFOx\_syntax.functions
  end
module \ Value =
  struct
     \mathsf{module}\ S\ =\ UFOx\_syntax
     module Q = Algebra.Q
     type builtin =
          Sqrt
          Exp | Log | Log10
          Sin | Asin
           Cos | Acos
           Tan | Atan
          Sinh | Asinh
           Cosh | Acosh
           Tanh | Atanh
          Sec | Asec
          Csc \mid Acsc
          Conj | Abs
     let builtin\_to\_string = function
          Sqrt \rightarrow "sqrt"
           Exp \rightarrow "exp"
          Log \rightarrow "log"
          Log10 \rightarrow "log10"
          Sin \rightarrow "sin"
           Cos \rightarrow "cos"
           Tan \rightarrow "tan"
          Asin \rightarrow "asin"
          Acos \rightarrow "acos"
           Atan \rightarrow "atan"
          Sinh \rightarrow "sinh"
           Cosh \rightarrow "cosh"
           Tanh \rightarrow "tanh"
          Asinh 
ightarrow "asinh"
          Acosh \rightarrow "acosh"
          A tanh \rightarrow "atanh"
          Sec \rightarrow "sec"
          \mathit{Csc} \to \texttt{"csc"}
          Asec 
ightarrow "asec"
          Acsc 
ightarrow "acsc"
          Conj \rightarrow "conjg"
          Abs \rightarrow "abs"
     let \ \mathit{builtin\_of\_string} \ = \ \mathsf{function}
          "cmath.sqrt" 	o Sqrt
           "cmath.exp" \rightarrow Exp
           \texttt{"cmath.log"} \to \textit{Log}
           "cmath.log10" \rightarrow Log10
           "cmath.sin" 
ightarrow Sin
           "cmath.cos" \rightarrow Cos
           "cmath.tan" \rightarrow Tan
           "cmath.asin" 	o \ Asin
           "cmath.acos" 
ightarrow Acos
           "cmath.atan" \rightarrow Atan
           \texttt{"cmath.sinh"} \to \mathit{Sinh}
```

```
"cmath.cosh" \rightarrow Cosh
     "cmath.tanh" \rightarrow Tanh
     "cmath.asinh" \rightarrow Asinh
     "cmath.acosh" \rightarrow Acosh
     "cmath.atanh" \rightarrow A tanh
     "\sec" \to Sec
     "\mathtt{csc"} \to \mathit{Csc}
     "asec" 	o Asec
     "acsc" \rightarrow Acsc
     "complexconjugate" \rightarrow Conj
     "abs" 
ightarrow Abs
     name \rightarrow failwith ("UFOx.Value: unsupported_function: " ^ name)
type t =
     Integer of int
     Rational \ {\it of} \ {\it Q.t}
     Real of float
     Complex 	ext{ of } float 	imes float
     Variable of string
     Sum 	ext{ of } t 	ext{ } list
     Difference of t \times t
     Product of t list
     Quotient of t \times t
     Power of t \times t
     Application of builtin \times t list
let rec to\_string = function
     Integer i \rightarrow string\_of\_int i
     Rational\ q\ 	o\ Q.to\_string\ q
     Real \ x \rightarrow string\_of\_float \ x
     Complex (0.0, 1.0) \rightarrow "I"
     Complex~(0.0,~-1.0)~\rightarrow~\text{"-I"}
     Complex (0.0, i) \rightarrow string\_of\_float i ^ "*I"
     Complex (r, 1.0) \rightarrow string\_of\_float r ^ "+I"
     Complex (r, -1.0) \rightarrow string\_of\_float r ^ "-I"
     Complex (r, i) \rightarrow
       string\_of\_float\ r ^ (if i<0.0 then "-" else "+") ^
         string_of_float (abs_float i) ^ "*I"
     Variable \ s \rightarrow s
     Sum [] \rightarrow "0"
     Sum [e] \rightarrow to\_string e
     Sum\ es\ 	o\ "("^String.concat"+"(List.map\ maybe\_parentheses\ es)^")"
     Difference\ (e1,\ e2)\ 	o\ to\_string\ e1\ ^"-"\ ^maybe\_parentheses\ e2
     Product [] \rightarrow "1"
     Product\ ((Integer\ (-1)\ |\ Real\ (-1.))\ ::\ es)\ 	o
       "-" ^ maybe_parentheses (Product es)
     Product\ es\ \rightarrow\ String.concat\ "*"\ (List.map\ maybe\_parentheses\ es)
     Quotient (e1, e2) \rightarrow to_string e1 ^{"}" ^{"} maybe_parentheses e2
     Power\ ((Integer\ i\ \mathsf{as}\ e),\ Integer\ p)\ \to
      if p < 0 then
         maybe\_parentheses (Real (float\_of\_int i)) ^
            "^(" ^ string_of_int p ^ ")"
       else if p = 0 then
         "1"
       else if p \leq 4 then
         maybe\_parentheses\ e\ ^{""}\ ^string\_of\_int\ p
         maybe\_parentheses (Real (float\_of\_int i)) ^
            "^{-}" ^{-} string_of_int p
   | Power (e1, e2) \rightarrow
       maybe_parentheses e1 ^ "^" ^ maybe_parentheses e2
  \mid Application (f, [Integer i]) \rightarrow
```

```
to\_string\ (Application\ (f,\ [Real\ (float\ i)]))
  \mid Application (f, es) \rightarrow
      builtin_to_string f
         "(" ^ String.concat "," (List.map to_string es) ^ ")"
and maybe\_parentheses = function
   | Integer\ i as e \rightarrow
      if i < 0 then
         "(" ^ to_string e ^ ")"
      else
         to_string e
  Real x as e \rightarrow
      if x < 0.0 then
         "(" \hat{to} to_string e \hat{r} ")"
      else
         to\_string\ e
     Complex (x, 0.0) \rightarrow to\_string (Real x)
     Complex (0.0, 1.0) \rightarrow "I"
     Variable = | Power (\_, \_) | Application (\_, \_) as e \rightarrow to\_string e
     Sum [e] \rightarrow to\_string e
     Product [e] \rightarrow maybe\_parentheses e
     e \rightarrow "(" \hat{} to_string e \hat{} ")"
\mathsf{let} \ \mathsf{rec} \ \mathit{to\_coupling} \ \mathit{atom} \ = \ \mathsf{function}
    Integer i \rightarrow Coupling.Integer i
    Rational q \rightarrow
      let n, d = Q.to\_ratio q in
      Coupling.Quot (Coupling.Integer n, Coupling.Integer d)
     Real \ x \rightarrow Coupling.Float \ x
     Product \ es \rightarrow Coupling.Prod \ (List.map \ (to\_coupling \ atom) \ es)
     Variable s \rightarrow Coupling.Atom (atom s)
     Complex (r, 0.0) \rightarrow Coupling.Float r
     Complex (0.0, 1.0) \rightarrow Coupling.I
     Complex (0.0, -1.0) \rightarrow Coupling.Prod [Coupling.I; Coupling.Integer (-1)]
     Complex (0.0, i) \rightarrow Coupling.Prod [Coupling.I; Coupling.Float i]
     Complex (r, 1.0) \rightarrow
      Coupling.Sum [Coupling.Float r; Coupling.I]
     Complex (r, -1.0) \rightarrow
      Coupling.Diff (Coupling.Float r, Coupling.I)
   | Complex(r, i) \rightarrow
      Coupling.Sum [Coupling.Float r;
                         Coupling.Prod [Coupling.I; Coupling.Float i]]
     Sum \ es \rightarrow Coupling.Sum \ (List.map \ (to\_coupling \ atom) \ es)
     Difference (e1, e2) \rightarrow
      Coupling.Diff (to_coupling atom e1, to_coupling atom e2)
    Quotient (e1, e2) \rightarrow
       Coupling. Quot (to_coupling atom e1, to_coupling atom e2)
   | Power (e1, Integer e2) \rightarrow
       Coupling. Pow (to_coupling atom e1, e2)
   | Power (e1, e2) \rightarrow
      Coupling.PowX (to_coupling atom e1, to_coupling atom e2)
     Application (f, [e]) \rightarrow apply1 (to\_coupling atom e) f
   Application (f, []) \rightarrow
      fail with
         ": _empty_argument_list")
  Application (f, \_ :: \_ :: \_) \rightarrow
      failwith
         ("UFOx.Value.to_coupling:_{\sqcup} ^{\smallfrown} builtin\_to\_string\ f ^{\smallfrown}
             ": \( \text{more} \) than \( \text{one} \) argument \( \text{in} \) list" \)
and apply1 e = function
  \mid Sqrt \rightarrow Coupling.Sqrt e
```

```
Exp \rightarrow Coupling.Exp e
     Log \rightarrow Coupling.Log e
     Log10 \rightarrow Coupling.Log10 e
     Sin \rightarrow Coupling.Sin e
     Cos \rightarrow Coupling.Cos e
     Tan \rightarrow Coupling. Tan e
     Asin \rightarrow Coupling. Asin e
     Acos \rightarrow Coupling.Acos e
     Atan \rightarrow Coupling.Atan e
     Sinh \rightarrow Coupling.Sinh e
     Cosh \rightarrow Coupling.Cosh e
     Tanh \rightarrow Coupling. Tanh e
     Sec \rightarrow Coupling.Quot (Coupling.Integer 1, Coupling.Cos e)
     Csc \rightarrow Coupling.Quot (Coupling.Integer 1, Coupling.Sin e)
     Asec \rightarrow Coupling.Acos (Coupling.Quot (Coupling.Integer 1, e))
     Acsc \rightarrow Coupling.Asin (Coupling.Quot (Coupling.Integer 1, e))
     Conj \rightarrow Coupling.Conj e
     Abs \rightarrow Coupling.Abs e
     (Asinh \mid Acosh \mid Atanh \text{ as } f) \rightarrow
       failwith
          ("UFOx. Value.to_coupling: __function__'"
             builtin\_to\_string\ f ^ "', \( \text{not}\) \( \text{supported}\) \( \text{yet!"} \)
let \ compress \ terms = terms
let rec of_expr e =
   compress (of \_expr' e)
and of \_expr' = function
     S.Integer i \rightarrow Integer i
     S.Float \ x \rightarrow Real \ x
     S.Variable "cmath.pi" \rightarrow Variable "pi"
     S.Quoted\ name\ 
ightarrow
       invalid_arg ("UFOx.Value.of_expr:_unexpected_quoted_variable_'," ^
                             name \ \hat{\ } ",")
     S.Variable\ name\ 	o\ Variable\ name
   | S.Sum(e1, e2) \rightarrow
       begin match of\_expr\ e1, of\_expr\ e2 with
         (Integer 0 \mid Real \ 0.), e \rightarrow e
         e, (Integer 0 \mid Real 0.) \rightarrow e
         Sum \ e1, Sum \ e2 \rightarrow Sum \ (e1 @ e2)
         e1, Sum e2 \rightarrow Sum (e1 :: e2)
         Sum \ e1, \ e2 \rightarrow Sum \ (e2 :: e1)
        e1, e2 \rightarrow Sum [e1; e2]
       end
   \mid S.Difference (e1, e2) \rightarrow
       begin match of\_expr\ e1, of\_expr\ e2 with
       \mid e1, (Integer 0 \mid Real 0.) \rightarrow e1
       | e1, e2 \rightarrow Difference (e1, e2)
       end
   \mid S.Product\ (e1,\ e2) \rightarrow
       begin match of _{-}expr e1, of _{-}expr e2 with
         (Integer \ 0 \ | \ Real \ 0.), \ \_ \rightarrow \ Integer \ 0
         \_, (Integer 0 \mid Real \ 0.) \rightarrow Integer \ 0
         (Integer 1 | Real 1.), e \rightarrow e
         e, (Integer 1 \mid Real 1.) \rightarrow e
         Product \ e1, \ Product \ e2 \rightarrow Product \ (e1 @ e2)
         e1, Product e2 \rightarrow Product (e1 :: e2)
         Product \ e1, \ e2 \rightarrow Product \ (e2 :: e1)
         e1, e2 \rightarrow Product [e1; e2]
       end
  \mid S.Quotient (e1, e2) \rightarrow
       begin match of expr e1, of expr e2 with
```

```
\mid e1, (Integer 0 \mid Real 0.) \rightarrow
                invalid_arg "UFOx.Value:idivideibyi0"
             e1, (Integer 1 | Real 1.) \rightarrow e1
            | e1, e2 \rightarrow Quotient (e1, e2)
            end
        | S.Power(e, p) \rightarrow
            begin match of \_expr \ e, of \_expr \ p with
            | (Integer \ 0 \ | Real \ 0.), (Integer \ 0 \ | Real \ 0.) \rightarrow
                invalid\_arg "UFOx.Value:_{\sqcup}0^0"
              \_, (Integer 0 | Real 0.) \rightarrow Integer 1
              e, (Integer 1 \mid Real 1.) \rightarrow e
            | Integer e, Integer p \rightarrow
                if p < 0 then
                   Power (Real (float\_of\_int e), Integer p)
                else if p = 0 then
                   Integer 1
                else if p \leq 4 then
                   Power (Integer e, Integer p)
                else
                   Power (Real (float\_of\_int e), Integer p)
            | e, p \rightarrow Power(e, p)
            end
        | S.Application ("complex", [r; i]) \rightarrow
            begin match of_{-}expr \ r, of_{-}expr \ i with
              r, (Integer 0 \mid Real \ 0.0) \rightarrow r
              Real r, Real i \rightarrow Complex (r, i)
              Integer r, Real i \rightarrow Complex (float_of_int r, i)
              Real r, Integer i \rightarrow Complex (r, float\_of\_int i)
              Integer r, Integer i \rightarrow Complex (float_of_int r, float_of_int i)
            | \_ \rightarrow invalid\_arg "UFOx.Value:\_complex\_expects\_two\_numeric\_arguments"
            end
        | S.Application ("complex", \_) \rightarrow
            invalid\_arg "UFOx.Value: \_complex\_expects\_two\_arguments"
        | S.Application ("complexconjugate", [e]) \rightarrow
            Application (Conj, [of\_expr e])
         S.Application ("complexconjugate", _) \rightarrow
            invalid\_arg "UFOx. Value: \_complexconjugate\_expects\_single\_argument"
          S.Application ("cmath.sqrt", [e]) \rightarrow
            Application (Sqrt, [of\_expr e])
         S.Application ("cmath.sqrt", _) \rightarrow
            invalid\_arg "UFOx.Value:\_sqrt\_expects\_single\_argument"
        \mid \ S.Application \ (name, \ args) \ \rightarrow
            Application (builtin_of_string name, List.map of_expr args)
  end
let positive integers =
   List.filter (fun (i, \_) \rightarrow i > 0) integers
let not_positive integers =
   List.filter (fun (i, \_) \rightarrow i \le 0) integers
module type Index =
  sig
     \mathsf{type}\ t\ =\ int
     val position : t \rightarrow int
     val\ factor: t \rightarrow int
     \mathsf{val}\ unpack\ :\ t\ \to\ int\ \times\ int
     val pack : int \rightarrow int \rightarrow t
     val map\_position : (int \rightarrow int) \rightarrow t \rightarrow t
     val to\_string : t \rightarrow string
     \mathsf{val}\ \mathit{list\_to\_string}\ :\ t\ \mathit{list} \to\ \mathit{string}
```

```
val free : (t \times \rho) list \rightarrow (t \times \rho) list
     val summation : (t \times \rho) list \rightarrow (t \times \rho) list
     val classes\_to\_string : (\rho \rightarrow string) \rightarrow (t \times \rho) \ list \rightarrow string
     val\ fresh\_summation : unit \rightarrow t
     val\ named\_summation: string \rightarrow unit \rightarrow t
  end
module\ Index\ :\ Index\ =
  struct
     type t = int
     \mathsf{let} \; \mathit{free} \; i \; = \; \mathit{positive} \; i
     let \ summation \ i \ = \ not\_positive \ i
     let position i =
        if i > 0 then
           i \mod 1000
        else
           i
     let factor i =
        if i > 0 then
           i / 1000
        else
           invalid\_arg "UFOx.Index.factor:\squareargument\squarenot\squarepositive"
     let \ unpack \ i =
        \quad \text{if} \ i \ > \ 0 \ \text{then} \\
           (position i, factor i)
        else
           (i, 0)
     let pack i j =
        if j > 0 then
           if i > 0 then
             1000 \times j + i
           else
              invalid\_arg "UFOx.Index.pack: _position_not_positive"
        \mathsf{else}\;\mathsf{if}\;j\;=\;0\;\mathsf{then}
           i
        else
           invalid\_arg "UFOx.Index.pack: \Boxfactor \Boxnegative"
     let map\_position f i =
        let pos, fac = unpack i in
        pack (f pos) fac
     let to\_string i =
        let pos, fac = unpack i in
        if fac = 0 then
           Printf.sprintf "%d" pos
           Printf.sprintf "%d.%d" pos fac
     let to\_string' = string\_of\_int
     let list\_to\_string is =
        "[" \hat{S}tring.concat ",\Box" (List.map to_string is) \hat{D} "]"
     let classes_to_string rep_to_string index_classes =
        let reps =
           ThoList.uniq (List.sort compare (List.map snd index_classes)) in
        " [" ^
           String.concat ", "
           (List.map
```

```
(fun r \rightarrow
                 (rep\_to\_string\ r) ^ "=" ^
                    (list\_to\_string
                        (List.map
                            fst
                            (List.filter (fun (\_, r') \rightarrow r = r') index\_classes))))
               reps) ^ "]"
     type factory =
        { mutable named : int SMap.t;}
           mutable used : Sets.Int.t }
     let factory =
        \{ named = SMap.empty; \}
           used = Sets.Int.empty 
     let first\_anonymous = -1001
     let fresh\_summation() =
        let next\_anonymous =
          try
             pred (Sets.Int.min_elt factory.used)
           with
           | Not\_found \rightarrow first\_anonymous in
        factory.used \leftarrow Sets.Int.add\ next\_anonymous\ factory.used;
        next\_anonymous
     let named\_summation name() =
        try
           SMap.find name factory.named
        with
        \mid Not\_found \rightarrow
            begin
               let next\_named = fresh\_summation () in
               factory.named \leftarrow SMap.add\ name\ next\_named\ factory.named;
               next\_named
            end
  end
module type Atom =
  sig
     type t
     val\ map\_indices: (int \rightarrow int) \rightarrow t \rightarrow t
     val\ rename\_indices\ :\ (int
ightarrow\ int)\ 
ightarrow\ t\ 
ightarrow\ t
     \mathsf{val}\ contract\_pair\ :\ t\ \to\ t\ option
     val\ variable\ :\ t\ 	o\ string\ option
     val\ scalar\ :\ t\ 	o\ bool
     \mathsf{val}\ is\_unit\ :\ t\ \to\ bool
     \mathsf{val}\ invertible\ :\ t\ \to\ bool
     val invert: t \rightarrow t
     val\ of\_expr\ :\ string 
ightarrow\ UFOx\_syntax.expr\ list 
ightarrow\ t\ list
     val to\_string : t \rightarrow string
     type r
     val classify\_indices : t \ list \rightarrow (Index.t \times r) \ list
     val\ disambiguate\_indices : t\ list \rightarrow\ t\ list
     val\ rep\_to\_string: r \rightarrow string
     val\ rep\_to\_string\_whizard: r \rightarrow string
     val\ rep\_of\_int : bool \rightarrow int \rightarrow r
     \mathsf{val}\ rep\_conjugate\ :\ r\ \to\ r
     val rep_trivial : r \rightarrow bool
     type r\_omega
     \mathsf{val}\ omega\ :\ r\ \to\ r\_omega
  end
```

```
module type Tensor =
  sig
     type atom
     type \alpha linear = (\alpha \ list \times Algebra.QC.t) list
     type t =
         Linear of atom linear
          Ratios of (atom\ linear\ 	imes\ atom\ linear)\ list
     val\ map\_atoms: (atom \rightarrow atom) \rightarrow t \rightarrow t
     val map\_indices: (int \rightarrow int) \rightarrow t \rightarrow t
     val\ rename\_indices: (int \rightarrow int) \rightarrow t \rightarrow t
     val\ map\_coeff\ :\ (Algebra.QC.t\ 	o\ Algebra.QC.t)\ 	o\ t\ 	o\ t
     val\ contract\_pairs\ :\ t\ 	o\ t
     val\ variables\ :\ t\ 	o\ string\ list
     val\ of\_expr\ :\ UFOx\_syntax.expr\ 	o\ t
     val of\_string : string \rightarrow t
     val of\_strings : string\ list 	o t
     val to\_string : t \rightarrow string
     type r
     val\ classify\_indices\ :\ t\ 	o\ (Index.t\ 	imes\ r)\ list
     val\ rep\_to\_string: r \rightarrow string
     val rep\_to\_string\_whizard : r \rightarrow string
     val\ rep\_of\_int : bool \rightarrow int \rightarrow r
     val rep\_conjugate : r \rightarrow r
     val rep_trivial : r \rightarrow bool
     type r\_omega
     val omega: r \rightarrow r\_omega
module \ Tensor \ (A : Atom) : Tensor
  with type atom = A.t and type r = A.r and type r\_omega = A.r\_omega =
     module S = UFOx\_syntax
     (* TODO: we have to switch to Algebra, QC to support complex coefficients, as used in custom propagators.
     module Q = Algebra.Q
     module \ QC = Algebra.QC
     type atom = A.t
     type \alpha linear = (\alpha \ list \times Algebra.QC.t) list
     type t =
          Linear of atom linear
          Ratios of (atom\ linear\ 	imes\ atom\ linear)\ list
     let term\_to\_string\ (tensors,\ c) =
       if QC.is\_null\ c then
          11 11
       else
          match tensors with
          [] \rightarrow QC.to\_string c
          \mid tensors \rightarrow
              String.concat
                 "*" ((if QC.is\_unit\ c then [] else [QC.to\_string\ c]) @
                          List.map A.to_string tensors)
     let linear\_to\_string terms =
        String.concat \verb""" (List.map term\_to\_string terms)
     let to\_string = function
         Linear\ terms \rightarrow linear\_to\_string\ terms
        \mid Ratios \ ratios \rightarrow
            String.concat
              "_+_"
              (List.map
```

```
(fun (n, d) \rightarrow
                    Printf.sprintf "(%s)/(%s)"
                      (linear\_to\_string\ n)\ (linear\_to\_string\ d))\ ratios)
    let variables\_of\_atoms atoms =
       List.fold\_left
          (fun acc \ a \rightarrow
            match A.variable a with
              None \rightarrow acc
             | Some name \rightarrow Sets.String.add name acc)
          Sets.String.empty atoms
    let variables_of_linear linear =
        List.fold\_left
          (fun\ acc\ (atoms,\ \_)\ 	o\ Sets.String.union\ (variables\_of\_atoms\ atoms)\ acc)
          Sets.String.empty linear
    let \ variables\_set = function
         Linear\ linear \rightarrow variables\_of\_linear\ linear
        \mid Ratios \ ratios \rightarrow
           List.fold\_left
              (fun acc\ (numerator,\ denominator)\ 	o
                Sets.String.union
                   (variables_of_linear numerator)
                   (Sets.String.union (variables_of_linear denominator) acc))
              Sets. String. empty ratios
    let variables t =
        Sets.String.elements (variables\_set t)
     let map\_ratios f = function
         Linear n \rightarrow Linear (f n)
        | Ratios ratios \rightarrow Ratios (List.map (fun (n, d) \rightarrow (f n, f d)) ratios)
    let map\_summands f t =
       map\_ratios (List.map f) t
    let map\_numerators f = function
       \mid Linear \ n \rightarrow Linear \ (List.map \ f \ n)
        \mid Ratios \ ratios \rightarrow
           Ratios (List.map (fun (n, d) \rightarrow (List.map f n, d)) ratios)
    let map\_atoms\ f\ t\ =
       map\_summands (fun (atoms, q) \rightarrow (List.map f atoms, q)) t
    let \ map\_indices \ f \ t \ =
        map\_atoms (A.map\_indices f) t
     let rename\_indices\ f\ t\ =
       map\_atoms (A.rename\_indices f) t
    let map\_coeff f t =
       map\_numerators (fun (atoms, q) \rightarrow (atoms, f q)) t
     type result =
         Matched of atom list
          Unmatched of atom list
contract_pair a rev_prefix suffix returns Unmatched (a :: List.rev_append rev_prefix suffix if there is no
match (as defined by A.contract_pair) and Matched with the reduced list otherwise.
     let rec contract\_pair \ a \ rev\_prefix = function
       [] \rightarrow Unmatched (a :: List.rev rev\_prefix)
       a' :: suffix \rightarrow
           begin match A.contract\_pair \ a \ a' with
             None \rightarrow contract\_pair \ a \ (a' :: rev\_prefix) \ suffix
            Some a'' \rightarrow
               if A.is\_unit \ a'' then
```

```
Matched (List.rev_append rev_prefix suffix)
                               else
                                     Matched\ (List.rev\_append\ rev\_prefix\ (a''\ ::\ suffix))
                        end
Use contract_pair to find all pairs that match according to A.contract_pair.
          let rec contract\_pairs1 = function
                | ([] | [\_] \text{ as } t) \rightarrow t
                a :: t \rightarrow
                       begin match contract\_pair\ a\ [\ ]\ t with
                            Unmatched ([]) \rightarrow []
                            Unmatched\ (a'::t') \rightarrow a'::contract\_pairs1\ t'
                        | Matched t' \rightarrow contract\_pairs1 \ t'
                       end
          let contract\_pairs\ t\ =
                map\_summands (fun (t', c) \rightarrow (contract\_pairs1 \ t', c)) t
          let add t1 t2 =
               match t1, t2 with
                 | Linear l1, Linear l2 \rightarrow Linear (l1 @ l2)
                    Ratios \ r, \ Linear \ l \ \mid \ Linear \ l, \ Ratios \ r \ \rightarrow
                        Ratios ((l, [([], QC.unit)]) :: r)
                | Ratios r1, Ratios r2 \rightarrow Ratios (r1 @ r2)
          let multiply1 (t1, c1) (t2, c2) =
                (List.sort compare (t1 @ t2), QC.mul c1 c2)
          let multiply2 t1 t2 =
                Product.list2 multiply1 t1 t2
          let multiply t1 t2 =
               match t1, t2 with
                    Linear l1, Linear l2 \rightarrow Linear (multiply2 l1 l2)
                    Ratios r, Linear l | Linear l, Ratios r \rightarrow
                        Ratios (List.map (fun (n, d) \rightarrow (multiply2 \ l \ n, d)) \ r)
                | Ratios r1, Ratios r2 \rightarrow
                        Ratios (Product.list2
                                                  (\text{fun } (n1, d1) (n2, d2) \rightarrow
                                                       (multiply2 n1 n2, multiply2 d1 d2))
                                                  r1 r2)
          let rec power n t =
               if n < 0 then
                     invalid\_arg "UFOx.Tensor.power:_{\sqcup}n_{\sqcup}<_{\sqcup}0"
               \  \, {\rm else} \,\, {\rm if} \,\, n \,\, = \,\, 0 \,\, {\rm then} \,\,
                     Linear[([], QC.unit)]
               else if n = 1 then
                     t
               else
                     multiply \ t \ (power \ (pred \ n) \ t)
          let \ compress \ ratios =
               map\_ratios
                     (fun terms \rightarrow
                          List.map (fun (t, cs) \rightarrow (t, QC.sum cs)) (ThoList.factorize terms))
                     ratios
          let rec of expr e =
                contract\_pairs\ (compress\ (of\_expr'\ e))
          and of \_expr' = function
                   S.Integer i \rightarrow Linear [([], QC.make (Q.make i 1) Q.null)]
                    S.Float \_ \rightarrow invalid\_arg "UFOx.Tensor.of_expr:\unuexpected\underline{\text{toat}}"
                \mid S.Quoted \ name \rightarrow
                        invalid\_arg ("UFOx.Tensor.of_expr:\ununexpected\underlinequoted\underlineqvariable\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\underlineq\un
```

```
name ^ "',")
\mid S. Variable name \rightarrow
          (* There should be a gatekeeper here or in A.of\_expr: *)
          Linear [(A.of\_expr name [], QC.unit)]
S.Application ("complex", [re; im]) \rightarrow
          begin match of expr re, of expr im with
          | Linear[([], re)], Linear[([], im)] \rightarrow
                    if QC.is\_real\ re\ \land\ QC.is\_real\ im then
                           Linear [([], QC.make (QC.real re) (QC.real im))]
                           invalid\_arg ("UFOx.Tensor.of_expr:\_argument\_of\_complex\_is\_complex")
                     invalid_arg "UFOx.Tensor.of_expr:unexpected_argumentuofucomplex"
 | S.Application (name, args) \rightarrow
          Linear [(A.of\_expr name args, QC.unit)]
      S.Sum\ (e1,\ e2) \rightarrow add\ (of\_expr\ e1)\ (of\_expr\ e2)
     S.Difference\ (e1,\ e2)\ 	o
          add\ (of\_expr\ e1)\ (of\_expr\ (S.Product\ (S.Integer\ (-1),\ e2)))
      S.Product\ (e1,\ e2) \rightarrow multiply\ (of\_expr\ e1)\ (of\_expr\ e2)
 | S.Quotient(n, d) \rightarrow
          begin match of \_expr n, of \_expr d with
          \mid n, Linear [] \rightarrow
                    invalid\_arg "UFOx.Tensor.of_expr:\uzero\udenominator"
           [n, Linear [([], q)] \rightarrow map\_coeff (fun c \rightarrow QC.div c q) n]
          \mid n, Linear ([(invertibles, q)] \text{ as } d) \rightarrow
                    if List.for_all A.invertible invertibles then
                          let inverses = List.map A.invert invertibles in
                           multiply (Linear [(inverses, QC.inv q)]) n
                    else
                           multiply (Ratios [[([], QC.unit)], d]) n
          \mid n, (Linear \ d \ \mathsf{as} \ d') \rightarrow
                    if List.for\_all (fun (t, \_) \rightarrow List.for\_all A.scalar t) d then
                           multiply (Ratios [[([], QC.unit)], d]) n
                    else
                           invalid\_arg ("UFOx.Tensor.of_expr:\_non\_scalar\_denominator:\_\" \hat{}
                                                                             to\_string d'
          \mid n, (Ratios \_ as d) \rightarrow
                     invalid\_arg ("UFOx.Tensor.of_expr:_{\sqcup}illegal_{\sqcup}denominator:_{\sqcup}" \hat{}
                                                                       to\_string d)
          end
\mid S.Power(e, p) \rightarrow
          begin match of\_expr\ e,\ of\_expr\ p with
          | Linear [([], q)], Linear [([], p)] \rightarrow
                    if QC.is\_real p then
                          let re_p = QC.real p in
                          if Q.is\_integer re\_p then
                                  Linear [([], QC.pow \ q \ (Q.to\_integer \ re\_p))]
                          else
                                  invalid\_arg "UFOx.Tensor.of_expr:\Boxrational\Boxpower\Boxof\Boxnumber"
                    else
                           invalid\_arg "UFOx.Tensor.of_expr:\u00cdcomplex\u00cdpower\u00cdof\u00cdnumber"
          |Linear[([], q)], \_ \rightarrow
                    invalid\_arg "UFOx.Tensor.of_expr:\underlinen-numeric\uppower\uppower\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppomer\uppome
          t, Linear [([], p)] \rightarrow
                    if QC.is\_integer p then
                           power (Q.to\_integer (QC.real p)) t
                   else
                           invalid\_arg "UFOx.Tensor.of_expr:\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under\under
           | \_ \rightarrow invalid\_arg "UFOx.Tensor.of_expr:\underline{\tensor}non\underline{\tensor}numeric\underline{\tensor}power\underline{\tensor}of\underline{\tensor}tensor"
          end
```

```
type r = A.r
     let rep_to_string = A.rep_to_string
     {\tt let} \ \mathit{rep\_to\_string\_whizard} \ = \ \mathit{A.rep\_to\_string\_whizard}
     let rep_of_int = A.rep_of_int
     let rep\_conjugate = A.rep\_conjugate
     let rep_trivial = A.rep_trivial
     let numerators = function
          \mathit{Linear\ tensors}\ \rightarrow\ \mathit{tensors}
          Ratios \ ratios \rightarrow \ ThoList.flatmap \ fst \ ratios
     let classify_indices' filter tensors =
            ThoList.uniq
              (List.sort compare
                  (List.map
                      (\text{fun }(t,\ c)\ 	o\ filter\ (A.classify\_indices\ t))
                      (numerators \ tensors)))
NB: the number of summation indices is not guarateed to be the same! Therefore it was foolish to try to check
for uniqueness ...
     let \ \mathit{classify\_indices} \ \mathit{tensors} \ =
       match classify_indices' Index.free tensors with
           (* There's always at least an empty list! *)
           failwith "UFOx.Tensor.classify_indices: can't happen!"
         [f] \rightarrow f
         _{-} \rightarrow
           invalid\_arg "UFOx.Tensor.classify_indices:\sqcupincompatible\sqcupfree\sqcupindices!"
     let disambiguate\_indices1 (atoms, q) =
        (A.disambiguate\_indices \ atoms, \ q)
     let \ disambiguate\_indices \ tensors =
        map_ratios (List.map disambiguate_indices1) tensors
     let check\_indices\ t\ =
        ignore\ (classify\_indices\ t)
     let of_expr e =
       let t = disambiguate\_indices (of\_expr e) in
        check\_indices\ t;
     let of string s =
        of\_expr(Expr.of\_string\ s)
     let of\_strings \ s =
        of\_expr(Expr.of\_strings\ s)
     type r\_omega = A.r\_omega
     let omega = A.omega
module type Lorentz\_Atom =
  sig
     type dirac = private
          C 	ext{ of } int 	imes int
          Gamma of int \times int \times int
          Gamma5 of int \times int
          Identity of int \times int
          ProjP of int \times int
          ProjM of int \times int
         Sigma 	ext{ of } int 	imes int 	imes int 	imes int
     type vector = (* private *)
        | Epsilon of int \times int \times int \times int
```

```
Metric of int \times int
          P 	ext{ of } int \times int
     type scalar = (* private *)
          Mass of int
           Width of int
          P2 of int
          P12 of int \times int
          Variable of string
          Coeff of Value.t
     type t = (* private *)
          Dirac of dirac
           Vector of vector
          Scalar of scalar
          Inverse of scalar
     val\ map\_indices\_scalar : (int \rightarrow int) \rightarrow scalar \rightarrow scalar
     val\ map\_indices\_vector: (int \rightarrow int) \rightarrow vector \rightarrow vector
     val\ rename\_indices\_vector\ :\ (int 
ightarrow\ int)\ 
ightarrow\ vector\ 
ightarrow\ vector
  end
module Lorentz\_Atom =
  struct
     type dirac =
          C of int \times int
          Gamma of int \times int \times int
          Gamma5 of int \times int
          Identity of int \times int
          ProjP of int \times int
          ProjM of int \times int
          Sigma 	ext{ of } int 	imes int 	imes int 	imes int
     type vector =
          Epsilon of int \times int \times int \times int
          Metric of int \times int
          P of int \times int
     type scalar =
          Mass of int
          Width of int
          P2 of int
          P12 of int \times int
          Variable of string
          Coeff of Value.t
     type t =
          Dirac of dirac
           Vector of vector
          Scalar of scalar
          Inverse of scalar
     let map\_indices\_scalar f = function
          Mass i \rightarrow Mass (f i)
           Width \ i \rightarrow Width \ (f \ i)
          P2 i \rightarrow P2 (f i)
          P12(i, j) \rightarrow P12(f i, f j)
          (Variable \_ | Coeff \_ as s) \rightarrow s
     let \ map\_indices\_vector \ f \ = \ function
          Epsilon (mu, nu, ka, la) \rightarrow Epsilon (f mu, f nu, f ka, f la)
          Metric\ (mu,\ nu)\ 	o\ Metric\ (f\ mu,\ f\ nu)
          P(mu, n) \rightarrow P(fmu, fn)
     let rename\_indices\_vector f = function
```

```
Epsilon (mu, nu, ka, la) \rightarrow Epsilon (f mu, f nu, f ka, f la)
          Metric\ (mu,\ nu)\ 	o\ Metric\ (f\ mu,\ f\ nu)
         P(mu, n) \rightarrow P(fmu, n)
  end
module Lorentz\_Atom' : Atom
  with type t = Lorentz\_Atom.t and type r\_omega = Coupling.lorentz =
     type t = Lorentz\_Atom.t
     open Lorentz_Atom
     let map\_indices\_dirac\ f = function
          C(i, j) \rightarrow C(f i, f j)
          Gamma\ (mu,\ i,\ j)\ \rightarrow\ Gamma\ (f\ mu,\ f\ i,\ f\ j)
          Gamma5\ (i,\ j)\ \rightarrow\ Gamma5\ (f\ i,\ f\ j)
          Identity(i, j) \rightarrow Identity(f i, f j)
          ProjP(i, j) \rightarrow ProjP(f i, f j)
          ProjM(i, j) \rightarrow ProjM(f i, f j)
         Sigma\ (mu,\ nu,\ i,\ j) \rightarrow Sigma\ (f\ mu,\ f\ nu,\ f\ i,\ f\ j)
     let rename_indices_dirac = map_indices_dirac
     let map\_indices\_scalar f = function
          Mass i \rightarrow Mass (f i)
          Width \ i \rightarrow Width \ (f \ i)
          P2 i \rightarrow P2 (f i)
          P12(i, j) \rightarrow P12(f i, f j)
          Variable \ s \ 	o \ Variable \ s
          Coeff \ c \rightarrow Coeff \ c
     let map\_indices f = function
          Dirac\ d \rightarrow Dirac\ (map\_indices\_dirac\ f\ d)
          Vector \ v \rightarrow Vector \ (map\_indices\_vector \ f \ v)
          Scalar s \rightarrow Scalar (map\_indices\_scalar f s)
          Inverse \ s \rightarrow Inverse \ (map\_indices\_scalar \ f \ s)
     let rename\_indices2 fd fv = function
          Dirac\ d \rightarrow Dirac\ (rename\_indices\_dirac\ fd\ d)
          Vector \ v \rightarrow Vector \ (rename\_indices\_vector \ fv \ v)
          Scalar s \rightarrow Scalar s
          Inverse \ s \rightarrow Inverse \ s
     let rename\_indices\ f\ atom\ =
        rename_indices2 f f atom
     let contract_pair\ a1\ a2\ =
       match a1, a2 with
        | Vector (P (mu1, i1)), Vector (P (mu2, i2)) \rightarrow
           if mu1 \leq 0 \wedge mu1 = mu2 then
              if i1 = i2 then
                 Some (Scalar (P2 i1))
              else
                 Some (Scalar (P12 (i1, i2)))
            else
              None
        \mid Scalar \ s, \ Inverse \ s' \mid Inverse \ s, \ Scalar \ s' \rightarrow
           \text{if } s \ = \ s' \ \text{then} \\
              Some (Scalar (Coeff (Value.Integer 1)))
            else
              None
       \downarrow \rightarrow None
     let \ variable = function
       | Scalar (Variable s) | Inverse (Variable s) \rightarrow Some s
```

```
 \bot \rightarrow None 
let scalar = function
   | Dirac \_ | Vector \_ \rightarrow false
   \mid Scalar \_ \mid Inverse \_ \rightarrow true
let is\_unit = function
  \mid Scalar (Coeff c) \mid Inverse (Coeff c) \rightarrow
       begin match \it c with
         Value.Integer 1 \rightarrow true
         Value.Rational\ q \rightarrow Algebra.Q.is\_unit\ q
       \perp \rightarrow false
       end
  |  \rightarrow false
let invertible = scalar
let invert = function
    Dirac \_ \rightarrow invalid\_arg "UFOx.Lorentz_Atom.invert_Dirac"
     Vector \_ \rightarrow invalid\_arg "UFOx.Lorentz_Atom.invert_\UVector"
     Scalar \ s \ 	o \ Inverse \ s
     Inverse \ s \rightarrow Scalar \ s
let i2s = Index.to\_string
\mathsf{let}\ \mathit{dirac\_to\_string}\ =\ \mathsf{function}
   C(i, j) \rightarrow
       Printf.sprintf "C(%s,%s)" (i2s i) (i2s j)
  \mid Gamma\ (mu,\ i,\ j) \rightarrow
       Printf.sprintf \; \texttt{"Gamma(\%s,\%s,\%s)"} \; (i2s \; mu) \; (i2s \; i) \; (i2s \; j)
  \mid Gamma5 (i, j) \rightarrow
       Printf.sprintf "Gamma5(%s,%s)" (i2s \ i) \ (i2s \ j)
  | Identity (i, j) \rightarrow
       Printf.sprintf "Identity(%s,%s)" (i2s \ i) (i2s \ j)
   | ProjP(i, j) \rightarrow
       Printf.sprintf "ProjP(%s,%s)" (i2s \ i) (i2s \ j)
   \mid ProjM(i, j) \rightarrow
       Printf.sprintf "ProjM(%s,%s)" (i2s \ i) (i2s \ j)
   Sigma\ (mu,\ nu,\ i,\ j) \rightarrow
       Printf.sprintf "Sigma(%s,%s,%s,%s)" (i2s mu) (i2s nu) (i2s i) (i2s j)
let vector\_to\_string = function
   | Epsilon (mu, nu, ka, la) \rightarrow
       Printf.sprintf "Epsilon(%s,%s,%s,%s)" (i2s mu) (i2s nu) (i2s ka) (i2s la)
    Metric\ (mu,\ nu)\ 	o
       Printf.sprintf "Metric(%s,%s)" (i2s mu) (i2s nu)
   P(mu, n) \rightarrow
       Printf.sprintf "P(%s,%d)" (i2s mu) n
let scalar\_to\_string = function
     Mass\ id \rightarrow Printf.sprintf "Mass(%d)" id
     Width id \rightarrow Printf.sprintf "Width(%d)" id
     P2 id \rightarrow Printf.sprintf "P(%d)**2" id
     P12 \ (id1, id2) \rightarrow Printf.sprintf "P(%d)*P(%d)" id1 id2
     Variable s \rightarrow s
     Coeff \ c \rightarrow Value.to\_string \ c
\mathsf{let}\ to\_string\ =\ \mathsf{function}
    Dirac\ d\ 	o\ dirac\_to\_string\ d
     Vector v \rightarrow vector\_to\_string v
     Scalar s \rightarrow scalar\_to\_string s
     Inverse s \rightarrow "1/" \hat{s} calar\_to\_string s
module S = UFOx\_syntax
```



Here we handle some special cases in order to be able to parse propagators. This needs to be made more general, but unfortunately the syntax for the propagator extension is not well documented and appears to be a bit chaotic!

```
let quoted\_index s =
   Index.named\_summation s ()
let integer\_or\_id = function
    S.Integer n \rightarrow n
    S.Variable "id" 
ightarrow 1
    \_ \rightarrow failwith "UFOx.Lorentz_Atom.integer_or_id:\_impossible"
let \ vector\_index = function
    S.Integer n \rightarrow n
    S.Quoted\ mu\ 	o\ quoted\_index\ mu
    S.Variable\ id \rightarrow
      \mathsf{let}\ l\ =\ \mathit{String.length}\ \mathit{id}\ \mathsf{in}
      if l > 1 then
        if id.[0] = '1' then
           int_of_string (String.sub id 1 (pred l))
         else
           invalid_arg ("UFOx.Lorentz_Atom.vector_index:" ^ id)
      else
         invalid_arg "UFOx.Lorentz_Atom.vector_index: uempty uvariable"
  | _ → invalid_arg "UFOx.Lorentz_Atom.vector_index"
let spinor\_index = function
    S.Integer n \rightarrow n
   S.Variable\ id \rightarrow
      let \ l = String.length \ id \ in
      if l > 1 then
        if id.[0] = 's' then
           int\_of\_string\ (String.sub\ id\ 1\ (pred\ l))
         else
           invalid_arg ("UFOx.Lorentz_Atom.spinor_index:□" ^ id)
      else
         invalid\_arg "UFOx.Lorentz_Atom.spinor_index:\_empty\_variable"
  _ → invalid_arg "UFOx.Lorentz_Atom.spinor_index"
let of\_expr\ name\ args\ =
  match name, args with
     "C", [i; j] \rightarrow [Dirac (C (spinor\_index i, spinor\_index j))]
     "C", \_ \rightarrow
      invalid_arg "UFOx.Lorentz.of_expr: uinvalid_arguments to C()"
    "Epsilon", [mu; nu; ka; la] \rightarrow
      [Vector (Epsilon (vector_index mu, vector_index nu,
                             vector_index ka, vector_index la))]
    "Epsilon", \_ \rightarrow
      invalid_arg "UFOx.Lorentz.of_expr:uinvaliduargumentsutouEpsilon()"
     \texttt{"Gamma"}, \ [mu; \ i; \ j] \ \rightarrow
      [Dirac\ (Gamma\ (vector\_index\ mu,\ spinor\_index\ i,\ spinor\_index\ j))]
     "Gamma", _{-} \rightarrow
      invalid_arg "UFOx.Lorentz.of_expr:_invalid_arguments_to_Gamma()"
     "Gamma5", [i; j] \rightarrow [Dirac (Gamma5 (spinor\_index i, spinor\_index j))]
     "Gamma5", _{-} \rightarrow
      invalid_arg "UFOx.Lorentz.of_expr:_invalid_arguments_to_Gamma5()"
     "Identity", [i; j] \rightarrow [Dirac\ (Identity\ (spinor\_index\ i,\ spinor\_index\ j))]
     "Identity", _ \rightarrow
      invalid\_arg "UFOx.Lorentz.of_expr:\sqcupinvalid\sqcuparguments\sqcupto\sqcupIdentity()"
     "Metric", [mu; nu] \rightarrow [Vector (Metric (vector\_index mu, vector\_index nu))]
    "Metric", \_ \rightarrow
      invalid\_arg "UFOx.Lorentz.of_expr:_{\square}invalid_{\square}arguments_{\square}to_{\square}Metric()"
    "P", [mu; id] \rightarrow [Vector (P (vector\_index mu, integer\_or\_id id))]
```

```
| "P", _{-} \rightarrow
      invalid_arg "UFOx.Lorentz.of_expr:_invalid_arguments_ito_P()"
     "ProjP", [i; j] \rightarrow [Dirac\ (ProjP\ (spinor\_index\ i,\ spinor\_index\ j))]
     "ProjP", \rightarrow
      invalid_arg "UFOx.Lorentz.of_expr:uinvalid_arguments_to_ProjP()"
     "ProjM", [i; j] \rightarrow [Dirac (ProjM (spinor\_index i, spinor\_index j))]
     "ProjM", \_ \rightarrow
      invalid\_arg "UFOx.Lorentz.of_expr:\sqcupinvalid\sqcuparguments\sqcupto\sqcupProjM()"
    "Sigma", [mu;\ nu;\ i;\ j]
       if mu \neq nu then
         [Dirac (Sigma (vector_index mu, vector_index nu,
                             spinor\_index i, spinor\_index j))
       else
         invalid_arg "UFOx.Lorentz.of_expr:uimplausibleuargumentsutouSigma()"
   \mid "Sigma", \_ \rightarrow
       invalid_arg "UFOx.Lorentz.of_expr:uinvaliduargumentsutouSigma()"
   | "PSlash", [i; j; id] \rightarrow
      let mu = Index.fresh\_summation () in
       [Dirac\ (Gamma\ (mu,\ spinor\_index\ i,\ spinor\_index\ j));
        Vector\ (P\ (mu,\ integer\_or\_id\ id))]
     "PSlash", \_ \rightarrow
      invalid_arg "UFOx.Lorentz.of_expr:_invalid_arguments_to_PSlash()"
     "Mass", [id] \rightarrow [Scalar (Mass (integer\_or\_id id))]
     "Mass", _ 
ightarrow
       invalid_arg "UFOx.Lorentz.of_expr: invalid arguments to Mass()"
     "Width", [id] \rightarrow [Scalar (Width (integer\_or\_id id))]
     "Width", _{-} \rightarrow
       invalid_arg "UFOx.Lorentz.of_expr:_invalid_arguments_to_Width()"
   \mid name, [] \rightarrow
       [Scalar (Variable name)]
   | name, \_ \rightarrow
       invalid_arg ("UFOx.Lorentz.of_expr:_invalid_tensor_'," ^ name ^ ",")
type r = S \mid V \mid T \mid Sp \mid CSp \mid Maj \mid VSp \mid CVSp \mid VMaj \mid Ghost
let rep_trivial = function
  \mid S \mid Ghost \rightarrow true
   \mid V \mid T \mid Sp \mid CSp \mid Maj \mid VSp \mid CVSp \mid VMaj \rightarrow \mathsf{false}
\mathsf{let}\ \mathit{rep\_to\_string}\ =\ \mathsf{function}
   \mid S \rightarrow "0"
     V 
ightarrow "1"
     T \rightarrow "2"
     Sp \rightarrow "1/2"
     CSp \rightarrow "1/2bar"
     Maj \rightarrow "1/2M"
     VSp \rightarrow "3/2"
     \mathit{CVSp} \rightarrow "3/2bar"
     VMaj \rightarrow "3/2M"
     Ghost \rightarrow \texttt{"Ghost"}
let rep\_to\_string\_whizard = function
    S \rightarrow \text{"0"}
     V 
ightarrow "1"
     T 	o "2"
     Sp \mid CSp \mid Maj \rightarrow "1/2"
     VSp \mid CVSp \mid VMaj \rightarrow "3/2"
     Ghost \rightarrow \texttt{"Ghost"}
let rep\_of\_int \ neutral = function
   |-1 \rightarrow Ghost|
     1 \ \to \ S
    2 \rightarrow \text{if } neutral \text{ then } Maj \text{ else } Sp
    -2 \rightarrow \text{if } neutral \text{ then } Maj \text{ else } CSp \text{ (* used by } UFO.Particle.force\_conjspinor *)}
```

```
\mid 3 \rightarrow V
    4 \rightarrow \text{if } neutral \text{ then } VMaj \text{ else } VSp
     -4 \rightarrow if neutral then VMaj else CVSp (* used by UFO.Particle.force_conjspinor *)
    5 \rightarrow T
    s when s > 0 \rightarrow
      failwith "UFOx.Lorentz:\_spin_{\square}>_{\square}2_{\square}not_{\square}supported!"
      invalid\_arg "UFOx.Lorentz:\sqcupinvalid\sqcupnon-positive\sqcupspin\sqcupvalue"
let rep\_conjugate = function
   S \rightarrow S
     V \rightarrow V
     T \rightarrow T
     Sp \rightarrow CSp (*???*)
     CSp \rightarrow Sp \ (*~???~*)
     Maj \rightarrow Maj
     VSp \rightarrow CVSp
     CVSp \rightarrow VSp
     VMaj \rightarrow VMaj
     Ghost \rightarrow Ghost
let classify\_vector\_indices1 = function
     Epsilon (mu, nu, ka, la) \rightarrow [(mu, V); (nu, V); (ka, V); (la, V)]
     Metric\ (mu,\ nu)\ \rightarrow\ [(mu,\ V);\ (nu,\ V)]
   P(mu, n) \rightarrow [(mu, V)]
let classify\_dirac\_indices1 = function
     C(i, j) \rightarrow [(i, CSp); (j, Sp)] (*???*)
     Gamma5 (i, j) \mid Identity (i, j)
     ProjP(i, j) \mid ProjM(i, j) \rightarrow [(i, CSp); (j, Sp)]
     Gamma\ (mu,\ i,\ j) \rightarrow [(mu,\ V);\ (i,\ CSp);\ (j,\ Sp)]
     Sigma\ (mu,\ nu,\ i,\ j) \rightarrow [(mu,\ V);\ (nu,\ V);\ (i,\ CSp);\ (j,\ Sp)]
let classify\_indices1 = function
     Dirac\ d \rightarrow classify\_dirac\_indices1\ d
     Vector \ v \ 
ightarrow \ classify\_vector\_indices1 \ v
     module IMap = Map.Make (struct type t = int let compare = compare end)
exception Incompatible\_factors of r \times r
let product rep1 rep2 =
  match rep1, rep2 with
     V, V \rightarrow T
     V, Sp \rightarrow VSp
     V,~CSp~\rightarrow~CVSp
     V, Maj \rightarrow VMaj
     Sp, \ V \ \rightarrow \ VSp
     CSp, V \rightarrow CVSp
    Maj, V \rightarrow VMaj
   | \_, \_ \rightarrow raise (Incompatible\_factors (rep1, rep2))
let combine\_or\_add\_index\ (i,\ rep)\ map\ =
  let pos, fac = Index.unpack i in
  try
     let fac', rep' = IMap.find pos map in
     if pos < 0 then
        IMap.add pos (fac, rep) map
     else if fac \neq fac' then
        IMap.add pos (0, product rep rep') map
     else if rep \neq rep' then (* Can be disambiguated! *)
        IMap.add pos (0, product rep rep') map
     else
        invalid\_arg\ (Printf.sprintf\ "UFO:\_duplicate\_subindex\_\%d"\ pos)
```

```
with
         Not\_found \rightarrow IMap.add pos (fac, rep) map
         Incompatible\_factors\ (rep1,\ rep2)\ 	o
           invalid\_arg
             (Printf.sprintf
                 "UFO: uincompatible of factors (%s, %s) at \d"
                 (rep_to_string rep1) (rep_to_string rep2) pos)
    let combine\_or\_add\_indices atom map =
       List.fold\_right\ combine\_or\_add\_index\ (classify\_indices1\ atom)\ map
    let project\_factors (pos, (fac, rep)) =
       if fac = 0 then
          (pos, rep)
       else
          invalid_arg (Printf.sprintf "UFO: Lleftover Lsubindex L%d.%d" pos fac)
    let \ classify\_indices \ atoms =
       List.map
          project_factors
          (IMap.bindings (List.fold_right combine_or_add_indices atoms IMap.empty))
    let \ add\_factor \ fac \ indices \ pos =
       if pos > 0 then
         if Sets.Int.mem pos indices then
            Index.pack pos fac
          else
            pos
       else
          pos
    let \ disambiguate\_indices1 \ indices \ atom =
       rename_indices2 (add_factor 1 indices) (add_factor 2 indices) atom
     let vectorspinors atoms =
       List.fold\_left
          (fun acc\ (i,\ r)\ \rightarrow
            match r with
              S \mid V \mid T \mid Sp \mid CSp \mid Maj \mid Ghost \rightarrow acc
              VSp \mid CVSp \mid VMaj \rightarrow Sets.Int.add \ i \ acc)
          Sets.Int.empty (classify_indices atoms)
    let \ disambiguate\_indices \ atoms =
       let \ vectorspinor\_indices = vectorspinors \ atoms \ in
       List.map (disambiguate_indices1 vectorspinor_indices) atoms
    type r\_omega = Coupling.lorentz
    let omega = function
         S \rightarrow Coupling.Scalar
          V \rightarrow Coupling. Vector
         T \rightarrow Coupling.Tensor\_2
         Sp \rightarrow Coupling.Spinor
         CSp \rightarrow Coupling.ConjSpinor
         Maj \rightarrow Coupling.Majorana
          VSp \rightarrow Coupling. Vectorspinor
         CVSp \rightarrow Coupling.Vectorspinor (* TODO: not really! *)
          VMaj \rightarrow Coupling. Vectorspinor (* TODO: not really! *)
         Ghost \rightarrow Coupling.Scalar
  end
module Lorentz = Tensor(Lorentz\_Atom')
module type Color\_Atom =
  sig
     type t = (* private *)
       | Identity of int \times int
```

```
Identity8 of int \times int
           T 	ext{ of } int \times int \times int
           F of int \times int \times int
           D of int \times int \times int
           \textit{Epsilon} \ \text{of} \ \textit{int} \ 	imes \ \textit{int} \ 	imes \ \textit{int}
           EpsilonBar of int \times int \times int
           T6 	ext{ of } int \times int \times int
           K6 of int \times int \times int
           K6Bar of int \times int \times int
  end
\mathsf{module}\ \mathit{Color} \_\mathit{Atom}\ =
  struct
     type t =
           Identity of int \times int
           Identity8 of int \times int
           T of int \times int \times int
           F of int \times int \times int
           D of int \times int \times int
           Epsilon of int \times int \times int
           EpsilonBar of int \times int \times int
           T6 	ext{ of } int \times int \times int
           K6 	ext{ of } int \times int \times int
           K6Bar of int \times int \times int
  end
module\ Color\_Atom'\ :\ Atom
  with type t = Color\_Atom.t and type r\_omega = Color.t =
  struct
     type t = Color\_Atom.t
     module S = UFOx\_syntax
     open Color_Atom
     let map\_indices f = function
           Identity(i, j) \rightarrow Identity(f i, f j)
           Identity8\ (a,\ b)\ 	o\ Identity8\ (f\ a,\ f\ b)
           T(a, i, j) \rightarrow T(f a, f i, f j)
           F(a, i, j) \rightarrow F(f a, f i, f j)
           D(a, i, j) \rightarrow D(f a, f i, f j)
           Epsilon (i, j, k) \rightarrow Epsilon (f i, f j, f k)
           EpsilonBar~(i,~j,~k)~\rightarrow~EpsilonBar~(f~i,~f~j,~f~k)
           T6 (a, i', j') \rightarrow T6 (f a, f i', f j')
           K6 (i', j, k) \rightarrow K6 (f i', f j, f k)
          K6Bar(i', j, k) \rightarrow K6Bar(f i', f j, f k)
     let rename\_indices = map\_indices
     let contract_pair_{-} = None
     let variable _ = None
     let scalar _ = false
     let invertible _ = false
     let is\_unit\_ = false
     let invert _ =
        invalid_arg "UFOx.Color_Atom.invert"
     let of\_expr1 name args =
        match name, args with
           "Identity", [S.Integer\ i;\ S.Integer\ j] \rightarrow Identity\ (i,\ j)
           "Identity", \_ \rightarrow
            invalid_arg "UFOx.Color.of_expr: uinvalid arguments to Identity()"
           "T", [S.Integer\ a;\ S.Integer\ i;\ S.Integer\ j] \rightarrow T\ (a,\ i,\ j)
          "T", \_ \rightarrow
```

```
invalid\_arg "UFOx.Color.of_expr:\sqcupinvalid\sqcuparguments\sqcupto\sqcupT()"
     "f", [S.Integer\ a;\ S.Integer\ b;\ S.Integer\ c] \rightarrow F\ (a,\ b,\ c)
     "f", \_ \rightarrow
      invalid_arg "UFOx.Color.of_expr:_invalid_arguments_to_f()"
     "d", [S.Integer\ a;\ S.Integer\ b;\ S.Integer\ c]\ 	o\ D\ (a,\ b,\ c)
       invalid\_arg "UFOx.Color.of_expr:\sqcupinvalid\sqcuparguments\sqcupto\sqcupd()"
     "Epsilon", [S.Integer i; S.Integer j; S.Integer k] \rightarrow
       Epsilon (i, j, k)
     "Epsilon", \_ \rightarrow
       invalid\_arg "UFOx.Color.of_expr:\sqcupinvalid\sqcuparguments\sqcupto\sqcupEpsilon()"
     "EpsilonBar", [S.Integer\ i;\ S.Integer\ j;\ S.Integer\ k] \rightarrow
       EpsilonBar(i, j, k)
     "EpsilonBar", \_ \rightarrow
       invalid\_arg "UFOx.Color.of_expr:\sqcupinvalid\sqcuparguments\sqcupto\sqcupEpsilonBar()"
     "T6", [S.Integer\ a;\ S.Integer\ i';\ S.Integer\ j'] \rightarrow T6\ (a,\ i',\ j')
     "T6", \_ \rightarrow
      invalid_arg "UFOx.Color.of_expr:_invalid_arguments_to_T6()"
     "K6", [S.Integer\ i';\ S.Integer\ j;\ S.Integer\ k] \rightarrow K6\ (i',\ j,\ k)
     "K6", \rightarrow
      invalid_arg "UFOx.Color.of_expr:_invalid_arguments_ito_K6()"
     "K6Bar", [S.Integer i'; S.Integer j; S.Integer k] \rightarrow K6Bar (i', j, k)
     "K6Bar", \_ \rightarrow
       invalid\_arg "UFOx.Color.of_expr:_\(\_invalid\_arguments_\)\(\_ito_\)K6Bar()"
   name, \_ \rightarrow
       invalid_arg ("UFOx.Color.of_expr:\(\_\)invalid\(\_\)tensor\(\_\)'\" \(^\) name \(^\)\"'\")
let of\_expr\ name\ args\ =
   [of\_expr1 name args]
let to\_string = function
     Identity\ (i,\ j)\ 	o\ Printf.sprintf "Identity(%d,%d)" i\ j
     Identity8\ (a,\ b)\ 	o\ Printf.sprintf "Identity8(%d,%d)" a\ b
     T~(a,~i,~j)~
ightarrow~Printf.sprintf "T(%d,%d,%d)" a~i~j
     F(a, b, c) \rightarrow Printf.sprintf "f(%d,%d,%d)" a b c
     D(a, b, c) \rightarrow Printf.sprintf "d(%d,%d,%d)" a b c
     Epsilon(i, j, k) \rightarrow Printf.sprintf "Epsilon(%d,%d,%d)" i j k
     EpsilonBar(i, j, k) \rightarrow Printf.sprintf "EpsilonBar(%d,%d,%d)" i j k
     T6~(a,~i',~j')~\rightarrow~Printf.sprintf~"\text{T6 (%d,%d,%d)}"~a~i'~j'
     K6 (i', j, k) \rightarrow Printf.sprintf "K6(%d,%d,%d)" i' j k
    K6Bar(i', j, k) \rightarrow Printf.sprintf "K6Bar(%d,%d,%d)" i'jk
\mathsf{type}\ r\ =\ S\ \mid\ F\ \mid\ C\ \mid\ A
let rep_trivial = function
  \mid S \rightarrow \mathsf{true}
  \mid F \mid C \mid A \rightarrow \mathsf{false}
\mathsf{let}\ \mathit{rep\_to\_string}\ =\ \mathsf{function}
  \mid S \rightarrow "1"
     F 	o "3"
     C 	o "3bar"
    A 
ightarrow "8"
let rep\_to\_string\_whizard = function
  \mid S \rightarrow "1"
    F 	o "3"
     C 	o "-3"
     A 
ightarrow "8"
let rep\_of\_int \ neutral = function
   1 \rightarrow S
     3 \ \to \ F
     -3 \rightarrow C
   \mid 8 \rightarrow A
```

```
6 \mid -6 \rightarrow failwith "UFOx.Color: \_sextets\_not\_supported\_yet!"
        \mid 10 \mid -10 \rightarrow failwith "UFOx.Color: udecuplets unot usupported uyet!"
       \mid n \rightarrow
           invalid\_arg
              (Printf.sprintf
                  "UFOx.Color:\sqcupimpossible\sqcuprepresentation\sqcupcolor\sqcup=\sqcup%d!" n)
     \mathsf{let}\ \mathit{rep\_conjugate}\ =\ \mathsf{function}
         S \rightarrow S
          C \rightarrow F
          F \rightarrow C
          A \rightarrow A
     let classify\_indices1 = function
          Identity (i, j) \rightarrow [(i, C); (j, F)]
          Identity8 (a, b) \rightarrow [(a, A); (b, A)]
          T(a, i, j) \rightarrow [(i, F); (j, C); (a, A)]
          Color\_Atom.F\ (a,\ b,\ c)\ |\ D\ (a,\ b,\ c)\ \to\ [(a,\ A);\ (b,\ A);\ (c,\ A)]
          Epsilon (i, j, k) \rightarrow [(i, F); (j, F); (k, F)]
          EpsilonBar (i, j, k) \rightarrow [(i, C); (j, C); (k, C)]
          T6 (a, i', j') \rightarrow
           failwith "UFOx.Color: usextets unot usupported uyet!"
          K6 (i', j, k) \rightarrow
           failwith "UFOx.Color: usextets unot usupported uyet!"
         K6Bar(i', j, k) \rightarrow
           failwith "UFOx.Color: usextets not supported yet!"
     let classify\_indices tensors =
        List.sort compare
          (List.fold\_right
              (fun \ v \ acc \rightarrow \ classify\_indices1 \ v \ @ \ acc)
              tensors [])
     let \ disambiguate\_indices \ atoms =
        atoms
     \mathsf{type}\ r\_omega\ =\ Color.t
FIXME: N_C = 3 should not be hardcoded!
     let omega = function
         S \rightarrow Color.Singlet
          F \rightarrow Color.SUN (3)
          C \rightarrow Color.SUN (-3)
         A \rightarrow Color.AdjSUN (3)
  end
module\ Color\ =\ Tensor(Color\_Atom')
module type Test =
  sig
     val\ example\ :\ unit 
ightarrow\ unit
     \mathsf{val}\ suite\ :\ OUnit.test
  end
                                      14.7 Interface of UFO_syntax
                                               14.7.1 Abstract Syntax
exception Syntax\_Error of string \times Lexing.position \times Lexing.position
type name = string \ list
type string\_atom =
  | Macro of name
```

```
| Literal of string
type value =
    Name of name
    Integer of int
    Float of float
    Fraction of int \times int
    String of string
    String_Expr of string_atom list
    Empty\_List
    Name_List of name list
    Integer_List of int list
    String\_List of string list
    Order\_Dictionary of (string \times int) list
    Coupling\_Dictionary of (int \times int \times name) list
    Decay\_Dictionary of (name\ list \times string)\ list
type attrib =
  \{a\_name : string;
    a\_value : value }
type declaration =
  \{ name : string; 
    kind: name;
     attribs : attrib list }
type t = declaration list
A macro expansion is encoded as a special declaration, with kind = "\$" and a single attribute. There should
not never be the risk of a name clash.
\mathsf{val}\ macro\ :\ string \to\ value\ \to\ declaration
val\ to\_strings : t \rightarrow string\ list
                              14.8 Implementation of UFO_syntax
                                            14.8.1 Abstract Syntax
exception Syntax\_Error of string \times Lexing.position \times Lexing.position
type name = string \ list
type string\_atom =
    {\it Macro} of {\it name}
    Literal of string
type value =
    Name of name
    Integer of int
    Float of float
    Fraction of int \times int
    String of string
    String\_Expr of string\_atom list
    Empty\_List
    Name_List of name list
    Integer\_List of int\ list
    String_List of string list
    Order\_Dictionary of (string \times int) list
    Coupling\_Dictionary of (int \times int \times name) list
    Decay\_Dictionary of (name\ list \times string)\ list
type \ attrib =
  \{a\_name : string;
     a\_value : value }
```

Lexer

```
type \ declaration =
  \{ name : string; 
    kind : name;
    attribs : attrib list }
type t = declaration list
let macro name expansion =
  \{ name;
    kind = ["\$"];
    attribs = [ \{ a\_name = name; a\_value = expansion \} ] \}
let to\_strings\ declarations\ =
  14.9 Lexer
open Lexing
open UFO_parser
let string\_of\_char c =
  String.make\ 1\ c
let init\_position fname lexbuf =
  let curr_p = lexbuf.lex_curr_p in
  lexbuf.lex\_curr\_p \ \leftarrow
    \{ curr_p \text{ with } \}
      pos\_fname = fname;
      pos\_lnum = 1;
      pos\_bol = curr\_p.pos\_cnum };
  lexbuf
}
let digit = [,0,-,9,]
let upper = ['A'-'Z']
let \ lower = ['a'-'z']
let char = upper \mid lower
let word = char \mid digit \mid '_-'
let white = [', ', '\t']
let esc = [`,`,`,",`,`,`]
let crlf = ['\r', '\n']
let \ not\_crlf \ = \ [ ``\", \", \", \"]
rule token = parse
    white { token lexbuf } (* skip blanks *)
    '#' not_crlf* { token lexbuf } (* skip comments *)
    crlf { new_line lexbuf; token lexbuf }
    "from" not_crlf* { token lexbuf } (* skip imports *)
    "import" not_crlf* { token lexbuf } (* skip imports (for now) *)
    "try:" not\_crlf^* { token\ lexbuf } (* skip imports (for now) *)
    "except" not\_crlf^{\star} { token\ lexbuf } (* skip imports (for now) *)
    "pass" { token lexbuf } (* skip imports (for now) *)
    (' \{ LPAREN \})
    ')' { RPAREN }
    '{' { LBRACE }
    '}' { RBRACE }
    '[' \{ LBRACKET \}
    ']'
        \{RBRACKET\}
    '=' \{ EQUAL \}
    '+' { PLUS }
    '-' { MINUS }
    ',' { DIV }
```

```
'.' { DOT }
    ',' { COMMA }
    ':' { COLON }
    '-'? (digit^+'.' digit^* \mid digit^*'.' digit^+
          ( ['E''e']'-'? digit^+ )? as x
                         \{ FLOAT (float\_of\_string x) \}
    \cdot-\cdot? digit^+ as i \{ INT (int\_of\_string i) \}
    char \ word^* \ as \ s \ \{ \ ID \ s \ \}
    '\\', '[', (word^+ \text{ as } stem)']', (word^* \text{ as } suffix)
                         { ID (UFO_tools.mathematica_symbol stem suffix) }
  | ' ' ' ' |  let sbuf = Buffer.create 20 in
                            STRING (string1 sbuf lexbuf) }
  | ""  { let sbuf = Buffer.create 20 in
                            STRING (string2 sbuf lexbuf)
  \bot as c { raise (UFO\_tools.Lexical\_Error
                                      ("invalid_character_'" ^ string_of_char c ^ "',",
                                       lexbuf.lex\_start\_p, lexbuf.lex\_curr\_p)) }
  \mid eof \{END\}
and string1 sbuf = parse
    '\', \{Buffer.contents\ sbuf\ \}
    '\\' (esc \ as \ c) \ \{ \ Buffer.add\_char \ sbuf \ c; \ string1 \ sbuf \ lexbuf \ \}
    eof { raise End_of_file }
    '\\', '[' (word^+ \text{ as } stem) ']' (word^* \text{ as } suffix)
                         { Buffer.add_string
                              sbuf (UFO_tools.mathematica_symbol stem suffix);
                            string1 sbuf lexbuf }
  _ as c { Buffer.add_char sbuf c; string1 sbuf lexbuf }
and string2 \ sbuf = parse
    ", { Buffer.contents sbuf }
    eof { raise End_of_file }
    '\\', '[' (word^+ as stem) ']' (word^* as suffix)
                         { Buffer.add_string
                              sbuf (UFO_tools.mathematica_symbol stem suffix);
                            string2 sbuf lexbuf }
  \bot as c \in Buffer.add\_char\ sbuf\ c;\ string2\ sbuf\ lexbuf \}
```

14.10 Parser

Right recursion is more convenient for constructing the value. Since the lists will always be short, there is no performace or stack size reason for prefering left recursion.

Header

Token declarations

```
%token < int > INT
%token < float > FLOAT
```

```
\%token < string > STRING ID
%token DOT COMMA COLON
\%token EQUAL\ PLUS\ MINUS\ DIV
\%token LPAREN RPAREN
%token LBRACE RBRACE
%token LBRACKET RBRACKET
%token END
%start file
\%type < UFO\_syntax.t > file
                                           Grammar rules
file ::=
 | declarations END { $1 }
declarations ::=
 | { [] }
 | declaration declarations { $1 :: $2 }
declaration ::=
 | ID EQUAL name LPAREN RPAREN \{ \{ U.name = \$1; \} \}
                                                 U.kind = \$3;
                                                 U.attribs = [] \} 
| ID EQUAL name LPAREN attributes RPAREN { { U.name = $1;
                                                 U.kind = \$3;
                                                 U.attribs = \$5 \}
 | ID EQUAL STRING { U.macro $1 (U.String $3) }
 | ID EQUAL string_expr { U.macro $1 (U.String_Expr $3) }
name ::=
 | ID { [$1] }
 | name DOT ID { $3 :: $1 }
attributes ::=
  attribute \{ [\$1] \}
  attribute COMMA attributes { $1 :: $3 }
attribute ::=
 | ID EQUAL value \{ \{ U.a\_name = \$1; U.a\_value = \$3 \} \}
 | ID EQUAL list \{ \{ U.a\_name = \$1; U.a\_value = \$3 \} \}
 | ID EQUAL \ dictionary \{ \{ U.a\_name = \$1; \ U.a\_value = \$3 \} \}
value ::=
  INT \{ U.Integer \$1 \}
  INT \ DIV \ INT \ \{ \ U.Fraction \ (\$1, \ \$3) \ \}
  FLOAT \{ U.Float \$1 \}
  string \{ U.String \$1 \}
  string_expr { U.String_Expr $1 }
 | name \{ U.Name \$1 \}
list ::=
 \mid LBRACKET \mid RBRACKET \mid U.Empty\_List \mid
  LBRACKET names RBRACKET { U.Name_List $2 }
 | LBRACKET strings RBRACKET { U.String_List $2 }
 | LBRACKET integers RBRACKET { U.Integer_List $2 }
```

```
\begin{array}{l} \textit{dictionary} ::= \\ \mid \textit{LBRACE orders RBRACE} \left\{ \begin{array}{l} \textit{U.Order\_Dictionary} \$2 \right\} \\ \mid \textit{LBRACE couplings RBRACE} \left\{ \begin{array}{l} \textit{U.Coupling\_Dictionary} \$2 \right\} \\ \mid \textit{LBRACE decays RBRACE} \left\{ \begin{array}{l} \textit{U.Decay\_Dictionary} \$2 \right\} \\ \\ \textit{names} ::= \\ \mid \textit{name} \left\{ \begin{array}{l} \$1 \end{array} \right\} \\ \mid \textit{name COMMA names} \left\{ \begin{array}{l} \$1 :: \$3 \right\} \\ \\ \textit{integers} ::= \\ \mid \textit{INT} \left\{ \begin{array}{l} \$1 \end{array} \right\} \\ \mid \textit{INT COMMA integers} \left\{ \begin{array}{l} \$1 :: \$3 \right\} \\ \end{array} \right.
```

We demand that a $U.String_Expr$ contains no adjacent literal strings. Instead, they are concatenated already in the parser. Note that a $U.String_Expr$ must have at least two elements: singletons are parsed as U.Name or U.String instead.

```
string\_expr ::=
 | literal_string_expr { $1 }
 \mid macro\_string\_expr \{ \$1 \}
literal\_string\_expr ::=
 | string PLUS name { [U.Literal $1; U.Macro $3] }
 | string PLUS macro_string_expr { U.Literal $1 :: $3 }
macro\_string\_expr ::=
 | name PLUS string { [U.Macro $1; U.Literal $3] }
 | name PLUS string_expr { U.Macro $1 :: $3 }
strings ::=
 | string { [$1] }
 | string COMMA strings { $1 :: $3 }
string ::=
 \mid STRING \{ \$1 \}
 | string PLUS STRING { $1 ^ $3 }
orders ::=
  order { [$1] }
 | order COMMA orders { $1 :: $3 }
order ::=
\mid STRING\ COLON\ INT\ \{\ (\$1,\ \$3)\ \}
couplings ::=
  coupling { [$1] }
 | coupling COMMA couplings { $1 :: $3 }
 | LPAREN INT COMMA INT RPAREN COLON name { ($2, $4, $7) }
decays ::=
  decay \{ [\$1] \}
 | decay COMMA decays { $1 :: $3 }
```

```
\begin{array}{l} \textit{decay} ::= \\ \mid \textit{LPAREN} \ \textit{names} \ \textit{RPAREN} \ \textit{COLON} \ \textit{STRING} \ \{ \ (\$2, \ \$5) \ \} \end{array}
```

14.11 Interface of UFO_Lorentz

14.11.1 Processed UFO Lorentz Structures

Just like UFOx.Lorentz_Atom.dirac, but without the Dirac matrix indices.

```
\begin{array}{lll} \text{type } \textit{dirac} &= (* \text{ private } *) \\ | \textit{ Gamma5} \\ | \textit{ ProjM} \\ | \textit{ ProjP} \\ | \textit{ Gamma of } \textit{int} \\ | \textit{ Sigma of } \textit{int} \times \textit{int} \\ | \textit{ C} \\ | \textit{ Minus} \end{array}
```

A sandwich of a string of γ -matrices. bra and ket are positions of fields in the vertex, not spinor indices.

```
 \begin{array}{lll} \mbox{type } \mbox{\it dirac\_string} &= (* \mbox{ private } *) \\ \{ \mbox{\it bra} : \mbox{\it int}; \\ \mbox{\it ket} : \mbox{\it int}; \\ \mbox{\it conjugated} : \mbox{\it bool}; \\ \mbox{\it gammas} : \mbox{\it dirac list} \ \} \end{array}
```

In the case of Majorana spinors, we have to insert charge conjugation matrices.

```
\Gamma \to -\Gamma:
```

```
\mathsf{val}\ minus\ :\ dirac\_string\ \to\ dirac\_string
```

 $\Gamma \to C\Gamma$:

 $\mathsf{val}\ cc_times\ :\ dirac_string\ \to\ dirac_string$

 $\Gamma \to -\Gamma C$:

 $\mathsf{val}\ times_minus_cc\ :\ dirac_string\ \to\ dirac_string$

 $\Gamma \to \Gamma^T$:

 $val\ transpose\ :\ dirac_string\ o\ dirac_string$

 $\Gamma \to C\Gamma C^{-1}$:

 $val\ conjugate\ :\ dirac_string\ o\ dirac_string$

 $\Gamma \to C\Gamma^T C^{-1}$, i. e. the composition of *conjugate* and *transpose*:

```
val\ conjugate\_transpose\ :\ dirac\_string\ 	o\ dirac\_string
```

The Lorentz indices appearing in a term are either negative internal summation indices or positive external polarization indices. Note that the external indices are not really indices, but denote the position of the particle in the vertex.

```
\begin{array}{ll} \text{type } \alpha \ term &= \ (* \ \text{private} \ *) \\ \{ \ indices \ : \ int \ list; \\ atom \ : \ \alpha \ \} \end{array}
```

Split the list of indices into summation and polarization indices.

```
val\ classify\_indices\ :\ int\ list\ 	o\ int\ list\ 	imes\ int\ list
```

Replace the atom keeping the associated indices.

```
val\ map\_atom\ :\ (\alpha\ 	o\ \beta)\ 	o\ \alpha\ term\ 	o\ \beta\ term
```

A contraction consists of a (possibly empty) product of Dirac strings and a (possibly empty) product of Lorentz tensors with a rational coefficient. The *denominator* is required for the poorly documented propagator extensions. The type *atom linear* is a *list* and an empty list is interpreted as 1.



The denominator is a contraction list to allow code reuse, though a (A. scalar list \times A. scalar list \times QC.t) list would suffice

```
type contraction = (* private *)
  \{ coeff : Algebra.QC.t; \}
     dirac : dirac_string term list;
     vector : UFOx.Lorentz_Atom.vector term list;
     scalar : UFOx.Lorentz_Atom.scalar list;
     inverse: UFOx.Lorentz_Atom.scalar list;
     denominator : contraction list }
A sum of contractions.
type t = contraction list
Fermion line connections.
val\ fermion\_lines: t \rightarrow Coupling.fermion\_lines
\Gamma \to C\Gamma C^{-1}
val\ charge\_conjugate\ :\ int \times int \to\ t \to\ t
parse spins lorentz uses the spins to parse the UFO lorentz structure as a list of contractions.
val parse: ?allow\_denominator:bool \rightarrow Coupling.lorentz\ list \rightarrow UFOx.Lorentz.t \rightarrow t
map\_indices\ f\ lorentz applies the map f to the free indices in lorentz.
val\ map\_indices: (int \rightarrow int) \rightarrow t \rightarrow t
val map\_fermion\_lines:
  (int \rightarrow int) \rightarrow Coupling.fermion\_lines \rightarrow Coupling.fermion\_lines
Create a readable representation for debugging and documenting generated code.
val to\_string : t \rightarrow string
val\ fermion\_lines\_to\_string: Coupling.fermion\_lines \rightarrow string
Punting ...
val\ dummy : t
More debugging and documenting.
val\ dirac\_string\_to\_string : dirac\_string \rightarrow string
dirac\_string\_to\_matrix substitute ds take a string of \gamma-matrices ds, applies substitute to the indices and returns
the product as a matrix.
val dirac\_string\_to\_matrix : (int \rightarrow int) \rightarrow dirac\_string \rightarrow Dirac.Chiral.t
module type Test =
  sig
     val suite : OUnit.test
module Test: Test
```

14.12 Implementation of UFO_Lorentz

14.12.1 Processed UFO Lorentz Structures

```
\begin{array}{lll} \operatorname{module} \ Q &= \ Algebra. \, Q \\ \operatorname{module} \ QC &= \ Algebra. \, QC \\ \operatorname{module} \ A &= \ UFOx. Lorentz\_Atom \\ \operatorname{module} \ D &= \ Dirac. \, Chiral \end{array}
```

Take a A.t list and return the corresponding pair A.dirac list \times A.vector list \times A.scalar list \times A.scalar list, without preserving the order (currently, the order is reversed).

```
let split\_atoms atoms =
```

```
\begin{array}{l} \textit{List.fold\_left} \\ (\mathsf{fun}\ (d,\ v,\ s,\ i)\ \to\ \mathsf{function} \\ \mid\ A.\textit{Vector}\ v'\ \to\ (d,\ v'\ ::\ v,\ s,\ i) \\ \mid\ A.\textit{Dirac}\ d'\ \to\ (d'\ ::\ d,\ v,\ s,\ i) \\ \mid\ A.\textit{Scalar}\ s'\ \to\ (d,\ v,\ s'\ ::\ s,\ i) \\ \mid\ A.\textit{Inverse}\ i'\ \to\ (d,\ v,\ s,\ i'\ ::\ i)) \\ ([],\ [],\ [],\ [])\ \textit{atoms} \end{array}
```

Just like UFOx.Lorentz_Atom.dirac, but without the Dirac matrix indices.

```
type dirac = | Gamma5 | ProjM | ProjP | Gamma of int | Sigma of int <math>\times int | C | Minus

let map\_indices\_gamma f = function | (Gamma5 | ProjM | ProjP | C | Minus as <math>g) \rightarrow g | Gamma mu \rightarrow Gamma (f mu) | Sigma (mu, nu) \rightarrow Sigma (f mu, f nu)
```

A sandwich of a string of γ -matrices. bra and ket are positions of fields in the vertex.

```
type dirac\_string = \{bra : int; \\ ket : int; \\ conjugated : bool; \\ gammas : dirac list \}
let map\_indices\_dirac f d = \{bra = f d.bra; \\ ket = f d.ket; \\ conjugated = d.conjugated; \\ gammas = List.map (map\_indices\_gamma f) d.gammas \}
let toggle\_conjugated ds = \\ \{ds \text{ with } conjugated = \neg ds.conjugated \}
let flip\_bra\_ket ds = \\ \{ds \text{ with } bra = ds.ket; ket = ds.bra \}
```

The implementation of couplings for Dirac spinors in omega_spinors uses conjspinor_spinor which is a straightforward positive inner product

psibar0 * psi1 =
$$\bar{\psi}_0 \psi_1 = \sum_{\alpha} \bar{\psi}_{0,\alpha} \psi_{1,\alpha}$$
. (14.1)

Note that the row spinor $\bar{\psi}_0$ is the actual argument, it is *not* conjugated and multplied by γ_0 ! In contrast, JRR's implementation of couplings for Majorana spinors uses spinor_product in omega_bispinors

$$chi0 * chi1 = \chi_0^T C \chi_1 \tag{14.2}$$

with a charge antisymmetric and unitary conjugation matrix: $C^{-1} = C^{\dagger}$ and $C^{T} = -C$. This product is obviously antisymmetric:

$$chi0 * chi1 = \chi_0^T C \chi_1 = \chi_1^T C^T \chi_0 = -\chi_1^T C \chi_0 = - chi1 * chi0.$$
 (14.3)

In the following, we assume to be in a realization with $C^{-1} = -C$, i. e. $C^2 = -1$:

```
let inv_{-}C = [Minus; C]
```

In JRR's implementation of Majorana fermions (see page 429), all fermion-boson fusions are realized with the $f_-\phi f(g,phi,chi)$ functions, where $\phi \in \{v,a,...\}$. This is different from the original Dirac implementation, where both $f_-\phi f(g,phi,psi)$ and $f_-f\phi(g,psibar,phi)$ are used. However, the latter plays nicer with the

permutations in the UFO version of *fuse*. Therefore, we can attempt to automatically map $f_{\phi}f(g, \phi)$ to $f_{f}\phi(g, \phi)$ by an appropriate transformation of the γ -matrices involved. Starting from

$$f_{-}\phi f(g, phi, chi) = \Gamma^{\mu}_{\phi} \chi$$
 (14.4)

where Γ_{ϕ} is the contraction of the bosonic field ϕ with the appropriate product of γ -matrices, we obtain a condition on the corresponding matrix $\tilde{\Gamma}_{\phi}$ that appears in $\mathbf{f}_{-}\mathbf{f}\phi$:

$$\mathbf{f}_{-}\mathbf{f}\phi(\mathbf{g},\mathbf{chi},\mathbf{phi}) = \chi^{T}\tilde{\Gamma}^{\mu}_{\phi} = \left((\tilde{\Gamma}_{\phi})^{T}\chi\right)^{T} \stackrel{!}{=} \left(\Gamma_{\phi}\chi\right)^{T}. \tag{14.5}$$

This amounts to requiring $\tilde{\Gamma} = \Gamma^T$, as one might have expected. Below we will see that this is *not* the correct approach.

In any case, we can use the standard charge conjugation matrix relations

$$\mathbf{1}^T = \mathbf{1} \tag{14.6a}$$

$$\gamma_{\mu}^{T} = -C\gamma_{\mu}C^{-1} \tag{14.6b}$$

$$\sigma_{\mu\nu}^{T} = C\sigma_{\nu\mu}C^{-1} = -C\sigma_{\mu\nu}C^{-1} \tag{14.6c}$$

$$(\gamma_5 \gamma_\mu)^T = \gamma_\mu^T \gamma_5^T = -C \gamma_\mu \gamma_5 C^{-1} = C \gamma_5 \gamma_\mu C^{-1}$$
(14.6d)

$$\gamma_5^T = C\gamma_5 C^{-1} \tag{14.6e}$$

to perform the transpositions symbolically. For the chiral projectors

$$\gamma_{\pm} = 1 \pm \gamma_5 \tag{14.7}$$

this means¹

$$\gamma_{+}^{T} = (\mathbf{1} \pm \gamma_{5})^{T} = C(\mathbf{1} \pm \gamma_{5})C^{-1} = C\gamma_{\pm}C^{-1}$$
(14.8a)

$$(\gamma_{\mu}\gamma_{\pm})^{T} = \gamma_{\pm}^{T}\gamma_{\mu}^{T} = -C\gamma_{\pm}\gamma_{\mu}C^{-1} = -C\gamma_{\mu}\gamma_{\mp}C^{-1}$$
(14.8b)

$$(\gamma_{\mu} \pm \gamma_{\mu} \gamma_5)^T = -C(\gamma_{\mu} \mp \gamma_{\mu} \gamma_5)C^{-1}$$
(14.8c)

and of course

$$C^T = -C. (14.9)$$

The implementation starts from transposing a single factor using (14.6) and (14.8):

let transpose1 = function

```
\begin{array}{c|cccc} (Gamma5 & ProjM & ProjP \text{ as } g) \rightarrow [C; \ g] @ inv\_C \\ (Gamma\_ & Sigma\ (\_,\ \_) \text{ as } g) \rightarrow [Minus] @ [C; \ g] @ inv\_C \\ C \rightarrow [Minus; \ C] \\ Minus \rightarrow [Minus] \end{array}
```

In general, this will leave more than one *Minus* in the result and we can pull these out:

let rec $collect_signs_rev$ (negative, acc) = function

```
 \begin{array}{l} [] \rightarrow (negative,\ acc) \\ |\ Minus\ ::\ g\_list\ \rightarrow\ collect\_signs\_rev\ (\neg\ negative,\ acc)\ g\_list \\ |\ g\ ::\ g\_list\ \rightarrow\ collect\_signs\_rev\ (negative,\ g\ ::\ acc)\ g\_list \end{array}
```

Also, there will be products CC inside the result, these can be canceled, since we assume $C^2 = -1$:

let rec $compress_ccs_rev$ (negative, acc) = function

```
 \begin{array}{lll} [ & | & | & | & | \\ | & C & | & | & | \\ | & C & | & | & | \\ | & g & | & | & | \\ | & g & | & | & | \\ | & g & | & | & | \\ | & g & | &
```

Compose collect_signs_rev and compress_ccs_rev. The two list reversals will cancel.

 $let compress_signs g_list =$

```
let negative, g\_list\_rev = collect\_signs\_rev (false, []) g\_list in match compress\_ccs\_rev (negative, []) g\_list\_rev with | true, g\_list \rightarrow Minus :: g\_list
```

¹The final two equations are two different ways to obtain the same result, of course.

```
| false, g\_list \rightarrow g\_list
```

Transpose all factors in reverse order and clean up:

 $\mathsf{let}\ transpose\ d\ =$

 $\{ d \text{ with }$

gammas = compress_signs (ThoList.rev_flatmap transpose1 d.gammas) }

We can also easily flip the sign:

 $\mathsf{let}\ minus\ d\ =$

 $\{ d \text{ with } gammas = compress_signs (Minus :: d.gammas) \}$

Also in omega_spinors

$$\phi_{\text{-ff}}(g,psibar1,psi2) = \bar{\psi}_1 \Gamma_{\phi} \psi_2,$$
 (14.10)

while in omega_bispinors

$$\phi_{\text{-ff}}(g,\text{chi1},\text{chi2}) = \chi_1^T C \Gamma_{\phi} \chi_2. \tag{14.11}$$

The latter has mixed symmetry, depending on the γ -matrices in Γ_{ϕ} according to (14.6) and (14.8)

$$\phi_{\text{-ff}}(\text{g,chi2,chi1}) = \chi_{2}^{T} C \Gamma_{\phi} \chi_{1} = \chi_{1}^{T} \Gamma_{\phi}^{T} C^{T} \chi_{2} = -\chi_{1}^{T} \Gamma_{\phi}^{T} C \chi_{2} = \pm \chi_{1}^{T} C \Gamma_{\phi} C^{-1} C \chi_{2} = \pm \chi_{1}^{T} C \Gamma_{\phi} \chi_{2}. \quad (14.12)$$

In the tests keystones_omegalib and keystones_UFO, we check that the vertex $\bar{\psi}_0\Gamma_{\phi_1}\psi_2$ can be expressed in three ways, which must all agree. In the case of keystones_omegalib, the equivalences are

$$psibar0 * f_{\phi}f(g,phi1,psi2) = \bar{\psi}_0 \Gamma_{\phi_1} \psi_2$$
 (14.13a)

$$f_-f\phi(g,psibar0,phi1) * psi2 = \bar{\psi}_0\Gamma_{\phi_1}\psi_2$$
 (14.13b)

$$phi1 * \phi_{-}ff(g,psibar0,psi2) = \bar{\psi}_0 \Gamma_{\phi_1} \psi_2. \qquad (14.13c)$$

In the case of keystones_UFO, we use cyclic permutations to match the use in $UFO_targets$, as described in the table following (14.25)

psibar0 * f
$$\phi$$
f_p012(g,phi1,psi2) = $\bar{\psi}_0 \Gamma_{\phi_1} \psi_2$ (14.14a)

$$f\phi f_p = 201(g_p sibar 0, phi 1) * psi 2 = \bar{\psi}_0 \Gamma_{\phi_1} \psi_2$$
 (14.14b)

$$phi1 * f\phi f_p 120(g,psi2,psibar0) = tr \left(\Gamma_{\phi_1} \psi_2 \otimes \bar{\psi}_0\right) = \bar{\psi}_0 \Gamma_{\phi_1} \psi_2. \tag{14.14c}$$

In both cases, there is no ambiguity regarding the position of spinors and conjugate spinors, since the inner product conjspinor_spinor is not symmetrical.

Note that, from the point of view of permutations, the notation $\operatorname{tr}(\Gamma\psi'\otimes\bar{\psi})$ is more natural than the equivalent $\bar{\psi}\Gamma\psi'$ that inspired the ϕ -ff functions in the omegalib more than 20 years ago.

We would like to perform the same tests in keystones_omegalib_bispinors and keystones_UFO_bispinors, but now we have to be more careful in positioning the Majorana spinors, because we can not rely on the Fortran type system to catch cofusions of spinor and conjspinor fields. In addition, we must make sure to insert charge conjugation matrices in the proper places [7].

Regarding the tests in keystones_omegalib_bispinors, we observe

$$chi0 * f_{-}\phi f(g,phi1,chi2) = \chi_0^T C \Gamma_{\phi_1} \chi_2$$
 (14.15a)

phi1 *
$$\phi$$
-ff(g,chi0,chi2) = $\chi_0^T C\Gamma_{\phi_1} \chi_2$ (14.15b)

and

$$chi2 * f_-f\phi(g, chi0, phi1) = \chi_2^T C (\chi_0^T \tilde{\Gamma}_{\phi_1}^{\mu})^T = \chi_2^T C (\tilde{\Gamma}_{\phi_1}^{\mu})^T \chi_0 = \chi_2^T C \Gamma_{\phi_1} \chi_0$$
 (14.16a)

$$phi1 * \phi_{-}ff(g,chi2,chi0) = \chi_2^T C \Gamma_{\phi_1} \chi_0, \qquad (14.16b)$$

while

$$\mathbf{f}_{-}\mathbf{f}\phi(\mathbf{g},\mathbf{chi0},\mathbf{phi1}) * \mathbf{chi2} = \chi_0^T \tilde{\Gamma}_{\phi_1} C \chi_2 = \chi_0^T \Gamma_{\phi_1}^T C \chi_2 = (\Gamma_{\phi_1} \chi_0)^T C \chi_2$$
 (14.17)

is different. JRR solved this problem by abandoning $f_f\phi$ altogether and using ϕ_ff only in the form $\phi_ff(g,chi0,chi2)$. Turning to the tests in keystones_UFO_bispinors, it would be convenient to be able to use

chi0 *
$$f\phi f_p012(g,phi1,chi2) = \chi_0^T C \Gamma_{\phi_1}^{012} \chi_2$$
 (14.18a)

$$f\phi f_p = 201(g, chi0, phi1) * chi2 = \chi_0^T \Gamma_{\phi_1}^{201} C \chi_2$$
 (14.18b)

$$\text{phi1} * \text{f}\phi \text{f_p120(g,chi2,chi0)} = \text{tr}\left(\Gamma_{\phi_1}^{120}\chi_2 \otimes \chi_0^T\right) = \chi_0^T \Gamma_{\phi_1}^{120}\chi_2 = \chi_2^T (\Gamma_{\phi_1}^{120})^T \chi_0, \tag{14.18c}$$

where $\Gamma^{012} = \Gamma$ is the string of γ -matrices as written in the Lagrangian. Obviously, we should require

$$\Gamma^{120} = C\Gamma^{012} = C\Gamma \tag{14.19}$$

as expected from omega_bispinors.

For Γ^{201} we must require²

$$\Gamma^{201}C = C\Gamma^{012} = C\Gamma \tag{14.20}$$

i.e.

$$\Gamma^{201} = C\Gamma C^{-1} \neq \Gamma^T. \tag{14.21}$$

14.12.3 From Dirac Strings to 4×4 Matrices

dirac_string bind ds applies the mapping bind to the indices of γ_{μ} and $\sigma_{\mu\nu}$ and multiplies the resulting matrices in order using complex rational arithmetic.

```
module type To\_Matrix =
  sig
    val\ dirac\_string : (int \rightarrow int) \rightarrow dirac\_string \rightarrow D.t
  end
module To\_Matrix : To\_Matrix =
  struct
    let half = QC.make (Q.make 1 2) Q.null
    let half_i = QC.make\ Q.null\ (Q.make\ 1\ 2)
    let gamma\_L = D.times \ half \ (D.sub \ D.unit \ D.gamma5)
    let gamma_R = D.times \ half \ (D.add \ D.unit \ D.gamma5)
    let \ sigma = Array.make\_matrix \ 4 \ 4 \ D.null
    let() =
       for mu = 0 to 3 do
         for nu = 0 to 3 do
            sigma.(mu).(nu) \leftarrow
              D.times
                half_i
                (D.sub
                    (D.mul\ D.gamma.(mu)\ D.gamma.(nu))
                    (D.mul\ D.gamma.(nu)\ D.gamma.(mu)))
         done
       done
```

chi2 * f
$$\phi$$
f_p201(g,chi0,phi1) = $\chi_2^T C (\chi_0^T \Gamma_{\phi_1}^{201})^T = \chi_0^T \Gamma_{\phi_1}^{201} C^T \chi_2$.

We would find the condition

$$-\Gamma^{201}C = \Gamma^{201}C^T = C\Gamma$$

i.e. only a sign

$$\Gamma^{201} = -C\Gamma C^{-1} \neq \Gamma^T,$$

as was to be expected from the antisymmetry of spinor_product, of course.

²Note that we don't get anything new, if we reverse the scalar product

```
 \begin{array}{l} |\mbox{ let } dirac \ bind\_indices \ = \ \mbox{function} \\ |\mbox{ } Gamma5 \ \rightarrow \mbox{ } D.gamma5 \\ |\mbox{ } ProjM \ \rightarrow \mbox{ } gamma\_L \\ |\mbox{ } ProjP \ \rightarrow \mbox{ } gamma\_R \\ |\mbox{ } Gamma \ (mu) \ \rightarrow \mbox{ } D.gamma.(bind\_indices \ mu) \\ |\mbox{ } Sigma \ (mu, \ nu) \ \rightarrow \mbox{ } sigma.(bind\_indices \ mu).(bind\_indices \ nu) \\ |\mbox{ } C \ \rightarrow \mbox{ } D.cc \\ |\mbox{ } Minus \ \rightarrow \mbox{ } D.neg \ D.unit \\ |\mbox{ } let \ dirac\_string \ bind\_indices \ ds \ = \\ |\mbox{ } D.product \ (List.map \ (dirac \ bind\_indices) \ ds.gammas) \\ |\mbox{ } end \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string \\ |\mbox{ } let \ dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string\_to\_matrix \ = \ To\_Matrix.dirac\_string\_to\_matr
```

The Lorentz indices appearing in a term are either negative internal summation indices or positive external polarization indices. Note that the external indices are not really indices, but denote the position of the particle in the vertex.

```
type \alpha term = 
{ indices : int list; atom : \alpha } 
let map\_atom f term = 
{ term with atom = f term.atom } 
let map\_term f\_index f\_atom term = 
{ indices = List.map f\_index term.indices; atom = f\_atom term.atom }
```

Return a pair of lists: first the (negative) summation indices, second the (positive) external indices.

```
\begin{array}{l} \text{let } classify\_indices \ ilist = \\ List.partition \\ \text{(fun } i \rightarrow \\ \text{if } i < 0 \text{ then} \\ \text{true} \\ \text{else if } i > 0 \text{ then} \\ \text{false} \\ \text{else} \\ invalid\_arg \ "classify\_indices") \\ ilist \end{array}
```

Recursions on this type only stop when we come across an empty *denominator*. In practice, this is no problem (we never construct values that recurse more than once), but it would be cleaner to use polymorphic variants as suggested for *UFOx.Tensor.t*.

```
type contraction =
  \{ coeff : QC.t; 
    dirac: dirac_string term list;
    vector: A.vector term list;
    scalar : A.scalar list;
    inverse: A.scalar list;
     denominator : contraction list }
let fermion_lines_of_contraction contraction =
  List.sort
    compare
    (List.map (fun \ term \rightarrow (term.atom.ket, \ term.atom.bra)) \ contraction.dirac)
let rec map\_indices\_contraction f c =
  \{ coeff = c.coeff;
    dirac = List.map (map\_term f (map\_indices\_dirac f)) c.dirac;
    vector = List.map (map\_term f (A.map\_indices\_vector f)) c.vector;
    scalar = List.map (A.map\_indices\_scalar f) c.scalar;
    inverse = List.map (A.map\_indices\_scalar f) c.inverse;
    denominator = List.map (map\_indices\_contraction f) c.denominator
```

```
type t = contraction list
let dummy =
  let rec charge\_conjugate\_dirac (ket, bra as fermion\_line) = function
  | \ | \ | \rightarrow \ | \ |
  | dirac :: dirac\_list \rightarrow
      if dirac.atom.bra = bra \wedge dirac.atom.ket = ket then
         map\_atom\ toggle\_conjugated\ dirac\ ::\ dirac\_list
         dirac :: charge_conjugate_dirac fermion_line dirac_list
let \ charge\_conjugate\_contraction \ fermion\_line \ c = 1
  { c with dirac = charge_conjugate_dirac fermion_line c.dirac }
let charge\_conjugate\ fermion\_line\ l\ =
  List.map\ (charge\_conjugate\_contraction\ fermion\_line)\ l
let fermion_lines contractions =
  let pairs = List.map fermion_lines_of_contraction contractions in
  match ThoList.uniq (List.sort compare pairs) with
    [] \rightarrow invalid\_arg "UFO_Lorentz.fermion_lines:\sqcupimpossible"
    [pairs] \rightarrow pairs
  {\scriptscriptstyle \perp} \rightarrow invalid\_arg "UFO_Lorentz.fermion_lines:{\scriptscriptstyle \perp}ambiguous"
let map\_indices\ f\ contractions\ =
  List.map\ (map\_indices\_contraction\ f)\ contractions
let map\_fermion\_lines f pairs =
   List.map (fun (i, j) \rightarrow (f i, f j)) pairs
let dirac\_of\_atom = function
    A.Identity (\_, \_) \rightarrow []
    A.C (\_, \_) \rightarrow [C]
    A.Gamma5 (\_, \_) \rightarrow [Gamma5]
    A.ProjP (_, _) \rightarrow [ProjP]
    A.ProjM (_, _) \rightarrow [ProjM]
    A.Gamma\ (mu, \_, \_) \rightarrow [Gamma\ mu]
    A.Sigma\ (mu,\ nu,\ \_,\ \_) \rightarrow [Sigma\ (mu,\ nu)]
let dirac\_indices = function
    A.Identity(i, j) \mid A.C(i, j)
    A.Gamma5\ (i,\ j)\ |\ A.ProjP\ (i,\ j)\ |\ A.ProjM\ (i,\ j)
    A.Gamma(-, i, j) \mid A.Sigma(-, -, i, j) \rightarrow (i, j)
let rec scan\_for\_dirac\_string\ stack\ =\ function
  | [] \rightarrow
      (* We're done with this pass. There must be no leftover atoms on the stack of spinor atoms, but we'll
check this in the calling function. *)
      (None, List.rev stack)
    atom :: atoms \rightarrow
      \mathsf{let}\ i,\ j\ =\ dirac\_indices\ atom\ \mathsf{in}
      if i > 0 then
        if j > 0 then
           (* That's an atomic Dirac string. Collect all atoms for further processing. *)
           (Some { bra = i; ket = j; conjugated = false;
                     gammas = dirac\_of\_atom \ atom \},
             List.rev\_append\ stack\ atoms)
           (* That's the start of a new Dirac string. Search for the remaining elements, not forgetting matrices
that we might pushed on the stack earlier. *)
           collect\_dirac\_string
              i j (dirac_of_atom atom) [] (List.rev_append stack atoms)
      else
```

```
(* The interior of a Dirac string. Push it on the stack until we find the start. *)
        scan\_for\_dirac\_string (atom :: stack) atoms
Complete the string starting with i and the current summation index j.
and collect\_dirac\_string \ i \ j \ rev\_ds \ stack = function
      (* We have consumed all atoms without finding the end of the string. *)
      invalid_arg "collect_dirac_string: uopen ustring"
  \mid atom :: atoms \rightarrow
      \mathsf{let}\ i',\ j'\ =\ \mathit{dirac\_indices}\ \mathit{atom}\ \mathsf{in}
      if i' = i then
        if j' > 0 then
           (* Found the conclusion. Collect all atoms on the stack for further processing. *)
           (Some { bra = i; ket = j'; conjugated = false;
                     gammas = List.rev\_append rev\_ds (dirac\_of\_atom atom),
            List.rev_append stack atoms)
        else
           (* Found the continuation. Pop the stack of open indices, since we're looking for a new one. *)
           collect\_dirac\_string
             i \ j' \ (dirac\_of\_atom \ atom \ @ \ rev\_ds) \ [] \ (List.rev\_append \ stack \ atoms)
      else
        (* Either the start of another Dirac string or a non-matching continuation. Push it on the stack until
we're done with the current one. *)
        collect\_dirac\_string \ i \ j \ rev\_ds \ (atom :: stack) \ atoms
let dirac\_string\_of\_dirac\_atoms atoms =
  scan_for_dirac_string[] atoms
let rec dirac_strings_of_dirac_atoms' rev_ds atoms =
  match dirac_string_of_dirac_atoms atoms with
    (None, []) \rightarrow List.rev rev_ds
    (None, _) → invalid_arg "dirac_string_of_dirac_atoms: _leftover_atoms"
  (Some \ ds, \ atoms) \rightarrow dirac\_strings\_of\_dirac\_atoms' \ (ds :: rev\_ds) \ atoms
let \ dirac\_strings\_of\_dirac\_atoms \ atoms =
  dirac\_strings\_of\_dirac\_atoms' [] atoms
let indices\_of\_vector = function
    A.Epsilon (mu1, mu2, mu3, mu4) \rightarrow [mu1; mu2; mu3; mu4]
    A.Metric\ (mu1,\ mu2) \rightarrow [mu1;\ mu2]
    A.P (mu, n) \rightarrow
      \quad \text{if } n \ > \ 0 \ \text{then} \\
        [mu]
      else
        invalid\_arg "indices_of_vector:_invalid_momentum"
{\tt let} \ \mathit{classify\_vector} \ \mathit{atom} \ = \\
  \{ indices = indices\_of\_vector \ atom; \}
     atom }
let indices\_of\_dirac = function
    Gamma5 \mid ProjM \mid ProjP \mid C \mid Minus \rightarrow []
     Gamma\ (mu)\ 	o\ [mu]
    Sigma\ (mu,\ nu)\ 	o\ [mu;\ nu]
let indices\_of\_dirac\_string \ ds =
  ThoList.flatmap indices_of_dirac ds.gammas
let \ classify\_dirac \ atom =
  \{ indices = indices\_of\_dirac\_string \ atom; \}
     atom }
let contraction\_of\_lorentz\_atoms denominator (atoms, coeff) =
  let dirac_atoms, vector_atoms, scalar, inverse = split_atoms atoms in
  let dirac =
```

```
List.map classify_dirac (dirac_strings_of_dirac_atoms dirac_atoms)
  and vector =
     List.map classify_vector vector_atoms in
  { coeff; dirac; vector; scalar; inverse; denominator }
type redundancy =
    Trace of int
    Replace of int \times int
let rec redundant\_metric' rev\_atoms = function
    [] \rightarrow (None, List.rev rev\_atoms)
  \{ atom = A.Metric (mu, nu) \} as atom :: atoms \rightarrow
      \quad \text{if } mu \ < \ 1 \ \text{then} \\
        if nu = mu then
           (Some (Trace mu), List.rev_append rev_atoms atoms)
        else
           (Some\ (Replace\ (mu,\ nu)),\ List.rev\_append\ rev\_atoms\ atoms)
      else if nu < 0 then
        (Some (Replace (nu, mu)), List.rev_append rev_atoms atoms)
      else
        redundant\_metric' (atom :: rev\_atoms) atoms
  \mid \{ atom = (A.Epsilon(\_, \_, \_, \_) \mid A.P(\_, \_) \}  as atom :: atoms \rightarrow
      redundant\_metric' (atom :: rev\_atoms) atoms
{\tt let} \ \mathit{redundant\_metric} \ \mathit{atoms} \ = \\
  redundant_metric' [] atoms
Substitude any occurance of the index mu by the index nu:
let substitute\_index\_vector1 mu nu = function
  | A.Epsilon (mu1, mu2, mu3, mu4) as eps \rightarrow
      if mu = mu1 then
        A.Epsilon (nu, mu2, mu3, mu4)
      else if mu = mu2 then
        A.Epsilon (mu1, nu, mu3, mu4)
      else if mu = mu3 then
        A.Epsilon (mu1, mu2, nu, mu4)
      else if mu = mu4 then
        A.Epsilon (mu1, mu2, mu3, nu)
      else
        eps
  \mid A.Metric\ (mu1,\ mu2)\ \mathsf{as}\ g \to
      if mu = mu1 then
        A.Metric (nu, mu2)
      else if mu = mu2 then
        A.Metric\ (mu1,\ nu)
      else
  A.P (mu1, n) \text{ as } p \rightarrow
      if mu = mu1 then
        A.P (nu, n)
      else
        p
let \ remove \ a \ alist \ =
  List.filter ((\neq) \ a) alist
let \ substitute\_index1 \ mu \ nu \ mu1 \ =
  if mu = mu1 then
    nu
  else
     mu1
let \ substitute\_index \ mu \ nu \ indices =
  List.map (substitute_index1 mu nu) indices
```

```
This assumes that mu is a summation index and nu is a polarization index.
{\tt let} \ substitute\_index\_vector \ mu \ nu \ vectors \ = \\
  List.map
    (fun v \rightarrow
       \{ indices = substitute\_index mu nu v.indices; \}
          atom = substitute\_index\_vector1 mu nu v.atom \})
     vectors
Substitude any occurance of the index mu by the index nu:
let substitute\_index\_dirac1 mu nu = function
    (Gamma5 \mid ProjM \mid ProjP \mid C \mid Minus) as g \rightarrow g
    Gamma\ (mu1)\ {\sf as}\ g\ 	o
      if mu = mu1 then
        Gamma (nu)
      else
  | Sigma\ (mu1,\ mu2) as g \rightarrow
      if mu = mu1 then
        Sigma (nu, mu2)
      else if mu = mu2 then
        Sigma (mu1, nu)
      else
        g
This assumes that mu is a summation index and nu is a polarization index.
let \ substitute\_index\_dirac \ mu \ nu \ dirac\_strings =
  List.map
    (fun ds \rightarrow
       { indices = substitute_index mu nu ds.indices;
          atom = \{ ds.atom \text{ with } \}
                    gammas =
                       List.map
                         (substitute\_index\_dirac1 \ mu \ nu)
                         ds.atom.gammas \} \} )
     dirac\_strings
let trace\_metric = QC.make (Q.make 4 1) Q.null
FIXME: can this be made typesafe by mapping to a type that only contains P and Epsilon?
let rec compress\_metrics \ c =
  {\tt match}\ \mathit{redundant\_metric}\ c.vector\ {\tt with}
    None, \ \_ \ \rightarrow \ c
  | Some (Trace mu), vector' \rightarrow
      compress\_metrics
        \{ coeff = QC.mul\ trace\_metric\ c.coeff; \}
           dirac = c.dirac;
           vector = vector';
           scalar = c.scalar;
           inverse = c.inverse;
           denominator = c.denominator 
  | Some (Replace (mu, nu)), vector' \rightarrow
      compress\_metrics
        \{ coeff = c.coeff;
           dirac = substitute_index_dirac mu nu c.dirac;
           vector = substitute\_index\_vector mu nu vector';
           scalar = c.scalar;
           inverse = c.inverse;
           denominator = c.denominator 
let compress\_denominator = function
    [([], q)] as denominator \rightarrow if QC.is\_unit q then [] else denominator
    denominator \rightarrow denominator
```

```
let parse1 spins denominator atom =
  compress_metrics (contraction_of_lorentz_atoms denominator atom)
let parse\ ?(allow\_denominator = false)\ spins\ =\ function
    UFOx.Lorentz.Linear\ l \rightarrow List.map\ (parse1\ spins\ [])\ l
    UFOx.Lorentz.Ratios r \rightarrow
      ThoList.flatmap
        (fun (numerator, denominator) \rightarrow
           match compress_denominator denominator with
           [] \rightarrow List.map (parse1 spins []) numerator
              if allow\_denominator then
                 let parsed\_denominator =
                    List.map
                      (parse1 [Coupling.Scalar; Coupling.Scalar] [])
                      denominator in
                 List.map (parse1 spins parsed_denominator) numerator
                 invalid\_arg
                    (Printf.sprintf
                       "UFO_Lorentz.parse:udenominatoru%suinu%sunotualloweduhere!"
                        (UFOx.Lorentz.to\_string\ (UFOx.Lorentz.Linear\ d))
                        (UFOx.Lorentz.to\_string\ (UFOx.Lorentz.Ratios\ r))))
let i2s = UFOx.Index.to\_string
let vector\_to\_string = function
  A.Epsilon (mu, nu, ka, la) \rightarrow
      Printf.sprintf "Epsilon(%s,%s,%s,%s)" (i2s mu) (i2s nu) (i2s ka) (i2s la)
    A.Metric\ (mu,\ nu)\ 	o
      Printf.sprintf "Metric(%s,%s)" (i2s mu) (i2s nu)
  \mid A.P (mu, n) \rightarrow
      Printf.sprintf "P(%s,%d)" (i2s mu) n
let \ dirac\_to\_string \ = \ function
    Gamma5 \rightarrow "g5"
    ProjM \rightarrow "(1-g5)/2"
    ProjP \rightarrow "(1+g5)/2"
    Gamma\ (mu) \rightarrow Printf.sprintf\ "g(%s)"\ (i2s\ mu)
    Sigma\ (mu,\ nu) \rightarrow Printf.sprintf\ "s(%s,%s)"\ (i2s\ mu)\ (i2s\ nu)
    C \rightarrow \text{"C"}
    Minus \rightarrow "-1"
let dirac\_string\_to\_string ds =
  match ds.gammas with
    [] \rightarrow Printf.sprintf "<\%s|\%s>" (i2s ds.bra) (i2s ds.ket)
    gammas \rightarrow
      Printf.sprintf
        "<%s|%s|%s>"
        (i2s \ ds.bra)
        (String.concat "*" (List.map dirac_to_string gammas))
        (i2s \ ds.ket)
let scalar\_to\_string = function
    A.Mass \_ \rightarrow "m"
    A.Width \_ \rightarrow "w"
    A.P2 \ i \rightarrow Printf.sprintf "p\d**2" i
    A.P12(i, j) \rightarrow Printf.sprintf "p\d*p\d" i j
    A. Variable s \rightarrow s
    A.Coeff \ c \rightarrow UFOx.Value.to\_string \ c
let rec contraction\_to\_string \ c =
  String.concat
     "_*_"
```

```
(List.concat
         [if QC.is\_unit\ c.coeff then
             else
             [QC.to\_string\ c.coeff];
          List.map (fun ds \rightarrow dirac\_string\_to\_string ds.atom) c.dirac;
          List.map  (fun v \rightarrow vector\_to\_string v.atom) <math>c.vector;
          List.map\ scalar\_to\_string\ c.scalar])
     (match c.inverse with
        [] \rightarrow ""
        inverse \rightarrow
          " \sqcup / \sqcup (" \hat String.concat "*" (List.map scalar_to_string inverse) ^ ")") ^
     (match c.denominator with
        [\,] \ \rightarrow \ ""
        denominator \rightarrow " \_ / \_ (" \hat to\_string denominator \hat ")")
and to\_string\ contractions\ =
  String.concat "_{\sqcup}+_{\sqcup}" (List.map contraction_to_string contractions)
let \ fermion\_lines\_to\_string \ fermion\_lines \ =
   ThoList.to_string
     (\text{fun } (ket, bra) \rightarrow Printf.sprintf "%s->%s" (i2s ket) (i2s bra))
     fermion\_lines
module type Test =
  sig
     val\ suite\ :\ OUnit.test
  end
module Test : Test =
  struct
     open OUnit
     let braket gammas =
        \{ bra = 11; ket = 22; conjugated = false; gammas \}
     \mathsf{let}\ assert\_transpose\ gt\ g\ =
        assert_equal ~printer: dirac_string_to_string
          (braket \ gt) \ (transpose \ (braket \ g))
     let \ assert\_conjugate\_transpose \ gct \ g \ = \\
        assert\_equal\ \~printer: dirac\_string\_to\_string
          (braket gct) (conjugate_transpose (braket g))
     let suite\_transpose =
        "transpose" >:::
          [ "identity" >::
               (fun () \rightarrow
                  assert_transpose [] []);
             "gamma_mu" >::
               (fun () \rightarrow
                  assert_transpose [C; Gamma 1; C] [Gamma 1]);
             "sigma_munu" >::
               (fun () \rightarrow
                  assert\_transpose [C; Sigma (1, 2); C] [Sigma (1, 2)]);
             "gamma_5*gamma_mu" >::
               (fun () \rightarrow
                  assert\_transpose
                     [C; Gamma\ 1; Gamma5; C]
                    [Gamma5; Gamma 1]);
             "gamma5" >::
               (\mathsf{fun}\ ()\ \to
```

```
assert_transpose [Minus; C; Gamma5; C] [Gamma5]);
          "gamma+" >::
            (fun () \rightarrow
               assert_transpose [Minus; C; ProjP; C] [ProjP]);
          "gamma-" >::
            (fun () \rightarrow
               assert\_transpose [Minus; C; ProjM; C] [ProjM]);
          "gamma_mu*gamma_nu" >::
            (fun () \rightarrow
               assert\_transpose
                 [Minus; C; Gamma 2; Gamma 1; C]
                 [Gamma\ 1;\ Gamma\ 2]);
          "gamma_mu*gamma_nu*gamma_la" >::
            (fun () \rightarrow
               assert\_transpose
                 [C; Gamma\ 3; Gamma\ 2; Gamma\ 1;\ C]
                 [Gamma 1; Gamma 2; Gamma 3]);
          "gamma_mu*gamma+" >::
            (fun () \rightarrow
               assert\_transpose
                 [C; ProjP; Gamma 1; C]
                 [Gamma\ 1;\ ProjP]);
          "gamma_mu*gamma-" >::
            (fun () \rightarrow
               assert\_transpose
                 [C; ProjM; Gamma 1; C]
                 [Gamma\ 1;\ ProjM])
  let suite_conjugate_transpose =
     "conjugate_transpose" >:::
       ["identity">::
            (fun () \rightarrow
               assert_conjugate_transpose [] []);
          "gamma_mu" >::
            (fun () \rightarrow
               assert_conjugate_transpose [Minus; Gamma 1] [Gamma 1]);
          "sigma_munu" >::
            (fun () \rightarrow
               assert\_conjugate\_transpose [Minus; Sigma (1, 2)] [Sigma (1, 2)]);
          "gamma_mu*gamma5" >::
            (fun () \rightarrow
               assert\_conjugate\_transpose
                 [\mathit{Minus}; \; \mathit{Gamma5}; \; \mathit{Gamma} \; 1] \; [\mathit{Gamma} \; 1; \; \mathit{Gamma5}]);
          "gamma5" >::
            (\mathsf{fun}\ ()\ \to
               assert\_conjugate\_transpose [Gamma5] [Gamma5])
  let suite =
     "UFO_Lorentz" >:::
       [suite\_transpose;
        suite\_conjugate\_transpose
end
```

14.13 Interface of UFO

 $val\ parse_string : string \rightarrow UFO_syntax.t$

```
val\ parse\_file\ :\ string 
ightarrow\ UFO\_syntax.t
```

These are the contents of the Python files after lexical analysis as context-free variable declarations, before any semantic interpretation.

```
module type Files =
  sig
    type t = private
       \{ particles : UFO\_syntax.t; \}
         couplings : UFO\_syntax.t;
         coupling\_orders : UFO\_syntax.t;
         vertices : UFO_syntax.t;
         lorentz : UFO\_syntax.t;
         parameters : UFO_syntax.t;
         propagators: UFO\_syntax.t;
         decays : UFO\_syntax.t }
    val parse\_directory : string \rightarrow t
  end
type t
exception Unhandled of string
module\ Model\ :\ Model\ T
val\ parse\_directory : string \rightarrow t
module type Fortran_Target =
```

fuse c v s fl g wfs ps fusion fuses the wavefunctions named wfs with momenta named ps using the vertex named v with legs reordered according to fusion. The overall coupling constant named g is multiplied by the rational coefficient c. The list of spins s and the fermion lines fl are used for selecting the appropriately transformed version of the vertex v.

```
val fuse :
        Algebra.QC.t \rightarrow string \rightarrow
        Coupling.lorentzn \rightarrow Coupling.fermion\_lines \rightarrow
        string \rightarrow string \ list \rightarrow string \ list \rightarrow Coupling.fusen \rightarrow unit
     val\ lorentz\_module :
        ?only : Sets.String.t \rightarrow ?name :string \rightarrow
        ?fortran\_module : string \rightarrow ?parameter\_module : string \rightarrow 
        Format\_Fortran.formatter \rightarrow unit \rightarrow unit
  end
module Targets:
  sig
     module Fortran : Fortran_Target
  end
Export some functions for testing:
module Propagator\_UFO :
  sig
     type t = (* private *)
        \{ name : string; 
          numerator : UFOx.Lorentz.t;
           denominator : UFOx.Lorentz.t }
  end
module Propagator :
  sig
     type t = (* private *)
        \{ name : string; 
          spins : Coupling.lorentz \times Coupling.lorentz;
```

```
numerator: UFO\_Lorentz.t; \\ denominator: UFO\_Lorentz.t; \\ variables: string list \} \\ \text{val } of\_propagator\_UFO: ?majorana:bool $\rightarrow$ Propagator\_UFO.t $\rightarrow$ t \\ \text{val } transpose: t $\rightarrow$ t \\ \text{end} \\ \\ \text{module type } Test = \\ \text{sig} \\ \text{val } suite: OUnit.test \\ \text{end} \\ \\ \text{module } Test: Test \\ \\ \end{aligned}
```

14.14 Implementation of UFO

Unfortunately, ocamlweb will not typeset all multi character operators nicely. E. g. f @< g comes out as f @ < g.

```
let (<*>) f g x =
f(g|x)
\mathsf{let} \; (<**>) \; f \; g \; x \; y \; = \;
  f(g x y)
module SMap = Map.Make (struct type t = string let compare = compare end)
module SSet = Sets.String
module CMap =
  Map.Make
    (struct
      type t = string
      let\ compare\ =\ ThoString.compare\_caseless
module \ CSet = Sets.String\_Caseless
let error_in_string text start_pos end_pos =
  let i = start\_pos.Lexing.pos\_cnum
  and j = end\_pos.Lexing.pos\_cnum in
  String.sub\ text\ i\ (j\ -\ i)
let error_in_file name start_pos end_pos =
  Printf.sprintf
    "%s:%d.%d-%d.%d"
    name
    start\_pos.Lexing.pos\_lnum
    (start\_pos.Lexing.pos\_cnum - start\_pos.Lexing.pos\_bol)
    end_pos.Lexing.pos_lnum
    (end\_pos.Lexing.pos\_cnum - end\_pos.Lexing.pos\_bol)
let parse\_string text =
  try
     UFO_parser.file
       UFO\_lexer.token
       (UFO_lexer.init_position "" (Lexing.from_string text))
  with
  UFO\_tools.Lexical\_Error\ (msg,\ start\_pos,\ end\_pos) \rightarrow
     invalid_arg (Printf.sprintf "lexical_error_(%s)_at:_'%s'"
                       msg (error_in_string text start_pos end_pos))
    UFO\_syntax.Syntax\_Error\ (msg,\ start\_pos,\ end\_pos) \rightarrow
     invalid_arg (Printf.sprintf "syntax_error_(%s)_at:_'%s'"
                       msg (error_in_string text start_pos end_pos))
  \mid Parsing.Parse\_error \rightarrow
     invalid_arg ("parse_error: " ^ text)
exception File\_missing of string
```

```
let parse\_file name =
  let ic =
    try open\_in\ name\ with
    \mid Sys\_error \ msg \ as \ exc \rightarrow
        if msg = name \ ^ ":\_No\_such\_file\_or\_directory" then
           raise (File\_missing name)
           raise exc in
  let result =
    begin
       try
          UFO\_parser.file
            UFO\_lexer.token
            (UFO_lexer.init_position name (Lexing.from_channel ic))
       with
       UFO\_tools.Lexical\_Error\ (msg,\ start\_pos,\ end\_pos) \rightarrow
           begin
             close\_in\ ic;
             invalid\_arg\ (Printf.sprintf
                                "%s:_lexical_error_(%s)"
                               (error_in_file name start_pos end_pos) msq)
         UFO\_syntax.Syntax\_Error\ (msg,\ start\_pos,\ end\_pos) \rightarrow
           begin
             close\_in\ ic;
             invalid_arg (Printf.sprintf
                                "%s:_{\sqcup}syntax_{\sqcup}error_{\sqcup}(%s)"
                                (error_in_file name start_pos end_pos) msg)
       \mid Parsing.Parse\_error \rightarrow
           begin
             close\_in\ ic;
             invalid_arg ("parse_error: " ^ name)
    end in
  close\_in ic;
  result
These are the contents of the Python files after lexical analysis as context-free variable declarations, before any
semantic interpretation.
module type \mathit{Files} =
  sig
    type t = private
       \{ particles : UFO\_syntax.t; 
          couplings : UFO\_syntax.t;
          coupling_orders : UFO_syntax.t;
          vertices : UFO_syntax.t;
         lorentz : UFO\_syntax.t;
         parameters : UFO\_syntax.t;
         propagators : UFO\_syntax.t;
          decays : UFO\_syntax.t }
    val\ parse\_directory : string \rightarrow t
  end
module\ Files\ :\ Files\ =
  struct
    type t =
       { particles : UFO_syntax.t;
          couplings : UFO\_syntax.t;
```

 $coupling_orders$: $UFO_syntax.t$;

```
vertices : UFO\_syntax.t;
          lorentz : UFO\_syntax.t;
         parameters : UFO_syntax.t;
         propagators : UFO_syntax.t;
         decays : UFO\_syntax.t 
    let parse\_directory dir =
       let filename stem = Filename.concat dir (stem ^ ".py") in
       \mathsf{let}\ parse\ stem\ =\ parse\_\mathit{file}\ (\mathit{filename}\ stem)\ \mathsf{in}
       let parse\_optional stem =
         try parse stem with File\_missing\_ \rightarrow [] in
       { particles = parse "particles";
         couplings = parse "couplings";
          coupling_orders = parse_optional "coupling_orders";
          vertices = parse "vertices";
         lorentz = parse "lorentz";
         parameters = parse "parameters";
         propagators = parse_optional "propagators";
          decays = parse_optional "decays" }
  end
let dump\_file pfx f =
  List.iter
     (\text{fun } s \rightarrow print\_endline (pfx ^ ": " ^ s))
     (UFO\_syntax.to\_strings\ f)
type charge =
    Q\_Integer of int
    Q_Fraction of int \times int
let \ charge\_to\_string \ = \ function
    Q\_Integer i \rightarrow Printf.sprintf "%d" i
    Q-Fraction (n, d) \rightarrow Printf.sprintf "%d/%d" n d
module S = UFO\_syntax
let find_attrib name attribs =
  try
     (List.find (fun a \rightarrow name = a.S.a\_name) attribs).S.a\_value
  with
  | Not\_found \rightarrow failwith ("UFO.find\_attrib:_\\"" ^ name ^ "\"_not_found")
let find_attrib name attribs =
  (List.find (fun a \rightarrow name = a.S.a\_name) attribs).S.a\_value
let name_to_string ?strip name =
  \mathsf{let}\ stripped\ =
     begin match strip, List.rev name with
     | Some pfx, head :: tail \rightarrow
        if pfx = head then
          tail
        else
           failwith ("UFO.name_to_string:_expected_prefix_'" ^ pfx ^
                           "', _got_', " ^ head ^ "', ")
    \mid _, name \rightarrow name
    end in
  String.concat "." stripped
{\tt let} \ name\_attrib \ ?strip \ name \ attribs \ =
  match find_attrib name attribs with
  | S.Name n \rightarrow name\_to\_string ?strip n
  | \_ \rightarrow invalid\_arg ("UFO.name\_attrib:_\" ^ name)
let integer\_attrib name attribs =
  match find\_attrib\ name\ attribs with
  \mid S.Integer i \rightarrow i
```

```
| _ → invalid_arg ("UFO.integer_attrib:__" ^ name)
let charge\_attrib name attribs =
  match find_attrib name attribs with
    S.Integer i \rightarrow Q\_Integer i
    S.Fraction(n, d) \rightarrow Q\_Fraction(n, d)
   | \_ \rightarrow invalid\_arg ("UFO.charge\_attrib: \_" ^ name)
{\tt let} \ string\_attrib \ name \ attribs \ =
  match find\_attrib\ name\ attribs with
    S.String \ s \rightarrow s
  | _ → invalid_arg ("UFO.string_attrib:__" ^ name)
let string\_expr\_attrib name attribs =
  match find\_attrib\ name\ attribs with
    S.Name n \rightarrow [S.Macro n]
    S.String \ s \rightarrow [S.Literal \ s]
    S.String\_Expr\ e\ 
ightarrow\ e
   \mid _ 
ightarrow invalid_arg ("UFO.string_expr_attrib:_{\sqcup}" \hat{\ } name)
let\ boolean\_attrib\ name\ attribs\ =
  try
     match ThoString.lowercase (name_attrib name attribs) with
       "true" \rightarrow true
       "false" 	o false
         → invalid_arg ("UFO.boolean_attrib:" ^ name)
  with
  | Not\_found \rightarrow false
type value =
    Integer of int
    Fraction of int \times int
    Float of float
    Expr of UFOx.Expr.t
    Name of string list
let map\_expr f default = function
    Integer \_ \mid Fraction (\_, \_) \mid Float \_ \mid Name \_ \rightarrow default
    Expr \ e \rightarrow f \ e
let \ variables = map\_expr \ UFOx.Expr.variables \ CSet.empty
let functions = map_expr UFOx.Expr.functions CSet.empty
let \ add\_to\_set\_in\_map \ key \ element \ map =
  let set = try \ CMap.find \ key \ map \ with \ Not\_found \rightarrow \ CSet.empty \ in
  CMap.add key (CSet.add element set) map
Add all variables in value to the map from variables to the names in which they appear, indicating that name
depends on these variables.
let dependency name value map = 
   CSet.fold
     (fun variable \ acc \rightarrow add\_to\_set\_in\_map \ variable \ name \ acc)
     (variables value)
     map
let dependencies name_value_list =
  List.fold\_left
     (fun\ acc\ (name,\ value)\ 	o\ dependency\ name\ value\ acc)
     CMap.empty
     name\_value\_list
let dependency_to_string (variable, appearences) =
  Printf.sprintf
     "%s<sub>U</sub>-><sub>U</sub>{%s}"
     variable (String.concat ", □" (CSet.elements appearences))
let dependencies\_to\_strings map =
```

```
List.map dependency_to_string (CMap.bindings map)
let \ expr_to_string =
   UFOx. Value.to\_string < * > UFOx. Value.of\_expr
let value\_to\_string = function
     Integer~i~\rightarrow~Printf.sprintf~"\mbox{"d"}~i
     Fraction (n, d) \rightarrow Printf.sprintf "%d/%d" n d
     Float \ x \rightarrow string\_of\_float \ x
     Expr \ e \rightarrow "'," ^ expr\_to\_string \ e ^ ","
    Name \ n \rightarrow name\_to\_string \ n
let value\_to\_expr\ substitutions\ =\ \mathsf{function}
    Integer i \rightarrow Printf.sprintf "%d" i
     Fraction (n, d) \rightarrow Printf.sprintf "%d/%d" n d
     Float \ x \rightarrow string\_of\_float \ x
     Expr \ e \rightarrow expr\_to\_string \ (substitutions \ e)
    Name \ n \rightarrow name\_to\_string \ n
let value\_to\_coupling \ substitutions \ atom = function
    Integer i \rightarrow Coupling.Integer i
     Fraction (n, d) \rightarrow Coupling.Quot (Coupling.Integer n, Coupling.Integer d)
     Float \ x \rightarrow Coupling.Float \ x
     Expr\ e\ 	o
       UFOx. Value.to_coupling atom (UFOx. Value.of_expr (substitutions e))
   | Name \ n \rightarrow failwith "UFO.value_to_coupling: Name not supported yet!"
\mathsf{let}\ value\_to\_numeric\ =\ \mathsf{function}
     Integer \ i \ 	o \ Printf.sprintf "%d" i
     Fraction (n, d) \rightarrow Printf.sprintf "%g" (float n /. float d)
     Float x \rightarrow Printf.sprintf "%g" x
     Expr\ e \rightarrow invalid\_arg\ ("UFO.value\_to\_numeric:\_expr_=_" ^ (expr\_to\_string\ e))
     Name \ n \ \rightarrow \ invalid\_arg \ ("UFO.value\_to\_numeric:\_name\_=\_" \ \hat{\ } \ name\_to\_string \ n)
let \ value\_to\_float = function
     \mathit{Integer}\ i\ \to\ \mathit{float}\ i
     Fraction (n, d) \rightarrow float n / float d
     Float \ x \ \rightarrow \ x
     Expr \ e \rightarrow invalid\_arg ("UFO.value\_to\_float: \_string\_= \_" ^ (expr\_to\_string \ e))
    Name\ n \rightarrow invalid\_arg\ ("UFO.value\_to\_float: \_name\_= \_" ^ name\_to\_string\ n)
{\tt let} \ value\_attrib \ name \ attribs \ =
  match find_attrib name attribs with
     S.Integer i \rightarrow Integer i
     S.Fraction(n, d) \rightarrow Fraction(n, d)
     S.Float x \rightarrow Float x
     S.String \ s \rightarrow Expr \ (UFOx.Expr.of\_string \ s)
     S.Name n \rightarrow Name n
     \rightarrow invalid\_arg ("UFO.value\_attrib:_\" ^ name)
let string\_list\_attrib name attribs =
  match find_attrib name attribs with
     S.String\_List \ l \rightarrow l
   | \_ \rightarrow invalid\_arg ("UFO.string\_list\_attrib:_\" ^ name)
let \ name\_list\_attrib \ \tilde{\ } strip \ name \ attribs \ =
  match find_attrib name attribs with
    S.Name\_List\ l \rightarrow List.map\ (name\_to\_string\ \ strip)\ l
   | \_ \rightarrow invalid\_arg ("UFO.name\_list\_attrib:_\" ^ name)
let integer\_list\_attrib name attribs =
  match find_attrib name attribs with
    S.Integer\_List \ l \rightarrow \ l
    _ → invalid_arg ("UFO.integer_list_attrib:_" ^ name)
let \ order\_dictionary\_attrib \ name \ attribs =
   match find_attrib name attribs with
```

```
S.Order\_Dictionary\ d \rightarrow d
            → invalid_arg ("UFO.order_dictionary_attrib: " ^ name)
{\tt let} \ coupling\_dictionary\_attrib\ \tilde{\ } strip\ name\ attribs\ =
    match find_attrib name attribs with
     | S.Coupling\_Dictionary d \rightarrow
            List.map (fun (i, j, c) \rightarrow (i, j, name\_to\_string \ \ strip \ c)) \ d
        \rightarrow invalid\_arg ("UFO.coupling\_dictionary\_attrib:_\" ^ name)
let \ decay\_dictionary\_attrib \ name \ attribs =
    match find_attrib name attribs with
     | S.Decay\_Dictionary d \rightarrow
            List.map (fun (p, w) \rightarrow (List.map List.hd p, w)) d
     \bot \rightarrow invalid\_arg ("UFO.decay\_dictionary\_attrib:_\" ^ name)
let required_handler kind symbol attribs query name =
    trv
         query name attribs
    with
    \mid Not\_found \rightarrow
            invalid\_arg
                 (Printf.sprintf
                        "fatal_UFO_error:_mandatory_attribute_'%s'_missing_for_%s_'%s'!"
                       name kind symbol)
let optional_handler attribs query name default =
         query name attribs
    with
     | Not\_found \rightarrow default
The UFO paper [17] is not clear on the question whether the name attribute of an instance must match its
Python name. While the examples appear to imply this, there are examples of UFO files in the wild that
violate this constraint.
let warn_symbol_name file symbol name =
    if name \neq symbol then
         Printf.eprintf
              "UFO:_warning:_symbol_'%s'_<>_name_'%s'_in_%s.py:_\
       \verb| uuuuuuuwhile_legal_uin_UFO, uit_uis_uunusual_uand_ucan_ucause_problems! \\ | musual_uand_ucan_ucause_problems! \\ | musual_uand_ucause_problems! \\ | musual_uand_ucause_pro
              symbol name file
let valid_fortran_id kind name =
    if \neg (ThoString.valid\_fortran\_id\ name) then
         invalid\_arg
              (Printf.sprintf
                     "fatal_UFO_error: uthe_%s_'%s'_is_not_a_valid_fortran_id!"
                     kind name)
let map\_to\_alist map =
     SMap.fold (fun key value acc \rightarrow (key, value) :: acc) map []
let keys map =
    SMap.fold (fun key \_ acc \rightarrow key :: acc) map []
let keys\_caseless map =
     CMap.fold (fun key \_ acc \rightarrow key :: acc) map []
let values map =
     SMap.fold (fun \_value\ acc \rightarrow value\ ::\ acc)\ map\ [\ ]
module SKey =
    struct
         type t = string
         let hash = Hashtbl.hash
         let equal = (=)
    end
```

```
module SHash = Hashtbl.Make (SKey)
module type Particle =
  sig
     type t = private
       \{ pdg\_code : int;
          name : string;
          antiname : string;
          spin : UFOx.Lorentz.r;
          color : UFOx.Color.r;
          mass: string;
          width : string;
          propagator : string option;
          texname : string;
          antitexname : string;
          charge: charge;
          ghost\_number: int;
          lepton\_number : int;
          y : charge;
          goldstone : bool;
          propagating : bool; (* NOT HANDLED YET! *)
          line: string option; (* NOT HANDLED YET! *)
          is\_anti : bool \}
    \mathsf{val}\ of\_file\ :\ S.t\ \to\ t\ SMap.t
    \mathsf{val}\ to\_string\ :\ string\ \to\ t\ \to\ string
    val\ conjugate\ :\ t\ 	o\ t
    val\ force\_spinor:\ t\ 	o\ t
    val force\_conjspinor : t \rightarrow t
    val\ force\_majorana\ :\ t\ 	o\ t
    \mathsf{val}\ is\_majorana\ :\ t\ \to\ bool
    val is\_ghost : t \rightarrow bool
    val\ is\_goldstone\ :\ t\ 	o\ bool
    \mathsf{val}\ is\_physical\ :\ t\ \to\ bool
    \mathsf{val}\ filter\ :\ (t\ 	o\ bool)\ 	o\ t\ SMap.t\ 	o\ t\ SMap.t
  end
module\ Particle\ :\ Particle\ =
  struct
     \mathsf{type}\ t\ =
       \{ pdg\_code : int; 
          name : string;
          antiname : string;
          spin : UFOx.Lorentz.r;
          color: UFOx.Color.r;
          mass: string;
          width : string;
          propagator: string option;
          texname : string;
          antitexname : string;
          charge : charge;
          ghost\_number: int;
          lepton\_number : int;
          y : charge;
          goldstone : bool;
          propagating : bool; (* NOT HANDLED YET! *)
          line: string option; (* NOT HANDLED YET! *)
          is\_anti : bool }
    let to\_string\ symbol\ p\ =
       Printf.sprintf
          "particle:_{\square}%s_{\square}=>_{\square}[pdg_{\square}=_{\square}%d,_{\square}name_{\square}=_{\square}'%s',_{\square}\
```

```
\verb| uuuuuuuuuuuuuuuspin_u=_u\%s,_ucolor_u=_u\%s,_u |
   \verb| uuuuuuuuuuuuuuuuuuuuuumass_u=_u\%s,_uwidth_u=_u\%s,\%s_u \\
   ____TeX_=_'%s'/'%s'%s]"
        symbol\ p.pdg\_code\ p.name\ p.antiname
        (UFOx.Lorentz.rep\_to\_string\ p.spin)
        (UFOx.Color.rep\_to\_string\ p.color)
        p.mass p.width
        (match p.propagator with
           None \rightarrow ""
           Some \ p \rightarrow " propagator p^ ",")
        (charge\_to\_string\ p.charge)
        p.ghost_number p.lepton_number
        (charge\_to\_string\ p.y)
        p.texname\ p.antitexname
        (if p.goldstone then ", \Box GB" else "")
    let conjugate\_charge = function
        Q\_Integer i \rightarrow Q\_Integer (-i)
        Q_Fraction (n, d) \rightarrow Q_Fraction (-n, d)
    let is\_neutral p =
      (p.name = p.antiname)
We must not mess with pdg\_code and color if the particle is neutral!
    let conjugate p =
      if is\_neutral p then
        p
      else
        \{ pdg\_code = - p.pdg\_code; \}
          name = p.antiname;
          antiname = p.name;
          spin = UFOx.Lorentz.rep\_conjugate\ p.spin;
          color = UFOx.Color.rep\_conjugate p.color;
          mass = p.mass;
          width = p.width;
          propagator = p.propagator;
          texname = p.antitexname;
          antitexname = p.texname;
          charge = conjugate\_charge \ p.charge;
          ghost\_number = - p.ghost\_number;
          lepton\_number = -p.lepton\_number;
          y = conjugate\_charge p.y;
          goldstone = p.goldstone;
          propagating = p.propagating;
          line = p.line;
          is\_anti = \neg p.is\_anti 
    let of_file1 \ map \ d =
      let \ symbol = d.S.name \ in
      match d.S.kind, d.S.attribs with
      \mid [ "Particle" \mid, attribs 
ightarrow
         let required query name =
           required_handler "particle" symbol attribs query name
         and optional query name default =
           optional_handler attribs query name default in
         let name = required string_attrib "name"
         and antiname = required string\_attrib "antiname" in
         let neutral = (name = antiname) in
         let pdg\_code = required integer\_attrib "pdg\_code" in
         SMap.add symbol
           { (* The required attributes per UFO docs. *)
             pdg\_code;
```

```
name; antiname;
           spin =
              UFOx.Lorentz.rep_of_int neutral (required integer_attrib "spin");
           color =
              UFOx.Color.rep_of_int neutral (required integer_attrib "color");
           mass = required (name_attrib ~strip:"Param") "mass";
           width = required (name_attrib ~strip :"Param") "width";
           texname = required string_attrib "texname";
           antitexname = required string_attrib "antitexname";
           charge = required charge_attrib "charge";
           (* The optional attributes per UFO docs. *)
           ghost_number = optional integer_attrib "GhostNumber" 0;
           lepton_number = optional integer_attrib "LeptonNumber" 0;
           y = optional \ charge\_attrib "Y" (Q_Integer 0);
           goldstone = optional \ boolean\_attrib "goldstone" false;
           propagating = optional boolean_attrib "propagating" true;
           line =
             (try Some\ (name\_attrib\ "line"\ attribs) with \_\to\ None);
           (* Undocumented extensions. *)
           propagator =
             (try Some\ (name\_attrib\ \tilde{\ }strip\ :"Prop"\ "propagator"\ attribs) with _{-}\to\ None);
           (* O'Mega extensions. *)
           (* Instead of "first come is particle" rely on a negative PDG code to identify antiparticles. *)
           is\_anti = pdg\_code < 0 \} map
   | [ "anti"; p ], [ ] \rightarrow
      begin
        try
           SMap.add symbol (conjugate (SMap.find p map)) map
         with
        \mid Not\_found \rightarrow
            invalid\_arg
              ("Particle.of_file: " ^ p ^ ".anti() not yet defined!")
       \rightarrow invalid\_arg ("Particle.of\_file:_\" ^ name\_to\_string d.S.kind)
let of_file particles =
   List.fold_left of_file1 SMap.empty particles
let is\_spinor p =
   match UFOx.Lorentz.omega p.spin with
     Coupling.Spinor \mid Coupling.ConjSpinor \mid Coupling.Majorana \rightarrow true
   |  \rightarrow false
TODO: this is a bit of a hack: try to expose the type UFOx.Lorentz_Atom'.r instead.
let force\_spinor p =
   if is\_spinor p then
     \{ p \text{ with } spin = UFOx.Lorentz.rep\_of\_int false 2 \}
   else
     p
let force\_conjspinor p =
   if is\_spinor p then
     { p \text{ with } spin = UFOx.Lorentz.rep\_of\_int }  false (-2) }
   else
     p
let force\_majorana p =
   if is\_spinor p then
     \{ p \text{ with } spin = UFOx.Lorentz.rep\_of\_int true 2 \}
   else
     p
```

```
let is\_majorana p =
       match UFOx.Lorentz.omega p.spin with
         Coupling.Majorana \mid Coupling.Vectorspinor \mid Coupling.Maj\_Ghost \rightarrow true
         _{-} 
ightarrow false
    let is\_ghost p =
       p.ghost\_number \neq 0
    let is\_goldstone p =
       p.goldstone
    let is_physical p =
       \neg (is\_ghost \ p \lor is\_goldstone \ p)
    let filter predicate map =
       SMap.filter (fun symbol p \rightarrow predicate p) <math>map
  end
module type UFO\_Coupling =
  sig
     type t = private
       \{ name : string; 
          value : UFOx.Expr.t;
          order : (string \times int) \ list \}
    val of_file : S.t \rightarrow t SMap.t
    val to\_string : string \rightarrow t \rightarrow string
  end
module UFO\_Coupling : UFO\_Coupling =
     type t =
       \{ name : string; 
          value : UFOx.Expr.t;
          order : (string \times int) \ list \}
    let order_to_string orders =
       String.concat ", "
          (List.map (fun (s, i) \rightarrow Printf.sprintf ",%s',:%d" s i) orders)
    let to\_string\ symbol\ c\ =
       Printf.sprintf
          "coupling: _\%s_=>_ [name_=_'\%s', _value_=_'\%s', _order_=_ [\%s]]"
          symbol c.name (expr_to_string c.value) (order_to_string c.order)
    \mathsf{let}\ \mathit{of\_file1}\ \mathit{map}\ \mathit{d}\ =
       \mathsf{let}\ symbol\ =\ d.S.name\ \mathsf{in}
       match d.S.kind, d.S.attribs with
       | ["Coupling"], attribs \rightarrow
           let required query name =
             required_handler "coupling" symbol attribs query name in
           let name = required string\_attrib "name" in
           warn_symbol_name "couplings" symbol name;
           valid_fortran_id "coupling" name;
           SMap.add symbol
             \{ name; 
                value = UFOx.Expr.of_string (required string_attrib "value");
                order = required order_dictionary_attrib "order" } map
       \bot → invalid\_arg ("UFO_Coupling.of_file:\Box" \hat{} name\_to\_string d.S.kind)
     let of_{-}file \ couplings =
       List.fold_left of_file1 SMap.empty couplings
  end
module type Coupling\_Order =
```

```
sig
    type t = private
       \{ name : string; 
         expansion\_order : int;
         hierarchy: int }
    val of_file : S.t \rightarrow t SMap.t
    \mathsf{val}\ to\_string\ :\ string\ \to\ t\ \to\ string
  end
module Coupling\_Order : Coupling\_Order =
    type t =
       \{ name : string;
         expansion\_order : int;
         hierarchy : int }
    let to\_string\ symbol\ c\ =
       Printf.sprintf
         "coupling_order:_{\sqcup}%s_{\sqcup}=>_{\sqcup}[name_{\sqcup}=_{\sqcup}'%s',_{\sqcup}\
   uuuuuuuuuuuuuuuuuuexpansion_order_{u}=_{u}'%d',_{u}\
   uuuuuuuuuuuuuuuuuuuuuuuuhierarchyu=u%d]"
         symbol c.name c.expansion_order c.hierarchy
    let of\_file1 \ map \ d =
       let \ symbol = d.S.name \ in
       match d.S.kind, d.S.attribs with
       | [ "CouplingOrder" |, attribs \rightarrow
          let required query name =
             required\_handler "coupling\sqcuporder" symbol attribs query name in
          let name = required string\_attrib "name" in
          warn_symbol_name "coupling_orders" symbol name;
          SMap.add symbol
             \{ name;
               expansion\_order \ = \ required \ integer\_attrib \ "expansion\_order";
               hierarchy = required integer_attrib "hierarchy" } map
       | \_ \rightarrow invalid\_arg ("Coupling\_order.of\_file:_\" ^ name\_to\_string d.S.kind)
    let of_file coupling_orders =
       List.fold_left of_file1 SMap.empty coupling_orders
module type Lorentz_UFO =
If the name attribute of a Lorentz object does not match the the name of the object, we need the latter for
weeding out unused Lorentz structures (see Vertex.contains below). Therefore, we keep it around.
    \mathsf{type}\ t\ =\ \mathsf{private}
       \{ name : string; 
         symbol : string;
         spins : int list;
         structure : UFOx.Lorentz.t }
    val of_file : S.t \rightarrow t SMap.t
    val to\_string : string \rightarrow t \rightarrow string
  end
module Lorentz\_UFO : Lorentz\_UFO =
  struct
    type t =
       \{ name : string;
         symbol: string;
         spins : int list;
```

```
structure : UFOx.Lorentz.t }
     let to\_string\ symbol\ l\ =
        Printf.sprintf
          "lorentz:_{\square}%s_{\square}=>_{\square}[name_{\square}=_{\square}'%s',_{\square}spins_{\square}=_{\square}[%s],_{\square}\
    uuuuuuuuuuuuuuustructureu=u%s]"
          symbol l.name
          (String.concat ", " (List.map string\_of\_int l.spins))
          (UFOx.Lorentz.to_string l.structure)
    let of_fle1 map d =
       let symbol = d.S.name in
       match d.S.kind, d.S.attribs with
       \mid [ "Lorentz" \mid, attribs \rightarrow
           let required query name =
              required_handler "lorentz" symbol attribs query name in
           let \ name \ = \ required \ string\_attrib \ "name" \ in
           warn_symbol_name "lorentz" symbol name;
           valid_fortran_id "lorentz" symbol;
           SMap.add symbol
              \{ name;
                symbol;
                spins = required integer_list_attrib "spins";
                structure =
                   UFOx.Lorentz.of_string (required string_attrib "structure") } map
       \bot \rightarrow invalid\_arg ("Lorentz.of_file:\Box" \hat{} name\_to\_string d.S.kind)
    let of_file lorentz =
       List.fold_left of_file1 SMap.empty lorentz
  end
module type Vertex =
  sig
     type lcc = private (* Lorentz-color-coupling *)
       { lorentz : string;
          color: UFOx.Color.t;
          coupling : string }
    type t = private
       \{ name : string; 
          particles: string array;
          lcc : lcc list }
    val of_file : Particle.t \ SMap.t \rightarrow S.t \rightarrow t \ SMap.t
    val to\_string : string \rightarrow t \rightarrow string
    val to_string_expanded :
       Lorentz\_UFO.t\ SMap.t\ 	o\ UFO\_Coupling.t\ SMap.t\ 	o\ t\ 	o\ string
    val\ contains\ :\ Particle.t\ SMap.t\ 	o\ (Particle.t\ 	o\ bool)\ 	o\ t\ 	o\ bool
    \mathsf{val}\ \mathit{filter}\ :\ (t\ \to\ \mathit{bool})\ \to\ t\ \mathit{SMap.t}\ \to\ t\ \mathit{SMap.t}
  end
module Vertex : Vertex =
  struct
     type lcc =
       { lorentz : string;
          color: UFOx.Color.t;
          coupling : string }
    type t =
       \{ name : string; 
          particles: string array;
          lcc : lcc list }
    let to\_string\ symbol\ c\ =
```

```
Printf.sprintf
      "vertex:_{\square}%s_{\square}=>_{\square}[name_{\square}=_{\square}'%s',_{\square}particles_{\square}=_{\square}[%s],_{\square}\
uuuuuuuuuuuuuulorentz-color-couplingsu=u[%s]"
      symbol\ c.name
      (String.concat
          ", \_" (Array.to\_list \ c.particles))
      (String.concat
          ",∟"
          (List.map
              (fun lcc \rightarrow
                 Printf.sprintf
                    "%s_*_%s_*_%s"
                    lcc.coupling lcc.lorentz
                    (UFOx.Color.to\_string\ lcc.color))
               c.lcc))
 let to\_string\_expanded lorentz couplings c =
   let expand\_lorentz s =
      try
         UFOx.Lorentz.to\_string\ (SMap.find\ s\ lorentz).Lorentz\_UFO.structure
      with
      \mid Not\_found \rightarrow "?" in
    Printf.sprintf
      "expanded: $_{\square}[\%s]_{\square} - >_{\square} \{_{\square}lorentz-color-couplings_{\square} =_{\square}[\%s]_{\square}\}"
      (String.concat ", " (Array.to\_list c.particles))
      (String.concat
          ",□"
          (List.map
               (fun lcc \rightarrow
                 Printf.sprintf
                    "%s_*_%s_*_%s"
                    lcc.coupling (expand_lorentz lcc.lorentz)
                    (UFOx.Color.to\_string\ lcc.color))
               c.lcc))
 let \ contains \ particles \ predicate \ v =
   let p = v.particles in
   let rec contains' i =
      if i < 0 then
         false
      else if predicate (SMap.find p.(i) particles) then
         true
      else
         contains' (pred i) in
    contains' (Array.length p-1)
 let \ \mathit{force\_adj\_identity1} \ \ \mathit{adj\_indices} \ = \ \mathsf{function}
    | UFOx.Color\_Atom.Identity(a, b) as atom \rightarrow
        begin match List.mem a adj_indices, List.mem b adj_indices with
          true, true \rightarrow UFOx.Color\_Atom.Identity8 (a, b)
          \mathsf{false},\,\mathsf{false} \to \ \mathit{atom}
         true, false | false, true \rightarrow
            invalid_arg "force_adj_identity:∟mixed_representations!"
   | \ atom \ \rightarrow \ atom
 let force_adj_identity adj_indices tensor =
    UFOx. Color.map_atoms (force_adj_identity1 adj_indices) tensor
 let find_adj_indices map particles =
   let adj\_indices = ref[] in
    Array.iteri
      (fun i p \rightarrow
```

```
(* We must pattern match against the O'Mega representation, because UFOx.Color.r is abstract.
*)
           match UFOx. Color.omega (SMap.find p map). Particle.color with
              Color.AdjSUN \_ \rightarrow adj\_indices := succ i :: !adj\_indices
              - \rightarrow ())
         particles;
       !adj\_indices
    let classify_color_indices map particles =
       let fund\_indices = ref[]
       and conj\_indices = ref[]
       and adj\_indices = ref[] in
       Array.iteri
         (fun i p \rightarrow
           (* We must pattern match against the O'Mega representation, because UFOx.Color.r is abstract.
*)
           match UFOx. Color.omega (SMap.find p map).Particle.color with
             Color.SUN \ n \rightarrow
               if n > 0 then
                 fund\_indices := succ i :: !fund\_indices
               else if n < 0 then
                 conj\_indices := succ i :: !conj\_indices
               else
                 failwith "classify_color_indices:_\SU(0)"
             Color.AdjSUN \ n \rightarrow
               if n \neq 0 then
                 adj\_indices := succ i :: !adj\_indices
                 failwith "classify_color_indices: □SU(0)"
            | - \rightarrow () \rangle
         particles;
       (!fund\_indices, !conj\_indices, !adj\_indices)
FIXME: would have expected the opposite order ...
    let \ force\_identity1 \ (fund\_indices, \ conj\_indices, \ adj\_indices) \ = \ function
       | UFOx.Color\_Atom.Identity\ (a,\ b) as atom\ 	o
          if List.mem a fund_indices then
             begin
               if List.mem\ b\ conj\_indices then
                  UFOx.Color\_Atom.Identity(b, a)
               else
                 invalid\_arg "force_adj_identity:\_mixed\_representations!"
             end
          else if List.mem a conj_indices then
             begin
               if List.mem\ b\ fund\_indices then
                  UFOx.Color\_Atom.Identity(a, b)
                 invalid_arg "force_adj_identity: _mixed_representations!"
             end else if List.mem a adj_indices then begin
               if List.mem\ b\ adj\_indices then
                  UFOx.Color\_Atom.Identity8 (a, b)
                 invalid\_arg "force_adj_identity:\(\_\mixed\)\(\_\representations!\)"
             end
          else
             atom
       | atom \rightarrow atom |
    let force_identity indices tensor =
       UFOx.Color.map_atoms (force_identity1 indices) tensor
```

Here we don't have the Lorentz structures available yet. Thus we set $fermion_lines = []$ for now and correct this later.

```
let of\_file1 particle\_map map d =
       let \ symbol = d.S.name \ in
       match d.S.kind, d.S.attribs with
       | ["Vertex"], attribs \rightarrow
           let required query name =
             required\_handler "vertex" symbol attribs query name in
           let name = required string\_attrib "name" in
           warn_symbol_name "vertices" symbol name;
           let particles =
             Array.of_list (required (name_list_attrib ~strip: "P") "particles") in
           let color =
             let indices = classify\_color\_indices particle\_map particles in
             Array.of_list
               (List.map)
                   (force\_identity\ indices\ <*>\ UFOx.Color.of\_string)
                   (required string_list_attrib "color"))
           and lorentz =
             Array.of_list (required (name_list_attrib ~strip:"L") "lorentz")
           and couplings\_alist =
             required (coupling_dictionary_attrib ~strip:"C") "couplings" in
           let \ lcc =
             List.map
               (\text{fun }(i, j, c) \rightarrow
                  \{ lorentz = lorentz.(j); 
                    color = color.(i);
                    coupling = c 
               couplings_alist in
           SMap.add symbol { name; particles; lcc } map
       \bot \rightarrow invalid\_arg ("Vertex.of_file:\Box" \hat{} name\_to\_string d.S.kind)
    let of_file particles vertices =
       List.fold_left (of_file1 particles) SMap.empty vertices
    let filter predicate map =
       SMap.filter (fun symbol p \rightarrow predicate p) map
  end
module type Parameter =
  sig
    type nature = private Internal \mid External
    \mathsf{type}\ ptype\ =\ \mathsf{private}\ Real\ \mid\ Complex
    type t = private
       \{ name : string;
         nature: nature;
         ptype: ptype;
         value: value;
          texname : string;
          lhablock : string option;
         lhacode: int list option;
         sequence \ : \ int \ \}
    val of_file : S.t \rightarrow t SMap.t
    val to\_string : string \rightarrow t \rightarrow string
    val missing : string \rightarrow t
  end
module Parameter : Parameter =
  struct
```

```
type nature = Internal \mid External
 let nature\_to\_string = function
    |Internal \rightarrow "internal"
   \mid External 
ightarrow "external"
 let nature\_of\_string = function
      \texttt{"internal"} \to \mathit{Internal}
      "external" 	o External
    s \rightarrow invalid\_arg ("Parameter.nature\_of\_string:_\" ^ s)
 type ptype = Real \mid Complex
let ptype\_to\_string = function
     Real \rightarrow "real"
     Complex \rightarrow "complex"
let ptype\_of\_string = function
     "real" \rightarrow Real
      "complex" \rightarrow Complex
    s \rightarrow invalid\_arg ("Parameter.ptype\_of\_string:_\" ^ s)
 type t =
   \{ name : string; 
      nature: nature;
      ptype: ptype;
      value: value;
      texname : string;
      lhablock: string option;
      lhacode: int list option;
      sequence : int }
let to\_string\ symbol\ p\ =
   Printf.sprintf
      "parameter:_{\square}%_{\square}=_{\square}[#%d,_{\square}name_{\square}=_{\square}'%s',_{\square}nature_{\square}=_{\square}%s,_{\square}\"parameter:_{\square}%s,_{\square}\"
uuuuuuuuuuuuuuuuuuuuvalue<sub>u</sub>=<sub>u</sub>%s,<sub>u</sub>texname<sub>u</sub>=<sub>u</sub>'%s',<sub>u</sub>\
\verb| uuuuuuuuuuuuuuuuuuuuuulhablock_= | %s, ulhacode_= | [%s] ] "
      symbol\ p.sequence\ p.name
      (nature\_to\_string\ p.nature)
      (ptype\_to\_string\ p.ptype)
      (value_to_string p.value) p.texname
      (\mathsf{match}\ p.\mathit{lhablock}\ \mathsf{with}\ \mathit{None}\ \rightarrow\ "???"\ |\ \mathit{Some}\ s\ \rightarrow\ s)
      (match p.lhacode with
        None \rightarrow ""
       Some \ c \rightarrow String.concat ", " (List.map string\_of\_int c))
let of_file1 (map, n) d =
   let symbol = d.S.name in
   match d.S.kind, d.S.attribs with
    \mid [ "Parameter" \mid, attribs \rightarrow
       let required query name =
          required_handler "particle" symbol attribs query name in
       let \ name = required \ string\_attrib "name" in
       warn_symbol_name "parameters" symbol name;
       valid_fortran_id "parameter" name;
       (SMap.add symbol
           \{ name;
              nature = nature_of_string (required string_attrib "nature");
              ptype = ptype_of_string (required string_attrib "type");
              value = required value_attrib "value";
              texname = required string_attrib "texname";
              lhablock =
                 (try Some (string_attrib "lhablock" attribs) with
                     Not\_found \rightarrow None);
              lhacode =
```

```
(try Some (integer_list_attrib "lhacode" attribs) with
                       Not\_found \rightarrow None);
                 sequence = n \} map, succ n
       | \_ \rightarrow invalid\_arg ("Parameter.of\_file:_\" ^ name\_to\_string d.S.kind)
    let of_file parameters =
       let map, _ = List.fold_left of_file1 (SMap.empty, 0) parameters in
       map
    let missing name =
       \{ name; 
         nature = External;
         ptype = Real;
         value = Integer 0;
          texname = Printf.sprintf "\\texttt{%s}" name;
          lhablock = None;
          lhacode = None;
         sequence = 0
  end
Macros are encoded as a special S. declaration with S.kind = "\$". This is slightly hackish, but general enough
and the overhead of a special union type is probably not worth the effort.
module type Macro =
  sig
    \mathsf{type}\ t
    val\ empty : t
The domains and codomains are still a bit too much ad hoc, but it does the job.
    val\ define: t \rightarrow string \rightarrow S.value \rightarrow t
    val\ expand\_string: t \rightarrow string \rightarrow S.value
    val\ expand\_expr\ :\ t\ 	o\ S.string\_atom\ list\ 	o\ string
Only for documentation:
    \mathsf{val}\ expand\_atom\ :\ t\ \to\ S.string\_atom\ \to\ string
  end
module Macro : Macro =
  struct
    type t = S.value SMap.t
    let empty = SMap.empty
    let define macros name expansion =
       SMap.add name expansion macros
    let\ expand\_string\ macros\ name\ =
       SMap.find name macros
    let rec expand\_atom \ macros = function
       \mid S.Literal \ s \rightarrow s
       | S.Macro [name] \rightarrow
           begin
             try
               begin match SMap.find name macros with
                \mid S.String \ s \rightarrow s
                 S.String\_Expr\ expr\ 	o \ expand\_expr\ macros\ expr
               | \_ \rightarrow invalid\_arg \; ("expand\_atom:\_not\_a\_string:\_" \; \hat{\ } \; name)
               end
             with
               Not\_found \rightarrow invalid\_arg ("expand\_atom: \_not\_found: \_" ^ name)
         S.Macro[] \rightarrow invalid\_arg "expand_atom:_empty"
       \mid S.Macro \ name \rightarrow
           invalid_arg ("expand_atom: □compound □name: □" ^ String.concat "." name)
```

```
and expand\_expr\ macros\ expr\ =
       String.concat "" (List.map (expand_atom macros) expr)
  end
module type Propagator\_UFO =
  sig
    type t = (* private *)
       \{ name : string;
         numerator : UFOx.Lorentz.t;
          denominator : UFOx.Lorentz.t }
    val of_file : S.t \rightarrow t SMap.t
    val to\_string : string \rightarrow t \rightarrow string
  end
module Propagator\_UFO : Propagator\_UFO =
  struct
    type t =
       \{ name : string; 
         numerator : UFOx.Lorentz.t;
          denominator : UFOx.Lorentz.t }
    let to\_string\ symbol\ p\ =
       Printf.sprintf
          "propagator:_{\square}%s_{\square}=>_{\square}[name_{\square}=_{\square}'%s',_{\square}numerator_{\square}=_{\square}'%s',_{\square}\
   uuuuuuuuuuuuuuuuuudenominatoru=u'%s']"
         symbol p.name
          (UFOx.Lorentz.to\_string\ p.numerator)
         (UFOx.Lorentz.to\_string\ p.denominator)
The denominator attribute is optional and there is a default (cf. arXiv:1308.1668)
    let default\_denominator =
       "P('mu', _id) _ * _P('mu', _id) _ \
   uuuuuuu-uMass(id)u*uMass(id)u\
   \verb| uuuuuuu+ucomplex(0,1)u*uMass(id)u*uWidth(id)"|
    let of\_string\_with\_error\_correction\ symbol\ num\_or\_den\ s =
       try
          UFOx.Lorentz.of\_string\ s
       with
       | Invalid\_argument \ msg \rightarrow
           begin
             let \mathit{fixed} = s \, \hat{\ } ")" in
             try
               let \ tensor = UFOx.Lorentz.of\_string \ fixed \ in
               Printf.eprintf
                  \verb|"UFO.Propagator.of_string:|| added_{\sqcup}missing_{\sqcup}closing_{\sqcup}parenthesis_{\sqcup} \setminus
   \verb"uuuuuuuuuuuuuuin_0\%s_0of_0\%s:_0\"\%s\"\""
                  num\_or\_den\ symbol\ s;
               tensor
             with
             \mid Invalid\_argument \_ \rightarrow
                 invalid\_arg
                   (Printf.sprintf
                       num_or_den symbol msg fixed)
           end
    let of_file1 (macros, map) d =
       let \ symbol = d.S.name \ in
       match d.S.kind, d.S.attribs with
       \mid [ "Propagator" \mid, attribs 
ightarrow
```

```
let required query name =
             required_handler "particle" symbol attribs query name
           and optional query name default =
             optional_handler attribs query name default in
          let name = required string\_attrib "name" in
           warn_symbol_name "propagators" symbol name;
           {\tt let} \ num\_string\_expr \ = \ required \ string\_expr\_attrib \ \verb"numerator"
           and den\_string =
             begin match optional find_attrib "denominator"
                                       (S.String default_denominator) with
               S.String \ s \rightarrow s
               S.Name [n] \rightarrow
                 begin match Macro.expand_string macros n with
                 | S.String s \rightarrow s
                 \mid _ 
ightarrow invalid_arg "Propagator.denominator"
                 end
             \bot \rightarrow invalid\_arg "Propagator.denominator:\Box"
             end in
           \mathsf{let}\ num\_string\ =\ Macro.expand\_expr\ macros\ num\_string\_expr\ \mathsf{in}
           let numerator =
             of_string_with_error_correction symbol "numerator" num_string
           and denominator =
             of\_string\_with\_error\_correction\ symbol "denominator" den\_string\ in
           (macros, SMap.add symbol { name; numerator; denominator } map)
       ["\$"], [macro] \rightarrow
           begin match macro.S.a\_value with
           \mid S.String \_ as s \rightarrow
               (Macro.define\ macros\ symbol\ s,\ map);
           \mid S.String\_Expr\ expr\ \rightarrow
              let \ expanded = S.String \ (Macro.expand\_expr \ macros \ expr) \ in
              (Macro.define macros symbol expanded, map)
           \bot \rightarrow invalid\_arg ("Propagator:of\_file:\_not\_a\_string\_" ^ symbol)
           end
       \mid [ "$" \mid, \mid \mid \rightarrow
           invalid_arg ("Propagator:of_file: _ empty_declaration_" ^ symbol)
       | ["\$"], \_ \rightarrow
           invalid_arg ("Propagator:of_file:umultipleudeclarationu" ^ symbol)
       | \_ \rightarrow invalid\_arg ("Propagator:of\_file:_\" ^ name\_to\_string d.S.kind)
    let of_file propagators =
       let _{-}, propagators' =
          List.fold\_left\ of\_file1\ (Macro.empty,\ SMap.empty)\ propagators\ in
       propagators'
  end
module type Decay =
  sig
     type t = private
       \{ name : string; 
          particle : string;
          widths : (string \ list \times string) \ list \}
     val of_file : S.t \rightarrow t SMap.t
    \mathsf{val}\ to\_string\ :\ string\ \to\ t\ \to\ string
  end
module Decay : Decay =
  struct
     type t =
       \{ name : string; 
          particle : string;
```

```
widths : (string \ list \times string) \ list \}
    let width_to_string ws =
       String.concat ",\Box"
          (List.map
             (\mathsf{fun}\ (ps,\ w)\ \to
               "(" ^ String.concat ", _ " ps ^ ") _ ->, ' " ^ w ^ ", ")
    let to\_string\ symbol\ d\ =
       Printf.sprintf
          "decay:_{\square}%s_{\square}=>_{\square}[name_{\square}=_{\square}'%s',_{\square}particle_{\square}=_{\square}'%s',_{\square}widths_{\square}=_{\square}[%s]]"
          symbol d.name d.particle (width_to_string d.widths)
    let of\_file1 \ map \ d =
       let symbol = d.S.name in
       match d.S.kind, d.S.attribs with
       | ["Decay"], attribs \rightarrow
           let required query name =
             required_handler "particle" symbol attribs query name in
           let name = required string\_attrib "name" in
           warn_symbol_name "decays" symbol name;
           SMap.add symbol
             \{ name;
               particle = required (name_attrib ~strip:"P") "particle";
               widths = required decay_dictionary_attrib "partial_widths" } map
       \bot \rightarrow invalid\_arg ("Decay.of_file:\bot" \hat{} name\_to\_string d.S.kind)
    let of_file decays =
       List.fold_left of_file1 SMap.empty decays
  end
We can read the spinor representations off the vertices to check for consistency.
   Note that we have to conjugate the representations!
let collect_spinor_reps_of_vertex particles lorentz v sets =
  List.fold\_left
    (fun sets' lcc \rightarrow
       let l = (SMap.find\ lcc.Vertex.lorentz\ lorentz).Lorentz\_UFO.structure\ in
       List.fold\_left
          (fun (spinors, conj\_spinors as sets'') (i, rep) \rightarrow
            let p = v.Vertex.particles.(pred i) in
            match UFOx.Lorentz.omega rep with
              Coupling.ConjSpinor \rightarrow (SSet.add\ p\ spinors,\ conj\_spinors)
              Coupling.Spinor \rightarrow (spinors, SSet.add \ p \ conj\_spinors)
            |  \rightarrow  sets'')
         sets' (UFOx.Lorentz.classify_indices l))
    sets v. Vertex.lcc
let collect_spinor_reps_of_vertices particles lorentz vertices =
  SMap.fold
     (fun \ v \rightarrow collect\_spinor\_reps\_of\_vertex \ particles \ lorentz \ v)
     vertices (SSet.empty, SSet.empty)
let lorentz\_reps\_of\_vertex\ particles\ v\ =
  (List.map)
        (fun p \rightarrow
           (* Why do we need to conjugate??? *)
           UFOx.Lorentz.rep\_conjugate
             (SMap.find p particles).Particle.spin)
        (Array.to\_list\ v.Vertex.particles))
let rep_compatible rep_vertex rep_particle =
```

```
let open UFOx.Lorentz in
  let open Coupling in
  match omega rep_vertex, omega rep_particle with
     (Spinor \mid ConjSpinor), Majorana \rightarrow true
    r1, r2 \rightarrow r1 = r2
{\tt let} \ reps\_compatible \ reps\_vertex \ reps\_particles \ =
  List.for\_all2
     (\text{fun } (iv, rv) (ip, rp) \rightarrow iv = ip \land rep\_compatible rv rp)
     reps_vertex reps_particles
{\tt let} \ check\_lorentz\_reps\_of\_vertex \ particles \ lorentz \ v \ = \\
  let reps_particles =
     List.sort\ compare\ (lorentz\_reps\_of\_vertex\ particles\ v) in
  List.iter
     (fun lcc \rightarrow
       let l = (SMap.find\ lcc.Vertex.lorentz\ lorentz).Lorentz\_UFO.structure in
       let reps\_vertex = List.sort\ compare\ (UFOx.Lorentz.classify\_indices\ l) in
       if \neg (reps\_compatible reps\_vertex reps\_particles) then begin
          Printf.eprintf "%s_{\sqcup}<>_{\sqcup}%s_{\sqcup}[%s] \n"
             (UFOx.Index.classes\_to\_string)
                 UFOx.Lorentz.rep_to_string reps_particles)
             (UFOx.Index.classes_to_string)
                 UFOx.Lorentz.rep_to_string reps_vertex)
             v.Vertex.name \ (*(Vertex.to\_string \ v.Vertex.name \ v) \ *);
          (* invalid_arg "check_lorentz_reps_of_vertex" *) ()
       end)
     v.\ Vertex.lcc
let color\_reps\_of\_vertex\ particles\ v\ =
   ThoList.alist\_of\_list \ \ \ predicate: (\neg <*> UFOx.Color.rep\_trivial) \ \ \ \ \ \ \ offset: 1
     (List.map)
         (fun \ p \rightarrow (SMap.find \ p \ particles).Particle.color)
         (Array.to\_list\ v.Vertex.particles))
let \ check\_color\_reps\_of\_vertex \ particles \ v =
  let reps_particles =
     List.sort compare (color_reps_of_vertex particles v) in
  List.iter
     (fun lcc \rightarrow
       let reps_vertex =
          List.sort compare (UFOx.Color.classify_indices lcc.Vertex.color) in
       if reps\_vertex \neq reps\_particles then begin
          Printf.printf "%s_{\sqcup}<>_{\sqcup}%s_{\square}"
             (\mathit{UFOx}.\mathit{Index}.\mathit{classes\_to\_string}\ \mathit{UFOx}.\mathit{Color}.\mathit{rep\_to\_string}\ \mathit{reps\_particles})
             (UFOx.Index.classes_to_string UFOx.Color.rep_to_string reps_vertex);
          invalid_arg "check_color_reps_of_vertex"
      end)
     v.Vertex.lcc
module P = Permutation. Default
module type Lorentz =
  sig
     type spins = private
          Unused
          Unique of Coupling.lorentz array
          Ambiguous of Coupling.lorentz array SMap.t
     type t = private
       \{ name : string; 
          n : int;
          spins: spins;
          structure : UFO_Lorentz.t;
```

```
fermion_lines : Coupling.fermion_lines;
           variables : string list }
     val\ required\_charge\_conjugates: t \rightarrow t\ list
     val permute : P.t \rightarrow t \rightarrow t
     val of \_lorentz \_UFO:
        Particle.t \ SMap.t \rightarrow Vertex.t \ SMap.t \rightarrow
        Lorentz\_UFO.t\ SMap.t\ 	o \ t\ SMap.t
     val\ lorentz\_to\_string : Coupling.lorentz \rightarrow string
     val to\_string : string \rightarrow t \rightarrow string
module Lorentz : Lorentz =
  struct
     let rec lorentz\_to\_string = function
          Coupling.Scalar 
ightarrow "Scalar"
          Coupling.Spinor 
ightarrow "Spinor"
          Coupling.ConjSpinor \rightarrow "ConjSpinor"
          Coupling.Majorana \rightarrow "Majorana"
          Coupling.Maj\_Ghost \rightarrow "Maj\_Ghost"
          Coupling.Vector \rightarrow "Vector"
          Coupling.Massive\_Vector \rightarrow "Massive\_Vector"
          Coupling.Vectorspinor 
ightarrow  "Vectorspinor"
          Coupling.Tensor_1 \rightarrow "Tensor_1"
          Coupling.Tensor_2 \rightarrow "Tensor_2"
          Coupling.BRS\ l \rightarrow "BRS(" \cap lorentz\_to\_string\ l \cap ")"
```

Unlike UFO, O'Mega distinguishes bewteen spinors and conjugate spinors. However, we can inspect the particles in the vertices in which a Lorentz structure is used to determine the correct quantum numbers.

Most model files in the real world contain unused Lorentz structures. This is not a problem, we can just ignore them.

```
type spins =
    | Unused
    | Unique of Coupling.lorentz array
    | Ambiguous of Coupling.lorentz array SMap.t
```



Use UFO_targets.Fortran.fusion_name below in order to avoid communication problems. Or even move away from strings alltogether.

```
type t =
    { name : string;
        n : int;
        spins : spins;
        structure : UFO_Lorentz.t;
        fermion_lines : Coupling.fermion_lines;
        variables : string list }

Add one charge conjugated fermion lines.

let charge_conjugate1 l (ket, bra as fermion_line) =
        { name = l.name ^ Printf.sprintf "_c%x%x" ket bra;
        n = l.n;
        spins = l.spins;
        structure = UFO_Lorentz.charge_conjugate fermion_line l.structure;
        fermion_lines = l.fermion_lines;
        variables = l.variables }
```

Add several charge conjugated fermion lines.

```
\label{let:charge_conjugate} \begin{array}{l} let \ charge\_conjugate \ l \ fermion\_lines \\ \\ List.fold\_left \ charge\_conjugate1 \ l \ fermion\_lines \\ \end{array}
```

Add all combinations of charge conjugated fermion lines that don't leave the fusion.

```
\begin{array}{l} \mathsf{let}\ \mathit{required\_charge\_conjugates}\ l = \\ \mathsf{let}\ \mathit{saturated\_fermion\_lines}\ = \\ \mathit{List.filter} \\ (\mathsf{fun}\ (\mathit{ket},\ \mathit{bra})\ \rightarrow\ \mathit{ket}\ \not\equiv\ 1\ \land\ \mathit{bra}\ \not\equiv\ 1) \\ \mathit{l.fermion\_lines}\ \mathsf{in} \\ \mathit{List.map}\ (\mathit{charge\_conjugate}\ l)\ (\mathit{ThoList.power}\ \mathit{saturated\_fermion\_lines}) \\ \mathsf{let}\ \mathit{permute\_spins}\ p = \ \mathsf{function} \\ |\ \mathit{Unused}\ \rightarrow\ \mathit{Unused} \\ |\ \mathit{Unique}\ s\ \rightarrow\ \mathit{Unique}\ (P.\mathit{array}\ p\ s) \\ |\ \mathit{Ambiguous}\ \mathit{map}\ \rightarrow\ \mathit{Ambiguous}\ (\mathit{SMap.map}\ (P.\mathit{array}\ p)\ \mathit{map}) \end{array}
```

Note that we apply the *inverse* permutation to the indices in order to match the permutation of the particles/spins.

```
let permute\_structure \ n \ p \ (l, \ f) =
  let permuted = P.array (P.inverse p) (Array.init n succ) in
  let permute\_index i =
    if i > 0 then
       UFOx.Index.map\_position (fun pos \rightarrow permuted.(pred pos)) i
    else
       i in
  (UFO_Lorentz.map_indices permute_index l,
    UFO\_Lorentz.map\_fermion\_lines\ permute\_index\ f)
let permute p l =
  let structure, fermion_lines =
     permute_structure l.n p (l.structure, l.fermion_lines) in
  \{ name = l.name ` "_p" ` P.to_string (P.inverse p); \}
     n = l.n;
     spins = permute\_spins p l.spins;
     structure;
     fermion\_lines;
     variables = l.variables 
let \ omega\_lorentz\_reps \ n \ alist =
  let reps = Array.make n Coupling.Scalar in
  List.iter
     (\text{fun } (i, rep) \rightarrow reps.(pred i) \leftarrow UFOx.Lorentz.omega rep)
     alist:
  reps
let contained lorentz vertex =
     (fun \ lcc1 \rightarrow lcc1.Vertex.lorentz = lorentz.Lorentz\_UFO.symbol)
     vertex. Vertex.lcc
```

Find all vertices in with the Lorentz structure *lorentz* is used and build a map from those vertices to the O'Mega Lorentz representations inferred from UFO's Lorentz structure and the *particles* involved. Then scan the bindings and check that we have inferred the same Lorentz representation from all vertices.

```
 \begin{array}{l} \text{let } \mathit{lorentz\_reps\_of\_structure } \; \mathit{particles } \; \mathit{vertices } \; \mathit{lorentz} \; = \\ \text{let } \mathit{uses} \; = \\ S\mathit{Map.fold} \\ \text{(fun } \mathit{name } \; \mathit{v} \; \mathit{acc} \; \rightarrow \\ \text{if } \; \mathit{contained } \; \mathit{lorentz} \; \mathit{v} \; \mathsf{then} \\ S\mathit{Map.add} \\ \quad \mathit{name} \\ \quad (\mathit{omega\_lorentz\_reps} \\ \quad (\mathit{Array.length } \; \mathit{v.Vertex.particles}) \\ \quad (\mathit{lorentz\_reps\_of\_vertex } \; \mathit{particles} \; \mathit{v})) \; \mathit{acc} \\ \text{else} \\ \quad \mathit{acc}) \; \mathit{vertices } \; \mathit{SMap.empty} \; \mathsf{in} \\ \text{let } \; \mathit{variants} \; = \\ \quad \mathit{ThoList.uniq } \; (\mathit{List.sort } \; \mathit{compare } \; (\mathit{List.map } \; \mathit{snd } \; (\mathit{SMap.bindings } \; \mathit{uses}))) \; \mathsf{in} \\ \end{aligned}
```

```
match variants with
         [] \rightarrow Unused
        [s] \rightarrow Unique s
          (fun variant \rightarrow
               Printf.eprintf
                 "UFO.Lorentz.lorentz_reps_of_structure: \\"\%s\n"
                 (ThoList.to_string lorentz_to_string (Array.to_list variant)))
             variants:
          Ambiguous\ uses
    let of_lorentz_tensor spins lorentz =
       match spins with
       \mid Unique \ s \rightarrow
          begin
             try
               Some (UFO_Lorentz.parse (Array.to_list s) lorentz)
             \mid Failure msg \rightarrow
                begin
                  prerr\_endline\ msg;
                   Some (UFO_Lorentz.dummy)
          end
       \mid Unused \rightarrow
          Printf.eprintf
             "UFO.Lorentz: ustripping unused structure %s\n"
             (UFOx.Lorentz.to_string lorentz);
          None
       \mid Ambiguous \_ 
ightarrow invalid\_arg "UFO.Lorentz.of_lorentz_tensor:\sqcupAmbiguous"
NB: if the name attribute of a Lorentz object does not match the the name of the object, the former has a
better chance to correspond to a valid Fortran name. Therefore we use it.
    let of_lorentz_UFO particles vertices lorentz_UFO =
       SMap.fold
         (fun name\ l\ acc\ 
ightarrow
           let spins = lorentz\_reps\_of\_structure\ particles\ vertices\ l in
           match of _lorentz_tensor spins l.Lorentz_UFO.structure with
             None \rightarrow acc
             Some \ structure \rightarrow
               SMap.add
                 name
                 \{ name = l.Lorentz\_UFO.symbol; \}
                    n = List.length \ l.Lorentz\_UFO.spins;
                    spins;
                    structure:
                   fermion_lines = UFO_Lorentz.fermion_lines structure;
                    variables = UFOx.Lorentz.variables l.Lorentz_UFO.structure }
                 acc)
         lorentz_UFO SMap.empty
    let to\_string\ symbol\ l\ =
       Printf.sprintf
         "lorentz:_{\square}%s_{\square}=>_{\square}[name_{\square}=_{\square}'%s',_{\square}spins_{\square}=_{\square}%s,_{\square}\
   " fermion_lines_=u%s, _fermion_lines_u=u%s]
         symbol\ l.name
         (match l.spins with
          \mid Unique s \rightarrow
              "[" ^ String.concat
                        ", " (List.map\ lorentz\_to\_string\ (Array.to\_list\ s)) ^ "]"
```

```
 | Ambiguous \_ \rightarrow "AMBIGUOUS!" \\ | Unused \rightarrow "UNUSED!") \\ (UFO\_Lorentz.to\_string \ l.structure) \\ (UFO\_Lorentz.fermion\_lines\_to\_string \ l.fermion\_lines) \\
```

According to arxiv:1308:1668, there should not be a factor of i in the numerators of propagators, but the (unused) propagators.py in most models violate this rule!

```
let divide\_propagators\_by\_i = ref false
module type Propagator =
  sig
    type t = (* private *)
       \{ name : string; 
         spins : Coupling.lorentz × Coupling.lorentz;
         numerator : UFO_Lorentz.t;
         denominator : UFO_Lorentz.t;
         variables : string list }
    val\ of\_propagator\_UFO: ?majorana:bool \rightarrow Propagator\_UFO.t \rightarrow t
    val\ of\_propagators\_UFO: ?majorana:bool 
ightarrow Propagator\_UFO.t\ SMap.t 
ightarrow t\ SMap.t
    val\ transpose\ :\ t\ 
ightarrow\ t
    val to\_string : string \rightarrow t \rightarrow string
module Propagator : Propagator =
  struct
    type t = (* private *)
       \{ name : string; 
         spins : Coupling.lorentz \times Coupling.lorentz;
         numerator : UFO\_Lorentz.t;
         denominator : UFO_Lorentz.t;
         variables : string list }
    let lorentz\_rep\_at rep\_classes i =
       try
         UFOx.Lorentz.omega\ (List.assoc\ i\ rep\_classes)
       with
       | Not\_found \rightarrow Coupling.Scalar
    let imaginary = Algebra. QC. make Algebra. Q. null Algebra. Q. unit
    let scalars = [Coupling.Scalar; Coupling.Scalar]
```

If 51 and 52 show up as indices, we must map $(1,51) \rightarrow (1001,2001)$ and $(2,52) \rightarrow (1002,2002)$, as per the UFO conventions for Lorentz structures.



end

This does not work yet, because $UFOx.Lorentz.map_indices$ affects also the position argument of P, Mass and Width.

```
let contains\_51\_52 tensor = List.exists (fun (i, \_) \rightarrow i = 51 \lor i = 52) (UFOx.Lorentz.classify\_indices\ tensor)

let remap\_51\_52 = function
\mid 1 \rightarrow 1001 \mid 51 \rightarrow 2001
\mid 2 \rightarrow 1002 \mid 52 \rightarrow 2002
\mid i \rightarrow i

let canonicalize\_51\_52 tensor = if\ contains\_51\_52 tensor\ then\ UFOx.Lorentz.rename\_indices\ remap\_51\_52 tensor
```

```
else
          tensor
     let force\_majorana = function
        \mid Coupling.Spinor \mid Coupling.ConjSpinor \rightarrow Coupling.Majorana
         s \rightarrow s
    let string\_list\_union \ l1 \ l2 =
       Sets. String. elements
          (Sets.String.union
             (Sets.String.of_list l1)
             (Sets.String.of\_list \ l2))
In the current conventions, the factor of i is not included:
     let of\_propagator\_UFO ?(majorana = false) p =
       let \ numerator = canonicalize\_51\_52 \ p.Propagator\_UFO.numerator \ in
       let lorentz_reps = UFOx.Lorentz.classify_indices numerator in
       let spin1 = lorentz\_rep\_at\ lorentz\_reps\ 1
       and spin2 = lorentz\_rep\_at\ lorentz\_reps\ 2 in
       let numerator_sans_i =
          if !divide\_propagators\_by\_i then
            UFOx.Lorentz.map\_coeff (fun q \rightarrow Algebra.QC.div q imaginary) numerator
         else
            numerator in
       \{ name = p.Propagator\_UFO.name; \}
          spins =
            if majorana then
               (force_majorana spin1, force_majorana spin2)
               (spin1, spin2);
          numerator =
            UFO_Lorentz.parse ~allow_denominator :true [spin1; spin2] numerator_sans_i;
          denominator = UFO\_Lorentz.parse\ scalars\ p.Propagator\_UFO.denominator;
          variables =
            string\_list\_union
               (UFOx.Lorentz.variables\ p.Propagator\_UFO.denominator)
               (UFOx.Lorentz.variables numerator_sans_i) }
    let of _propagators_UFO ?majorana propagators_UFO =
       SMap.fold
          (fun name p acc \rightarrow SMap.add name (of_propagator_UFO ?majorana p) acc)
          propagators_UFO SMap.empty
    let permute12 = function
        1 \rightarrow 2
         2 \rightarrow 1
       \mid n \rightarrow n
    let transpose\_positions\ t\ =
       UFOx.Index.map\_position\ permute12\ t
    let transpose p =
       \{ name = p.name; 
          spins = (snd \ p.spins, fst \ p.spins);
          numerator = UFO\_Lorentz.map\_indices\ transpose\_positions\ p.numerator;
          denominator = p.denominator;
          variables = p.variables 
     let to\_string\ symbol\ p\ =
       Printf.sprintf
          "propagator:_{\mathsf{U}}%_{\mathsf{S}}=>_{\mathsf{U}}[name_{\mathsf{U}}=_{\mathsf{U}}'%s',_{\mathsf{U}}$pin_{\mathsf{U}}=_{\mathsf{U}}'(%s,_{\mathsf{U}}%s)',_{\mathsf{U}}numerator/_{\mathsf{U}}=_{\mathsf{U}}'%s',_{\mathsf{U}}\
   uuuuuuuuuuuuuuuuuuuuuuuuuuudenominator_=_',%s']"
          symbol p.name
          (Lorentz.lorentz\_to\_string\ (fst\ p.spins))
          (Lorentz.lorentz\_to\_string\ (snd\ p.spins))
```

```
(UFO_Lorentz.to_string p.numerator)
         (UFO\_Lorentz.to\_string\ p.denominator)
  end
type t =
  { particles : Particle.t SMap.t;
    particle_array : Particle.t array; (* for diagnostics *)
    couplings : UFO\_Coupling.t SMap.t;
    coupling_orders : Coupling_Order.t SMap.t;
    vertices: Vertex.t SMap.t;
    lorentz_UFO: Lorentz_UFO.t SMap.t;
    lorentz: Lorentz.t SMap.t;
    parameters : Parameter.t SMap.t;
    propagators_UFO : Propagator_UFO.t SMap.t;
    propagators : Propagator.t SMap.t;
    decays : Decay.t SMap.t;
    nc : int 
let use\_majorana\_spinors = ref false
let fallback_to_majorana_if_necessary particles vertices lorentz_UFO =
  let majoranas =
    SMap.fold
      (fun p particle acc \rightarrow
         if Particle.is_majorana particle then
           SSet.add p acc
         else
           acc)
      particles SSet.empty in
  let spinors, conj\_spinors =
    collect_spinor_reps_of_vertices particles lorentz_UFO vertices in
  let ambiguous =
    SSet.diff (SSet.inter spinors conj_spinors) majoranas in
  let no\_majoranas = SSet.is\_empty majoranas
  and no\_ambiguities = SSet.is\_empty ambiguous in
  if no\_majoranas \land no\_ambiguities \land \neg !use\_majorana\_spinors then
    (SMap.mapi
        (fun p particle \rightarrow
          if SSet.mem p spinors then
            Particle.force_spinor particle
          else if SSet.mem\ p\ conj\_spinors then
            Particle.force_conjspinor particle
          else
            particle)
        particles,
     false)
  else
    begin
      if !use\_majorana\_spinors then
         Printf.eprintf "O'Mega: \Majorana fermions requested. \n";
      if \neg no\_majoranas then
         Printf.eprintf "O'Mega: Lound Lound Majorana Lound fermions! \n";
      if \neg no\_ambiguities then
         Printf.eprintf
           "O'Mega: _found_ambiguous_spinor_representations_for_%s!\n"
           (String.concat ", " (SSet.elements ambiguous));
       Printf.eprintf
         "0'Mega: | falling | back | to | the | Majorana | representation | for | all | fermions. \n";
       (SMap.map Particle.force_majorana particles,
        true)
    end
let nc\_of\_particles particles =
```

```
let nc\_set =
     List.fold\_left
       (fun nc\_set\ (\_,\ p)\ 	o
         match UFOx.Color.omega p.Particle.color with
           Color.Singlet \rightarrow nc\_set
           Color.SUN \ nc \rightarrow Sets.Int.add \ (abs \ nc) \ nc\_set
           Color.AdjSUN \ nc \rightarrow Sets.Int.add \ (abs \ nc) \ nc\_set)
       Sets.Int.empty (SMap.bindings particles) in
  match Sets.Int.elements nc_set with
    [] \rightarrow 0
    [n] \rightarrow n
   nc\_list \rightarrow
      invalid\_arg
        ("UFO.Model: |more| |than| |one| |value| |of| |N_C:| | ^
            String.concat ",\Box" (List.map string\_of\_int nc\_list))
let of_file u =
  let particles = Particle.of_file u.Files.particles in
  let vertices = Vertex.of_file particles u.Files.vertices
  and lorentz\_UFO = Lorentz\_UFO.of\_file\ u.Files.lorentz
  and propagators_UFO = Propagator_UFO.of_file u.Files.propagators in
  let particles, majorana =
    fallback_to_majorana_if_necessary particles vertices lorentz_UFO in
  let particle\_array = Array.of\_list (values particles)
  and lorentz = Lorentz.of_lorentz_UFO particles vertices lorentz_UFO
  and propagators = Propagator of propagators UFO majorana propagators UFO in
  let model =
     { particles;
       particle\_array;
       couplings = UFO_Coupling.of_file u.Files.couplings;
       coupling_orders = Coupling_Order.of_file u.Files.coupling_orders;
       vertices;
       lorentz_UFO;
       lorentz;
       parameters = Parameter.of\_file u.Files.parameters;
       propagators_UFO;
       propagators;
       decays = Decay.of\_file u.Files.decays;
       nc = nc\_of\_particles \ particles \ \} in
  SMap.iter
     (fun \_v \rightarrow
       check_color_reps_of_vertex model.particles v;
       check_lorentz_reps_of_vertex model.particles model.lorentz_UFO v)
     model.vertices;
  model
let parse\_directory dir =
  of_file (Files.parse_directory dir)
let dump model =
  Printf.printf "NC\sqsubseteq=\sqsubseteq%d\n" model.nc;
  SMap.iter\ (print\_endline\ < **>\ Particle.to\_string)\ model.particles;
  SMap.iter\ (print\_endline\ <**>\ UFO\_Coupling.to\_string)\ model.couplings;
  SMap.iter\ (print\_endline\ < ** > Coupling\_Order.to\_string)\ model.coupling\_orders;
  (* SMap.iter (print_endline < ** > Vertex.to_string) model.vertices; *)
  SMap.iter
     (fun symbol\ v\ \rightarrow
       (print\_endline < ** > Vertex.to\_string) \ symbol \ v;
       print\_endline
         (Vertex.to\_string\_expanded\ model.lorentz\_UFO\ model.couplings\ v))
     model.vertices;
  SMap.iter\ (print\_endline\ <**>\ Lorentz\_UFO.to\_string)\ model.lorentz\_UFO;
  SMap.iter\ (print\_endline\ < ** > Lorentz.to\_string)\ model.lorentz;
```

```
SMap.iter\ (print\_endline\ < **>\ Parameter.to\_string)\ model.parameters;
  SMap.iter\ (print\_endline\ < ** > Propagator\_UFO.to\_string)\ model.propagators\_UFO;
  SMap.iter\ (print\_endline\ < **>\ Propagator.to\_string)\ model.propagators;
  SMap.iter\ (print\_endline\ < **>\ Decay.to\_string)\ model.decays;
  SMap.iter
    (fun symbol\ d\ 	o
       List.iter (fun (\_, w) \rightarrow ignore (UFOx.Expr.of\_string w)) d.Decay.widths)
     model.decays
exception Unhandled of string
let unhandled s = raise (Unhandled s)
\mathsf{module}\ \mathit{Model}\ =
  struct
NB: we could use type flavor = Particle.t, but that would be very inefficient, because we will use flavor as a
key for maps below.
    type flavor = int
    type constant = string
    \mathsf{type} \,\, \mathit{gauge} \,\, = \,\, \mathit{unit}
    \mathsf{module}\ M\ =\ Modeltools.Mutable
         (struct type f = flavor type g = gauge type c = constant end)
    let flavors = M.flavors
    let external\_flavors = M.external\_flavors
    let external\_flavors = M.external\_flavors
    let lorentz = M.lorentz
    let color = M.color
    \mathsf{let}\ nc\ =\ M.nc
    let propagator = M.propagator
    \mathsf{let}\ \mathit{width}\ =\ \mathit{M.width}
    let \ goldstone = M.goldstone
    let conjugate = M.conjugate
    let fermion = M.fermion
    let vertices = M.vertices
    let fuse2 = M.fuse2
    let fuse3 = M.fuse3
    let fuse = M.fuse
    let max\_degree = M.max\_degree
    let parameters = M.parameters
    let flavor\_of\_string = M.flavor\_of\_string
    let flavor\_to\_string = M.flavor\_to\_string
    let flavor\_to\_TeX = M.flavor\_to\_TeX
    let flavor\_symbol = M.flavor\_symbol
    let gauge\_symbol = M.gauge\_symbol
    let pdg = M.pdg
    let \ mass\_symbol = M.mass\_symbol
    let \ width\_symbol = M.width\_symbol
    let constant\_symbol = M.constant\_symbol
    module Ch = M.Ch
    let charges = M.charges
    let rec fermion\_of\_lorentz = function
         Coupling.Spinor \rightarrow 1
         Coupling.ConjSpinor \rightarrow -1
         Coupling.Majorana \rightarrow 2
         Coupling.Maj\_Ghost \rightarrow 2
         Coupling. Vectorspinor \rightarrow 1
         Coupling.Vector \mid Coupling.Massive\_Vector \rightarrow 0
         Coupling.Scalar \mid Coupling.Tensor\_1 \mid Coupling.Tensor\_2 \rightarrow 0
         Coupling.BRS\ f \rightarrow fermion\_of\_lorentz\ f
    module Q = Algebra.Q
```

```
module QC = Algebra.QC
let dummy\_tensor3 = Coupling.Scalar\_Scalar\_Scalar 1
let dummy\_tensor4 = Coupling.Scalar4 1
let triplet p = (p.(0), p.(1), p.(2))
\mathsf{let}\ quartet\ p\ =\ (p.(0),\ p.(1),\ p.(2),\ p.(3))
let half\_times \ q1 \ q2 =
  Q.mul\ (Q.make\ 1\ 2)\ (Q.mul\ q1\ q2)
let name \ q =
  g. UFO_Coupling.name
let fractional\_coupling g r =
  let g = name g in
  match Q.to\_ratio \ r with
    0, \; \_ \; \rightarrow \; "0.0_default"
    1, 1 \rightarrow g
    -1, 1 \rightarrow Printf.sprintf "(-\%s)" q
  n, 1 \rightarrow Printf.sprintf "(%d*%s)" n g
  | 1, d \rightarrow Printf.sprintf "(%s/%d)" g d
    -1, d \rightarrow Printf.sprintf "(-\%s/\%d)" g d
  | n, d \rightarrow Printf.sprintf "(%d*%s/%d)" n g d
let lorentz_of_symbol model symbol =
  try
     SMap.find symbol model.lorentz
  with
  | Not\_found \rightarrow invalid\_arg ("lorentz\_of\_symbol: " ^ symbol)|
let lorentz_UFO_of_symbol model symbol =
  try
     SMap.find symbol model.lorentz_UFO
  with
  Not\_found \rightarrow invalid\_arg ("lorentz\_UFO\_of\_symbol: \_" ^ symbol)
let coupling_of_symbol model symbol =
     SMap.find symbol model.couplings
  with
  Not\_found \rightarrow invalid\_arg ("coupling\_of\_symbol:_\" ^ symbol)
let spin_triplet model name =
  match (lorentz_of_symbol model name).Lorentz.spins with
    Lorentz. Unique [|s0; s1; s2|] \rightarrow (s0, s1, s2)
    Lorentz.Unique \_ \rightarrow invalid\_arg "spin_triplet: \_wrong\_number\_of\_spins"
    Lorentz.Unused \rightarrow invalid\_arg "spin_triplet: \sqcupUnused"
    Lorentz.Ambiguous \_ \rightarrow invalid\_arg "spin_triplet:\sqcupAmbiguous"
let spin\_quartet\ model\ name\ =
  match (lorentz_of_symbol model name).Lorentz.spins with
    Lorentz. Unique [|s0; s1; s2; s3|] \rightarrow (s0, s1, s2, s3)
    Lorentz.Unique \_ \rightarrow invalid\_arg "spin_quartet:\_wrong\_number\_of\_spins"
    Lorentz.Unused \rightarrow invalid\_arg "spin_quartet:_\Unused"
    Lorentz.Ambiguous \_ \rightarrow invalid\_arg "spin_quartet:\BoxAmbiguous"
let spin\_multiplet model name =
  match (lorentz_of_symbol model name).Lorentz.spins with
    Lorentz.Unique\ sarray\ 	o\ sarray
    Lorentz.Unused \rightarrow invalid\_arg "spin_multiplet: \sqcupUnused"
    Lorentz.Ambiguous \_ \rightarrow invalid\_arg "spin_multiplet:\sqcupAmbiguous"
```

If we have reason to believ that a δ_{ab} -vertex is an effective $\operatorname{tr}(T_a T_b)$ -vertex generated at loop level, like $gg \to H \dots$ in the SM, we should interpret it as such and use the expression (6.2) from [16].

AFAIK, there is no way to distinguish these cases directly in a UFO file. Instead we rely in a heuristic, in which each massless color octet vector particle or ghost is a gluon and colorless scalars are potential Higgses.

```
let is\_massless p =
  match ThoString.uppercase p.Particle.mass with
     "ZERO" \rightarrow true
       \rightarrow false
let is\_gluon \ model \ f =
  let p = model.particle\_array.(f) in
  match UFOx. Color. omega p. Particle. color,
          UFOx.Lorentz.omega p.Particle.spin with
     Color.AdjSUN _, Coupling.Vector \rightarrow is\_massless p
     Color.AdjSUN _, Coupling.Scalar \rightarrow
      if p.Particle.ghost\_number \neq 0 then
         is\_massless p
      else
         false

ightarrow false
let is\_color\_singlet model f =
  let p = model.particle\_array.(f) in
  match UFOx. Color. omega p. Particle. color with
     Color.Singlet \rightarrow true

ightarrow false
let is_higgs_gluon_vertex model p adjoints =
  if Array.length p > List.length adjoints then
     List.for\_all
        (\mathsf{fun}\ (i,\ p)\ \to
          if List.mem i adjoints then
             is\_gluon \ model \ p
          else
             is\_color\_singlet model p)
        (ThoList.enumerate\ 1\ (Array.to\_list\ p))
  else
     false
let delta8\_heuristics model p a b =
  if is\_higgs\_gluon\_vertex \ model \ p \ [a; \ b] then
     Color. Vertex. delta8_loop a b
  else
     Color. Vertex. delta8 a b
let verbatim\_higgs\_glue = ref false
let translate\_color\_atom \ model \ p = function
     UFOx.Color\_Atom.Identity\ (i,\ j) \rightarrow Color.Vertex.delta3\ j\ i
     UFOx.Color\_Atom.Identity8\ (a,\ b)\ \rightarrow
      if !verbatim_higgs_glue then
         Color. Vertex. delta8 a b
      else
         delta8\_heuristics\ model\ p\ a\ b
     UFOx.Color\_Atom.T (a, i, j) \rightarrow Color.Vertex.t a i j
     UFOx.Color\_Atom.F~(a,~b,~c)~\rightarrow~Color.Vertex.f~a~b~c
     UFOx.Color\_Atom.D\ (a,\ b,\ c) \rightarrow Color.Vertex.d\ a\ b\ c
     UFOx.Color\_Atom.Epsilon\ (i,\ j,\ k) \rightarrow Color.Vertex.epsilon\ [i;\ j;\ k]
     UFOx.Color\_Atom.EpsilonBar\ (i,\ j,\ k) \rightarrow Color.Vertex.epsilon\_bar\ [i;\ j;\ k]
     UFOx.Color\_Atom.T6 (a, i, j) \rightarrow Color.Vertex.t6 a i j
     UFOx.Color\_Atom.K6\ (i,\ j,\ k) \rightarrow Color.Vertex.k6\ i\ j\ k
     UFOx.Color\_Atom.K6Bar\ (i,\ j,\ k) \rightarrow Color.Vertex.k6bar\ i\ j\ k
let translate\_color\_term \ model \ p = function
   | \ | \ |, \ q \rightarrow
       Color. Vertex.scale q Color. Vertex.one
   | [atom], q \rightarrow
       Color.Vertex.scale\ q\ (translate\_color\_atom\ model\ p\ atom)
  \mid atoms, q \rightarrow
```

```
let atoms = List.map (translate\_color\_atom model p) atoms in
      Color. Vertex.scale q (Color. Vertex.multiply atoms)
{\tt let} \ translate\_color \ model \ p \ terms \ =
  match terms with
     [] \rightarrow invalid\_arg "translate_color:_empty"
     [term] \rightarrow translate\_color\_term\ model\ p\ term
   | terms \rightarrow
      Color. Vertex.sum (List.map (translate_color_term model p) terms)
let translate_coupling_1 model p lcc =
  let l = lcc. Vertex. lorentz in
  \mathsf{let}\ s\ =\ \mathit{Array}.to\_\mathit{list}\ (\mathit{spin\_multiplet}\ \mathit{model}\ \mathit{l})
  and fl = (SMap.find \ l \ model.lorentz).Lorentz.fermion\_lines
  and c = name (coupling\_of\_symbol model lcc.Vertex.coupling) in
  match lcc. Vertex.color with
  \mid UFOx.Color.Linear\ color \rightarrow
      let col = translate\_color model p color in
      (Array.to\_list\ p,\ Coupling.UFO\ (QC.unit,\ l,\ s,\ fl,\ col),\ c)
  \mid UFOx.Color.Ratios \_ as color \rightarrow
      invalid\_arq
         ("UFO.Model.translate_coupling: _invalid_color_structure" ^
             UFOx. Color.to_string color)
let translate\_coupling \ model \ p \ lcc =
   List.map\ (translate\_coupling\_1\ model\ p)\ lcc
let long\_flavors = ref false
module type Lookup =
  sig
     \mathsf{type}\,f \ = \ \mathsf{private}
       { flavors : flavor list;
          flavor\_of\_string : string \rightarrow flavor;
          flavor\_of\_symbol : string \rightarrow flavor;
          particle: flavor \rightarrow Particle.t;
          flavor\_symbol : flavor \rightarrow string;
          conjugate : flavor \rightarrow flavor }
     type flavor\_format =
         Long
          Decimal
         Hexadecimal
     val flavor_format : flavor_format ref
     val\ of\_model:\ t\ 	o\ f
  end
module \ Lookup : Lookup =
  struct
     type f =
        { flavors : flavor list;
          flavor\_of\_string : string \rightarrow flavor;
          flavor\_of\_symbol : string \rightarrow flavor;
          particle : flavor \rightarrow Particle.t;
          flavor\_symbol : flavor \rightarrow string;
          conjugate : flavor \rightarrow flavor 
     type flavor\_format =
         Long
          Decimal
        | Hexadecimal
     let flavor\_format = ref Hexadecimal
     let conjugate_of_particle_array particles =
        Array.init
```

```
(Array.length particles)
           (fun i \rightarrow
             let f' = Particle.conjugate particles.(i) in
             match \ \mathit{ThoArray}.match\_\mathit{all}\ \mathit{f'}\ \mathit{particles}\ \mathsf{with}
               [i'] \rightarrow i'
              [] \rightarrow
                 invalid\_arg ("no\sqcupcharge\sqcupconjugate:\sqcup" ^ f'.Particle.name)
                 invalid_arg ("multiple_charge_conjugates:_" ^ f'.Particle.name))
     let invert\_flavor\_array a =
        let table = SHash.create 37 in
        Array.iteri (fun i \ s \rightarrow SHash.add \ table \ s \ i) \ a;
         (fun name \rightarrow
          try
             SHash.find table name
           with
           Not\_found \rightarrow invalid\_arg ("not\_found: \_" ^ name))
     let digits \ base \ n =
        let rec digits' \ acc \ n =
          if n < 1 then
             acc
           else
             digits' (succ acc) (n / base) in
        if n < 0 then
           digits' \ 1 \ (-n)
        else if n = 0 then
           1
        else
           digits' \ 0 \ n
     let of\_model model =
        let particle\_array = Array.of\_list (values model.particles) in
        let conjugate_array = conjugate_of_particle_array particle_array
        and name\_array = Array.map (fun f \rightarrow f.Particle.name) particle\_array
        and symbol\_array = Array.of\_list (keys model.particles) in
        let flavor\_symbol f =
           begin match !flavor_format with
             Long \rightarrow symbol\_array.(f)
            Decimal \rightarrow
              let w = digits 10 (Array.length particle\_array - 1) in
               Printf.sprintf "%0*d" w f
           \mid Hexadecimal \rightarrow
              let w = digits \ 16 \ (Array.length \ particle\_array - 1) in
               Printf.sprintf "%0*X" w f
         \{ flavors = ThoList.range\ 0\ (Array.length\ particle\_array\ -\ 1);
           flavor\_of\_string = invert\_flavor\_array name\_array;
          flavor\_of\_symbol = invert\_flavor\_array \ symbol\_array;
           particle = Array.get particle_array;
           flavor\_symbol = flavor\_symbol;
           conjugate = Array.get conjugate_array }
   end
We appear to need to conjugate all flavors. Why???
let translate\_vertices model tables =
   let vn =
      List.fold\_left
        (fun acc \ v \rightarrow
           let p = Array.map \ tables.Lookup.flavor\_of\_symbol \ v.Vertex.particles
```

```
and lcc = v.Vertex.lcc in
         let p = Array.map \ conjugate \ p \ in (* FIXME: why? *)
         translate_coupling model p lcc @ acc)
         (values model.vertices) in
  ([], [], vn)
let propagator\_of\_lorentz = function
    Coupling.Scalar \rightarrow Coupling.Prop\_Scalar
    Coupling.Spinor \rightarrow Coupling.Prop\_Spinor
    Coupling.ConjSpinor \rightarrow Coupling.Prop\_ConjSpinor
     Coupling.Majorana \rightarrow Coupling.Prop\_Majorana
    Coupling.Maj\_Ghost \rightarrow invalid\_arg
      "UFO.Model.propagator\_of\_lorentz: \_SUSY\_ghosts\_do\_not\_propagate"
     Coupling.Vector \rightarrow Coupling.Prop\_Feynman
     Coupling.Massive\_Vector \rightarrow Coupling.Prop\_Unitarity
    Coupling.Tensor\_2 \rightarrow Coupling.Prop\_Tensor\_2
    Coupling.Vectorspinor \rightarrow invalid\_arg
      "UFO.Model.propagator_of_lorentz:_Vectorspinor"
    Coupling.Tensor_1 \rightarrow invalid\_arg
      "UFO.Model.propagator_of_lorentz:_Tensor_1"
  | Coupling.BRS \_ \rightarrow invalid\_arg |
      "UFO.Model.propagator_of_lorentz:_no_BRST"
let filter\_unphysical model =
  let physical\_particles =
     Particle.filter Particle.is_physical model.particles in
  let physical_particle_array =
     Array.of_list (values physical_particles) in
  let physical\_vertices =
     Vertex.filter
       (\neg < * > (Vertex.contains\ model.particles\ (\neg < * > Particle.is\_physical)))
       model.vertices in
  \{ model with \}
     particles = physical\_particles;
     particle\_array = physical\_particle\_array;
     vertices = physical\_vertices }
let \ whizard\_constants =
  SSet.of\_list
     ["ZERO"]
let filter_constants parameters =
  List.filter
    (\mathsf{fun}\ p\ \to
       \neg (SSet.mem (ThoString.uppercase p.Parameter.name) whizard\_constants))
     parameters
let \ add\_name \ set \ parameter =
  CSet.add parameter.Parameter.name set
let hardcoded\_parameters =
  CSet.of\_list
     ["cmath.pi"]
let missing_parameters input derived couplings =
  let input\_parameters =
     List.fold_left add_name hardcoded_parameters input in
  let all\_parameters =
     List.fold_left add_name input_parameters derived in
  let derived\_dependencies =
     dependencies
       (List.map)
          (\text{fun } p \rightarrow (p.Parameter.name, p.Parameter.value}))
          derived) in
  let coupling\_dependencies =
```

```
dependencies
                   (List.map)
                          (\text{fun } p \rightarrow (p.UFO\_Coupling.name, Expr p.UFO\_Coupling.value}))
                          (values couplings)) in
       let missing\_input =
              CMap.filter
                   (fun parameter\ derived\_parameters\ 
ightarrow
                        \neg (CSet.mem\ parameter\ all\_parameters))
                   derived\_dependencies
       and missing =
              CMap.filter
                   (fun parameter couplings \rightarrow
                        \neg (CSet.mem\ parameter\ all\_parameters))
                   coupling_dependencies in
        CMap.iter
             (fun parameter\ derived\_parameters\ 
ightarrow
                   Printf.eprintf
                        parameter (String.concat "; " (CSet.elements derived_parameters)))
             missing\_input;
        CMap.iter
             (fun parameter couplings \rightarrow
                   Printf.eprintf
                        "UFO \sqcup warning: \sqcup undefined \sqcup parameter \sqcup \%s \sqcup appears \sqcup in \sqcup couplings \sqcup \{\%s\}: \sqcup \setminus S \sqcup Appears \sqcup S \sqcup Appears \sqcup Ap
\verb| uuuuuuuuuuuuuwill_ube_uadded_uto_uthe_ulist_uof_uinput_uparameters! \\ \verb| n" |
                        parameter (String.concat "; " (CSet.elements couplings)))
             missing;
       keys\_caseless\ missing\_input\ @\ keys\_caseless\ missing
  let \ classify\_parameters \ model =
       let compare\_parameters p1 p2 =
             compare\ p1. Parameter. sequence\ p2. Parameter. sequence\ in
       let input, derived =
             List.fold\_left
                   (fun (input, derived) p \rightarrow
                       match p.Parameter.nature with
                             Parameter.Internal \rightarrow (input, p :: derived)
                          Parameter.External \rightarrow
                                begin match p.Parameter.ptype with
                                    Parameter.Real \rightarrow ()
                                    Parameter.Complex \rightarrow
                                         Printf.eprintf
                                              "UFO_{\sqcup}warning:_{\sqcup}invalid_{\sqcup}complex_{\sqcup}declaration_{\sqcup}of_{\sqcup}input_{\sqcup}\setminus
uuuuuuuuuuuuuparameteru'%s'uignored!\n"
                                             p.Parameter.name
                                end;
                                (p :: input, derived))
                   ([], []) (filter_constants (values model.parameters)) in
       let additional = missing_parameters input derived model.couplings in
       (List.sort compare_parameters input @ List.map Parameter.missing additional,
          List.sort compare_parameters derived)
  let translate\_name \ map \ name =
        try SMap.find\ name\ map\ with\ Not\_found\ 	o\ name
  let translate\_input map p =
        (translate_name map p.Parameter.name, value_to_float p.Parameter.value)
  let alpha_s_half\ e\ =
        UFOx.Expr.substitute "aS" (UFOx.Expr.half "aS") e
  let \ alpha\_s\_half\_etc \ map \ e =
        UFOx.Expr.rename (map_to_alist map) (alpha_s_half e)
```

```
let translate\_derived map p =
  let make\_atom s = s in
  let c = make\_atom (translate\_name map p.Parameter.name)
  and v =
     value_to_coupling (alpha_s_half_etc map) make_atom p.Parameter.value in
  match p.Parameter.ptype with
    Parameter.Real \rightarrow (Coupling.Real \ c, \ v)
   | Parameter.Complex \rightarrow (Coupling.Complex c, v) |
let translate\_coupling\_constant map c =
  let <math>make\_atom \ s = s in
  (Coupling.Complex\ c.UFO\_Coupling.name,
    Coupling. Quot
      (value_to_coupling
          (alpha_s_half_etc map) make_atom
          (Expr\ c.UFO\_Coupling.value),
        Coupling.I)
module Lowercase\_Parameters =
  struct
     \mathsf{type}\ \mathit{elt}\ =\ \mathit{string}
     type base = string
     let compare\_elt = compare
     let \ compare\_base = compare
     let pi = ThoString.lowercase
module Lowercase_Bundle = Bundle.Make (Lowercase_Parameters)
let coupling\_names model =
  SMap.fold
     (fun \ \_c \ acc \ \rightarrow \ c.UFO\_Coupling.name :: acc)
     model.couplings []
let parameter_names model =
  SMap.fold
     (fun \ \_ c \ acc \ \rightarrow \ c.Parameter.name :: acc)
     model.parameters []
let \ ambiguous\_parameters \ model =
  \mathsf{let} \ \mathit{all\_names} \ = \\
     List.rev_append (coupling_names model) (parameter_names model) in
  let lc\_bundle = Lowercase\_Bundle.of\_list all\_names in
  let lc\_set =
     List.fold\_left
        (fun\ acc\ s\ 	o\ SSet.add\ s\ acc)
        SSet.empty (Lowercase_Bundle.base lc\_bundle)
  and ambiguities =
     List.filter
       (fun (\_, names) \rightarrow List.length names > 1)
       (Lowercase\_Bundle.fibers\ lc\_bundle) in
  (lc\_set, ambiguities)
{\tt let} \ \textit{disambiguate1} \ \textit{lc\_set} \ \textit{name} \ = \\
  \mathsf{let} \ \mathsf{rec} \ \mathit{disambiguate1'} \ i \ = \\
     let name' = Printf.sprintf "%s_%d" name i in
     let lc\_name' = ThoString.lowercase name' in
     if SSet.mem\ lc\_name'\ lc\_set then
        disambiguate1' (succ i)
       (SSet.add lc_name' lc_set, name') in
  disambiguate1' 1
let disambiguate\ lc\_set\ names\ =
  let _, replacements =
```

```
List.fold\_left
            (fun (lc\_set', acc) name \rightarrow
              let lc\_set'', name' = disambiguate1 lc\_set' name in
              (lc_set", SMap.add name name' acc))
            (lc\_set, SMap.empty) names in
       replacements
    let \ omegalib\_names =
       ["u"; "ubar"; "v"; "vbar"; "eps"]
    let translate\_parameters model =
      let lc\_set, ambiguities = ambiguous\_parameters model in
      let replacements =
         disambiguate lc_set (ThoList.flatmap snd ambiguities) in
       SMap.iter
         (Printf.eprintf
             "warning:_case_sensitive_parameter_names:_renaming_'%s'_->_'%s'\n")
         replacements;
      let replacements =
         List.fold\_left
            (fun acc\ name\ \rightarrow\ SMap.add\ name\ ("UFO_"\ ^name)\ acc)
            replacements omegalib_names in
      let input\_parameters, derived\_parameters = classify\_parameters model
      and couplings = values \ model.couplings in
       \{ Coupling.input =
            List.map (translate_input replacements) input_parameters;
         Coupling.derived =
            List.map (translate_derived replacements) derived_parameters @
              List.map\ (translate\_coupling\_constant\ replacements)\ couplings;
         Coupling.derived\_arrays = [] 
UFO requires us to look up the mass parameter to distinguish between massless and massive vectors.
   TODO: this is a candidate for another lookup table.
    let lorentz\_of\_particle p =
       match UFOx.Lorentz.omega p.Particle.spin with
         Coupling.Vector \rightarrow
          begin match ThoString.uppercase p.Particle.mass with
            "ZERO" \rightarrow Coupling. Vector
            \_ \rightarrow Coupling.Massive\_Vector
          end
       | s \rightarrow s
    type state =
       \{ directory : string; \}
         model: t
    let initialized = ref None
    let is\_initialized\_from dir =
      match !initialized with
         None \rightarrow \mathsf{false}
        Some \ state \rightarrow dir = state.directory
    let dump\_raw = ref false
    let init dir =
      let model = filter\_unphysical\ (parse\_directory\ dir) in
      if !dump\_raw then
         dump model:
      let \ tables = Lookup.of\_model \ model \ in
      let vertices () = translate_vertices model tables in
      let particle f = tables.Lookup.particle f in
      let \ lorentz \ f \ = \ lorentz \ \_of \ \_particle \ (particle \ f) \ in
      let propagator f =
         let p = particle f in
```

```
match p.Particle.propagator with
       None \rightarrow propagator\_of\_lorentz (lorentz\_of\_particle p)
       Some \ s \rightarrow Coupling.Prop\_UFO \ s in
  let gauge\_symbol () = "?GAUGE?" in
  \mathsf{let}\ constant\_symbol\ s\ =\ s\ \mathsf{in}
  let parameters = translate_parameters model in
   M.setup
      Color: (fun \ f \ 
ightarrow \ UFOx. Color. omega \ (particle \ f). Particle. color)
      \tilde{n}c: (\mathsf{fun}\ () \ \to \ model.nc)
      \tilde{p}dg: (\mathsf{fun}\ f \ 	o \ (particle\ f).Particle.pdg\_code)
      \tilde{lorentz}
      propagator
      Width: (fun \ f \ 	o \ Coupling.Constant)
       (goldstone: (fun f \rightarrow None))
      \~conjugate: tables. Lookup. conjugate
      \tilde{f}ermion: (fun \ f \rightarrow fermion\_of\_lorentz \ (lorentz \ f))
      \tilde{vertices}
      \tilde{flavors}: [("All_{\sqcup}Flavors", tables.Lookup.flavors)]
      parameters : (fun () \rightarrow parameters)
      \tilde{f}lavor\_of\_string: tables.Lookup.flavor\_of\_string
       flavor\_to\_string: (fun \ f \ 
ightarrow \ (particle \ f).Particle.name)
       flavor\_to\_TeX : (fun f \rightarrow (particle f).Particle.texname)
       flavor\_symbol: tables.Lookup.flavor\_symbol
       gauge\_symbol
       mass\_symbol: (fun f \rightarrow (particle f).Particle.mass)
      \widetilde{\ \ } width\_symbol: (fun \ f \ 	o \ (particle \ f).Particle.width)
      \tilde{constant\_symbol};
   initialized := Some \{ directory = dir; model = model \}
let ufo_directory = ref Config.default_UFO_dir
lot load () =
  if is\_initialized\_from\ !ufo\_directory then
     ()
  else
      init !ufo_directory
let include\_all\_fusions = ref false
```

In case of Majorana spinors, also generate all combinations of charge conjugated fermion lines. The naming convention is to append $_cnm$ if the γ -matrices of the fermion line $n \to m$ has been charge conjugated (this could become impractical for too many fermions at a vertex, but shouldn't matter in real life).

Here we alway generate all charge conjugations, because we treat all fermions as Majorana fermion, if there is at least one Majorana fermion in the model!

```
let is\_majorana = function
     Coupling.Majorana \mid Coupling.Vectorspinor \mid Coupling.Maj\_Ghost \rightarrow true
     _{-} \rightarrow false
let name\_spins\_structure\ spins\ l\ =
   (l.Lorentz.name, spins, l.Lorentz.structure)
let fusions\_of\_model ? only model =
  let include\_fusion =
     match !include_all_fusions, only with
       true. _
       false, None \rightarrow (fun \ name \rightarrow true)
       false, Some\ names \rightarrow (fun\ name \rightarrow SSet.mem\ name\ names)
  SMap.fold
     (fun name\ l\ acc\ 	o
        if include\_fusion\ name\ then
          List.fold\_left
             (fun acc p \rightarrow
               let l' = Lorentz.permute p l in
```

```
match l'.Lorentz.spins with
                Lorentz.Unused \rightarrow acc
                Lorentz.Unique\ spins\ 	o
                 if Array.exists is_majorana spins then
                    List.map
                      (name\_spins\_structure\ spins)
                      (Lorentz.required\_charge\_conjugates\ l')
                 else
                    name\_spins\_structure\ spins\ l'\ ::\ acc
              | Lorentz.Ambiguous → failwith "fusions: Lorentz.Ambiguous")
            [] (Permutation.Default.cyclic l.Lorentz.n) @ acc
       else
         acc)
    model.lorentz []
let fusions ?only() =
  match !initialized with
    None \rightarrow []
  | Some \{ model = model \} \rightarrow fusions\_of\_model ?only model |
let propagators_of_model ?only model =
  let include\_propagator =
    match \ !include\_all\_fusions, \ only \ with
      true.
      false, None \rightarrow (fun \ name \rightarrow true)
      \mathsf{false}, \ Some \ names \ \rightarrow \ (\mathsf{fun} \ name \ \rightarrow \ SSet.mem \ name \ names)
  SMap.fold
     (fun name \ p \ acc \rightarrow
       if include\_propagator name then
         (name, p) :: acc
       else
         acc)
    model.propagators []
let propagators ?only () =
  match !initialized with
    None \rightarrow []
    Some \{ model = model \} \rightarrow propagators\_of\_model ? only model 
\mathsf{let}\ include\_hadrons\ =\ ref\ \mathsf{true}
let ufo_majorana_warnings =
  "*_____*"
     "*_CAVEAT:_____*"
     "*_____*";
     "*_{\sqcup\sqcup\sqcup} These_{\sqcup} amplitudes_{\sqcup} have_{\sqcup} been_{\sqcup} computed_{\sqcup} for_{\sqcup} a_{\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup} *";
     "*_{\sqcup \sqcup \sqcup} UFO_{\sqcup} model_{\sqcup} containing_{\sqcup} Majorana_{\sqcup} fermions._{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} *";
     "*_UU_This_version_of_O'Mega_contains_some_known_UU_U*";
     "*_UULbugs_for_this_case.ULIt_was_released_early_at_*";
     "*___the_request_of_the_Linear_Collider_community._*";
     "*.....*";
     "*___These_amplitudes_MUST_NOT_be_used_for_____*";
     "*_{\sqcup\sqcup\sqcup}publications_{\sqcup}without_{\sqcup}prior_{\sqcup}consulation_{\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup}*";
     "*uuuwithutheuWHIZARDuauthorsu!!!uuuuuuuuuuuu*";
     "*<sub>-----</sub>";
let \ caveats \ () =
  if !use_majorana_spinors then
     ufo\_majorana\_warnings
  else
```

```
module Whizard: sig val write: unit \rightarrow unit end =
  struct
    let write\_header dir =
       Printf.printf \ \verb"#_UWHIZARD_UModel_Ufile_Uderived_Ufrom_UFO_Udirectory\\ \verb"n"};
       Printf.printf "#\sqcup \sqcup \sqcup'%s'\n\n" dir;
       List.iter (fun s \rightarrow Printf.printf "#\square%s\n" s) (M.caveats ());
       Printf.printf "model_\"%s\"\n\n" (Filename.basename\ dir)
    let write\_input\_parameters parameters =
       let open Parameter in
       Printf.printf "#\BoxIndependent\Box(input)\BoxParameters\n";
       List.iter
         (fun p \rightarrow
            Printf.printf
              "parameter_%s_=_%s"
              p.name (value_to_numeric p.value);
           begin match p.lhablock, p.lhacode with
             None, None \rightarrow ()
            \mid Some \ name, \ Some \ (index :: indices) \rightarrow
               Printf.printf "_\slha_entry\\%s\\%d" name index;
               | Some name, None \rightarrow
               Printf.eprintf
                 "UFO: _parameter_%: _slhablock_%s_without_slhacode\n"
                 p.name name
            | Some name, Some [] \rightarrow
               Printf.eprintf
                 "UFO:\_parameter\_\%s:\_slhablock\_\%s\_with\_empty\_slhacode\n"
                 p.name name
            | None, Some \_ \rightarrow
               Printf.eprintf
                 "UF0:\_parameter\_\%s:\_slhacode\_without\_slhablock\n"
            end;
            Printf.printf "\n")
         parameters;
       Printf.printf "\n"
    let write_derived_parameters parameters =
       let open Parameter in
       Printf.printf "#□Dependentu(derived)uParameters\n";
       List.iter
         (fun p \rightarrow
            Printf.printf
              "derived<sub>\\\</sub>"s<sub>\\\</sub>=<sub>\\\</sub>"s\\n"
              p.name (value_to_expr alpha_s_half p.value))
         parameters
    let write\_particles particles =
       let open Particle in
       Printf.printf "#⊔Particles\n";
       Printf.printf "#UNB: hypercharge assignments appear to be unreliable ";
       Printf.printf "#_\uu\uu\utherefore\uwe\ucan't\uinfer\uthe\uisospin\n";
       Printf.printf "#□NB: □parton-, □gauge-□&□handedness□are□unavailable\n";
       List.iter
         (fun p \rightarrow
           if \neg p.is\_anti then begin
                Printf.printf
                   "particle_\"%s\"_\%d_\###_parton?_gauge?_left?\n"
                  p.name \ p.pdg\_code;
                Printf.printf
```

```
"_uspin_%s_charge_%s_color_%s_###_isospin?\n"
                                             (UFOx.Lorentz.rep_to_string_whizard p.spin)
                                             (charge_to_string p.charge)
                                             (UFOx.Color.rep\_to\_string\_whizard\ p.color);
                                      Printf.printf "\sqcup \sqcup name \sqcup \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ p.name;
                                      if p.antiname \neq p.name then
                                             Printf.printf "\sqcup \sqcupanti\sqcup \backslash "%s\backslash"\backslashn" p.antiname;
                                      Printf.printf "\sqcup \sqcup tex_name \sqcup \"%s\" n" p.texname;
                                     if p.antiname \neq p.name then
                                             Printf.printf "\sqcup \sqcup \max \sqcup \slass \sqcup \sl
                             end)
               (values particles);
       Printf.printf "\n"
let write\_hadrons() =
        Printf.printf "#_Hadrons_(protons_and_beam_remnants)\n";
        Printf.printf "#, NB:, these, are, NOT, part, of, the, UFO, model\n";
       Printf.printf "#_\uu_\ubut_added_\ufor_\WHIZARD's_\uconvenience!\n";
       Printf.printf "particle_PROTON_2212\n";
        Printf.printf "⊔⊔spin⊔1/2⊔⊔charge⊔1\n";
       Printf.printf "\sqcup \sqcup name \sqcup p \sqcup \backslash "p+ \backslash " \backslash n";
        Printf.printf "_{\sqcup\sqcup}anti_{\sqcup}pbar_{\sqcup}\"p-\"\";
       Printf.printf "particle_HADRON_REMNANT_90\n";
        Printf.printf "\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersame\undersa
        Printf.printf "_{\sqcup\sqcup} tex_name_{\sqcup} \' had_r \' n";
       Printf.printf "particle_HADRON_REMNANT_SINGLET_91\n";
       Printf.printf "_uname_hr1\n";
        Printf.printf "_{\sqcup\sqcup} tex_name_{\sqcup} \' had_r^{(1)} \' n";
        Printf.printf "particle_HADRON_REMNANT_TRIPLET_92\n";
        Printf.printf "⊔⊔color⊔3\n";
        Printf.printf "⊔⊔name⊔hr3\n";
        Printf.printf "⊔⊔anti⊔hr3bar\n";
       Printf.printf "_{\sqcup\sqcup} tex_anti_{\sqcup} ''had_r^{(\\Delta)}''n";
       Printf.printf "particle_HADRON_REMNANT_OCTET_93\n";
       Printf.printf "⊔⊔color⊔8\n";
       Printf.printf "\u\u\name\u\n";
        Printf.printf "\n"
let \ vertex\_to\_string \ model \ v =
       String.concat
               "<sub>''</sub>"
               (List.map
                          (fun s \rightarrow
                                  "\"" \hat{SMap.find\ s\ model.particles}.Particle.name \hat{T}"\"")
                          (Array.to\_list\ v.Vertex.particles))
let write_vertices3 model vertices =
       Printf.printf "#uVerticesu(foruphasespaceugenerationuonly)\n";
       \textit{Printf.printf} \ \texttt{"#$\_$NB:$\_particles$\_should$\_be$\_sorted$\_increasing$\_in$\_mass.$\\ \texttt{""};}
        Printf.printf "#____This_is_NOT_implemented_yet!\n";
       List.iter
               (fun v \rightarrow
                     if Array.length \ v.Vertex.particles = 3 then
                              Printf.printf "vertex_\%s\n" (vertex_to_string\ model\ v))
               (values vertices);
        Printf.printf "\n"
let write_vertices_higher model vertices =
        Printf.printf
                "\#_{\sqcup} Higher_{\sqcup} 0 rder_{\sqcup} Vertices_{\sqcup} (ignored_{\sqcup} by_{\sqcup} phasespace_{\sqcup} generation) \\ \backslash n";
```

```
List.iter
              (fun v \rightarrow
                 if Array.length \ v.Vertex.particles \neq 3 then
                   Printf.printf "#\uvertex\und{\subset}\n\" (vertex\_to\_string \ model \ v))
              (values vertices);
            Printf.printf "\n"
         let write_vertices model vertices =
            write_vertices3 model vertices;
            write\_vertices\_higher\ model\ vertices
         let write() =
            match !initialized with
              None \rightarrow failwith "UFO.Whizard.write: UFO umodel unot uinitialized"
              Some { directory = dir; model = model \} \rightarrow
               let input_parameters, derived_parameters =
                  classify_parameters model in
               write_header dir;
               write_input_parameters input_parameters;
                write_derived_parameters derived_parameters;
               write_particles model.particles;
               if !include\_hadrons then
                  write\_hadrons ();
                write_vertices model model.vertices;
                exit 0
       end
    let options =
       Options.create
         [ ("UFO_dir", Arg.String (fun name \rightarrow ufo\_directory := name),
             "UFO_{\sqcup}model_{\sqcup}directory_{\sqcup}(default:_{\sqcup}" ^{^{\circ}}!ufo\_directory ^{^{\circ}}")");
            ("Majorana", Arg.Set use_majorana_spinors,
             "use_\Majorana\spinors\((must\(\)come\(\)_before\(\)(exec!)");
            ("divide_propagators_by_i", Arg.Set divide_propagators_by_i,
             "divide⊔propagators⊔by⊔I⊔(pre⊔2013⊔FeynRules⊔convention)");
            ("verbatim_Hg", Arg.Set verbatim_higgs_glue,
             "don'tucorrectutheucoloruflowsuforueffectiveuHiggsuGluonucouplings");
            ("write_WHIZARD", Arg. Unit Whizard.write,
             "write_the_WHIZARD_model_file_(required_once_per_model)");
            ("long_flavors",
             Arg.Unit (fun () \rightarrow Lookup.flavor\_format := Lookup.Long),
             "write_use_the_UFO_flavor_names_instead_of_integers");
            ("dump", Arg.Set dump_raw,
             "dump_{\sqcup}UFO_{\sqcup}model_{\sqcup}for_{\sqcup}debugging_{\sqcup}the_{\sqcup}parser_{\sqcup}(must_{\sqcup}come_{\sqcup}-before_{-\sqcup}exec!)");
            ("all_fusions", Arg.Set include_all_fusions,
             "include_all_fusions_in_the_fortran_module");
            ("no_hadrons", Arg. Clear include_hadrons,
             "don't add any particle not in the UFO file");
            ("add_hadrons", Arg.Set include_hadrons,
             "add_protons_and_beam_remants_for_WHIZARD");
            ("exec", Arg.Unit\ load,
             "load_the_UFO_model_files_(required__before_using_particles_names)");
            ("help", Arg.Unit (fun () \rightarrow prerr\_endline "..."),
             "print_{\sqcup}information_{\sqcup}on_{\sqcup}the_{\sqcup}model")]
  end
module type Fortran\_Target =
    val fuse :
       Algebra.QC.t \rightarrow string \rightarrow
       Coupling.lorentzn \rightarrow Coupling.fermion\_lines \rightarrow
       string \rightarrow string \ list \rightarrow string \ list \rightarrow Coupling.fusen \rightarrow unit
```

```
{\tt val}\ lorentz\_module\ :
       ?only: SSet.t \rightarrow ?name:string \rightarrow
       ?fortran\_module:string \rightarrow ?parameter\_module:string \rightarrow
        Format\_Fortran.formatter \rightarrow unit \rightarrow unit
  end
module Targets =
  struct
     module Fortran : Fortran\_Target =
       struct
          open Format_Fortran
          \mathsf{let}\ \mathit{fuse}\ =\ \mathit{UFO\_targets}.\mathit{Fortran.fuse}
          let lorentz_functions ff fusions () =
             List.iter
                (fun (name, s, l) \rightarrow
                  UFO\_targets.Fortran.lorentz\ ff\ name\ s\ l)
               fusions
          let propagator_functions ff parameter_module propagators () =
             List.iter
                (fun (name, p) \rightarrow
                  UFO\_targets.Fortran.propagator
                    ff name
                    parameter\_module\ p.Propagator.variables
                    p.Propagator.spins
                    p.Propagator.numerator\ p.Propagator.denominator)
               propagators
          let lorentz_module
                  ?only ?(name = "omega_amplitude_ufo")
                  ?(fortran_module ="omega95")
                  ?(parameter_module = "parameter_module") ff () =
             let printf fmt = fprintf ff fmt
             and nl = pp\_newline ff in
             printf "module_\%s" name; nl ();
             printf "\sqcup \sqcup use \sqcup kinds"; nl();
             printf "_uuse_\%s" fortran_module; nl ();
             printf ~" \verb| llimplicit_lnone"; ~nl~();
             printf "\sqcup \sqcup private"; nl();
             let fusions = Model.fusions ?only ()
             and propagators = Model.propagators () in
             List.iter
                (fun (name, \_, \_) \rightarrow printf "_{\sqcup \sqcup}public_{\sqcup} : _{\sqcup}%s" name; nl ())
                fusions;
             List.iter
                (\mathsf{fun}\ (name,\ \_)\ \to\ printf\ \verb"$\sqsubseteq$ \verb"public$$\sqsubseteq$::$\sqsubseteq$ \verb"pr$\_$U\_$$s"}\ name;\ nl\ ())
                propagators;
             UFO\_targets.Fortran.eps4\_g4\_g44\_decl\ ff\ ();
             UFO\_targets.Fortran.eps4\_g4\_g44\_init\ ff\ ();
             printf "contains"; nl ();
             UFO_targets.Fortran.inner_product_functions ff ();
             lorentz_functions ff fusions ();
             propagator_functions ff parameter_module propagators ();
             printf "end_module_%s" name; nl ();
             pp_{-}flush ff ()
       end
  end
module type Test =
  sig
```

```
val suite : OUnit.test
module \ Test : Test =
  struct
    open OUnit
    let \ lexer \ s =
       UFO_lexer.token (UFO_lexer.init_position "" (Lexing.from_string s))
    let suite\_lexer\_escapes =
       "escapes" >:::
         [ "single-quote" >::
              (fun () \rightarrow
                 assert_equal (UFO_parser.STRING "a'b'c") (lexer "'a\\'c'"));
            "unterminated" >::
              (fun () \rightarrow
                 assert\_raises\ End\_of\_file\ (fun\ ()\ \rightarrow\ lexer\ "'a\\'c"))\ ]
    let suite\_lexer =
       "lexer" >:::
         [suite\_lexer\_escapes]
    let suite =
       "UFO" >:::
         [suite\_lexer]
  end
```

14.15 Targets

14.16 Interface of UFO_targets

14.16.1 Generating Code for UFO Lorentz Structures

```
\begin{array}{ccc} \mathsf{module} \ \mathsf{type} \ T \ = \\ \mathsf{sig} \end{array}
```

 $lorentz\ ff\ name\ spins\ lorentz\ writes$ the Fortran code implementing the fusion corresponding to the Lorentz structure $lorentz\ to\ ff$. NB: The $spins\ :\ int\ list\ element\ of\ UFO.Lorentz.t$ from the UFO file is $not\ sufficient\ to\ determine$ the domain and codomain of the function. We had to inspect the flavors, where the Lorentz structure is referenced to heuristically compute the $spins\ as\ a\ Coupling.lorentz\ array\ .$

```
 \begin{array}{l} \textbf{val} \ lorentz : \\ Format\_Fortran.formatter \rightarrow string \rightarrow \\ Coupling.lorentz \ array \rightarrow \ UFO\_Lorentz.t \rightarrow unit \\ \\ \textbf{val} \ propagator : \\ Format\_Fortran.formatter \rightarrow string \rightarrow string \rightarrow string \ list \rightarrow \\ Coupling.lorentz \times \ Coupling.lorentz \rightarrow \\ UFO\_Lorentz.t \rightarrow \ UFO\_Lorentz.t \rightarrow unit \\ \end{array}
```

 $fusion_name\ name\ perm\ cc_list$ forms a name for the fusion $name\ with\ the\ permutations\ perm\ and\ charge\ conjugations\ applied\ to\ the\ fermion\ lines\ cc_list.$

```
val fusion\_name: string \rightarrow Permutation.Default.t \rightarrow Coupling.fermion\_lines \rightarrow string
```

fuse c v s fl g wfs ps fusion fuses the wavefunctions named wfs with momenta named ps using the vertex named v with legs reordered according to fusion. The overall coupling constant named g is multiplied by the rational coefficient c. The list of spins s and the fermion lines fl are used for selecting the appropriately transformed version of the vertex v.

```
\begin{array}{c} \text{val } \textit{fuse} \ : \\ \textit{Algebra}.\textit{QC.t} \ \rightarrow \ \textit{string} \ \rightarrow \end{array}
```

```
Coupling.lorentzn \rightarrow Coupling.fermion\_lines \rightarrow string \rightarrow string \ list \rightarrow string \ list \rightarrow Coupling.fusen \rightarrow unit val\ eps4\_g4\_g44\_decl\ : \ Format\_Fortran.formatter \rightarrow unit \rightarrow unit val\ eps4\_g4\_g44\_init\ : \ Format\_Fortran.formatter \rightarrow unit \rightarrow unit val\ inner\_product\_functions\ : \ Format\_Fortran.formatter \rightarrow unit \rightarrow unit module\ type\ Test\ = sig \qquad val\ suite\ : \ OUnit.test end module\ Test\ : \ Test end module\ Fortran\ : \ T
```

14.17 Implementation of UFO_targets

14.17.1 Generating Code for UFO Lorentz Structures



O'Caml before 4.02 had a module typing bug that forced us to put these definitions outside of *Lorentz_Fusion*. Since then, they might have appeared in more places. Investigate, if it is worthwhile to encapsulate them again.

```
\begin{array}{lll} \operatorname{module} \ Q &= \ Algebra. Q \\ \operatorname{module} \ QC &= \ Algebra. QC \\ \operatorname{module} \ \operatorname{type} \ T &= \\ \operatorname{sig} \end{array}
```

lorentz formatter name spins v writes a representation of the Lorentz structure v of particles with the Lorentz representations spins as a (Fortran) function name to formatter.

```
val\ lorentz:
        Format\_Fortran.formatter \rightarrow string \rightarrow
        Coupling.lorentz \ array \rightarrow \ UFO\_Lorentz.t \rightarrow \ unit
     val propagator :
        Format\_Fortran.formatter \rightarrow string \rightarrow string \rightarrow string \ list \rightarrow
        Coupling.lorentz \times Coupling.lorentz \rightarrow
        UFO\_Lorentz.t \rightarrow UFO\_Lorentz.t \rightarrow unit
     val\ fusion\_name:
        string \rightarrow Permutation.Default.t \rightarrow Coupling.fermion\_lines \rightarrow string
     val\ fuse:
        Algebra.QC.t \rightarrow string \rightarrow
        Coupling.lorentzn \rightarrow Coupling.fermion\_lines \rightarrow
        string \rightarrow string \ list \rightarrow string \ list \rightarrow Coupling.fusen \rightarrow unit
     val\ eps4\_g4\_g44\_decl\ :\ Format\_Fortran.formatter\ 	o\ unit\ 	o\ unit
     val\ eps4\_g4\_g44\_init\ :\ Format\_Fortran.formatter\ 	o\ unit\ 	o\ unit
     \verb|val|| inner\_product\_functions|: Format\_Fortran.formatter| \rightarrow unit \rightarrow unit|
     module type Test =
        sig
           val\ suite\ :\ OUnit.test
        end
     module Test: Test
module Fortran : T =
  struct
     open Format_Fortran
```

```
let pp\_divide\ ?(indent = 0)\ ff\ () =
       fprintf ff "%*s!_1%s" indent "" (String.make (70 - indent) '-');
       pp\_newline ff()
    let conjugate = function
         Coupling.Spinor \rightarrow Coupling.ConjSpinor
          Coupling.ConjSpinor \rightarrow Coupling.Spinor
         r \rightarrow r
    let spin\_mnemonic = function
          Coupling.Scalar → "phi"
          Coupling.Spinor 
ightarrow "psi"
          Coupling.ConjSpinor \rightarrow "psibar"
          Coupling.Majorana → "chi"
         Coupling.Maj\_Ghost \rightarrow
           invalid\_arg "UFO_targets:\BoxMaj_Ghost"
          Coupling.Vector \rightarrow "a"
          Coupling.Massive\_Vector \rightarrow "v"
          Coupling.Vectorspinor \rightarrow "grav" (* itino *)
          Coupling.Tensor\_1 \rightarrow
           invalid\_arg "UFO_targets:_\Tensor_1"
          Coupling.Tensor_2 \rightarrow \text{"h"}
         Coupling.BRS\ l\ 
ightarrow
           invalid\_arg "UFO_targets:_{\square}BRS"
    let fortran_type = function
          Coupling.Scalar \rightarrow "complex(kind=default)"
          Coupling.Spinor \rightarrow "type(spinor)"
          Coupling.ConjSpinor \rightarrow "type(conjspinor)"
          Coupling.Majorana \rightarrow "type(bispinor)"
          Coupling.Maj\_Ghost \rightarrow
           invalid\_arg "UFO_targets:\BoxMaj_Ghost"
          Coupling.Vector \rightarrow "type(vector)"
          Coupling.Massive\_Vector \rightarrow "type(vector)"
          Coupling.Vectorspinor \rightarrow "type(vectorspinor)"
         Coupling.Tensor\_1 \rightarrow
           invalid_arg "UFO_targets: _Tensor_1"
          Coupling. Tensor_2 \rightarrow "type(tensor)"
        \mid Coupling.BRS \mid l \rightarrow
           invalid\_arg "UFO_targets:\_BRS"
The omegalib separates time from space. Maybe not a good idea after all. Mend it locally ...
     type wf =
       \{ pos : int; 
          spin : Coupling.lorentz;
          name : string;
          local\_array : string\ option;
          momentum : string;
          momentum\_array : string;
          fortran_type : string }
    let wf\_table\ spins\ =
        Array.mapi
          (fun i \ s \rightarrow
            let spin =
               \quad \text{if } i \ = \ 0 \ \text{then} \\
                  conjugate s
               else
                 s in
            \mathsf{let}\ pos\ =\ succ\ i\ \mathsf{in}
            let i = string\_of\_int pos in
            let name = spin\_mnemonic s \hat{\ } i in
            let local\_array =
```

```
begin match spin with
                Coupling. Vector | Coupling. Massive _ Vector → Some (name ^ "a")
                \_ \rightarrow None
              end in
            \{ pos;
              spin;
              name;
              local\_array;
              momentum = "k" \hat{i};
              momentum\_array = "p" \hat{i};
              fortran_type = fortran_type spin } )
         spins
    module L = UFO\_Lorentz
Format rational (Q.t) and complex rational (QC.t) numbers as fortran values.
    let format_rational q =
       if Q.is\_integer \ q then
         string\_of\_int (Q.to\_integer q)
       else
         let n, d = Q.to\_ratio q in
         Printf.sprintf "%d.0_default/%d" n \ d
    let format\_complex\_rational \ cq =
       let real = QC.real cq
       and imag = QC.imag cq in
       if Q.is\_null\ imag then
         begin
            if Q.is\_negative\ real then
              "(" ^ format_rational real ^ ")"
              format\_rational\ real
         end
       else if Q.is\_integer\ real\ \land\ Q.is\_integer\ imag then
         Printf.sprintf "(%d,%d)" (Q.to\_integer\ real) (Q.to\_integer\ imag)
       else
         Printf.sprintf
            "cmplx(%s,%s,kind=default)"
            (format_rational real) (format_rational imag)
Optimize the representation if used as a prefactor of a summand in a sum.
    let format\_rational\_factor q =
       if Q.is\_unit\ q then
         "+_"
       else if Q.is\_unit\ (Q.neg\ q) then
       else if Q.is\_negative \ q then
         "-_{\sqcup}" \hat{format\_rational} (Q.neg \ q) \hat{"}*"
         "+_{\Box}" ^ format_rational q ^ "*"
    let format_complex_rational_factor cq =
       let real = QC.real cq
       and imag = QC.imag cq in
       if Q.is\_null\ imag then
         begin
            if Q.is\_unit\ real then
            else if Q.is\_unit (Q.neg\ real) then
            else if Q.is\_negative\ real then
              "-_{\sqcup}" ^ format\_rational (Q.neg\ real) ^ "*"
            else
```

```
"+_{\sqcup}" \hat{\ } format\_rational \ real \hat{\ } "*"
          end
       else if Q.is\_integer\ real\ \land\ Q.is\_integer\ imag then
          Printf.sprintf "+\sqcup(%d,%d)*" (Q.to\_integer\ real) (Q.to\_integer\ imag)
       else
          Printf.sprintf
             "+\cmplx(%s,%s,kind=default)*"
             (format_rational real) (format_rational imag)
Append a formatted list of indices to name.
     let append\_indices name = function
       | [] \rightarrow name
        \mid indices \rightarrow
           name \ ^ "(" \ ^ String.concat "," (List.map string\_of\_int indices) \ ^ ")"
Dirac string variables and their names.
     type dsv =
          Ket of int
          Bra of int
         Braket of int
     let dsv\_name = function
          Ket \ n \rightarrow Printf.sprintf "ket%02d" n
          Bra\ n\ 	o\ Printf.sprintf "bra%02d" n
        \mid Braket \ n \rightarrow Printf.sprintf "bkt%02d" n
     let \ dirac\_dimension \ dsv \ indices =
       let tail ilist =
          String.concat "," (List.map (fun \rightarrow "0:3") ilist) \hat{} ")" in
        match dsv, indices with
          Braket \_, [] \rightarrow ""
         (Ket \_ \mid Bra \_), [] \rightarrow ", \_dimension(1:4)"
          Braket _, indices 
ightarrow ", _dimension(" \hat{} tail indices
        |(Ket \_|Bra \_), indices \rightarrow ", \_dimension(1:4, " ^ tail indices)|
```

Write Fortran code to decl and eval: apply the Dirac matrix gamma with complex rational entries to the spinor ket from the left. ket must be the name of a scalar variable and cannot be an array element. The result is stored in dsv_name (Ket n) which can have additional indices. Return Ket n for further processing.

```
let dirac_ket_to_fortran_decl ff n indices =
  let printf fmt = fprintf ff fmt
  and nl = pp\_newline ff in
  \mathsf{let}\ dsv\ =\ Ket\ n\ \mathsf{in}
  printf
     "_{\cup\cup\cup\cup\cup}0[<2>complex(kind=default)%s_{\cup}::0_{\cup}%s0]"
     (dirac_dimension dsv indices) (dsv_name dsv);
  nl()
let \ dirac_ket_to_fortran_eval \ ff \ n \ indices \ gamma \ ket =
  let printf fmt = fprintf ff fmt
  and nl = pp\_newline ff in
  \mathsf{let}\ dsv\ =\ Ket\ n\ \mathsf{in}
  for i = 0 to 3 do
     let name = append\_indices (dsv\_name dsv) (succ i :: indices) in
     printf "_{ \sqcup \sqcup \sqcup \sqcup \sqcup} @ [<\%d>\%s_{ \sqcup} = \sqcup 0" (String.length name + 4) name;
     for i = 0 to 3 do
        if \neg (QC.is\_null\ gamma.(i).(j)) then
          printf
             "@<sub>\\</sub>s\\s\\\a(\\d)\"
             (format\_complex\_rational\_factor\ gamma.(i).(j))
             ket.name\ (succ\ j)
     done;
     printf "@]";
     nl()
```

```
done; \\ dsv
```

The same as $dirac_ket_to_fortran$, but apply the Dirac matrix gamma to bra from the right and return Bra n.

```
let dirac\_bra\_to\_fortran\_decl ff n indices =
        let printf fmt = fprintf ff fmt
        and nl = pp\_newline ff in
        \mathsf{let}\ dsv\ =\ Bra\ n\ \mathsf{in}
           "_{\cup\cup\cup\cup\cup}0[<2>complex(kind=default)%s_{\cup}::0_{\cup}%s0]"
          (dirac_dimension dsv indices) (dsv_name dsv);
        nl()
     let dirac_bra_to_fortran_eval ff n indices bra gamma =
        let printf fmt = fprintf ff fmt
        and nl = pp\_newline \ f\!\!f in
        \mathsf{let}\ dsv\ =\ Bra\ n\ \mathsf{in}
        for i = 0 to 3 do
          let name = append\_indices (dsv\_name dsv) (succ j :: indices) in
          for i = 0 to 3 do
             if \neg (QC.is\_null\ gamma.(i).(j)) then
                printf
                   "@<sub>\\</sub>s\\s\\\a(\\d)\"
                   (format\_complex\_rational\_factor\ gamma.(i).(j))
                   bra.name (succ i)
          done:
          printf "@]";
          nl()
        done:
        dsv
More of the same, but evaluating a spinor sandwich and returning Braket n.
     let dirac_braket_to_fortran_decl ff n indices =
        \mathsf{let}\ \mathit{printf}\ \mathit{fmt}\ =\ \mathit{fprintf}\ \mathit{ff}\ \mathit{fmt}
        and nl = pp\_newline ff in
        let \ dsv \ = \ Braket \ n \ in
        printf
           "_{\cup\cup\cup\cup\cup}0[<2>complex(kind=default)%s_{\cup}::0_{\cup}%s0]"
          (dirac_dimension dsv indices) (dsv_name dsv);
        nl()
     let \ dirac\_braket\_to\_fortran\_eval \ ff \ n \ indices \ bra \ gamma \ ket =
        \mathsf{let}\ \mathit{printf}\ \mathit{fmt}\ =\ \mathit{fprintf}\ \mathit{ff}\ \mathit{fmt}
        and nl = pp\_newline ff in
        \mathsf{let}\ dsv\ =\ Braket\ n\ \mathsf{in}
        let name = append\_indices (dsv\_name dsv) indices in
        printf "_{ \sqcup \sqcup \sqcup \sqcup \sqcup} @[<\%d>\%s_{ \sqcup} = _{\sqcup} 0" (String.length name + 4) name;
        for i = 0 to 3 do
          for j = 0 to 3 do
             if \neg (QC.is\_null\ gamma.(i).(j)) then
               printf
                   "@<sub>\\\\</sub>s\\\\a(\\\d)*\\\s\\\\a(\\\d)"
                   (format\_complex\_rational\_factor\ gamma.(i).(j))
                   bra.name (succ i) ket.name (succ j)
          done
        done;
        printf "@]";
        nl();
        dsv
```

Choose among the previous functions according to the position of *bra* and *ket* among the wavefunctions. If any is in the first position evaluate the spinor expression with the corresponding spinor removed, otherwise evaluate the spinir sandwich.

```
let dirac_bra_or_ket_to_fortran_decl ff n indices bra ket =
  if bra = 1 then
      dirac_ket_to_fortran_decl ff n indices
  else if ket = 1 then
      dirac_bra_to_fortran_decl ff n indices
  else
      dirac_braket_to_fortran_decl ff n indices

let dirac_bra_or_ket_to_fortran_eval ff n indices wfs bra gamma ket =
  if bra = 1 then
      dirac_ket_to_fortran_eval ff n indices gamma wfs.(pred ket)
  else if ket = 1 then
      dirac_bra_to_fortran_eval ff n indices wfs.(pred bra) gamma
  else
      dirac_braket_to_fortran_eval
      ff n indices wfs.(pred bra) gamma wfs.(pred ket)
```

UFO summation indices are negative integers. Derive a valid Fortran variable name.

```
let prefix\_summation = "mu"
\  \  \, \mathsf{let} \,\, \mathit{prefix\_polarization} \,\, = \,\, \mathtt{"nu"}
let index\_spinor = "alpha"
let index\_tensor = "nu"
let index_variable mu =
  if mu < 0 then
     Printf.sprintf "%s%d" prefix_summation (- mu)
  else if mu \equiv 0 then
   prefix\_polarization
  else
     Printf.sprintf "%s%d" prefix_polarization mu
let format_indices indices =
  String.concat "," (List.map index_variable indices)
  Partial.Make (struct type t = int let compare = compare end)
type \ tensor =
    DS of dsv
     V of string
    T of UFOx.Lorentz_Atom.vector
    S of UFOx.Lorentz\_Atom.scalar
    Inv of UFOx.Lorentz_Atom.scalar
```

Transform the Dirac strings if we have Majorana fermions involved, in order to implement the algorithm from JRR's thesis. NB: The following is for reference only, to better understand what JRR was doing... If the vertex is (suppressing the Lorentz indices of ϕ_2 and Γ)

$$\bar{\psi}\Gamma\phi\psi = \Gamma_{\alpha\beta}\bar{\psi}_{\alpha}\phi\psi_{\beta} \tag{14.22}$$

(cf. Coupling. FBF in the hardcoded O'Mega models), then this is the version implemented by fuse below.

The corresponding UFO fuse exchanges the arguments in the case of two fermions. This is the natural choice for cyclic permutations.

let $tho_print_FBF_current\ f\ c\ wf1\ wf2\ fusion =$ match fusion with $|\ [3;\ 1]\ \to\ printf\ "f\%sf_p120\ (\%s,\%s,\%s)"\ f\ c\ wf1\ wf2\ (*\ \Gamma_{\alpha\beta}\psi_{1,\beta}\bar{\psi}_{2,\alpha}\ *) \\ |\ [1;\ 3]\ \to\ printf\ "f\%sf_p120\ (\%s,\%s,\%s)"\ f\ c\ wf2\ wf1\ (*\ \Gamma_{\alpha\beta}\psi_{1,\beta}\bar{\psi}_{2,\alpha}\ *) \\ |\ [2;\ 3]\ \to\ printf\ "f\%sf_p012\ (\%s,\%s,\%s)"\ f\ c\ wf1\ wf2\ (*\ \Gamma_{\alpha\beta}\phi_1\psi_{2,\beta}\ *) \\ |\ [3;\ 2]\ \to\ printf\ "f\%sf_p201\ (\%s,\%s,\%s)"\ f\ c\ wf1\ wf2\ (*\ \Gamma_{\alpha\beta}\bar{\psi}_{1,\alpha}\phi_2\ *) \\ |\ [2;\ 1]\ \to\ printf\ "f\%sf_p201\ (\%s,\%s,\%s)"\ f\ c\ wf2\ wf1\ (*\ \Gamma_{\alpha\beta}\bar{\psi}_{1,\alpha}\phi_2\ *) \\ |\ [2;\ 1]\ \to\ printf\ "f\%sf_p201\ (\%s,\%s,\%s)"\ f\ c\ wf2\ wf1\ (*\ \Gamma_{\alpha\beta}\bar{\psi}_{1,\alpha}\phi_2\ *) \\ |\ [-\to\ ()$

This is how JRR implemented (see subsection Y.26.1) the Dirac matrices that don't change sign under $C\Gamma^TC^{-1} = \Gamma$, i. e. 1, γ_5 and $\gamma_5\gamma_\mu$ (see Targets.Fortran_Majorana_Fermions.print_fermion_current)

- In the case of two fermions, the second wave function wf2 is always put into the second slot, as described in JRR's thesis.
- In the case of a boson and a fermion, there is no need for both "f_%sf" and "f_f%s", since the latter can be obtained by exchanging arguments.

let $jrr_print_majorana_current_S_P_A$ f c wf1 wf2 fusion = match fusion with $| [1; 3] \rightarrow printf$ "%s_ff(%s,%s,%s)" f c wf1 wf2 (* $(C\Gamma)_{\alpha\beta}\bar{\psi}_{1,\alpha}\psi_{2,\beta}\cong C\Gamma$ *) $| [3; 1] \rightarrow printf$ "%s_ff(%s,%s,%s)" f c wf1 wf2 (* $(C\Gamma)_{\alpha\beta}\psi_{1,\alpha}\bar{\psi}_{2,\beta}\cong C\Gamma = C$ $C\Gamma^TC^{-1}$ *) $| [2; 3] \rightarrow printf$ "f_%sf(%s,%s,%s)" f c wf1 wf2 (* $\Gamma_{\alpha\beta}\phi_1\psi_{2,\beta}\cong \Gamma$ *) $| [3; 2] \rightarrow printf$ "f_%sf(%s,%s,%s)" f c wf2 wf1 (* $\Gamma_{\alpha\beta}\phi_1\psi_{2,\beta}\cong \Gamma$ *) $| [1; 2] \rightarrow printf$ "f_%sf(%s,%s,%s)" f c wf2 wf1 (* $\Gamma_{\alpha\beta}\phi_1\bar{\psi}_{2,\beta}\cong \Gamma = C\Gamma^TC^{-1}$ *) $| [2; 1] \rightarrow printf$ "f_%sf(%s,%s,%s)" f c wf1 wf2 (* $\Gamma_{\alpha\beta}\phi_1\bar{\psi}_{2,\beta}\cong \Gamma = C\Gamma^TC^{-1}$ *) $| [2; 1] \rightarrow printf$ "f_%sf(%s,%s,%s)" | f | c | wf1 | wf2 | e $| \Gamma_{\alpha\beta}\phi_1\bar{\psi}_{2,\beta}\cong \Gamma = C\Gamma^TC^{-1}$ *) $| \Gamma_{\alpha\beta}\phi_1\bar{\psi}_{2,\beta}\cong \Gamma = C\Gamma^TC^{-1}$ *)

This is how JRR implemented the Dirac matrices that do change sign under $C\Gamma^T C^{-1} = -\Gamma$, i.e. γ_{μ} and $\sigma_{\mu\nu}$ (see Targets.Fortran_Majorana_Fermions.print_fermion_current_vector).

 $\begin{array}{l} \text{let } \textit{jrr_print_majorana_current_V } \textit{f } \textit{c } \textit{wf1 } \textit{wf2 } \textit{fusion } = \\ \text{match } \textit{fusion } \text{with} \\ | \; [1; \; 3] \; \rightarrow \; \textit{printf} \; \text{"%s_ff}(\sqcup \%s, \%s, \%s) \text{"} \textit{f } \textit{c } \textit{wf1 } \textit{wf2 } (* \; (C\Gamma)_{\alpha\beta}\bar{\psi}_{1,\alpha}\psi_{2,\beta} \cong C\Gamma \; *) \\ | \; [3; \; 1] \; \rightarrow \; \textit{printf} \; \text{"%s_ff}(-\%s, \%s, \%s) \text{"} \textit{f } \textit{c } \textit{wf1 } \textit{wf2 } (* \; -(C\Gamma)_{\alpha\beta}\psi_{1,\alpha}\psi_{2,\beta} \cong -C\Gamma = C\; C\Gamma^TC^{-1} \; *) \\ | \; [2; \; 3] \; \rightarrow \; \textit{printf} \; \text{"f_\%sf}(\sqcup \%s, \%s, \%s) \text{"} \textit{f } \textit{c } \textit{wf1 } \textit{wf2 } (* \; \Gamma_{\alpha\beta}\phi_1\psi_{2,\beta} \cong \Gamma \; *) \\ | \; [3; \; 2] \; \rightarrow \; \textit{printf} \; \text{"f_\%sf}(-\%s, \%s, \%s) \text{"} \textit{f } \textit{c } \textit{wf2 } \textit{wf1 } (* \; \Gamma_{\alpha\beta}\phi_1\psi_{2,\beta} \cong \Gamma \; *) \\ | \; [1; \; 2] \; \rightarrow \; \textit{printf} \; \text{"f_\%sf}(-\%s, \%s, \%s) \text{"} \textit{f } \textit{c } \textit{wf1 } \textit{wf2 } (* \; -\Gamma_{\alpha\beta}\phi_1\psi_{2,\beta} \cong -\Gamma = C\Gamma^TC^{-1} \; *) \\ | \; [2; \; 1] \; \rightarrow \; \textit{printf} \; \text{"f_\%sf}(-\%s, \%s, \%s) \text{"} \textit{f } \textit{c } \textit{wf1 } \textit{wf2 } (* \; -\Gamma_{\alpha\beta}\phi_1\psi_{2,\beta} \cong -\Gamma = C\Gamma^TC^{-1} \; *) \\ | \; - \; \rightarrow \; () \end{array}$

These two can be unified, if the $_c$ functions implement $\Gamma' = C\Gamma^T C^{-1}$, but we must make sure that the multiplication with C from the left happens after the transformation $\Gamma \to \Gamma'$.

 $\label{let_print_majorana_current} \ \ let \ \mathit{jrr_print_majorana_current} \ f \ \ \mathit{c} \ \ \mathit{wf1} \ \ \mathit{wf2} \ \mathit{fusion} \ = \\ \ \ \mathsf{match} \ \mathit{fusion} \ \ \mathsf{with}$

Since we may assume $C^{-1} = -C = C^T$, this can be rewritten if the $_c$ functions implement

$$\Gamma'^T = (C\Gamma^T C^{-1})^T = (C^{-1})^T \Gamma C^T = C\Gamma C^{-1}$$
 (14.23)

instead.

 $\label{let_print_majorana_current_transposing} \begin{array}{l} f \ c \ wf1 \ wf2 \ fusion \ = \\ \\ \text{match } fusion \ \text{with} \end{array}$

```
 \begin{array}{l} \mid [1;\; 3] \; \to \; printf \; \text{"%s_ff}_{\sqcup\sqcup}(\text{%s,\%s,\%s}) \text{"} \; f \; c \; wf1 \; wf2 \; (*\; (C\Gamma)_{\alpha\beta}\bar{\psi}_{1,\alpha}\psi_{2,\beta} \cong C\Gamma \; *) \\ \mid [3;\; 1] \; \to \; printf \; \text{"%s_ff}_{\sqcup C}(\text{%s,\%s,\%s}) \text{"} \; f \; c \; wf2 \; wf1 \; (*\; (C\Gamma')_{\alpha\beta}^T\bar{\psi}_{1,\alpha}\psi_{2,\beta} \cong (C\Gamma')^T = -C\Gamma \; *) \\ \mid [2;\; 3] \; \to \; printf \; \text{"f_\%sf}_{\sqcup\sqcup}(\text{%s,\%s,\%s}) \text{"} \; f \; c \; wf1 \; wf2 \; (*\; \Gamma_{\alpha\beta}\phi_1\psi_{2,\beta} \cong \Gamma \; *) \\ \mid [3;\; 2] \; \to \; printf \; \text{"f_\%sf}_{\sqcup\sqcup}(\text{%s,\%s,\%s}) \text{"} \; f \; c \; wf2 \; wf1 \; (*\; \Gamma_{\alpha\beta}\phi_1\psi_{2,\beta} \cong \Gamma \; *) \\ \mid [1;\; 2] \; \to \; printf \; \text{"f_f\%s_c(\%s,\%s,\%s)} \text{"} \; f \; c \; wf1 \; wf2 \; (*\; \Gamma_{\alpha\beta}^T\bar{\psi}_{1,\alpha}\phi_2 \cong \Gamma'^T = C\Gamma C^{-1} \; *) \\ \mid [2;\; 1] \; \to \; printf \; \text{"f_f\%s_c(\%s,\%s,\%s)} \text{"} \; f \; c \; wf2 \; wf1 \; (*\; \Gamma_{\alpha\beta}^T\bar{\psi}_{1,\alpha}\phi_2 \cong \Gamma'^T = C\Gamma C^{-1} \; *) \\ \mid \; - \; \to \; () \end{array}
```

where we have used

$$(C\Gamma')^{T} = \Gamma'^{T}C^{T} = C\Gamma C^{-1}C^{T} = C\Gamma C^{-1}(-C) = -C\Gamma.$$
(14.24)

This puts the arguments in the same slots as $tho_print_dirac_current$ above and can be implemented by fuse, iff we inject the proper transformations in dennerize below. We notice that we do not need the conjugated version for all combinations, but only for the case of two fermions. In the two cases of one column spinor ψ , only the original version appears and in the two cases of one row spinor $\bar{\psi}$, only the conjugated version appears. Before we continue, we must however generalize from the assumption (14.22) that the fields in the vertex are always ordered as in Coupling.FBF. First, even in this case the slots of the fermions must be exchanged to accommodate the cyclic permutations. Therefore we exchange the arguments of the [1; 3] and [3; 1] fusions.

```
let jrr\_print\_majorana\_FBF\ f\ c\ wf1\ wf2\ fusion =  match fusion\ with\ (*fline\ =\ (3,\ 1)\ *) | [3;\ 1]\ \to\ printf\ "f\%sf\_p120\_c(\%s,\%s,\%s)"\ f\ c\ wf1\ wf2\ (*(C\Gamma')_{\alpha\beta}^T\psi_{1,\beta}\bar{\psi}_{2,\alpha}\cong (C\Gamma')^T=-C\Gamma\ *) | [1;\ 3]\ \to\ printf\ "f\%sf\_p120_{\sqcup\sqcup}(\%s,\%s,\%s)"\ f\ c\ wf2\ wf1\ (*(C\Gamma)_{\alpha\beta}\psi_{1,\beta}\bar{\psi}_{2,\alpha}\cong C\Gamma\ *) | [2;\ 3]\ \to\ printf\ "f\%sf\_p012_{\sqcup\sqcup}(\%s,\%s,\%s)"\ f\ c\ wf1\ wf2\ (*\Gamma_{\alpha\beta}\phi_1\psi_{2,\beta}\cong \Gamma\ *) | [3;\ 2]\ \to\ printf\ "f\%sf\_p012_{\sqcup\sqcup}(\%s,\%s,\%s)"\ f\ c\ wf2\ wf1\ (*\Gamma_{\alpha\beta}\phi_1\psi_{2,\beta}\cong \Gamma\ *) | [1;\ 2]\ \to\ printf\ "f\%sf\_p201_{\sqcup\sqcup}(\%s,\%s,\%s)"\ f\ c\ wf1\ wf2\ (*\Gamma'_{\alpha\beta}\bar{\psi}_{1,\alpha}\phi_2\cong \Gamma'^T=C\Gamma C^{-1}\ *) | [2;\ 1]\ \to\ printf\ "f\%sf\_p201_{\sqcup\sqcup}(\%s,\%s,\%s)"\ f\ c\ wf2\ wf1\ (*\Gamma'_{\alpha\beta}\bar{\psi}_{1,\alpha}\phi_2\cong \Gamma'^T=C\Gamma C^{-1}\ *) | [2;\ 1]\ \to\ printf\ "f\%sf\_p201_{\sqcup\sqcup}(\%s,\%s,\%s)"\ f\ c\ wf2\ wf1\ (*\Gamma'_{\alpha\beta}\bar{\psi}_{1,\alpha}\phi_2\cong \Gamma'^T=C\Gamma C^{-1}\ *) | [2;\ 1]\ \to\ printf\ "f\%sf\_p201_{\sqcup\sqcup}(\%s,\%s,\%s)"\ f\ c\ wf2\ wf1\ (*\Gamma'_{\alpha\beta}\bar{\psi}_{1,\alpha}\phi_2\cong \Gamma'^T=C\Gamma C^{-1}\ *)
```

The other two permutations:

```
let \ jrr\_print\_majorana\_FFB \ f \ c \ wf1 \ wf2 \ fusion =
        match fusion with (*fline = (1, 2) *)
               [3; 1] \rightarrow printf "ff%s_p120_\(\subseteq\) (%s,%s,%s)" f \ c \ wf1 \ wf2 \ (* \Gamma_{\alpha\beta}\phi_1\psi_{2,\beta} \constant \Gamma *)
               [1;\ 3]\ \rightarrow\ printf\ \texttt{"ff\%s\_p120}_{\sqcup\sqcup}(\texttt{\%s,\%s,\%s})\ \texttt{"}\ f\ c\ wf2\ wf1\ (*\ \Gamma_{\alpha\beta}\phi_1\psi_{2,\beta}\cong\Gamma\ *)
        [2;\ 3] \ \rightarrow \ printf \ "ff%s_p012_{\sqcup\sqcup}(\%s,\%s,\%s)" \ f \ c \ wf1 \ wf2 \ (* \ \Gamma'^T_{\alpha\beta}\bar{\psi}_{1,\alpha}\phi_2 \cong \Gamma'^T = C\Gamma C^{-1} \ *)
        [3;\ 2]\ \to\ printf\ \texttt{"ff\%s\_p012}_{\sqcup\sqcup}(\texttt{\%s},\texttt{\%s},\texttt{\%s}) \texttt{"}\ f\ c\ wf2\ wf1\ (*\ \Gamma'^T_{\alpha\beta}\bar{\psi}_{1,\alpha}\phi_2\cong\Gamma'^T=C\Gamma C^{-1}\ *)
         [1;\ 2]\ \rightarrow\ printf\ \texttt{"ff\%s\_p201}_{\sqcup\sqcup}(\texttt{\%s,\%s,\%s})\ \texttt{"}\ f\ c\ wf1\ wf2\ (*\ (C\Gamma)_{\alpha\beta}\psi_{1,\beta}\bar{\psi}_{2,\alpha}\cong C\Gamma\ *)
         [2; 1] \rightarrow \textit{printf} \ \texttt{"ff%s\_p201\_c(\%s,\%s,\%s)"} \ \textit{f} \ \textit{c} \ \textit{wf2} \ \textit{wf1} \ (* \ (C\Gamma')_{\alpha\beta}^T \psi_{1,\beta} \bar{\psi}_{2,\alpha} \cong (C\Gamma')^T = -C\Gamma \ *)
let jrr\_print\_majorana\_BFF\ f\ c\ wf1\ wf2\ fusion\ =
        match fusion with (*fline = (2, 3) *)
         | \ [3; \ 1] \ \to \ printf \ "\%sff_p120_{\sqcup\sqcup} (\%s, \%s, \%s) \ " \ f \ c \ wf1 \ wf2 \ (* \ \Gamma'^T_{\alpha\beta} \bar{\psi}_{1,\alpha} \phi_2 \cong \Gamma'^T = C\Gamma C^{-1} \ *) 
        [1;\ 3]\ \rightarrow\ printf\ \text{"%sff_pl20}_{\sqcup\sqcup}\text{(%s,\%s,\%s)}\ \text{"}\ f\ c\ wf2\ wf1\ (*\ \Gamma'T_{\alpha\beta}\bar{\psi}_{1,\alpha}\phi_2\cong\Gamma'^T=C\Gamma C^{-1}\ *)
        | [2; 3] \rightarrow printf "%sff_p012_{\square\square}(%s,%s,%s)" f \ c \ wf1 \ wf2 \ (* (C\Gamma)_{\alpha\beta}\psi_{1,\beta}\bar{\psi}_{2,\alpha}\cong C\Gamma \ *)
            [3;\ 2] \ \to \ printf \ \text{``%sff_p012\_c (\%s,\%s,\%s)''} \ f \ c \ wf2 \ wf1 \ (* \ (C\Gamma')_{\alpha\beta}^T \psi_{1,\beta} \bar{\psi}_{2,\alpha} \cong (C\Gamma')^T = -C\Gamma \ *)
              [1;\ 2]\ \rightarrow\ printf\ \text{"%sff_p201}_{\sqcup\sqcup}(\text{\%s,\%s,\%s})\ \text{"}\ f\ c\ wf1\ wf2\ (*\ \Gamma_{\alpha\beta}\phi_1\overline{\psi}_{2,\beta}\cong\Gamma\ *)
         [2; 1] \rightarrow printf "%sff_p201_\(\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00ed\u00
```

In the model, the necessary information is provided as $Coupling.fermion_lines$, encoded as (right, left) in the usual direction of the lines. E. g. the case of (14.22) is (3,1). Equivalent information is available as (ket, bra) in $UFO_Lorentz.dirac_string$.

```
and nl = pp\_newline eval in
if is\_majorana\ wfs.(pred\ atom.L.bra).spin\ \lor
      is_majorana wfs.(pred atom.L.ket).spin then
  if atom.L.bra = 1 then
     (* Fusing one or more bosons with a ket like fermion: \chi \leftarrow \Gamma \chi. *)
     (* Don't do anything, as per subsection Y.26.1. *)
     atom
  else if atom.L.ket = 1 then
     (* We fuse one or more bosons with a bra like fermion: \bar{\chi} \leftarrow \bar{\chi}\Gamma. *)
     (*\Gamma \to C\Gamma C^{-1}. *)
     begin
       let atom = L.conjugate atom in
       printf "_\underset \underset conjugated_\underset for \underset Majorana"; nl();
       printf ~" \verb| | s " ~ (L.dirac\_string\_to\_string ~atom); ~nl ~ ();
        atom
     end
  else if \neg atom.L.conjugated then
     (* We fuse zero or more bosons with a sandwich of fermions. \phi \leftarrow \bar{\chi}\gamma\chi.*)
     (* Multiply by C from the left, as per subsection Y.26.1. *)
     begin
       let atom = L.cc\_times atom in
       printf "____!_multiplied_by_CC_for_Majorana"; nl();
       printf "_{\sqcup\sqcup\sqcup\sqcup}!_{\sqcup}\%s" (L.dirac\_string\_to\_string \ atom); \ nl \ ();
        atom
     end
  else
     (* Transposed: multiply by -C from the left. *)
       let atom = L.minus (L.cc\_times atom) in
       printf "□□□□□!□multiplied□by□-CC□for□Majorana"; nl();
       end
else
   atom
```

Write the *i*th Dirac string ds as Fortran code to eval, including a shorthand representation as a comment. Return ds with ds.L.atom replaced by the dirac string variable, i, e. DS dsv annotated with the internal and external indices. In addition write the declaration to decl.

```
let printf fmt = fprintf eval fmt
  and nl = pp\_newline eval in
  let \ bra = ds.L.atom.L.bra
  and ket = ds.L.atom.L.ket in
  pp_divide ~indent : 4 eval ();
  let atom = dennerize ~~eval ~wfs ~ds.L.atom ~in
  begin match ds.L.indices with
  | [] \rightarrow
     let gamma = L.dirac\_string\_to\_matrix (fun \_ \rightarrow 0) atom in
     dirac_bra_or_ket_to_fortran_decl decl i [] bra ket;
       dirac_bra_or_ket_to_fortran_eval eval i [] wfs bra gamma ket in
     L.map\_atom (fun \_ \rightarrow DS \ dsv) \ ds
   indices \rightarrow
     dirac_bra_or_ket_to_fortran_decl decl i indices bra ket;
     let combinations = Product.power (List.length indices) [0; 1; 2; 3] in
     let dsv =
       List.map
         (fun combination \rightarrow
           let \ substitution = IntPM.of\_lists \ indices \ combination \ in
```

```
let substitute = IntPM.apply\ substitution in let indices = List.map\ substitute\ indices in let gamma = L.dirac\_string\_to\_matrix\ substitute\ atom in dirac\_bra\_or\_ket\_to\_fortran\_eval\ eval\ i\ indices\ wfs\ bra\ gamma\ ket) combinations in begin match ThoList.uniq\ (List.sort\ compare\ dsv) with |\ [dsv] \ \to \ L.map\_atom\ (fun\ \_\ \to\ DS\ dsv)\ ds |\ \_\ \to\ failwith\ "dirac\_string\_to\_fortran: \_impossible" end end
```

Write the Dirac strings in the list ds_list as Fortran code to eval, including shorthand representations as comments. Return the list of variables and corresponding indices to be contracted.

Perform a nested sum of terms, as printed by *print_term* (which takes the number of spaces to indent as only argument) of the cartesian product of *indices* running from 0 to 3.

```
let nested\_sums\ \~decl\ \~eval\ initial\_indent\ indices\ print\_term\ =\ let\ rec\ nested\_sums'\ indent\ =\ function
|\ [\ ]\ \to\ print\_term\ indent\ |\ lindex\ ::\ indices\ \to\ let\ var\ =\ index\_variable\ index\ in\ fprintf\ eval\ "%*s@[<2>do_\%s_\=_\0,\_3@]"\ indent\ ""\ var;\ pp\_newline\ eval\ ();\ nested\_sums'\ (indent\ +\ 2)\ indices;\ pp\_newline\ eval\ ();\ fprintf\ eval\ "%*s@[<2>end_\do@]"\ indent\ ""\ in\ nested\_sums'\ (initial\_indent\ +\ 2)\ indices
```

Polarization indices also need to be summed over, but they appear only once.

```
let \ indices\_of\_contractions \ contractions \ =
  let index\_pairs, polarizations =
     L.classify\_indices
       (ThoList.flatmap (fun \ ds \rightarrow \ ds.L.indices) \ contractions) \ in
     ThoList.pairs index_pairs @ ThoList.uniq (List.sort compare polarizations)
  with
  | Invalid\_argument s \rightarrow
      invalid\_arg
        ("indices_of_contractions:_" ^
            ThoList.to_string_of_int_index_pairs)
let format_dsv dsv indices =
  match dsv, indices with
    Braket \_, [] \rightarrow dsv\_name dsv
    Braket \_, ilist \rightarrow
      Printf.sprintf "%s(%s)" (dsv_name dsv) (format_indices indices)
    (Bra \_ | Ket \_), [] \rightarrow
      Printf.sprintf "%s(%s)" (dsv\_name\ dsv) index\_spinor
  | (Bra \_ | Ket \_), ilist \rightarrow
      Printf.sprintf
        "%s(%s,%s)" (dsv_name dsv) index_spinor (format_indices indices)
let denominator\_name = "denom\_"
let \ mass\_name \ = \ "m\_"
let width\_name = "w\_"
let format\_tensor t =
  let indices = t.L.indices in
```

```
match t.L.atom with
    DS \ dsv \rightarrow format\_dsv \ dsv \ indices
    V \ vector \rightarrow Printf.sprintf "%s(%s)" vector \ (format\_indices \ indices)
    T\ UFOx.Lorentz\_Atom.P\ (mu,\ n) \rightarrow
      Printf.sprintf "p%d(%s)" n (index_variable mu)
    T\ UFOx.Lorentz\_Atom.Epsilon\ (mu1,\ mu2,\ mu3,\ mu4)\ 
ightarrow
      Printf.sprintf "eps4_(%s)" (format_indices [mu1; mu2; mu3; mu4])
    T\ UFOx.Lorentz\_Atom.Metric\ (mu1,\ mu2)\ 
ightarrow
      if mu1 > 0 \land mu2 > 0 then
        Printf.sprintf "g44_(%s)" (format_indices [mu1; mu2])
      else
        failwith "format_tensor:\Boxcompress_metrics\Boxhas\Boxfailed!"
    S(UFOx.Lorentz\_Atom.Mass\_) \rightarrow mass\_name
    S(UFOx.Lorentz\_Atom.Width\_) \rightarrow width\_name
    S(UFOx.Lorentz\_Atom.P2\ i) \rightarrow Printf.sprintf "g2\_(p%d)"\ i
    S(UFOx.Lorentz\_Atom.P12\ (i,\ j)) \rightarrow Printf.sprintf\ "g12\_(p%d,p%d)"\ i\ j
    Inv (UFOx.Lorentz\_Atom.Mass\_) \rightarrow "1/" ^ mass\_name
    Inv (UFOx.Lorentz\_Atom.Width\_) \rightarrow "1/" ^ width\_name
    Inv (UFOx.Lorentz\_Atom.P2 \ i) \rightarrow Printf.sprintf "1/g2\_(p%d)" \ i
    Inv (UFOx.Lorentz\_Atom.P12 (i, j)) \rightarrow
      Printf.sprintf "1/g12_(p%d,p%d)" i j
    S (UFOx.Lorentz\_Atom.Variable s) \rightarrow s
    Inv (UFOx.Lorentz\_Atom.Variable \ s) \rightarrow "1/" \ \hat{} \ s
    S(UFOx.Lorentz\_Atom.Coeff c) \rightarrow UFOx.Value.to\_string c
    Inv (UFOx.Lorentz\_Atom.Coeff \ c) \rightarrow "1/(" \ `UFOx.Value.to\_string \ c \ `")"
let rec multiply\_tensors ~ decl ~ eval = function
  [] \rightarrow fprintf \ eval "1";
    [t] \rightarrow fprintf \ eval "%s" (format\_tensor \ t)
  t :: tensors \rightarrow
     fprintf eval "%s@,*" (format_tensor t);
      multiply_tensors ~decl ~eval tensors
let pseudo_wfs_for_denominator =
  Array.init
    2
    (fun i \rightarrow
       let ii = string\_of\_int i in
       \{ pos = i; 
         spin = Coupling.Scalar;
         name = denominator\_name;
         local\_array = None;
         momentum = "k" \hat{i}i;
         momentum\_array = "p" ^ ii;
         fortran\_type = fortran\_type Coupling.Scalar \})
let contract_indices ~decl ~eval indent wf_indices wfs (fusion, contractees) =
  let printf fmt = fprintf eval fmt
  and nl = pp\_newline eval in
  let sum_var =
    begin match wf\_indices with
      [] \rightarrow wfs.(0).name
    | ilist \rightarrow
        let indices = String.concat "," ilist in
        begin match wfs.(0).local\_array with
        | None \rightarrow
            let component =
              begin match wfs.(0).spin with
                Coupling.Spinor \mid Coupling.ConjSpinor \mid Coupling.Majorana \rightarrow "a"
                Coupling.Tensor_2 \rightarrow "t"
              | Coupling.Vector | Coupling.Massive\_Vector \rightarrow
                  failwith "contract_indices:\_expected\_local_array\_for\_vectors"
              \bot \rightarrow failwith "contract_indices:\botunexpected\botspin"
```

```
end in
            Printf.sprintf "%s%%%s(%s)" wfs.(0).name component indices
        | Some a \rightarrow Printf.sprintf "%s(%s)" a indices
        end
    end in
  let indices =
     List.filter
       (fun i \rightarrow UFOx.Index.position i \neq 1)
       (indices_of_contractions contractees) in
  nested\_sums
     \tilde{decl} \tilde{eval}
     indent indices
     (fun indent \rightarrow
       printf "%*s@[<2>%s_{\perp}=_{\perp}%s" indent "" sum_{-}var sum_{-}var;
       printf "@_\%s" (format_complex_rational_factor fusion.L.coeff);
       List.iter~(\texttt{fun}~i~\rightarrow~printf~\texttt{"@,g4\_(\%s)*"}~(index\_variable~i))~indices;
       printf "@,(";
       multiply_tensors ~decl ~eval contractees;
       printf ")":
       begin match fusion.L.denominator with
       | [] \rightarrow ()
       end;
       printf "@]");
  printf "@]";
  nl()
let scalar_expression1 ~decl ~eval fusion =
  let printf fmt = fprintf eval fmt in
  match fusion.L.dirac, fusion.L.vector with
  | [], [] \rightarrow
      let scalars =
        List.map  (fun t \rightarrow \{ L.atom = S t; L.indices = [] \}) fusion. L.scalar
      and inverses =
        List.map (fun t \rightarrow \{L.atom = Inv \ t; L.indices = [] \}) fusion.L.inverse in
      let \ contractees = scalars @ inverses in
      printf "Q_{\square}%s" (format\_complex\_rational\_factor\ fusion.L.coeff);
      multiply\_tensors\ \tilde{\ }decl\ \tilde{\ }eval\ contractees
  | -, [] \rightarrow
      invalid\_arg
        "UFO\_targets.Fortran.scalar\_expression1: \_unexpected\_spinor\_indices"
  | \ | \ |, \ \_ \rightarrow
      invalid\_arg
        "UFO_targets.Fortran.scalar_expression1: unexpected vector indices"
  _, _ →
      invalid\_arg
        "UFO\_targets.Fortran.scalar\_expression1: \_unexpected\_indices"
let scalar_expression ~decl ~eval indent name fusions =
  \mathsf{let}\ \mathit{printf}\ \mathit{fmt}\ =\ \mathit{fprintf}\ \mathit{eval}\ \mathit{fmt}
  and nl = pp\_newline \ eval \ in
  let sum_var = name in
  printf "%*s@[<2>%s<sub>\\\\</sub>=" indent "" sum_var;
  List.iter\ (scalar\_expression1\ \~decl\ \~eval)\ fusions;
  printf "@]";
  nl()
let local_vector_copies ~decl ~eval wfs =
  begin match wfs.(0).local\_array with
    None \rightarrow ()
   | Some \ a \rightarrow
      fprintf
```

```
pp\_newline \ decl \ ()
  end;
  let n = Array.length wfs in
  for i = 1 to n - 1 do
     match wfs.(i).local\_array with
        None \rightarrow ()
     \mid Some \ a \rightarrow
         fprintf
            decl = (<2>complex(kind=default), @_dimension(0:3)_::@_%s@] = a;
          pp\_newline \ decl \ ();
          fprintf\ eval\ "$=$\ldots$ (0)$=$\ldots$%t0] " $a\ wfs.(i).name;
          pp\_newline eval ();
          fprintf\ eval\ "$=$_\sigma s(1:3)$=$_\%s\%x@] " a wfs.(i).name;
          pp\_newline eval ()
  done
let return\_vector ff wfs =
  \mathsf{let}\ \mathit{printf}\ \mathit{fmt}\ =\ \mathit{fprintf}\ \mathit{ff}\ \mathit{fmt}
  and nl = pp\_newline ff in
  match wfs.(0).local\_array with
   | None \rightarrow ()
   \mid Some \ a \rightarrow
       pp\_divide \ \tilde{} indent : 4 ff \ ();
       printf "_{\Box\Box\Box\Box} @ [<2>\%s\%t_{\Box} = _{\Box}\%s(0)@] " wfs.(0).name a; nl ();
       printf "_{ \sqcup \sqcup \sqcup \sqcup \sqcup } @[<2>\%s\%x_{ \sqcup = \sqcup}\%s(1:3)@] " wfs.(0).name a; nl () 
let \ \mathit{multiply\_coupling\_and\_scalars} \ \mathit{ff} \ \mathit{g\_opt} \ \mathit{wfs} \ = \\
  let printf fmt = fprintf ff fmt
  and nl = pp\_newline ff in
  pp\_divide ~ indent : 4 ff ();
  let g =
     match g_-opt with
     | None \rightarrow ""
      \mid Some \ g \rightarrow g \ \hat{\ } "*" in
  let wfs0name =
     match wfs.(0).local\_array with
       None \rightarrow wfs.(0).name
      \mid Some \ a \rightarrow a \ in
  printf " \subseteq 0 <2 > %s = %s%s" wfs0name g wfs0name;
  for i = 1 to Array.length \ wfs - 1 do
     match wfs.(i).spin with
       Coupling.Scalar \rightarrow printf "@,*%s" wfs.(i).name
      | - \rightarrow ()
  done;
  printf "@]"; nl ()
let local_momentum_copies ~decl ~eval wfs =
  let n = Array.length wfs in
  fprintf
      decl "_____@[<2>real(kind=default), @_dimension(0:3)_::@_\%s"
      wfs.(0).momentum\_array;
  for i = 1 to n - 1 do
     fprintf\ decl\ ", @_{\sqcup}%s"\ wfs.(i).momentum\_array;
     fprintf
        eval "_{\Box \Box \Box \Box} @ [<2>\%s(0)_{\Box} = _{\Box}\%s\%\%t@] "
        wfs.(i).momentum\_array \ wfs.(i).momentum;
     pp\_newline eval ();
     fprintf
         eval "_{\Box \Box \Box \Box} @ [<2>\%s(1:3)_{\Box} =_{\Box}\%s\%%x@] "
        wfs.(i).momentum\_array \ wfs.(i).momentum;
     pp\_newline eval ()
  done;
  fprintf\ eval\ "_{\sqcup\sqcup\sqcup\sqcup\sqcup}@[<2>\%s_{\sqcup}="\ wfs.(0).momentum\_array;
```

```
for i = 1 to n - 1 do
    fprintf\ eval\ "Q_{\sqcup}-_{\sqcup}%s"\ wfs.(i).momentum\_array
  done;
  fprintf decl "@]";
  pp\_newline \ decl \ ();
  fprintf eval "@]";
  pp\_newline eval ()
let contractees_of_fusion
       \tilde{decl} \tilde{eval} wfs (max\_dsv, indices\_seen, contractees) fusion =
  let max\_dsv', dirac\_strings =
     dirac_strings_to_fortran ~decl ~eval wfs max_dsv fusion.L.dirac
  and vectors =
     List.fold\_left
       (fun acc \ wf \rightarrow
         match wf.spin, wf.local_array with
          | Coupling. Tensor_2, None \rightarrow
             \{L.atom =
                   V (Printf.sprintf "%s%d%%t" (spin_mnemonic wf.spin) wf.pos);
                L.indices = [UFOx.Index.pack\ wf.pos\ 1;
                                UFOx.Index.pack \ wf.pos \ 2] \} :: acc
          \mid _, None \rightarrow acc
           A, Some a \rightarrow \{L.atom = V \ a; L.indices = [wf.pos]\} :: acc)
       [] (List.tl (Array.to\_list wfs))
  and tensors =
     List.map\ (L.map\_atom\ (fun\ t\ 	o\ T\ t))\ fusion.L.vector
     List.map (fun t \rightarrow \{L.atom = S t; L.indices = [] \}) fusion.L.scalar
  and inverses =
     List.map (fun t \rightarrow \{L.atom = Inv \ t; L.indices = [] \}) fusion.L.inverse in
  let contractees' = dirac_strings @ vectors @ tensors @ scalars @ inverses in
  let indices\_seen' =
     Sets.Int.of_list (indices_of_contractions contractees') in
  (max\_dsv',
   Sets.Int.union indices_seen indices_seen',
   (fusion, contractees') :: contractees)
let local\_name \ wf =
  match wf.local_array with
    Some \ a \rightarrow a
    None \rightarrow
      match wf.spin with
      | Coupling.Spinor | Coupling.ConjSpinor | Coupling.Majorana \rightarrow
          wf.name ^ "%a"
        Coupling.Scalar \rightarrow wf.name
        Coupling.Tensor_2 \rightarrow wf.name `"%t"
      | Coupling.Vector | Coupling.Massive\_Vector \rightarrow
         failwith "UFO_targets.Fortran.local_name:\(\_\underset\)unexpected\(\_\underset\)spin\(\_1\)"
         failwith "UFO_targets.Fortran.local_name:_uunhandled_spin"
let external_wf_loop ~decl ~eval ~indent wfs (fusion, _ as contractees) =
  pp_divide ~indent eval ();
  fprintf\ eval "%*s! \( \lambda\sigma\)" indent "" (L.to_string [fusion]); pp_newline\ eval\ ();
  pp\_divide ~ indent ~ eval ~ ();
  begin match fusion.L.denominator with
   | [] \rightarrow ()
  \mid denominator \rightarrow
      scalar\_expression\ \~eval\ 4\ denominator\_name\ denominator
  end;
  match wfs.(0).spin with
  \mid Coupling.Scalar \rightarrow
      contract_indices ~decl ~eval 2 [] wfs contractees
```

```
| Coupling.Spinor | Coupling.ConjSpinor | Coupling.Majorana \rightarrow
      let idx = index\_spinor in
     fprintf\ eval\ "%*s@[<2>do_\%s_\=\1,\_4@]"\ indent\ ""\ idx;\ pp\_newline\ eval\ ();
      contract_indices ~decl ~eval 4 [idx] wfs contractees;
      fprintf eval "%*send_do@]" indent ""; pp_newline eval ()
  | Coupling.Vector | Coupling.Massive\_Vector \rightarrow
      let idx = index\_variable 1 in
     fprintf\ eval\ "\%*s@[<2>do_{\square}\%s_{\square}=_{\square}0,_{\square}3@]\ "\ indent\ ""\ idx;\ pp\_newline\ eval\ ();
      contract_indices ~ decl ~ eval 4 [idx] wfs contractees;
     fprintf eval "%*send_do@]" indent ""; pp_newline eval ()
  \mid Coupling.Tensor\_2 \rightarrow
      let idx1 = index\_variable (UFOx.Index.pack 1 1)
      and idx2 = index\_variable (UFOx.Index.pack 1 2) in
      fprintf\ eval\ "%*s@[<2>do_\%s_=_0,_3@]"\ indent\ ""\ idx1;
      pp_newline eval ();
      fprintf\ eval\ "\%*s@[<2>do\%s\=\0,\3@]"\ (indent\ +\ 2)\ ""\ idx2;
      pp\_newline\ eval\ ();
      contract_indices ~ decl ~ eval 6 [idx1; idx2] wfs contractees;
      fprintf\ eval\ "%*send_do@]"\ (indent\ +\ 2)\ "";\ pp_newline\ eval\ ();
      fprintf eval "%*send_do@]" indent ""; pp_newline eval ()
    Coupling. Vectorspinor \rightarrow
      failwith "external_wf_loop: Uectorspinor not supported yet!"
    Coupling.Maj\_Ghost \rightarrow
      failwith "external_wf_loop:\unexpected\underpress{Maj_Ghost}"
  | Coupling.Tensor_1 \rightarrow
     failwith "external_wf_loop:_unexpected_Tensor_1"
  \mid Coupling.BRS \rightarrow
      failwith "external_wf_loop:\unexpected\uBRS"
let fusions_to_fortran ~decl ~eval wfs ?(denominator = []) ?coupling fusions =
  local_vector_copies ~ decl ~ eval wfs;
  local_momentum_copies ~decl ~eval wfs;
  begin match denominator with
  | [] \rightarrow ()
      fprintf decl "LULLO [<2>complex(kind=default) :: \"\s@] " denominator_name;
      pp_newline decl ()
  end:
  let max\_dsv, indices\_used, contractions =
     List.fold\_left
       (contractees_of_fusion ~decl ~eval wfs)
       (0, Sets.Int.empty, [])
       fusions in
  Sets.Int.iter
    (fun index \rightarrow
       pp\_newline \ decl \ ())
     indices\_used;
  begin match wfs.(0).spin with
  | Coupling.Spinor | Coupling.ConjSpinor | Coupling.Majorana \rightarrow
      fprintf\ decl\ "$\sqcup\sqcup\sqcup\sqcup\sqcup @[<2>$integer$\sqcup::@$\sqcup\%s@]$" index$\_spinor$;
     pp\_newline \ decl \ ()
  | - \rightarrow ()
  end;
  pp\_divide \ \tilde{} indent : 4 \ eval \ ();
  let wfs0name = local\_name \ wfs.(0) in
  \textit{fprintf eval "$$} \verb""=$$$ \verb"" wfs0name";
  pp\_newline\ eval\ ();
  List.iter (external_wf_loop ~decl ~eval ~indent: 4 wfs) contractions;
  multiply_coupling_and_scalars eval coupling wfs;
  begin match denominator with
```

```
 \begin{array}{l} |\hspace{.05cm} [\hspace{.05cm} ] \rightarrow () \\ |\hspace{.05cm} denominator \rightarrow \\ |\hspace{.05cm} pp\_divide ~ indent : 4 ~ eval ~ (); \\ |\hspace{.05cm} fprintf ~ eval ~ "%*s!\_%s" ~ 4 ~ "" ~ (L.to\_string ~ denominator); \\ |\hspace{.05cm} pp\_newline ~ eval ~ (); \\ |\hspace{.05cm} scalar\_expression ~ decl ~ eval ~ 4 ~ denominator\_name ~ denominator; \\ |\hspace{.05cm} fprintf ~ eval \\ |\hspace{.05cm} "_{ \sqcup \sqcup \sqcup \sqcup \sqcup } @ [<2>\%s_{\sqcup} = @_{\sqcup}\%s_{\sqcup}/_{\sqcup}\%s@] " ~ wfs0name ~ wfs0name ~ denominator\_name; \\ |\hspace{.05cm} pp\_newline ~ eval ~ () \\ |\hspace{.05cm} end; \\ |\hspace{.05cm} return\_vector ~ eval ~ wfs \\ |\hspace{.05cm} \end{array}
```

TODO: eventually, we should include the momentum among the arguments only if required. But this can wait for another day.

```
let lorentz ff name spins lorentz =
       let printf fmt = fprintf ff fmt
       and nl = pp\_newline ff in
       let wfs = wf\_table \ spins \ in
       let n = Array.length wfs in
       printf "⊔∪@[<4>pureufunctionu%s@u(g,@u" name;
       for i = 1 to n - 2 do
               printf "%s,@_{\sqcup}%s,@_{\sqcup}" wfs.(i).name wfs.(i).momentum
       done;
       printf "%s, @_{\sqcup}%s" wfs.(n-1).name\ wfs.(n-1).momentum;
       printf ")@_result_(%s)@] " wfs.(0).name; nl();
       printf " \subseteq \mathbb{C} = \mathbb{C}  \mathbb{C} = \mathbb{C}  \mathbb{C} = \mathbb{C}  \mathbb{C} = \mathbb{C}  \mathbb{C} = \mathbb{C} = \mathbb{C}  \mathbb{C} = \mathbb{C} 
       printf "_{\sqcup\sqcup\sqcup\sqcup} @[<2>complex(kind=default), @_intent(in)_{\sqcup}::@_{\sqcup} @]"; nl();
       for i = 1 to n - 1 do
               printf
                        "_{\cup\cup\cup\cup\cup}0[<2>%s,_intent(in)_::_\%s0]"
                       wfs.(i).fortran\_type\ wfs.(i).name;\ nl();
       done;
       printf "____@[<2>type(momentum),_intent(in)_::@_%s" wfs.(1).momentum;
       for i = 2 to n - 1 do
               printf ", @_{\sqcup}%s" wfs.(i).momentum
       done;
       printf "@]";
       nl();
       let width = 80 in (* get this from the default formatter instead! *)
       let \ decl_buf = Buffer.create \ 1024
       and eval\_buf = Buffer.create 1024 in
       let decl = formatter\_of\_buffer ~width ~decl\_buf
       and eval = formatter\_of\_buffer ~width ~eval\_buf in
       fusions_to_fortran ~decl ~eval ~coupling: "g" wfs lorentz;
       pp_flush decl ();
       pp\_flush \ eval \ ();
       pp\_divide ~indent : 4 ff ();
       printf "%s" (Buffer.contents decl_buf);
       pp\_divide ~ indent : 4 ff ();
       printf "_{\square \square \square \square} if_{\square}(g_{\square} = = _{\square} 0)_{\square} then"; nl ();
       printf ~ \verb"uuuuuucalluset_zerou(%s)" ~ wfs.(0).name; ~ nl ~ ();
       printf "____return"; nl ();
       printf "_\underset end\underset if"; nl ();
       pp\_divide ~ indent : 4 ff ();
        printf "%s" (Buffer.contents eval_buf);
       printf "_u_end_function_%s@] " name; nl ();
        Buffer.reset decl_buf;
         Buffer.reset eval_buf;
let use_variables ff parameter_module variables =
       let printf fmt = fprintf ff fmt
```

```
and nl = pp\_newline ff in
  match variables with
   [] \rightarrow ()
   v :: v\_list \rightarrow
       printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup} @[<2>use_{\sqcup}%s, _{\sqcup}only:_{\sqcup}%s" parameter\_module v;
       List.iter (fun s \rightarrow printf ", "%s" s) v\_list;
       printf "@]"; nl ()
let propagator ff name parameter_module variables
        (bra\_spin, ket\_spin) numerator denominator =
  let printf fmt = fprintf ff fmt
  and nl = pp\_newline ff in
  let width = 80 in (* get this from the default formatter instead! *)
  let wf\_name = spin\_mnemonic ket\_spin
  and wf\_type = fortran\_type \ ket\_spin \ in
  let wfs = wf\_table [| ket\_spin; ket\_spin |] in
  printf
     "_{\sqcup\sqcup}@[<4>pure_{\sqcup}function_{\sqcup}pr_{U_{s}@_{\sqcup}(k2,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s2)}"
     name mass_name width_name wf_name;
   printf "_result_(%s1)@] " wf_name; nl ();
   use_variables ff parameter_module variables;
  printf "_{\square \square \square \square} \%s_{\square} :: \square \%s1" wf_type wf_name; nl ();
  printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup} type (momentum), _{\sqcup} intent(in)_{\sqcup} : _{\sqcup} k2"; nl ();
  printf
      "____real(kind=default),_intent(in)_::_\%s,_\%s"
     mass_name width_name; nl();
  printf "_{\square\square\square\square}%s,_{\square}intent(in)_{\square}::_{\square}%s2" wf_{-}type \ wf_{-}name; \ nl \ ();
  \mathsf{let}\ \mathit{decl\_buf}\ =\ \mathit{Buffer.create}\ 1024
  and eval\_buf = Buffer.create 1024 in
  let decl = formatter\_of\_buffer ~width ~decl\_buf
  and eval = formatter\_of\_buffer ~width ~eval\_buf in
  fusions_to_fortran ~decl ~eval wfs ~denominator numerator;
  pp\_flush \ decl \ ();
  pp_{-}flush \ eval \ ();
  pp\_divide ~ indent : 4 ff ();
   printf "%s" (Buffer.contents decl_buf);
  pp\_divide ~indent : 4 ff ();
  printf "%s" (Buffer.contents eval_buf);
  printf "uuendufunctionupr_U_%s@] " name; nl ();
   Buffer.reset\ decl\_buf;
   Buffer.reset eval_buf;
let scale\_coupling \ c \ g =
  if c = 1 then
     g
  \mbox{else if } c \ = \ -1 \ \mbox{then}
      Printf.sprintf "%d*%s" c g
let scale\_coupling z g =
  format\_complex\_rational\_factor\ z\ \hat{\ }g
```

As a prototypical example consider the vertex

$$\bar{\psi} A \psi = \operatorname{tr} \left(\psi \otimes \bar{\psi} A \right) \tag{14.25a}$$

encoded as FFV in the SM UFO file. This example is useful, because all three fields have different type and we can use the Fortran compiler to check our implementation.

In this case we need to generate the following function calls with the arguments in the following order

```
\psi_1\bar{\psi}_2 \rightarrow A FFV_p201(g,psi1,p1,psibar2,p2)
F12:
F21:
       \bar{\psi}_1\psi_2 \to A
                      FFV_p201(g,psi2,p2,psibar1,p1)
       \psi_1 A_2 \to \bar{\psi}
F23:
                      FFV_p012(g,psibar1,p1,A2,p2)
F32:
       A_1\psi_2 \to \psi
                      FFV_p012(g,psibar2,p2,A1,p1)
F31:
       A_1\psi_2 \to \psi
                      FFV_p120(g,A1,p1,psi2,p2)
        \psi_1 A_2 \to \psi
                      FFV_p120(g,A2,p2,psi1,p1)
F13:
```

Fortunately, all Fermi signs have been taken care of by *Fusions* and we can concentrate on injecting the wave functions into the correct slots.

The other possible cases are

$$\bar{\psi}A\psi$$
 (14.25b)

which would be encoded as FVF in a UFO file

```
\begin{array}{lll} \text{F12:} & \bar{\psi}_1 A_2 \rightarrow \bar{\psi} & \text{FVF\_p201(g,psibar1,p1,A2,p2)} \\ \text{F21:} & A_1 \bar{\psi}_2 \rightarrow \bar{\psi} & \text{FVF\_p201(g,psibar2,p2,A1,p1)} \\ \text{F23:} & A_1 \psi_2 \rightarrow \psi & \text{FVF\_p012(g,A1,p1,psi2,p2)} \\ \text{F32:} & \psi_1 A_2 \rightarrow \psi & \text{FVF\_p012(g,A2,p2,psi1,p1)} \\ \text{F31:} & \psi_1 \bar{\psi}_2 \rightarrow A & \text{FVF\_p120(g,psi1,p1,psibar2,p2)} \\ \text{F13:} & \bar{\psi}_1 \psi_2 \rightarrow A & \text{FVF\_p120(g,psi2,p2,psibar1,p1)} \end{array}
```

and

$$\bar{\psi} A \psi = \operatorname{tr} \left(A \psi \otimes \bar{\psi} \right) , \qquad (14.25c)$$

corresponding to VFF

```
A_1\psi_2 \rightarrow \psi VFF_p201(g,A1,p1,psi2,p2)
F12:
       \psi_1 A_2 \to \psi
F21:
                      VFF_p201(g,A2,p2,psi1,p1)
F23:
       \psi_1\psi_2 \to A
                      VFF_p012(g,psi1,p1,psibar2,p2)
F32:
       \bar{\psi}_1\psi_2 \to A
                      VFF_p012(g,psi2,p2,psibar1,p1)
F31:
       \bar{\psi}_1 A_2 \to \bar{\psi}
                      VFF_p120(g,psibar1,p1,A2,p2)
F13:
       A_1\bar{\psi}_2 \to \psi
                      VFF_p120(g,psibar2,p2,A1,p1)
```



Once the Majorana code generation is fully debugged, we should replace the lists by reverted lists everywhere in order to become a bit more efficient.

```
module P = Permutation.Default
let factor\_cyclic f12\_\_n =
  let f12__, fn = ThoList.split_last <math>f12__n in
  let cyclic = ThoList.cycle\_until fn (List.sort compare f12\__n) in
  (P.of_list (List.map pred cyclic),
    P. of _lists (List.tl cyclic) f12__)
let ccs\_to\_string \ ccs =
  String.concat "" (List.map (fun (f, i) \rightarrow Printf.sprintf "_c%x%x" i f) ccs)
let fusion\_name \ v \ perm \ ccs =
  Printf.sprintf "%s_p%s%s" v (P.to\_string\ perm) (ccs\_to\_string\ ccs)
let fuse\_dirac\ c\ v\ s\ fl\ g\ wfs\ ps\ fusion\ =
  let g = scale\_coupling c g
  and cyclic, factor = factor_cyclic fusion in
  let wfs\_ps = List.map2 (fun wf p \rightarrow (wf, p)) wfs ps in
  let args = P.list (P.inverse factor) wfs_ps in
  let args\_string =
     String.concat "," (List.map (fun (wf, p) \rightarrow wf ^ ", "^ p) args) in
  printf "%s(%s,%s)" (fusion_name v cyclic []) g args_string
```

We need to look at the permuted fermion lines in order to decide wether to apply charge conjugations. It is not enough to look at the cyclic permutation used to move the fields into the correct arguments of the fusions . . .

```
let map_indices perm unit =
  let pmap = IntPM.of_lists unit (P.list perm unit) in
  IntPM.apply pmap
```

 \dots we also need to inspect the full permutation of the fields.

```
\begin{array}{lll} \text{let } map\_indices2 \ perm \ unit = \\ & \text{let } pmap = \\ & IntPM.of\_lists \ unit \ (1 \ :: \ P.list \ (P.inverse \ perm) \ (List.tl \ unit)) \ \text{in} \\ & IntPM.apply \ pmap \end{array}
```

This is a more direct implementation of the composition of $map_indices2$ and $map_indices$, that is used in the unit tests.

```
\begin{array}{lll} \text{let } map\_indices\_raw \; fusion \; = \\ & \text{let } unit = \; ThoList.range \; 1 \; (List.length \; fusion) \; \text{in} \\ & \text{let } f12\_\_, \; fn \; = \; ThoList.split\_last \; fusion \; \text{in} \\ & \text{let } fusion \; = \; fn \; :: \; f12\_\_ \; \text{in} \\ & \text{let } map\_index \; = \; IntPM.of\_lists \; fusion \; unit \; \text{in} \\ & IntPM.apply \; map\_index \end{array}
```

Map the fermion line indices in fl according to map_index .

```
let map\_fermion\_lines\ map\_index\ fl = List.map\ (fun\ (i,\ f)\ \rightarrow\ (map\_index\ i,\ map\_index\ f))\ fl
```

Map the fermion line indices in fl according to map_index , but keep a copy of the original.

```
let map\_fermion\_lines2 map\_index fl = List.map (fun (i, f) \rightarrow ((i, f), (map\_index\ i, map\_index\ f))) fl
let permute\_fermion\_lines cyclic unit fl = map\_fermion\_lines (map\_indices\ cyclic\ unit) fl
let permute\_fermion\_lines2 cyclic\ factor\ unit fl = map\_fermion\_lines2 (map\_indices2\ factor\ unit) (map\_fermion\_lines\ (map\_indices\ cyclic\ unit)\ fl)
```



TODO: this needs more more work for the fully general case with 4-fermion operators involving Majoranas.

```
let charge_conjugations fl2 =
   Tho List. filter map
     (\text{fun }((i, f), (i', f')) \rightarrow
        match (i, f), (i', f') with
          (1, 2), - (2, 1), - \rightarrow Some (f, i) (* \chi^T \Gamma' *)
          (2, 3) \rightarrow Some (f, i) (* \chi^T(C\Gamma')\chi *)
          \rightarrow None
     fl2
let charge_conjugations fl2 =
   ThoList.filtermap
     (\text{fun }((i, f), (i', f')) \rightarrow
        match (i, f), (i', f') with
          (2, 3) \rightarrow Some(f, i)
          \rightarrow None
     fl2
let fuse\_majorana \ c \ v \ s \ fl \ g \ wfs \ ps \ fusion =
  let q = scale\_coupling c q
  and cyclic, factor = factor\_cyclic fusion in
  let wfs\_ps = List.map2 (fun wf p \rightarrow (wf, p)) wfs ps in
  let args = P.list (P.inverse factor) wfs_ps in
  let args\_string =
     String.concat "," (List.map (fun (wf, p) \rightarrow wf \hat{} "," \hat{} p) args) in
  let unit = ThoList.range 1 (List.length fusion) in
  let \ ccs =
     charge_conjugations (permute_fermion_lines2 cyclic factor unit fl) in
  printf "%s(%s,%s)" (fusion_name v cyclic ccs) g args_string
let fuse \ c \ v \ s \ fl \ g \ wfs \ ps \ fusion =
```

```
if List.exists is_majorana s then
                   fuse_majorana c v s fl q wfs ps fusion
         else
                    fuse_dirac c v s fl g wfs ps fusion
let eps4\_g4\_g44\_decl \ ff \ () =
         let printf fmt = fprintf ff fmt
         and nl = pp\_newline ff in
         printf "_{\sqcup \sqcup} @[<2> integer, @_{\sqcup} dimension(0:3)";
         printf ", @_{\square}save, @_{\square}private_{\square}:: @_{\square}g4_{-}@] "; nl ();
         printf "_{\sqcup \sqcup} @[<2> integer, @_{\sqcup} dimension(0:3,0:3)";
         printf ", @_{\square}save, @_{\square}private_{\square}:: @_{\square}g44_{\square}g] "; nl ();
         printf "_{\sqcup\sqcup} @[<2> integer, @_{\sqcup} dimension(0:3,0:3,0:3,0:3)";
         printf ", @_{\square}save, @_{\square}private_{\square}:: @_{\square}eps4_@] "; nl ()
let eps4\_g4\_g44\_init ff () =
         let printf fmt = fprintf ff fmt
         and nl = pp\_newline ff in
         printf ~" \_ \_ @ [<2> data \_ g4 \_ @ \_ \_ \_ \_ / @ \_ \_ 1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1, \_ -1
         printf "_{UU}@[<2> data_Ug44_(1,:)@_{UU}U_UU}/@_{UU}0,_{U}-1,_{UU}0,_{UU}0_U/@]"; nl ();
         printf "_{UU}@[<2>data_{U}g44_{-}(2,:)@_{UU}U_{UU}U_{U}U_{U}, 0_{UU}0_{UU}0_{U}U_{U}0_{U}U_{U}]"; nl ();
         printf "_{ \sqcup \sqcup } @[<2> data_{ \sqcup } g44_{ \sqcup } (3,:) @_{ \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup } / @_{ \sqcup \sqcup } 0,_{ \sqcup \sqcup } 0,_{ \sqcup \sqcup } 0,_{ \sqcup \sqcup } 0 ] "; nl ();
         for mu1 = 0 to 3 do
                    for mu2 = 0 to 3 do
                             for mu3 = 0 to 3 do
                                        printf "_{\sqcup\sqcup} @[<2> data_{\sqcup} eps4_(%d,%d,%d,:)@_{\sqcup}/@_{\sqcup}" mu1 mu2 mu3;
                                       for mu4 = 0 to 3 do
                                                if mu4 \neq 0 then
                                                           printf ", @_{\sqcup}";
                                                let mus = [mu1; mu2; mu3; mu4] in
                                                if List.sort compare mus = [0; 1; 2; 3] then
                                                          printf "%2d" (Combinatorics.sign mus)
                                                else
                                                          printf "%2d" 0;
                                       done:
                                        printf "□/@]";
                                        nl()
                             done
                    done
         done
let \ \mathit{inner\_product\_functions} \ \mathit{ff} \ () \ =
         let printf fmt = fprintf ff fmt
         and nl = pp\_newline ff in
         printf "_{\sqcup \sqcup} pure_{\sqcup} function_{\sqcup} g2_{-\sqcup}(p)_{\sqcup} result_{\sqcup}(p2) "; nl();
         \textit{printf} ~\texttt{"} \\ \texttt{\_} \\ \texttt{\_} \\ \texttt{real(kind=default),} \\ \texttt{\_} \\ \texttt{dimension(0:3),} \\ \texttt{\_} \\ \texttt{intent(in)} \\ \texttt{\_} \\ \texttt{:} \\ \texttt{\_} \\ \texttt{p"}; ~nl(); \\ \texttt{:} \\ \texttt{\_} \\ \texttt{p"}; ~nl(); \\ \texttt{:} \\ \texttt{\_} \\ \texttt{\_} \\ \texttt{p"}; ~nl(); \\ \texttt{:} \\ \texttt{\_} \\ \texttt{\_} \\ \texttt{p"}; ~nl(); \\ \texttt{:} \\ \texttt{\_} \\ \texttt{\_
         printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup} real(kind=default)_{\sqcup} :: _{\sqcup}p2"; nl();
         printf "_{\Box\Box\Box\Box}p2_{\Box} =_{\Box}p(0)*p(0)_{\Box} -_{\Box}p(1)*p(1)_{\Box} -_{\Box}p(2)*p(2)_{\Box} -_{\Box}p(3)*p(3)"; nl();
         printf "_uend_function_g2_"; nl();
         printf "_{\sqcup\sqcup}pure_{\sqcup}function_{\sqcup}g12_{-\sqcup}(p1,_{\sqcup}p2)_{\sqcup}result_{\sqcup}(p12)"; nl();
         printf "_{\square\square\square\square} real(kind=default), \_dimension(0:3), \_intent(in)_{\square}: \_p1, \_p2"; nl();
         printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup} real(kind=default)_{\sqcup}: _{\sqcup}p12"; nl();
         printf "_{\Box\Box\Box\Box} p12_{\Box} = _{\Box} p1(0) * p2(0)_{\Box} - _{\Box} p1(1) * p2(1)_{\Box} - _{\Box} p1(2) * p2(2)_{\Box} - _{\Box} p1(3) * p2(3) "; nl();
         printf "\sqcup \sqcupend\sqcupfunction\sqcupg12\_"; nl()
module type Test =
         sig
                   val suite : OUnit.test
         end
module Test : Test =
         struct
```

open OUnit ${\tt let} \ assert_mappings \ fusion \ =$ $let \ unit = ThoList.range \ 1 \ (List.length \ fusion) \ in$ $let \ cyclic, \ factor = factor_cyclic \ fusion \ in$ let $raw = map_indices_raw fusion$ and $map1 = map_indices \ cyclic \ unit$ and $map2 = map_indices2 \ factor \ unit$ in let map i = map2 (map1 i) in assert_equal ~printer: (ThoList.to_string string_of_int) (List.map raw unit) (List.map map unit) $let suite_mappings =$ "mappings" >::: ["1<-2" >:: $(fun () \rightarrow$ List.iter assert_mappings (Combinatorics.permute [1; 2; 3])); "1<-3" >:: $(\mathsf{fun}\ ()\ \to$ List.iter assert_mappings (Combinatorics.permute [1; 2; 3; 4])) let suite ="UFO_targets" >::: $[suite_mappings]$ end end

-15-

HARDCODED TARGETS

15.1 Interface of Format_Fortran

Mimic parts of the Format API with support for Fortran style line continuation.

```
type formatter
val\ std\_formatter : formatter
val fprintf: formatter \rightarrow (\alpha, Format.formatter, unit) format \rightarrow \alpha
val printf: (\alpha, Format.formatter, unit) format \rightarrow \alpha
Start a new line, not a continuation!
val pp\_newline : formatter \rightarrow unit \rightarrow unit
val\ newline : unit \rightarrow unit
\mathsf{val}\ pp\_\mathit{flush}\ :\ formatter\ \to\ unit\ \to\ unit
\mathsf{val}\ \mathit{flush}\ :\ \mathit{unit}\ \to\ \mathit{unit}
val\ formatter\_of\_out\_channel: ?width:int \rightarrow out\_channel \rightarrow formatter
val\ formatter\_of\_buffer: ?width:int \rightarrow Buffer.t \rightarrow formatter
\verb|val|| pp\_set\_formatter\_out\_channel|: formatter| \rightarrow ?width: int \rightarrow out\_channel| \rightarrow unit|
val\ set\_formatter\_out\_channel\ :\ ?width:int 
ightarrow\ out\_channel\ 
ightarrow\ unit
This must be exposed for the benefit of Targets. Make_Fortran().print_interface, because somebody decided to
use it for the K-matrix support. Is this really necessary?
val pp\_switch\_line\_continuation : formatter \rightarrow bool \rightarrow unit
val\ switch\_line\_continuation\ :\ bool 
ightarrow\ unit
module \ Test : sig \ val \ suite : OUnit.test \ end
```

15.2 Implementation of Format_Fortran

```
let default_width = 80
let max_clines = ref (-1) (* 255 *)
exception Continuation_Lines of int
Fortran style line continuation:
type formatter =
    { formatter : Format.formatter;
    mutable current_cline : int;
    mutable width : int }
let formatter_of_formatter ?(width = default_width) ff =
    { formatter = ff;
        current_cline = 1;
        width = width }
Default function to output new lines.
let pp_output_function ff =
```

```
fst (Format.pp_get_formatter_output_functions ff.formatter ())
Default function to output spaces (copied from format.ml).
let blank\_line = String.make 80 , ,
let rec pp\_display\_blanks ff n =
  if n > 0 then
    if n \leq 80 then
       pp\_output\_function \ ff \ blank\_line \ 0 \ n
    else begin
         pp_output_function ff blank_line 0 80;
         pp\_display\_blanks ff (n - 80)
       end
let pp\_display\_newline ff =
  pp\_output\_function ff "\n" 0 1
ff.current\_cline
   • \leq 0: not continuing: print a straight newline,
   • > 0: continuing: append "L\dark" until we run up to !max_clines. NB: !max_clines < 0 means unlimited
      continuation lines.
let pp\_switch\_line\_continuation ff = function
    false \rightarrow ff.current\_cline \leftarrow 0
    true \rightarrow ff.current\_cline \leftarrow 1
let pp\_fortran\_newline\ ff\ () =
  if ff.current\_cline > 0 then
    begin
       if !max\_clines \ge 0 \land ff.current\_cline > !max\_clines then
         raise (Continuation_Lines ff.current_cline)
       else
         begin
            pp\_output\_function ff "$\ldot\&\" 0 2;
            ff.current\_cline \leftarrow succ ff.current\_cline
         end
    end:
  pp\_display\_newline ff
let pp\_newline ff() =
  pp_switch_line_continuation ff false;
  Format.pp_print_newline ff.formatter ();
  pp_switch_line_continuation ff true
Make a formatter with default functions to output spaces and new lines.
let pp\_setup ff =
  let formatter\_out\_functions =
     Format.pp\_get\_formatter\_out\_functions\ ff.formatter\ ()\ in
  Format.pp\_set\_formatter\_out\_functions
    ff.formatter
    { formatter_out_functions with
       Format.out\_newline = pp\_fortran\_newline ff;
       Format.out\_spaces = pp\_display\_blanks ff };
  Format.pp\_set\_margin\ ff.formatter\ (ff.width\ -\ 2)
let std\_formatter =
  \mathsf{let}\ \mathit{ff}\ =\ \mathit{formatter\_of\_formatter}\ \mathit{Format.std\_formatter}\ \mathsf{in}
  pp\_setup ff;
let formatter_of_out_channel ?(width = default_width) oc =
  let ff = formatter\_of\_formatter\~vidth (Format.formatter\_of\_out\_channel oc) in
  pp\_setup ff;
  ff
```

```
let formatter\_of\_buffer ?(width = default\_width) b =
  let ff =
     \{ formatter = Format.formatter\_of\_buffer b; \}
       current\_cline = 1;
       width = width \} in
  pp\_setup ff;
  ff
let pp\_set\_formatter\_out\_channel ff ?(width = default\_width) oc =
  Format.pp_set_formatter_out_channel ff.formatter oc;
  ff.width \leftarrow width;
  pp\_setup ff
let set_formatter_out_channel ?(width = default_width) oc =
  Format.pp_set_formatter_out_channel std_formatter.formatter oc;
  std\_formatter.width \leftarrow width;
  pp\_setup\ std\_formatter
let \ fprintf \ ff \ fmt = Format.fprintf \ ff.formatter \ fmt
let pp\_flush ff = Format.pp\_print\_flush ff.formatter
let printf fmt = fprintf std\_formatter fmt
let \ newline \ = \ pp\_newline \ std\_formatter
\mathsf{let}\ \mathit{flush}\ =\ \mathit{pp\_flush}\ \mathit{std\_formatter}
let switch_line_continuation = pp_switch_line_continuation std_formatter
module Test =
  struct
    open OUnit
    let input\_line\_opt ic =
       try
          Some (input\_line ic)
       with
       \mid End\_of\_file \rightarrow None
    let read\_lines ic =
       let rec read\_lines' acc =
          match input\_line\_opt ic with
            Some line \rightarrow read_lines' (line :: acc)
           None \rightarrow List.rev \ acc
       in
       read_lines' []
     let lines_of_file filename =
       \mathsf{let}\ ic\ =\ open\_in\ filename\ \mathsf{in}
       let \ lines = read\_lines \ ic \ in
       close\_in\ ic;
       lines
    let equal_or_dump_lines lhs rhs =
       if lhs = rhs then
          true
       else
          begin
            Printf.printf "Unexpected output: \n";
            List.iter\ (Printf.printf\ "< \ld %s\n")\ lhs;
            List.iter\ (Printf.printf "> \sqcup %s \n")\ rhs;
            false
          end
    let format\_and\_compare f expected() =
       bracket_tmpfile
          \~prefix:"omega-"\~suffix:".f90"
          (fun (name, oc) \rightarrow
            (* There can be something left in the queue from OUnit! *)
```

```
Format.print_flush ();
         f \ oc;
         close\_out\ oc;
          (* OUnit uses Format.printf! *)
         Format.set_formatter_out_channel stdout;
          assert_bool "" (equal_or_dump_lines expected (lines_of_file name)))
      ()
let suite =
   "Format_Fortran" >:::
      [ "formatter_of_out_channel" >::
             format_and_compare
                (fun oc \rightarrow
                   let ff = formatter\_of\_out\_channel ~width: 20 oc in
                   let nl = pp\_newline ff in
                   List.iter
                      (fprintf ff)
                       ["@[<2>lhs_{\sqcup}=_{\sqcup}rhs";
                        "@_{\sqcup}+_{\sqcup}rhs"; "@_{\sqcup}+_{\sqcup}rhs"; "@_{\sqcup}+_{\sqcup}rhs"; "@_{\sqcup}+_{\sqcup}rhs"; "@_{\sqcup}+_{\sqcup}rhs";
                        "@_+_rhs"; "@_+_rhs"; "@_+_rhs"; "@_+_rhs"; "@_+_rhs"];
                   nl())
                ["lhs⊔=⊔rhs⊔+⊔rhs⊔&";
                   "___+_rhs__+_rhs_&";
                   "___+_rhs__+_rhs_&"
                   "_{\sqcup \sqcup} +_{\sqcup} rhs_{\sqcup} +_{\sqcup} rhs_{\sqcup} \&";
                   "_{\sqcup \sqcup} +_{\sqcup} rhs_{\sqcup} +_{\sqcup} rhs_{\sqcup} \&";
                   "___+_rhs" ];
          "formatter_of_buffer" >::
             format\_and\_compare
                (fun oc \rightarrow
                   let \ buffer = Buffer.create \ 1024 \ in
                   \mathsf{let}\ \mathit{ff}\ =\ \mathit{formatter\_of\_buffer}\ \tilde{\ \ }\mathit{width}: 20\ \mathit{buffer}\ \mathsf{in}
                   let \ nl \ = \ pp\_newline \ f\!\!f \ in
                   List.iter
                      (fprintf ff)
                      ["_{\sqcup\sqcup}@[<2>1hs_{\sqcup}=_{\sqcup}rhs";
                        "@||+||rhs"; "@||+||rhs"; "@||+||rhs"; "@||+||rhs"; "@||+||rhs";
                        "@_{\sqcup} +_{\sqcup} rhs"; \; "@_{\sqcup} +_{\sqcup} rhs"; \; "@_{\sqcup} +_{\sqcup} rhs"; \; "@_{\sqcup} +_{\sqcup} rhs"; \; "@_{\sqcup} +_{\sqcup} rhs"];
                   nl();
                   pp_{-}flush ff ();
                   let ff' = formatter\_of\_out\_channel ~width : 20 oc in
                   fprintf ff' "do_{\square}mu_{\square} = \square 0, \square 3"; pp_newline ff'();
                   fprintf ff' "%s" (Buffer.contents buffer);
                   fprintf ff' "end

do";
                   pp\_newline ff'()
                  "do_{\square}mu_{\square}=_{\square}0,_{\square}3";
                   "_{\sqcup\sqcup}lhs_{\sqcup}=_{\sqcup}rhs_{\sqcup}+_{\sqcup}rhs_{\sqcup}&";
                   "uuuu+urhsu+urhsu&";
                   "uuuu+urhsu+urhsu&";
                   "____+_rhs__+_rhs__&";
                   "_{ \sqcup \sqcup \sqcup \sqcup \sqcup} +_{ \sqcup} rhs_{ \sqcup} +_{ \sqcup} rhs_{ \sqcup} \&";
                   "uuuu+urhs";
                   "end<sub>□</sub>do"];
          "formatter_of_out_channel+indentation" >::
             format\_and\_compare
                (fun oc \rightarrow
                   let ff = formatter\_of\_out\_channel ~width: 20 oc in
                   let nl = pp\_newline ff in
                   List.iter
                      (fprintf ff)
                       ["_{\sqcup\sqcup}@[<4>1hs_{\sqcup}=_{\sqcup}rhs";
```

```
"@_{\sqcup}+_{\sqcup}rhs"; "@_{\sqcup}+_{\sqcup}rhs"; "@_{\sqcup}+_{\sqcup}rhs"; "@_{\sqcup}+_{\sqcup}rhs"; "@_{\sqcup}+_{\sqcup}rhs";
                 "@_+_rhs"; "@_+_rhs"; "@_+_rhs"; "@_+_rhs"; "@_+_rhs"];
           nl())
       [ "uulhsu=urhsu+urhsu&";
             '____+_rhs_+_rhs_&";
           "_{ \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} +_{ \sqcup} rhs_{ \sqcup} +_{ \sqcup} rhs_{ \sqcup} \&";
           "_{ \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} +_{ \sqcup} rhs_{ \sqcup} +_{ \sqcup} rhs_{ \sqcup} \&";
           "_{ \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} +_{ \sqcup} rhs_{ \sqcup} +_{ \sqcup} rhs_{ \sqcup} \&";
           "____+_rhs" ];
"set_formatter_out_channel" >::
   format\_and\_compare
       (fun oc \rightarrow
           \mathsf{let}\ nl\ =\ newline\ \mathsf{in}
           set\_formatter\_out\_channel \ \tilde{\ } width: 20\ oc;
               printf
                ["@[<2>1hs_{\sqcup}=_{\sqcup}rhs";
                 "@_{\sqcup}+_{\sqcup}rhs"; "@_{\sqcup}+_{\sqcup}rhs"; "@_{\sqcup}+_{\sqcup}rhs"; "@_{\sqcup}+_{\sqcup}rhs"; "@_{\sqcup}+_{\sqcup}rhs";
                 "@_+_rhs"; "@_+_rhs"; "@_+_rhs"; "@_+_rhs"; "@_+_rhs"];
           nl())
        [ "lhs_=urhs_+urhs_&";
           "___+_rhs__+_rhs_&";
           "___+_rhs__+_rhs__&";
           "_{\sqcup \sqcup} +_{\sqcup} rhs_{\sqcup} +_{\sqcup} rhs_{\sqcup} \&";
           "_{\sqcup \sqcup} +_{\sqcup} rhs_{\sqcup} +_{\sqcup} rhs_{\sqcup} \&";
           "___+_rhs" ]; ]
```

end

15.3 Interface of Targets

 $module\ Dummy\ :\ Target.Maker$

15.3.1 Supported Targets

module Fortran : Target.Maker

 $module\ Fortran_Majorana\ :\ Target.Maker$

 ${\sf module}\ VM\ :\ Target.Maker$

15.3.2 Potential Targets

```
\begin{array}{lll} \operatorname{module} \ Fortran 77 \ : \ Target.Maker \\ \operatorname{module} \ C \ : \ Target.Maker \\ \operatorname{module} \ Cpp \ : \ Target.Maker \\ \operatorname{module} \ Java \ : \ Target.Maker \\ \operatorname{module} \ Ocaml \ : \ Target.Maker \\ \operatorname{module} \ LaTeX \ : \ Target.Maker \\ \end{array}
```

15.4 Implementation of Targets

15.4.1 O'Mega Virtual Machine with Fortran 90/95

Preliminaries

```
module\ VM\ (Fusion\_Maker\ :\ Fusion.Maker)\ (P\ :\ Momentum.T)\ (M\ :\ Model.T)\ =
   open Coupling
   open Format
    module CM = Colorize.It(M)
    module F = Fusion\_Maker(P)(M)
    module \ CF = Fusion.Multi(Fusion\_Maker)(P)(M)
    module \ CFlow = Color.Flow
    type \ amplitudes = CF.amplitudes
Options.
    type diagnostic = All | Arguments | Momenta | Gauge
    let wrapper_module = ref "ovm_wrapper"
    let parameter_module_external = ref "some_external_module_with_model_info"
   let bytecode\_file = ref "bytecode.hbc"
   let md5sum = ref None
    let open mp = ref false
   let kind = ref "default"
   let whizard = ref false
    let options = Options.create
      ["wrapper_module", Arg.String (fun s \rightarrow wrapper_module := s),
          "name_of_wrapper_module";
        "bytecode_file", Arg.String (fun s \rightarrow bytecode\_file := s),
          "bytecode_file_to_be_used_in_wrapper";
        "parameter_module_external", Arg.String (fun s 
ightarrow
                                     parameter\_module\_external := s),
          "external_parameter_module_to_be_used_in_wrapper";
        "md5sum", Arg.String (fun s \rightarrow md5sum := Some s),
          "transfer⊔MD5⊔checksum⊔in⊔wrapper";
        "whizard", Arg.Set\ whizard, "include_WHIZARD_interface_in_wrapper";
        "openmp", Arg. Set openmp,
          "activate_parallel_computation_of_amplitude_with_OpenMP"]
Integers encode the opcodes (operation codes).
    let ovm\_ADD\_MOMENTA = 1
    let ovm\_CALC\_BRAKET = 2
    let ovm\_LOAD\_SCALAR = 10
   let ovm\_LOAD\_SPINOR\_INC = 11
   let ovm\_LOAD\_SPINOR\_OUT = 12
    let ovm\_LOAD\_CONJSPINOR\_INC = 13
   let ovm\_LOAD\_CONJSPINOR\_OUT = 14
   \mathsf{let}\ ovm\_LOAD\_MAJORANA\_INC\ =\ 15
   let ovm\_LOAD\_MAJORANA\_OUT = 16
   let ovm\_LOAD\_VECTOR\_INC = 17
   let ovm\_LOAD\_VECTOR\_OUT = 18
   let ovm\_LOAD\_VECTORSPINOR\_INC = 19
   let ovm\_LOAD\_VECTORSPINOR\_OUT = 20
   let ovm\_LOAD\_TENSOR2\_INC = 21
   let ovm\_LOAD\_TENSOR2\_OUT = 22
   let ovm\_LOAD\_BRS\_SCALAR = 30
    let ovm\_LOAD\_BRS\_SPINOR\_INC = 31
   let ovm\_LOAD\_BRS\_SPINOR\_OUT = 32
   let ovm\_LOAD\_BRS\_CONJSPINOR\_INC = 33
   let ovm\_LOAD\_BRS\_CONJSPINOR\_OUT = 34
```

```
let ovm\_LOAD\_BRS\_VECTOR\_INC = 37
let ovm\_LOAD\_BRS\_VECTOR\_OUT = 38
let ovm\_LOAD\_MAJORANA\_GHOST\_INC = 23
let ovm\_LOAD\_MAJORANA\_GHOST\_OUT = 24
let ovm\_LOAD\_BRS\_MAJORANA\_INC = 35
let ovm_LOAD_BRS_MAJORANA_OUT = 36
let ovm_PROPAGATE_SCALAR = 51
let ovm_PROPAGATE_COL_SCALAR = 52
let ovm\_PROPAGATE\_GHOST = 53
let ovm_PROPAGATE_SPINOR = 54
let ovm_PROPAGATE_CONJSPINOR = 55
let ovm_PROPAGATE_MAJORANA = 56
let ovm\_PROPAGATE\_COL\_MAJORANA = 57
let ovm_PROPAGATE_UNITARITY = 58
let ovm_PROPAGATE_COL_UNITARITY = 59
let ovm_PROPAGATE_FEYNMAN = 60
let ovm_PROPAGATE_COL_FEYNMAN = 61
let ovm_PROPAGATE_VECTORSPINOR = 62
let ovm\_PROPAGATE\_TENSOR2 = 63
```



 $ovm_PROPAGATE_NONE$ has to be split up to different types to work in conjunction with color MC ...

```
let ovm_PROPAGATE_NONE = 64
let ovm\_FUSE\_V\_FF = -1
let ovm_FUSE_F_VF = -2
let ovm_FUSE_F_FV = -3
let ovm_FUSE_VA_FF =
let ovm_FUSE_F_VAF =
let ovm_FUSE_F_FVA = -6
let ovm_FUSE_VA2_FF = -7
let ovm_FUSE_F_VA2F = -8
let ovm_FUSE_F_FVA2 = -9
let ovm_FUSE_A_FF = -10
let ovm_FUSE_F_AF = -11
let ovm_FUSE_F_FA = -12
let ovm_FUSE_VL_FF = -13
let ovm_FUSE_F_VLF = -14
let ovm_FUSE_F_FVL = -15
let ovm_FUSE_VR_FF = -16
let ovm_FUSE_F_VRF = -17
let ovm_FUSE_F_FVR = -18
let ovm_FUSE_VLR_FF = -19
let ovm_FUSE_F_VLRF = -20
let ovm_FUSE_F_FVLR = -21
let ovm\_FUSE\_SP\_FF = -22
let ovm_FUSE_F_SPF = -23
let ovm_FUSE_FFSP = -24
let ovm\_FUSE\_S\_FF = -25
let ovm\_FUSE\_F\_SF = -26
let ovm_FUSE_FFS = -27
let ovm_FUSE_P_FF = -28
let ovm_FUSE_F_PF = -29
let ovm\_FUSE\_F\_FP = -30
let ovm_FUSE_SL_FF = -31
let ovm_FUSE_F_SLF = -32
let ovm_FUSE_F_FSL = -33
let ovm_FUSE_SR_FF = -34
let ovm_FUSE_F_SRF = -35
let ovm_FUSE_FFSR = -36
let ovm_FUSE_SLR_FF = -37
```

```
let ovm_FUSE_F_SLRF = -38
    let ovm_FUSE_F_FSLR = -39
    let ovm_FUSE_GGG = -40
    let ovm\_FUSE\_V\_SS = -41
    \label{eq:constraint} \text{let } \textit{ovm}\_\textit{FUSE}\_\textit{S}\_\textit{VV} \ = \ -42
    let ovm_FUSE_S_VS = -43
    let ovm_FUSE_V_SV = -44
    let ovm_FUSE_SS = -45
    let ovm_FUSE_S_SVV = -46
    let ovm_FUSE_V_SSV = -47
    let ovm_FUSE_SSSS = -48
    let ovm_FUSE_V_VVV = -49
    let ovm_FUSE_S_G2 = -50
    let ovm\_FUSE\_G\_SG = -51
    let ovm_FUSE_GG_S = -52
    let ovm_FUSE_S_G2_SKEW = -53
    let ovm_FUSE_G_SG_SKEW = -54
    let ovm\_FUSE\_G\_GS\_SKEW = -55
    let inst_length = 8
Some helper functions.
    let printi \ \ "lhs: l \ "rhs1: r1 \ ?coupl: (cp = 0) \ ?coeff: (co = 0)
                ?rhs2:(r2 = 0) ?rhs3:(r3 = 0) ?rhs4:(r4 = 0) code =
      let nl () = printf "@\n"
    |\text{let } print\_int\_lst \ lst = nl \ (); \ lst \ | > List.iter \ (printf \ "%dull")
    let print\_str\_lst\ lst\ =\ nl\ ();\ lst\ |>\ List.iter\ (printf\ "%s_\")
    let break() = printi \ ^{\sim} lhs : 0 \ ^{\sim} rhs1 : 0 \ 0
Copied from below. Needed for header.
   Could be fused with lorentz_ordering.
    type declarations =
      \{ scalars : F.wf list; \}
        spinors : F.wf list;
         conjspinors : F.wf list;
        realspinors : F.wf list;
         ghostspinors : F.wf list;
         vectorspinors : F.wf list;
         vectors : F.wf list;
         ward\_vectors : F.wf list;
         massive\_vectors : F.wf list;
         tensors_1 : F.wf list;
         tensors \_2 : F.wf list;
         brs_scalars : F.wf list;
         brs\_spinors : F.wf\ list;
         brs\_conjspinors : F.wf list;
         brs\_realspinors : F.wf list;
         brs\_vectorspinors : F.wf list;
         brs\_vectors : F.wf list;
         brs_massive_vectors : F.wf list }
    let rec classify\_wfs' acc = function
      | [] \rightarrow acc
      | wf :: rest \rightarrow
           classify\_wfs'
             (match CM.lorentz (F.flavor wf) with
             Scalar \rightarrow \{acc \text{ with } scalars = wf :: acc.scalars\}
```

```
Spinor \rightarrow \{acc \text{ with } spinors = wf :: acc.spinors\}
                          ConjSpinor \rightarrow \{acc \text{ with } conjspinors = wf :: acc.conjspinors\}
                          Majorana \rightarrow \{acc \text{ with } realspinors = wf :: acc.realspinors\}
                          Maj\_Ghost \rightarrow \{acc \text{ with } ghostspinors = wf :: acc.ghostspinors\}
                           Vectorspinor \rightarrow
                                 \{acc \text{ with } vectorspinors = wf :: acc.vectorspinors\}
                           Vector \rightarrow \{acc \text{ with } vectors = wf :: acc.vectors\}
                          Massive\_Vector \rightarrow
                                 \{acc \text{ with } massive\_vectors = wf :: acc.massive\_vectors\}
                          Tensor_1 \rightarrow \{acc \text{ with } tensors_1 = wf :: acc.tensors_1\}
                          Tensor_2 \rightarrow \{acc \text{ with } tensors_2 = wf :: acc.tensors_2\}
                          BRS\ Scalar \rightarrow \{acc\ with\ brs\_scalars = wf :: acc.brs\_scalars\}
                          BRS \ Spinor \rightarrow \{acc \ with \ brs\_spinors = wf :: acc.brs\_spinors\}
                          BRS\ ConjSpinor\ 	o\ \{acc\ with\ brs\_conjspinors\ =
                                                                              wf :: acc.brs\_conjspinors
                         BRS\ Majorana\ 	o\ \{acc\ {\it with}\ brs\_realspinors\ =
                                                                       wf :: acc.brs\_realspinors
                         BRS\ Vectorspinor\ 	o\ \{acc\ {\it with}\ brs\_vectorspinors\ =
                                                                                   wf :: acc.brs\_vectorspinors
                          BRS\ Vector \rightarrow \{acc\ with\ brs\_vectors = wf :: acc.brs\_vectors\}
                          BRS\ Massive\_Vector \rightarrow \{acc\ with\ brs\_massive\_vectors =
                                                                                         wf :: acc.brs\_massive\_vectors
                      \mid BRS \_ \rightarrow invalid\_arg  "Targets.classify_wfs':\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandinged\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandinged\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandingeded\understandinged\understandingeded\understandingeded\understandingeded\unde
                      rest
let classify\_wfs wfs = classify\_wfs'
      \{ scalars = [];
           spinors = [];
           conjspinors = [];
           realspinors = [];
           ghostspinors = [];
           vectorspinors = [];
           vectors = [];
           ward\_vectors = [];
           massive\_vectors = [];
           tensors_1 = [];
           tensors_2 = [];
           brs\_scalars = [];
           brs\_spinors = [];
           brs\_conjspinors = [];
           brs\_realspinors = [];
           brs\_vectorspinors = [];
           brs\_vectors = [];
           brs\_massive\_vectors = []  wfs
```

Sets and maps

The OVM identifies all objects via integers. Therefore, we need maps which assign the abstract object a unique ID.

I want *int lists* with less elements to come first. Used in conjunction with the int list representation of momenta, this will set the outer particles at first position and allows the OVM to set them without further instructions.



Using the Momentum module might give better performance than integer lists?

```
let rec int\_lst\_compare (e1 : int\ list) (e2 : int\ list) = match e1, e2 with  |\ [],\ [] \to 0 \\ |\ \_,\ [] \to +1 \\ |\ [],\ \_ \to -1 \\ |\ [\_;\ \_],\ [\_] \to +1 \\ |\ [\_],\ [\_;\ \_] \to -1 \\ |\ hd1 :: tl1,\ hd2 :: tl2 \to
```

```
\begin{array}{lll} \text{let } c &=& compare \ hd1 \ hd2 \ \text{in} \\ \text{if } (c &\not\equiv 0 \ \land \ List.length \ tl1 \ = \ List.length \ tl2) \ \text{then} \\ c &=& \text{else} \\ &int\_lst\_compare \ tl1 \ tl2 \end{array}
```

We need a canonical ordering for the different types of wfs. Copied, and slightly modified to order wfs, from fusion.ml.

```
let lorentz\_ordering \ wf =
  match CM.lorentz (F.flavor wf) with
     Scalar \rightarrow 0
     Spinor \rightarrow 1
     ConjSpinor \rightarrow 2
     Majorana \rightarrow 3
     Vector \rightarrow 4
     Massive\_Vector \rightarrow 5
     Tensor\_2 \rightarrow 6
     Tensor_1 \rightarrow 7
     Vectorspinor \rightarrow 8
     BRS\ Scalar\ 	o\ 9
     BRS\ Spinor\ 	o\ 10
     BRS\ ConjSpinor\ 	o\ 11
     BRS\ Majorana\ 	o\ 12
     BRS\ Vector\ 	o\ 13
     BRS\ Massive\_Vector \rightarrow 14
    BRS\ Tensor\_2\ 	o\ 15
    BRS\ Tensor\_1\ 	o\ 16
    BRS\ Vectorspinor \rightarrow 17
    Maj\_Ghost \rightarrow invalid\_arg "lorentz_ordering:_\underline_implemented"
    BRS \rightarrow invalid\_arg "lorentz_ordering:_not_needed"
let wf\_compare\ (wf1,\ mult1)\ (wf2,\ mult2)\ =
  let c1 = compare (lorentz\_ordering wf1) (lorentz\_ordering wf2) in
  if c1 \neq 0 then
     c1
  else
     let c2 = compare \ wf1 \ wf2 in
     if c2 \neq 0 then
       c2
     else
       compare mult1 mult2
let amp\_compare \ amp1 \ amp2 =
  let cflow \ a = CM.flow \ (F.incoming \ a) \ (F.outgoing \ a) in
  let c1 = compare (cflow amp1) (cflow amp2) in
  if c1 \neq 0 then
     c1
  else
     let process\_sans\_color a =
       (List.map CM.flavor\_sans\_color (F.incoming a),
         List.map\ CM.flavor\_sans\_color\ (F.outgoing\ a)) in
     compare (process_sans_color amp1) (process_sans_color amp2)
let level\_compare\ (f1,\ amp1)\ (f2,\ amp2)\ =
  let p1 = F.momentum\_list (F.lhs f1)
  and p2 = F.momentum\_list (F.lhs f2) in
  \mathsf{let}\ c1\ =\ int\_lst\_compare\ p1\ p2\ \mathsf{in}
  if c1 \neq 0 then
     c1
  else
     let c2 = compare f1 f2 in
     if c2 \neq 0 then
       c2
```

```
else amp\_compare\ amp1\ amp2

module ISet = Set.Make (struct type t = int\ list let compare = int\_lst\_compare end)

module WFSet = Set.Make (struct type t = CF.wf \times int let compare = wf\_compare end)

module CSet = Set.Make (struct type t = CM.constant let compare = compare end)

module FSet = Set.Make (struct type t = F.fusion \times F.amplitude let compare = level\_compare end)
```



It might be preferable to use a PMap which maps mom to int, instead of this way. More standard functions like mem could be used. Also, get_ID would be faster, $\mathcal{O}(\log N)$ instead of $\mathcal{O}(N)$, and simpler. For 8 gluons: N=127 momenta. Minor performance issue.

```
module IMap = Map.Make (struct type t = int let compare = compare end)
```

For wfs it is crucial for the performance to use a different type of Maps.

```
 \begin{tabular}{ll} {\bf module} \begin{tabular}{ll} WFMap &= Map.Make (struct type $t = CF.wf \times int$ \\ &= let $compare = wf\_compare$ end) \\ \hline \\ {\bf type} \begin{tabular}{ll} lookups &= \{pmap : int \begin{tabular}{ll} WFMap.t; \\ wfmap : int \begin{tabular}{ll} WFMap.t; \\ cmap : CM.constant \begin{tabular}{ll} IMap.t \times CM.constant \begin{tabular}{ll} IMap.t; \\ amap : F.amplitude \begin{tabular}{ll} IMap.t; \\ n\_wfs : int \begin{tabular}{ll} list; \\ amplitudes : CF.amplitudes; \\ dict : F.amplitude \rightarrow F.wf \rightarrow int \begin{tabular}{ll} let \begin{tabular}{ll} largest\_key \begin{tabular}{ll} imap \begin{tabular}{ll} let \begin{tabular}{ll} largest\_key: $\sqcup$ Map$$$ $\sqcup$ imap) \end{tabular} \end{tabular}
```

OCaml's *compare* from pervasives cannot compare functional types, e.g. for type *amplitude*, if no specific equality function is given ("equal: functional value"). Therefore, we allow to specify the ordering.

```
let get\_ID' comp map elt : int =
let smallmap = IMap.filter (fun \_x \rightarrow (comp \ x \ elt) = 0) map in if IMap.is\_empty smallmap then raise Not\_found else fst (IMap.min\_binding smallmap)
```



Trying to curry map here leads to type errors of the polymorphic function get_ID?

```
let get\_ID\ map\ =\ match\ map\ with
|\ map\ 	o\ get\_ID'\ compare\ map\ 
let get\_const\_ID\ map\ x\ =\ match\ map\ with
|\ (map1,\ map2)\ 	o\ try\ get\_ID'\ compare\ map1\ x\ with
_{-}\ 	o\ failwith\ "Impossible"
```

Creating an integer map of a list with an optional argument that indicates where the map should start counting.

```
\begin{array}{lll} \text{let} \ map\_of\_list \ ?start : (st=1) \ lst \ = \\ & \text{let} \ g \ (ind, \ map) \ wf \ = \ (succ \ ind, \ IMap.add \ ind \ wf \ map) \ \text{in} \\ & lst \ | > \ List.fold\_left \ g \ (st, \ IMap.empty) \ | > \ snd \\ & \text{let} \ wf\_map\_of\_list \ ?start : (st=1) \ lst \ = \\ & \text{let} \ g \ (ind, \ map) \ wf \ = \ (succ \ ind, \ WFMap.add \ wf \ ind \ map) \ \text{in} \\ & lst \ | > \ List.fold\_left \ g \ (st, \ WFMap.empty) \ | > \ snd \\ \end{array}
```

Header



Bijan: It would be nice to save the creation date as comment. However, the Unix module doesn't seem to be loaded on default.

```
let \ version =
  String.concat "" [Config.version; Config.status; Config.date]
let model\_name =
  let basename = Filename.basename Sys.executable_name in
     Filename.chop\_extension\ basename
  with
      \rightarrow basename
  | _
let print_description cmdline =
  printf "Model_%s\n" model_name;
  printf "OVM<sub>□</sub>%s\n" version;
  printf "@\nBytecode_file_generated_automatically_by_0'Mega_for_0VM";
  printf "@\nDo_not_delete_any_lines._You_called_0'Mega_with";
  printf "@\n_\%s" cmdline;
         printf "@\n"
let num\_classified\_wfs wfs =
  let wfs' = classify\_wfs \ wfs in
  List.map List.length
    [ wfs'.scalars @ wfs'.brs\_scalars;
       wfs'.spinors @ wfs'.brs\_spinors;
       wfs'.conjspinors @ wfs'.brs\_conjspinors;
       wfs'.realspinors @ wfs'.brs\_realspinors @ wfs'.ghostspinors;
       wfs'.vectors @ wfs'.massive_vectors @ wfs'.brs_vectors
         @ wfs'.brs_massive_vectors @ wfs'.ward_vectors;
       wfs'.tensors\_2;
       wfs'.tensors\_1;
       wfs'.vectorspinors
let \ description\_classified\_wfs =
  [ "N_scalars";
    "N_spinors";
    "N_conjspinors";
    "N_bispinors";
    "N_vectors";
    "N_tensors_2";
     "N_tensors_1";
    "N_vectorspinors"]
let num\_particles\_in amp =
  match CF.flavors amp with
    [] \rightarrow 0
   | (fin, \_) :: \_ \rightarrow List.length fin
let num\_particles\_out amp =
  match CF.flavors amp with
   [] \rightarrow 0
  (\_, fout) :: \_ \rightarrow List.length fout
let num\_particles \ amp =
  match CF.flavors amp with
    [] \rightarrow 0
  | (fin, fout) :: \_ \rightarrow List.length fin + List.length fout
let num_color_indices_default = 2 (* Standard model and non-color-exotica *)
let num\_color\_indices \ amp =
  try CFlow.rank (List.hd (CF.color_flows amp)) with
  \_ \rightarrow num\_color\_indices\_default
```

```
let num\_color\_factors amp =
       let table = CF.color\_factors \ amp in
       let n\_cflow = Array.length table
       and n\_cfactors = ref 0 in
       for c1 = 0 to pred \ n\_cflow do
         for c2 = 0 to pred n\_cflow do
           if c1 \le c2 then begin
              match table.(c1).(c2) with
              | [] \rightarrow ()
                \rightarrow incr n\_cfactors
            end
         done
       done:
       !n\_cfactors
    |\text{let } num\_helicities | amp | > CF.helicities | > List.length
    let num\_flavors amp = amp \mid > CF.flavors \mid > List.length
    |\text{let } num\_ks \ amp = amp | > CF.processes | > List.length
    let num\_color\_flows \ amp = amp \mid > CF.color\_flows \mid > List.length
Use fst since WFSet.t = F.wf \times int.
    \mathsf{let}\ num\_w\mathit{fs}\ w\mathit{fset}\ =\ w\mathit{fset}\ |>\ \mathit{WFSet.elements}\ |>\ \mathit{List.map}\ \mathit{fst}
                                    \longrightarrow num\_classified\_wfs
largest_key gives the number of momenta if applied to pmap.
    let num\_lst lookups wfset =
       [ largest_key lookups.pmap;
         num_particles lookups.amplitudes;
         num_particles_in lookups.amplitudes;
         num_particles_out lookups.amplitudes;
         num_ks lookups.amplitudes;
         num_helicities lookups.amplitudes;
         num_color_flows lookups.amplitudes;
         num\_color\_indices\ lookups.amplitudes;
         num_flavors lookups.amplitudes;
         num_color_factors lookups.amplitudes | @ num_wfs wfset
    let \ description\_lst =
       [ "N_momenta";
         "N_particles":
         "N_prt_in";
         "N_prt_out";
         "N_amplitudes";
         "N_helicities";
         "N_col_flows";
         "N_col_indices";
         "N_flavors";
         "N_col_factors" ] @ description_classified_wfs
    let print\_header' numbers =
       let \ chopped\_num\_lst = ThoList.chopn \ inst\_length \ numbers
       and chopped\_desc\_lst = ThoList.chopn inst\_length description\_lst
       and printer \ a \ b = print\_str\_lst \ a; \ print\_int\_lst \ b in
       List.iter 2\ printer\ chopped\_desc\_lst\ chopped\_num\_lst
    let print_header lookups wfset = print_header' (num_lst lookups wfset)
    let print_zero_header() =
       let rec zero\_list' j =
         if j < 1 then []
         else 0 :: zero\_list' (j - 1) in
       let zero\_list \ i = zero\_list' \ (i + 1) in
       description\_lst \mid > List.length \mid > zero\_list \mid > print\_header'
```

Tables

```
let print\_spin\_table' tuples =
  match tuples with
   | \ | \ | \rightarrow \ ()
  | \_ \rightarrow tuples | > List.iter (fun (tuple1, tuple2) \rightarrow
        tuple1 @ tuple2 | > List.map (Printf.sprintf "%d_\")
                             --> String.concat "" --> printf "@\n%s")
let print\_spin\_table amplitudes =
  printf "@\nSpin_states_table";
  print_spin_table' @@ CF.helicities amplitudes
let print_flavor_table tuples =
  match tuples with
   | [] \rightarrow ()
  \downarrow \rightarrow List.iter (fun tuple \rightarrow tuple
                           \longrightarrow List.map \text{ (fun } f \rightarrow Printf.sprintf "%d" @@ M.pdg f)
                           \longrightarrow String.concat "" \longrightarrow printf "@\n%s"
                         ) tuples
let print\_flavor\_tables amplitudes =
  printf "@\nFlavor_states_table";
  print\_flavor\_table @@ List.map (fun (fin, fout) \rightarrow fin @ fout)
                            @@ CF.flavors amplitudes
let print_color_flows_table' tuple =
     match CFlow.to_lists tuple with
     | [] \rightarrow ()
     | cfs \rightarrow printf "@\n\s" @@ String.concat "" @@ List.map
                   ( fun cf \rightarrow cf \mid > List.map (Printf.sprintf "%d_\")
                                       -> String.concat ""
                   ) cfs
let print_color_flows_table tuples =
  \mathsf{match}\ \mathit{tuples}\ \mathsf{with}
   | [] \rightarrow ()
  \bot \rightarrow List.iter print_color_flows_table' tuples
let print\_ghost\_flags\_table tuples =
  match tuples with
   | [] \rightarrow ()
   |  \rightarrow
     List.iter (fun tuple \rightarrow
     match \ \mathit{CFlow.ghost\_flags} \ \mathit{tuple} \ with
          | [] \rightarrow ()
           | gfs \rightarrow printf "@\n"; List.iter (fun gf \rightarrow printf "%s_{\sqcup}"
             (if gf then "1" else "0") ) gfs
     ) tuples
let format_power
   \{CFlow.num = num; CFlow.den = den; CFlow.power = pwr\} =
  match num, den, pwr with
   | \ \_, \ 0, \ \_ \ 	o \ invalid\_arg "targets.format_power:\_zero\_denominator"
   [n, d, p \rightarrow [n; d; p]]
let format\_powers = function
  | [] \rightarrow [0]
  powers \rightarrow List.flatten (List.map format\_power powers)
```

Straightforward iteration gives a great speedup compared to the fancier approach which only collects nonzero colorfactors.

```
\begin{array}{lll} \text{let } print\_color\_factor\_table \ table \ = \\ & \text{let } n\_cflow \ = \ Array.length \ table \ \text{in} \\ & \text{if } n\_cflow \ > \ 0 \ \text{then begin} \end{array}
```

```
for c1 = 0 to pred n\_cflow do
       for c2 = 0 to pred n\_cflow do
         if c1 \leq c2 then begin
           match table.(c1).(c2) with
             [] \rightarrow ()
             cf \rightarrow printf "@\n"; List.iter (printf "%9d")
              ([succ\ c1;\ succ\ c2]\ @\ (format\_powers\ cf));
         end
       done
    done
  end
let option\_to\_binary = function
    Some \_ \rightarrow "1"
    None \rightarrow "0"
let print_flavor_color_table n_flv n_cflow table =
  if n_{-}flv > 0 then begin
    for c = 0 to pred n\_cflow do
       printf "@\n";
       for f = 0 to pred n_-flv do
         printf "%s_{\perp}" (option_to_binary table.(f).(c))
       done:
    done:
  end
let print\_color\_tables amplitudes =
  let \ cflows = CF.color\_flows \ amplitudes
  and cfactors = CF.color_factors amplitudes in
  printf "@\nColor_flows_table:_\[\(\(\(\disp\)_i\)\(\disp\)_\(\disp\)\(\disp\)...]";
  print_color_flows_table cflows;
  printf "@\nColor_ghost_flags_table:";
  print_ghost_flags_table cflows;
  printf "@\nColor_factors_table:_[_i,_j:_num_den_power],_%s"
    "i,_j_are_indexed_color_flows";
  print_color_factor_table cfactors;
  printf "@\nFlavor_color_combination_is_allowed:";
  print_flavor_color_table (num_flavors amplitudes) (List.length
     (CF.color_flows amplitudes)) (CF.process_table amplitudes)
```

Momenta

Add the momenta of a WFSet to a Iset. For now, we are throwing away the information to which amplitude the momentum belongs. This could be optimized for random color flow computations.

```
\begin{array}{lll} \text{let} \ momenta\_set \ wfset = \\ & \text{let} \ get\_mom \ wf = wf \mid > fst \mid > F.momentum\_list \ \text{in} \\ & \text{let} \ momenta = List.map \ get\_mom \ (WFSet.elements \ wfset) \ \text{in} \\ & momenta \mid > List.fold\_left \ (\text{fun } set \ x \ \rightarrow \ set \mid > ISet.add \ x) \ ISet.empty \\ & \text{let} \ chop\_in\_3 \ lst = \\ & \text{let} \ ceil\_div \ i \ j = \text{if} \ (i \ \text{mod} \ j = 0) \ \text{then} \ i/j \ \text{else} \ i/j \ + 1 \ \text{in} \\ & ThoList.chopn \ (ceil\_div \ (List.length \ lst) \ 3) \ lst \\ \end{array}
```

Assign momenta via instruction code. External momenta [_] are already set by the OVM. To avoid unnecessary look-ups of IDs we separate two cases. If we have more, we split up in two or three parts.

```
let add\_mom\ p\ pmap = let print\_mom\ lhs\ rhs1\ rhs2\ rhs3 = if\ (rhs1 \not\equiv 0) then printi\ \~lhs: lhs\ \~rhs1: rhs1\ \~rhs2: rhs2\ \~rhs3: rhs3\ ovm\_ADD\_MOMENTA\ in let get\_p\_ID = get\_ID\ pmap\ in match p with |\ []\ |\ [\_]\ \to\ print\_mom\ 0\ 0\ 0\ 0 |\ [rhs1; rhs2]\ \to\ print\_mom\ (get\_p\_ID\ [rhs1; rhs2])\ rhs1\ rhs2\ 0
```

```
 \begin{array}{l} [\mathit{rhs1}; \mathit{rhs2}; \mathit{rhs3}] \to \mathit{print\_mom} \; (\mathit{get\_p\_ID} \; [\mathit{rhs1}; \mathit{rhs2}; \mathit{rhs3}]) \; \mathit{rhs1} \; \mathit{rhs2} \; \mathit{rhs3} \\ | \; \mathit{more} \; \to \\ | \; \mathsf{let} \; \mathit{ids} \; = \; \mathit{List.map} \; \mathit{get\_p\_ID} \; (\mathit{chop\_in\_3} \; \mathit{more}) \; \mathsf{in} \\ | \; \mathsf{if} \; (\mathit{List.length} \; \mathit{ids} \; = \; 3) \; \mathsf{then} \\ | \; \; \mathit{print\_mom} \; (\mathit{get\_p\_ID} \; \mathit{more}) \; (\mathit{List.nth} \; \mathit{ids} \; 0) \; (\mathit{List.nth} \; \mathit{ids} \; 1) \\ | \; \; (\mathit{List.nth} \; \mathit{ids} \; 2) \\ | \; \mathsf{else} \\ | \; \; \mathit{print\_mom} \; (\mathit{get\_p\_ID} \; \mathit{more}) \; (\mathit{List.nth} \; \mathit{ids} \; 0) \; (\mathit{List.nth} \; \mathit{ids} \; 1) \; 0 \\ \end{aligned}
```

Hand through the current level and print level seperators if necessary.

```
\begin{array}{lll} \text{let } add\_all\_mom \ lookups \ pset &= \\ & \text{let } add\_all' \ level \ p &= \\ & \text{let } level' &= List.length \ p \ \text{in} \\ & \text{if } (level' > level \ \land \ level' > 3) \ \text{then } break \ (); \\ & add\_mom \ p \ lookups.pmap; \ level' \\ & \text{in} \\ & ignore \ (pset \ | > ISet.elements \ | > List.fold\_left \ add\_all' \ 1) \end{array}
```

Expand a set of momenta to contain all needed momenta for the computation in the OVM. For this, we create a list of sets which contains the chopped momenta and unify them afterwards. If the set has become larger, we expand again.

```
let rec expand\_pset p =
let momlst = ISet.elements p in
let pset\_of lst = List.fold\_left (fun s x \to ISet.add x s) ISet.empty lst in
let sets = List.map (fun x \to pset\_of (chop\_in\_3 x)) momlst in
let bigset = List.fold\_left ISet.union ISet.empty sets in
let biggerset = ISet.union bigset p in
if (List.length momlst < List.length (ISet.elements biggerset)) then
expand\_pset biggerset
else
biggerset
let mom\_ID pmap wf = get\_ID pmap (F.momentum\_list wf)
```

Wavefunctions and externals

 $mult_wf$ is needed because the wf with same combination of flavor and momentum can have different dependencies and content.

Build the union of all wfs of all amplitudes and a map of the amplitudes.

To obtain the Fortran index, we substract the number of precedent wave functions.

```
\begin{array}{lll} \text{let } lorentz\_ordering\_reduced \ wf &= \\ & \text{match } CM.lorentz \ (F.flavor \ wf) \ \text{with} \\ & \mid \ Scalar \ \mid \ BRS \ Scalar \ \rightarrow \ 0 \end{array}
```

```
Spinor \mid BRS Spinor \rightarrow 1
    ConjSpinor \mid BRS \ ConjSpinor \rightarrow 2
    Majorana \mid BRS \ Majorana \rightarrow 3
    Vector \mid BRS \ Vector \mid Massive\_Vector \mid BRS \ Massive\_Vector \rightarrow 4
    Tensor_2 \mid BRS \ Tensor_2 \rightarrow 5
    Tensor_1 \mid BRS \ Tensor_1 \rightarrow 6
    Vectorspinor \mid BRS \ Vectorspinor \rightarrow 7
    Maj\_Ghost \rightarrow invalid\_arg "lorentz_ordering:_\underingtontonimplemented"
    BRS \rightarrow invalid\_arg "lorentz_ordering:\_not\_needed"
let wf_index \ wfmap \ num_ilst \ (wf, i) =
  let wf\_ID = WFMap.find (wf, i) wfmap
  and sum\ lst\ =\ List.fold\_left\ (fun\ x\ y\ 	o\ x+y)\ 0\ lst in
    wf\_ID - sum (ThoList.hdn (lorentz\_ordering\_reduced wf) num\_lst)
let print_ext\ lookups\ amp_ID\ inc\ (wf,\ i)\ =
  let mom = (F.momentum\_list \ wf) in
  let outer\_index = if List.length mom = 1 then List.hd mom else
    failwith "targets.print_ext:\_called\_with\_non-external\_particle"
  and f = F.flavor wf in
  let pdg = CM.pdg f
  and wf\_code =
    match CM.lorentz f with
      Scalar \rightarrow ovm\_LOAD\_SCALAR
      BRS\ Scalar\ 	o\ ovm\_LOAD\_BRS\_SCALAR
         if inc then ovm\_LOAD\_SPINOR\_INC
         else ovm_LOAD_SPINOR_OUT
    \mid BRS Spinor \rightarrow
         if inc then ovm\_LOAD\_BRS\_SPINOR\_INC
         else ovm\_LOAD\_BRS\_SPINOR\_OUT
    | ConjSpinor \rightarrow
         if inc then ovm\_LOAD\_CONJSPINOR\_INC
         else ovm_LOAD_CONJSPINOR_OUT
    \mid BRS \ ConjSpinor \rightarrow
         if inc then ovm\_LOAD\_BRS\_CONJSPINOR\_INC
         else ovm_LOAD_BRS_CONJSPINOR_OUT
      Vector \mid Massive\_Vector \rightarrow
         if inc then ovm\_LOAD\_VECTOR\_INC
         else ovm\_LOAD\_VECTOR\_OUT
      BRS\ Vector\ |\ BRS\ Massive\_Vector\ 
ightarrow
         if inc then ovm\_LOAD\_BRS\_VECTOR\_INC
         else ovm_LOAD_BRS_VECTOR_OUT
    | Tensor_2 \rightarrow
         if inc then ovm_LOAD_TENSOR2_INC
         else ovm_LOAD_TENSOR2_OUT
      Vectorspinor \mid BRS \ Vectorspinor \rightarrow
         if inc then ovm\_LOAD\_VECTORSPINOR\_INC
         else ovm_LOAD_VECTORSPINOR_OUT
    \mid Majorana \rightarrow
         if inc then ovm\_LOAD\_MAJORANA\_INC
         else ovm_LOAD_MAJORANA_OUT
    \mid BRS \ Majorana \rightarrow
         if inc then ovm\_LOAD\_BRS\_MAJORANA\_INC
         else ovm\_LOAD\_BRS\_MAJORANA\_OUT
      Maj\_Ghost \rightarrow
         if inc then ovm_LOAD_MAJORANA_GHOST_INC
         else ovm\_LOAD\_MAJORANA\_GHOST\_OUT
    | Tensor_1 \rightarrow
         invalid_arg "targets.print_ext: _Tensor_1 only internal"
    \mid BRS \perp \rightarrow
         failwith "targets.print_ext: Not implemented"
```

```
and wf_ind = wf_index lookups.wfmap lookups.n_wfs (wf, i)
     printi wf_code ~lhs: wf_ind ~coupl: (abs(pdg)) ~rhs1: outer_index ~rhs4: amp_ID
let print_ext_amp lookups amplitude =
  \mathsf{let}\ incoming\ =\ (\mathit{List.map}\ (\mathsf{fun}\ \_\ \to\ \mathsf{true})\ (\mathit{F.incoming}\ \mathit{amplitude})\ @
                       List.map (fun _{-} \rightarrow false) (F.outgoing amplitude))
  and amp\_ID = get\_ID' \ amp\_compare \ lookups.amap \ amplitude in
  let wf_-tpl \ wf = mult_-wf \ lookups.dict \ amplitude \ wf in
  let print_ext_wf inc wf = wf \mid > wf_tpl \mid > print_ext lookups amp_tD inc in
     List.iter2 print_ext_wf incoming (F.externals amplitude)
let print_externals lookups seen_wfs amplitude =
  let externals =
     List.combine
        (F.externals amplitude)
       (List.map (fun \_ \rightarrow true) (F.incoming amplitude) @
         List.map (fun _{-} \rightarrow false) (F.outgoing \ amplitude)) in
  List.fold\_left (fun seen (wf, incoming) \rightarrow
     let \ amp\_ID = get\_ID' \ amp\_compare \ lookups.amap \ amplitude \ in
    let wf\_tpl = mult\_wf \ lookups.dict \ amplitude \ wf in
    if \neg (WFSet.mem \ wf\_tpl \ seen) then begin
       wf\_tpl \mid > print\_ext\ lookups\ amp\_ID\ incoming
     end:
     WFSet.add \ wf\_tpl \ seen) seen\_wfs \ externals
```

print_externals and print_ext_amp do in principle the same thing but print_externals filters out dublicate external wave functions. Even with print_externals the same (numerically) external wave function will be loaded if it belongs to a different color flow, just as in the native Fortran code. For color MC, print_ext_amp has to be used (redundant instructions but only one flow is computed) and the filtering of duplicate fusions has to be disabled.

Parallelization issues: All fusions have to be completed before the propagation takes place. Preferably each fusion and propagation is done by one thread. Solution: All fusions are subinstructions, i.e. if they are read by the main loop they are skipped. If a propagation occurs, all fusions have to be computed first. The additional control bit is the sign of the first int of an instruction.

```
let print\_fermion\_current\ code\_a\ code\_b\ code\_c\ coeff\ lhs\ c\ wf1\ wf2\ fusion\ =
let printc\ code\ r1\ r2\ =\ printi\ code\ ^c\ lhs\ :\ lhs\ ^c\ coupl\ :\ c\ ^c\ coeff\ :\ coeff\ ^c\ rhs1: r1\ ^c\ rhs2: r2\ in
match fusion\ with
|\ F13\ \to\ printc\ code\_a\ wf1\ wf2\ |\ F31\ \to\ printc\ code\_a\ wf2\ wf1\ |\ F23\ \to\ printc\ code\_b\ wf2\ wf1\ |\ F32\ \to\ printc\ code\_b\ wf2\ wf1\ |\ F12\ \to\ printc\ code\_c\ wf1\ wf2\ |\ F21\ \to\ printc\ code\_c\ wf2\ wf1\ |\ let\ ferm\_print\_current\ =\ function\ |\ coeff\ ,\ Psibar,\ V,\ Psi\ \to\ print\_fermion\_current\ ovm\_FUSE\_V\_FF\ ovm\_FUSE\_F\_FV\ coeff\ |\ coeff\ ,\ Psibar,\ VA,\ Psi\ \to\ print\_fermion\_current
```

```
ovm_FUSE_VA_FF ovm_FUSE_F_VAF ovm_FUSE_F_FVA coeff
      coeff, Psibar, VA2, Psi \rightarrow print\_fermion\_current
       ovm_FUSE_VA2_FF ovm_FUSE_F_VA2F ovm_FUSE_F_FVA2 coeff
       coeff, Psibar, A, Psi \rightarrow print\_fermion\_current
       ovm_FUSE_A_FF ovm_FUSE_F_AF ovm_FUSE_F_FA coeff
       coeff, Psibar, VL, Psi \rightarrow print\_fermion\_current
       ovm\_FUSE\_VL\_FF ovm\_FUSE\_F\_VLF ovm\_FUSE\_F\_FVL coeff
       coeff, Psibar, VR, Psi \rightarrow print\_fermion\_current
       ovm\_FUSE\_VR\_FF ovm\_FUSE\_F\_VRF ovm\_FUSE\_F\_FVR coeff
       coeff, Psibar, VLR, Psi → print_fermion_current
       ovm\_FUSE\_VLR\_FF ovm\_FUSE\_F\_VLRF ovm\_FUSE\_F\_FVLR coeff
       coeff, Psibar, SP, Psi \rightarrow print\_fermion\_current
        ovm_FUSE_SP_FF ovm_FUSE_F_SPF ovm_FUSE_F_FSP coeff
      coeff, Psibar, S, Psi \rightarrow print\_fermion\_current
       ovm_FUSE_S_FF ovm_FUSE_F_SF ovm_FUSE_F_FS coeff
      coeff, Psibar, P, Psi \rightarrow print\_fermion\_current
       ovm\_FUSE\_P\_FF ovm\_FUSE\_F\_PF ovm\_FUSE\_F\_FP coeff
       coeff, Psibar, SL, Psi \rightarrow print\_fermion\_current
       ovm\_FUSE\_SL\_FF ovm\_FUSE\_F\_SLF ovm\_FUSE\_F\_FSL coeff
       coeff, Psibar, SR, Psi \rightarrow print\_fermion\_current
       ovm\_FUSE\_SR\_FF ovm\_FUSE\_F\_SRF ovm\_FUSE\_F\_FSR coeff
       coeff, Psibar, SLR, Psi \rightarrow print\_fermion\_current
       ovm_FUSE_SLR_FF ovm_FUSE_F_SLRF ovm_FUSE_F_FSLR coeff
       \_, \ Psibar, \ \_, \ Psi \ \rightarrow \ invalid\_arg
       "Targets.Fortran.VM: _no_superpotential_here"
       \_, Chibar, \_, \_ | \_, \_, Chi \rightarrow invalid\_arg
       \verb"Targets.Fortran.VM: $\sqcup$ Majorana $\sqcup$ spinors $\sqcup$ not $\sqcup$ handled "
       \_, Gravbar, \_, \_ | \_, \_, \_, Grav <math>\rightarrow invalid\_arg
        "Targets.Fortran.VM: \_Gravitinos\_not\_handled"
{\sf let} \ \mathit{children2} \ \mathit{rhs} \ = \\
  match F.children rhs with
    [wf1; wf2] \rightarrow (wf1, wf2)
   \mid \ \_ 
ightarrow \mathit{failwith} "Targets.children2:\sqcupcan't\sqcuphappen"
let children3 \ rhs =
  match F.children rhs with
    [wf1; wf2; wf3] \rightarrow (wf1, wf2, wf3)
   \mid \ \_ 
ightarrow invalid\_arg "Targets.children3:\sqcupcan't\sqcuphappen"
let print_vector4 c lhs wf1 wf2 wf3 fusion (coeff, contraction) =
  let printc \ r1 \ r2 \ r3 = printi \ ovm_FUSE_V_VVV \ ~lhs : lhs \ ~coupl : c
      "coeff:coeff"" rhs1:r1"" rhs2:r2"" rhs3:r3 in
  match contraction, fusion with
     C_{-}12_{-}34, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
     C_{-13}_42, (F241 \mid F421 \mid F243 \mid F423 \mid F132 \mid F312 \mid F134 \mid F314)
    C_{-}14_{-}23, (F231 \mid F321 \mid F234 \mid F324 \mid F142 \mid F412 \mid F143 \mid F413) <math>\rightarrow
       printc wf1 wf2 wf3
     C_{-12-34}, (F_{134} | F_{143} | F_{234} | F_{243} | F_{312} | F_{321} | F_{412} | F_{421})
    C_{-13}_{-42}, (F_{124} | F_{142} | F_{324} | F_{342} | F_{213} | F_{231} | F_{413} | F_{431})
    C_{-}14_{-}23, (F123 \mid F132 \mid F423 \mid F432 \mid F214 \mid F241 \mid F314 \mid F341) <math>\rightarrow
       printc wf2 wf3 wf1
    C_{-12}_{-34}, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241)
    C_{-}13_{-}42, (F214 | F412 | F234 | F432 | F123 | F321 | F143 | F341)
  \mid C_{-}14_{-}23, (F213 \mid F312 \mid F243 \mid F342 \mid F124 \mid F421 \mid F134 \mid F431) \rightarrow
       printc wf1 wf3 wf2
let print_current lookups lhs amplitude rhs =
  let f = mult\_wf lookups.dict amplitude in
  match F.coupling \ rhs with
  V3 (vertex, fusion, constant) \rightarrow
       let ch1, ch2 = children2 rhs in
       let wf1 = wf\_index\ lookups.wfmap\ lookups.n\_wfs\ (f\ ch1)
```

```
and wf2 = wf\_index\ lookups.wfmap\ lookups.n\_wfs\ (f\ ch2)
   and p1 = mom\_ID \ lookups.pmap \ ch1
   and p2 = mom\_ID \ lookups.pmap \ ch2
   and const\_ID = get\_const\_ID \ lookups.cmap \ constant in
   let c = if (F.sign \ rhs) < 0 \ then - const\_ID \ else \ const\_ID \ in
   begin match vertex with
    \mid FBF (coeff, fb, b, f) \rightarrow
         begin match coeff, fb, b, f with
           _, Psibar, VLRM, Psi | _, Psibar, SPM, Psi
           _, Psibar, TVA, Psi | _, Psibar, TVAM, Psi
           _, Psibar, TLR, Psi | _, Psibar, TLRM, Psi
           \_, Psibar, TRL, Psi | \_, Psibar, TRLM, Psi \rightarrow failwith
"print_current: UV3: Momentum dependent fermion couplings not implemented"
              ferm_print_current (coeff, fb, b, f) lhs c wf1 wf2 fusion
         end
   \mid PBP (\_, \_, \_, \_) \rightarrow
         failwith "print_current: _{\square}V3:_{\square}PBP_{\square}not_{\square}implemented"
     BBB (\_, \_, \_, \_) \rightarrow
         failwith "print_current: UV3: UBBB Inot implemented"
    \mid GBG (\_, \_, \_, \_) \rightarrow
         failwith "print_current: UV3: UGBG not implemented"
   \mid Gauge\_Gauge\_Gauge\_coeff \rightarrow
         \label{eq:condition} \text{let } printc \ r1 \ r2 \ r3 \ r4 \ = \ printi \ ovm\_FUSE\_G\_GG
            \tilde{lhs}: lhs\ \tilde{coupl}: c\ \tilde{coeff}: coeff\ \tilde{rhs1}: r1\ \tilde{rhs2}: r2\ \tilde{rhs3}: r3
            rhs4: r4 in
         begin match \mathit{fusion} with
          \mid (F23 \mid F31 \mid F12) \rightarrow printc wf1 p1 wf2 p2
         \mid (F32 \mid F13 \mid F21) \rightarrow printc \ wf2 \ p2 \ wf1 \ p1
         end
   | I\_Gauge\_Gauge\_Gauge\_ \rightarrow
         failwith "print_current:\BoxI_Gauge_Gauge_Gauge:\Boxnot\Boximplemented"
     Scalar\_Vector\_Vector\ coeff \rightarrow
         let printc \ code \ r1 \ r2 = printi \ code
            \tilde{\ } lhs : lhs \tilde{\ } coupl : c \tilde{\ } coeff : coeff \tilde{\ } rhs1 : r1 \tilde{\ } rhs2 : r2 in
         begin match fusion with
         | (F23 | F32) \rightarrow printc\ ovm\_FUSE\_S\_VV\ wf1\ wf2
          (F12 \mid F13) \rightarrow printc \ ovm\_FUSE\_V\_SV \ wf1 \ wf2
         | (F21 | F31) \rightarrow printc\ ovm\_FUSE\_V\_SV\ wf2\ wf1
         end
    | Scalar\_Scalar\_Scalar coeff \rightarrow
         printi ovm_FUSE_S_SS ~ lhs: lhs ~ coupl: c ~ coeff: coeff ~ rhs1: wf1 ~ rhs2: wf2
      Vector\_Scalar\_Scalar\ coeff \rightarrow
         let printc\ code\ ?flip: (f = 1)\ r1\ r2\ r3\ r4\ =\ printi\ code
            \tilde{lhs}: lhs\ \tilde{coupl}: (c 	imes f)\ \tilde{coeff}: coeff\ \tilde{rhs1}: r1\ \tilde{rhs2}: r2\ \tilde{rhs3}: r3
            \tilde{r}hs4:r4 in
         begin match fusion with
           F23 \rightarrow printc \ ovm_FUSE_V_SS \ wf1 \ p1 \ wf2 \ p2
           F32 \rightarrow printc \ ovm_FUSE_V_SS \ wf2 \ p2 \ wf1 \ p1
           F12 \rightarrow printc \ ovm\_FUSE\_S\_VS \ wf1 \ p1 \ wf2 \ p2
           F21 \rightarrow printc \ ovm\_FUSE\_S\_VS \ wf2 \ p2 \ wf1 \ p1
           F13 \rightarrow printc \ ovm\_FUSE\_S\_VS \ wf1 \ p1 \ wf2 \ p2 \ ~flip : (-1)
          F31 \rightarrow printc \ ovm\_FUSE\_S\_VS \ wf2 \ p2 \ wf1 \ p1 \ ~flip : (-1)
         end
    Aux\_Vector\_Vector\_ \rightarrow
         failwith "print_current: UV3: Unot implemented"
   \mid Aux\_Scalar\_Scalar\_ \rightarrow
         failwith "print_current: UV3: unot implemented"
```

```
Aux\_Scalar\_Vector\_ \rightarrow
     failwith "print_current: UV3: unot implemented"
| Graviton\_Scalar\_Scalar\_ \rightarrow
     failwith "print_current: _{\square}V3:_{\square}not_{\square}implemented"
  Graviton\_Vector\_Vector\_ \rightarrow
     failwith "print_current: UV3: unot implemented"
| Graviton\_Spinor\_Spinor\_ \rightarrow
     failwith "print_current: UV3: unot implemented"
Dim4\_Vector\_Vector\_Vector\_T\_ \rightarrow
     failwith "print_current: _{\square}V3:_{\square}not_{\square}implemented"
Dim4\_Vector\_Vector\_Vector\_L\_ \rightarrow
     failwith "print_current: UV3: unot implemented"
| Dim6\_Gauge\_Gauge\_Gauge\_ \rightarrow
     failwith "print_current: UV3: unot implemented"
Dim4\_Vector\_Vector\_Vector\_T5\_ \rightarrow
     failwith "print_current: _{\square}V3: _{\square}not_{\square}implemented"
Dim4\_Vector\_Vector\_Vector\_L5\_ \rightarrow
     failwith "print_current: UV3: unot implemented"
| Dim6\_Gauge\_Gauge\_5 \_ \rightarrow
     failwith "print_current: UV3: unot implemented"
\mid Aux\_DScalar\_DScalar\_ \rightarrow
     failwith "print_current: UV3: unot implemented"
 Aux\_Vector\_DScalar\_ \rightarrow
     failwith "print_current: UV3: unot implemented"
| Dim5\_Scalar\_Gauge2 \ coeff \rightarrow
     let printc \ code \ r1 \ r2 \ r3 \ r4 = printi \ code
        \tilde{lhs}: lhs\ \tilde{coupl}: c\ \tilde{coeff}: coeff\ \tilde{rhs1}: r1\ \tilde{rhs2}: r2\ \tilde{rhs3}: r3
        rhs4: r4 in
     begin match fusion with
      (F23 \mid F32) \rightarrow printc \ ovm\_FUSE\_S\_G2 \ wf1 \ p1 \ wf2 \ p2
       (F12 \mid F13) \rightarrow printc \ ovm\_FUSE\_G\_SG \ wf1 \ p1 \ wf2 \ p2
     | (F21 | F31) \rightarrow printc \ ovm\_FUSE\_G\_GS \ wf2 \ p2 \ wf1 \ p1
     end
| Dim5\_Scalar\_Gauge2\_Skew coeff \rightarrow
     let printc\ code\ ?flip: (f = 1)\ r1\ r2\ r3\ r4\ =\ printi\ code
        \tilde{lhs}: lhs\ \tilde{coupl}: (c \times f)\ \tilde{coeff}: coeff\ \tilde{rhs1}: r1\ \tilde{rhs2}: r2\ \tilde{rhs3}: r3
        \tilde{r}hs4:r4 in
     begin match fusion with
     \mid (F23 \mid F32) \rightarrow printc\ ovm\_FUSE\_S\_G2\_SKEW\ wf1\ p1\ wf2\ p2
      (F12 \mid F13) \rightarrow printc \ ovm\_FUSE\_G\_SG\_SKEW \ wf1 \ p1 \ wf2 \ p2
     |(F21 | F31) \rightarrow printc\ ovm\_FUSE\_G\_GS\_SKEW\ wf2\ p1\ wf1\ p2\ ~flip:(-1)
     end
| Dim5\_Scalar\_Vector\_Vector\_T \_ \rightarrow
     failwith "print_current: UV3: unot implemented"
Dim5\_Scalar\_Vector\_Vector\_U\_ \rightarrow
     failwith "print_current: UV3: unot implemented"
Dim5\_Scalar\_Scalar2\_ \rightarrow
     failwith "print_current: UV3: unot implemented"
| Dim6\_Vector\_Vector\_Vector\_T\_ \rightarrow
     failwith "print_current: UV3: unot implemented"
| Tensor_2 - Vector_Vector_- \rightarrow
     failwith "print_current: _{\square}V3:_{\square}not_{\square}implemented"
```

```
Tensor_2\_Scalar\_Scalar\_ \rightarrow
    failwith "print_current: UV3: unot implemented"
Dim5\_Tensor\_2\_Vector\_Vector\_1\_ \rightarrow
    failwith "print_current: _{\square}V3:_{\square}not_{\square}implemented"
 Dim5\_Tensor\_2\_Vector\_Vector\_2\_ \rightarrow
    failwith "print_current: _{\square}V3:_{\square}not_{\square}implemented"
Dim7\_Tensor\_2\_Vector\_Vector\_T\_ \rightarrow
    | Dim5\_Scalar\_Vector\_Vector\_TU \_ \rightarrow
    failwith "print_current: _{\square}V3:_{\square}not_{\square}implemented"
| Scalar\_Vector\_Vector\_t \_ \rightarrow
    failwith "print_current: UV3: unot implemented"
Tensor_2 Vector_Vector_cf \rightarrow
    failwith "print_current: UV3: unot implemented"
| Tensor_2\_Scalar\_Scalar\_cf\_ \rightarrow
    failwith "print_current: _{\square}V3:_{\square}not_{\square}implemented"
  Tensor_2 \_Vector_Vector_1 \_ \rightarrow
    failwith "print_current: UV3: unot implemented"
| Tensor_2 \_Vector_Vector_t \_ \rightarrow
    failwith "print_current: UV3: unot implemented"
| TensorVector\_Vector\_Vector\_ \rightarrow
    failwith "print_current: UV3: unot implemented"
  TensorVector\_Vector\_Vector\_cf\_ \rightarrow
    failwith "print_current: UV3: unot implemented"
\mid TensorVector\_Scalar\_Scalar\_ \rightarrow
    failwith "print_current: UV3: Unot implemented"
| TensorVector\_Scalar\_Scalar\_cf \_ \rightarrow
    TensorScalar\_Vector\_Vector\_ \rightarrow
    failwith "print_current: UV3: unot implemented"
TensorScalar\_Vector\_Vector\_cf\_ \rightarrow
    failwith "print_current: UV3: unot implemented"
\mid TensorScalar\_Scalar\_Scalar\_ \rightarrow
    failwith "print_current: _{\square}V3:_{\square}not_{\square}implemented"
  TensorScalar\_Scalar\_cf \_ \rightarrow
    failwith "print_current: UV3: unot implemented"
| Dim6\_Scalar\_Vector\_Vector\_D \_ \rightarrow
    failwith "print_current: UV3: unot implemented"
Dim6\_Scalar\_Vector\_Vector\_DP\_ \rightarrow
    failwith "print_current: UV3: unot implemented"
 Dim6\_HAZ\_D\_ \rightarrow
    failwith "print_current: UV3: unot implemented"
\mid Dim6\_HAZ\_DP\_ \rightarrow
    failwith "print_current: UV3: unot implemented"
\mid Dim6\_HHH \_ \rightarrow
    failwith "print_current: UV3: unot implemented"
| Dim6\_Gauge\_Gauge\_i \_ \rightarrow
    failwith "print_current: UV3: unot implemented"
| Gauge\_Gauge\_i \_ \rightarrow
```

```
failwith "print_current: UV3: unot implemented"
            \mid Dim6\_GGG\_ \rightarrow
                 failwith "print_current: UV3: unot implemented"
            \mid Dim6\_AWW\_DP\_ \rightarrow
                 failwith "print_current: UV3: unot implemented"
            \mid Dim6\_AWW\_DW\_ \rightarrow
                 failwith "print_current: UV3: unot implemented"
            | Dim6\_WWZ\_DPWDW\_ \rightarrow
                 failwith "print_current: UV3: unot implemented"
            \mid Dim6\_WWZ\_DW\_ \rightarrow
                 failwith "print_current: UV3: unot implemented"
            \mid Dim6\_WWZ\_D\_ \rightarrow
                 Aux\_Gauge\_Gauge\_ \rightarrow
                 failwith "print_current: UV3 (Aux_Gauge_Gauge): _not_implemented"
   end
Flip the sign in c to account for the i^2 relative to diagrams with only cubic couplings.
       \mid V4 (vertex, fusion, constant) \rightarrow
            let ch1, ch2, ch3 = children3 rhs in
            let wf1 = wf\_index\ lookups.wfmap\ lookups.n\_wfs\ (f\ ch1)
            and wf2 = wf\_index\ lookups.wfmap\ lookups.n\_wfs\ (f\ ch2)
            and wf3 = wf\_index\ lookups.wfmap\ lookups.n\_wfs\ (f\ ch3)
                        and const\_ID = get\_const\_ID \ lookups.cmap \ constant in
            \mathsf{let}\ c\ =
              if (F.sign\ rhs) < 0 then const\_ID else - const\_ID in
            begin match vertex with
            \mid Scalar4 coeff \rightarrow
                 printi ovm_FUSE_S_SSS ~lhs: lhs ~coupl: c ~coeff: coeff ~rhs1: wf1
                    \tilde{r}hs2: wf2 \tilde{r}hs3: wf3
            \mid Scalar2\_Vector2 \ coeff \rightarrow
                 let printc \ code \ r1 \ r2 \ r3 \ = \ printi \ code
                    \tilde{l}lhs:lhs\ \tilde{\ }coupl:c\ \tilde{\ }coeff:coeff\ \tilde{\ }rhs1:r1\ \tilde{\ }rhs2:r2\ \tilde{\ }rhs3:r3 in
                 begin match fusion with
                 \mid F134 \mid F143 \mid F234 \mid F243 \rightarrow
                      printc ovm_FUSE_S_SVV wf1 wf2 wf3
                 \mid F314 \mid F413 \mid F324 \mid F423 \rightarrow
                      printc ovm_FUSE_S_SVV wf2 wf1 wf3
                 \mid F341 \mid F431 \mid F342 \mid F432 \rightarrow
                      printc ovm_FUSE_S_SVV wf3 wf1 wf2
                 \mid F312 \mid F321 \mid F412 \mid F421 \rightarrow
                      printc ovm_FUSE_V_SSV wf2 wf3 wf1
                 \mid F231 \mid F132 \mid F241 \mid F142 \rightarrow
                      printc ovm_FUSE_V_SSV wf1 wf3 wf2
                 \mid F123 \mid F213 \mid F124 \mid F214 \rightarrow
                      printc ovm_FUSE_V_SSV wf1 wf2 wf3
                 end
            | Vector4 contractions \rightarrow
                 List.iter (print_vector4 c lhs wf1 wf2 wf3 fusion) contractions
              Vector4\_K\_Matrix\_tho\_
              Vector4_K_Matrix_jr_
              Vector 4 \_K \_Matrix \_cf \_t0 \_
              Vector 4 \_K \_Matrix \_cf \_t1 \_
              Vector4\_K\_Matrix\_cf\_t2\_
              Vector 4 \_K \_Matrix \_cf \_t \_rsi \_
              Vector4\_K\_Matrix\_cf\_m0\_
```

```
Vector 4 \_K \_Matrix \_cf \_m1 \_
  Vector 4 \_K \_Matrix \_cf \_m7 \_
  DScalar2\_Vector2\_K\_Matrix\_ms
  DScalar2\_Vector2\_m\_0\_K\_Matrix\_cf
  DScalar2\_Vector2\_m\_1\_K\_Matrix\_cf
  DScalar2\_Vector2\_m\_7\_K\_Matrix\_cf
  DScalar4\_K\_Matrix\_ms\_ \rightarrow
    failwith \ "print\_current: \_V4: \_K\_Matrix\_not \_implemented"
  Dim8_Scalar2_Vector2_1 _
  Dim8_Scalar2_Vector2_2_
  Dim8\_Scalar2\_Vector2\_m\_0
  Dim8\_Scalar2\_Vector2\_m\_1\_
  Dim8\_Scalar2\_Vector2\_m\_7\_
  Dim8\_Scalar4\_ \rightarrow
    failwith "print_current: UV4: unot implemented"
  Dim8\_Vector4\_t\_0\_ \rightarrow
    failwith "print_current: UV4: unot implemented"
  Dim8\_Vector4\_t\_1\_ \rightarrow
    failwith "print_current: _{\square}V4:_{\square}not_{\square}implemented"
 Dim8\_Vector4\_t\_2\_ \rightarrow
    failwith "print_current: UV4: Unot implemented"
 Dim8\_Vector4\_m\_0\_ \rightarrow
    failwith "print_current: UV4: Unot implemented"
 Dim8\_Vector4\_m\_1\_ \rightarrow
    failwith "print_current: UV4: Unot implemented"
| Dim8\_Vector4\_m\_7\_ \rightarrow
    failwith "print_current:_{\sqcup} V4:_{\sqcup} not_{\sqcup} implemented"
  GBBG \rightarrow
    failwith "print_current: UV4: GBBG not implemented"
  DScalar4 _
 DScalar2\_Vector2\_ \rightarrow
    failwith "print_current: UV4: DScalars not implemented"
  Dim6_H4_P2_ \rightarrow
    failwith "print_current: UV4: Unot Unimplemented"
  Dim6\_AHWW\_DPB\_ \rightarrow
    failwith "print_current: UV4: unot implemented"
  Dim6\_AHWW\_DPW\_ \rightarrow
    failwith "print_current: UV4: unot implemented"
 Dim6\_AHWW\_DW\_ \rightarrow
    failwith "print_current: UV4: Unot Unimplemented"
| Dim6\_Vector4\_DW\_ \rightarrow
    failwith "print_current: UV4: Unot implemented"
| Dim6\_Vector4\_W\_ \rightarrow
    failwith "print_current: \Box V4: \Box not \Box implemented"
| Dim6\_Scalar2\_Vector2\_D \_ \rightarrow
    failwith "print_current: UV4: Unot implemented"
 Dim6\_Scalar2\_Vector2\_DP\_ \rightarrow
    failwith "print_current: UV4: Unot implemented"
  Dim6\_HWWZ\_DW\_ \rightarrow
    failwith "print_current: UV4: Unot Uimplemented"
  Dim6\_HWWZ\_DPB\_ \rightarrow
    failwith "print_current: UV4: Unot implemented"
 Dim6\_HWWZ\_DDPW\_ \rightarrow
    failwith "print_current: UV4: Unot implemented"
 Dim6\_HWWZ\_DPW\_ \rightarrow
    failwith "print_current: UV4: Unot Unimplemented"
| Dim6\_AHHZ\_D\_ \rightarrow
    failwith "print_current:_{\square}V4:_{\square}not_{\square}implemented"
 Dim6\_AHHZ\_DP\_ \rightarrow
    failwith "print_current: UV4: Unot implemented"
\mid Dim6\_AHHZ\_PB\_ \rightarrow
```

```
failwith "print_current: UV4: Unot implemented"
        Dim6\_Scalar2\_Vector2\_PB\_ \rightarrow
           failwith "print_current: UV4: Unot implemented"
        Dim6\_HHZZ\_T\_ \rightarrow
           failwith "print_current: UV4: Unot implemented"
  Vn(\_,\_,\_) \rightarrow invalid\_arg "Targets.print_current:_n-ary_fusion."
                                              Fusions
let print_fusion lookups lhs_momID fusion amplitude =
  if F.on\_shell amplitude (F.lhs\ fusion) then
    failwith "print_fusion: on_shell projectors not implemented!";
  if F.is\_gauss amplitude (F.lhs\ fusion) then
    failwith \ "print_fusion: \_gauss\_amplitudes\_not\_implemented!";
  let lhs\_wf = mult\_wf \ lookups.dict \ amplitude \ (F.lhs \ fusion) in
  let lhs\_wfID = wf\_index\ lookups.wfmap\ lookups.n\_wfs\ lhs\_wf\ in
  let f = F.flavor (F.lhs fusion) in
  let pdg = CM.pdg f in
  \mathsf{let}\ w\ =
    begin match CM.width f with
      Vanishing \mid Fudged \rightarrow 0
      Constant \rightarrow 1
      Timelike \rightarrow 2
      Complex\_Mass \rightarrow 3
      Running \rightarrow 4
      Custom \_ \rightarrow failwith "Targets.VM: \_custom\_width\_not\_available"
    end
  in
  \label{let_propagate_code} \mbox{let propagate code} \ = \ printi\ code\ \~{\ }lhs: lhs\_wfID\ \~{\ }rhs1: lhs\_momID
     in
  begin match \mathit{CM.propagator}\ f with
    Prop\_Scalar \rightarrow
      propagate ovm_PROPAGATE_SCALAR
    Prop\_Col\_Scalar \rightarrow
      propagate\ ovm\_PROPAGATE\_COL\_SCALAR
    Prop\_Ghost \rightarrow
      propagate ovm_PROPAGATE_GHOST
    Prop\_Spinor \rightarrow
      propagate\ ovm\_PROPAGATE\_SPINOR
    Prop\_ConjSpinor \rightarrow
      propagate ovm_PROPAGATE_CONJSPINOR
  | Prop_Majorana \rightarrow
       propagate\ ovm\_PROPAGATE\_MAJORANA
  | Prop\_Col\_Majorana \rightarrow
       propagate ovm_PROPAGATE_COL_MAJORANA
  | Prop\_Unitarity \rightarrow
      propagate ovm_PROPAGATE_UNITARITY
    Prop\_Col\_Unitarity \rightarrow
      propagate ovm_PROPAGATE_COL_UNITARITY
    Prop\_Feynman \rightarrow
      propagate ovm_PROPAGATE_FEYNMAN
    Prop\_Col\_Feynman \rightarrow
       propagate ovm_PROPAGATE_COL_FEYNMAN
    Prop\_Vectorspinor \rightarrow
      propagate ovm_PROPAGATE_VECTORSPINOR
   Prop\_Tensor\_2 \rightarrow
      propagate ovm_PROPAGATE_TENSOR2
```

```
|Aux\_Col\_Scalar| Aux\_Col\_Vector| Aux\_Col\_Tensor\_1 \rightarrow
    failwith "print_fusion: |Aux_Col_*| not | implemented!"
  Aux\_Vector \mid Aux\_Tensor\_1 \mid Aux\_Scalar \mid Aux\_Spinor \mid Aux\_ConjSpinor
  Aux\_Majorana \mid Only\_Insertion \rightarrow
     propagate\ ovm\_PROPAGATE\_NONE
  Prop\_Gauge \_ \rightarrow
    failwith "print_fusion: \squareProp_Gauge \squarenot \square implemented!"
  Prop\_Tensor\_pure \rightarrow
    failwith "print_fusion: Prop_Tensor_pure not implemented!"
 Prop\_Vector\_pure \rightarrow
    failwith "print_fusion:_Prop_Vector_pure_not_implemented!"
| Prop_Rxi_{-} \rightarrow
    failwith "print_fusion: Prop_Rxi not implemented!"
\mid Prop\_UFO\_ \rightarrow
    failwith \ "print_fusion: \_Prop\_UFO\_not\_implemented!"
end:
```

Since the OVM knows that we want to propagate a wf, we can send the necessary fusions now.

```
List.iter (print_current lookups lhs_wfID amplitude) (F.rhs fusion)
```

```
let print\_all\_fusions\ lookups =
let fusions = CF.fusions\ lookups.amplitudes in
let fset = List.fold\_left (fun s\ x \to FSet.add\ x\ s) FSet.empty\ fusions in
ignore\ (List.fold\_left\ (fun\ level\ (f,\ amplitude)\ \to
let wf = F.lhs\ f in
let lhs\_momID = mom\_ID\ lookups.pmap\ wf in
let level' = List.length\ (F.momentum\_list\ wf) in
if (level'\ >\ level\ \land\ level'\ >\ 2) then break\ ();
print\_fusion\ lookups\ lhs\_momID\ f\ amplitude;
level')
1 (FSet.elements\ fset))
```

Brakets

```
iT = i^{\text{#vertices}} i^{\text{#propagators}} \cdots = i^{n-2} i^{n-3} \cdots = -i(-1)^n \cdots (15.1)
```

All brakets for one cflow amplitude should be calculated by one thread to avoid multiple access on the same memory (amplitude).

```
\begin{array}{lll} \text{let } print\_brakets \ lookups \ (amplitude, \ i) = \\ & \text{let } n = List.length \ (F.externals \ amplitude) \ \text{in} \\ & \text{let } sign = \text{if } n \ \text{mod} \ 2 = 0 \ \text{then} \ -1 \ \text{else} \ 1 \\ & \text{and } sym = F.symmetry \ amplitude \ \text{in} \\ & printi \ ovm\_CALC\_BRAKET \ \tilde{\ } lhs: i \ \tilde{\ } rhs1: sym \ \tilde{\ } coupl: sign; \\ & amplitude \ | > F.brakets \ | > List.iter \ (print\_braket \ lookups \ amplitude) \end{array}
```

Fortran arrays/OCaml lists start on 1/0. The amplitude list is sorted by *amp_compare* according to their color flows. In this way the amp array is sorted in the same way as *table_color_factors*.

```
\begin{array}{lll} \text{let } print\_all\_brakets \ lookups \ = \\ \text{let } g \ i \ elt \ = \ print\_brakets \ lookups \ (elt, \ i+1) \ \text{in} \\ lookups.amplitudes} \ | > \ CF.processes \ | > \ List.sort \ amp\_compare \\ --> \ ThoList.iteri \ g \ 0 \end{array}
```

Couplings

For now we only care to catch the arrays gncneu, gnclep, gncup and gncdown of the SM. This will need an overhaul when it is clear how we store the type information of coupling constants.

```
let strip\_array\_tag = function
    Real\_Array x \rightarrow x
    Complex\_Array x \rightarrow x
let array_constants_list =
  let params = M.parameters()
  and strip\_to\_constant (lhs, \_) = strip\_array\_tag lhs in
     List.map\ strip\_to\_constant\ params.derived\_arrays
let is\_array x = List.mem x array\_constants\_list
let constants\_map =
  let first = fun(x, \_, \_) \rightarrow x in
  let second = fun(_-, y,_-) \rightarrow y in
  let third = fun(_-, _-, z) \rightarrow z in
  let v3 = List.map third (first (M.vertices ()))
  and v4 = List.map third (second (M.vertices ())) in
  let set = List.fold\_left (fun s \ x \rightarrow CSet.add \ x \ s) CSet.empty (v3 \ @ v4) in
  let (arrays, singles) = CSet.partition is\_array set in
    (singles \mid > CSet.elements \mid > map\_of\_list,
      arrays \mid > CSet.elements \mid > map\_of\_list)
                                              Output calls
let amplitudes_to_channel (cmdline : string) (oc : out_channel)
  (diagnostics : (diagnostic \times bool) list) (amplitudes : CF.amplitudes) =
  set_formatter_out_channel oc;
  if (num\_particles \ amplitudes = 0) then begin
    print_description cmdline;
    print_zero_header (); nl ()
  end else begin
    let (wfset, amap) = wfset\_amps amplitudes in
    let pset = expand\_pset (momenta\_set wfset)
    and n_-wfs = num_-wfs \ wfset in
    let wfmap = wf_map_of_list (WFSet.elements wfset)
    and pmap = map\_of\_list (ISet.elements pset)
    and cmap = constants\_map in
    let lookups = \{pmap = pmap; wfmap = wfmap; cmap = cmap; amap = amap;
       n\_wfs = n\_wfs; amplitudes = amplitudes;
       dict = CF.dictionary amplitudes} in
    print_description cmdline;
    print_header lookups wfset;
    print_spin_table amplitudes;
    print_flavor_tables amplitudes;
    print_color_tables amplitudes;
    printf "@\n%s" ("OVM_{\sqcup}instructions_{\sqcup}for_{\sqcup}momenta_{\sqcup}addition," ^
                       "_fusions_and_brakets_start_here:_");
     break();
     add\_all\_mom\ lookups\ pset;
    print\_ext\_amps\ lookups;
     break();
    print_all_fusions lookups;
     break();
    print_all_brakets lookups;
     break(); nl();
```

print_flush ()

```
end
```

```
let parameters\_to\_fortran\ oc\ \_\ =
         set_formatter_out_channel oc;
  let arrays\_to\_set = \neg (IMap.is\_empty (snd constants\_map)) in
  let set\_coupl ty dim\ cmap\ =\ IMap.iter (fun key\ elt\ \to
     printf "_{ \sqcup \sqcup \sqcup \sqcup \sqcup} %s (%s%d)_{ \sqcup} =_{ \sqcup} %s" ty dim key (M.constant_symbol elt);
     nl()) cmap in
  let \ declarations \ () =
     printf "□□complex(%s), □dimension(%d)□::□ovm_coupl_cmplx"
       !kind\ (constants\_map\ | > fst\ | > largest\_key);\ nl\ ();
     if arrays\_to\_set then
       printf "_{\sqcup\sqcup} complex(%s),_{\sqcup} dimension(2,_{\sqcup}%d)_{\sqcup}:_{\sqcup} ovm_coupl_cmplx2"
          !kind\ (constants\_map\ | > snd\ | > largest\_key);\ nl\ () in
  let print\_line \ str = printf "%s" str; \ nl() in
  let print_{-}md5sum = function
       \mid Some \ s \rightarrow
          print\_line "_{\sqcup \sqcup} function_{\sqcup} md5sum_{\sqcup}()";
          print_line "□□□□□character(len=32)□::□md5sum";
          print_line "___call_initialize_vm_(vm,_bytecode_file)";
          print_line "____!_DON'T_EVEN_THINK_of_modifying_the_following_line!";
          print\_line ("_{\square \square \square \square} md5sum_{\square} =_{\square}, " \hat{s} ^{"},");
          print\_line "uuendufunctionumd5sum";
       | None \rightarrow ()
  let print_inquiry_function_openmp() = begin
     print_line "⊔⊔pure⊔function⊔openmp_supported⊔()⊔result⊔(status)";
     print_line "⊔⊔⊔⊔logical⊔::⊔status";
     print\_line~("\verb|__| \verb|status| = \verb|_|"~(if~!openmp~then~".true."~else~".false."));
     print\_line " \sqcup \sqcup end \sqcup function \sqcup openmp\_supported";
     nl()
  end in
  let print_interface whizard =
  if whizard then begin
     print_line "□□subroutine□init□(par,□scheme)";
     print_line "____real(kind=default),_dimension(*),_intent(in)_::_par";
     print_line "□□□□□integer,□intent(in)□::□scheme";
     print\_line \ ("\verb||line|| bytecode\_file|| - || `" ^ ! bytecode\_file ^ "'");
     print_line "uuuucalluimport_from_whizardu(par,uscheme)";
     print_line "uuuucalluinitialize_vmu(vm,ubytecode_file)";
     print\_line "\sqcup \sqcupend\sqcupsubroutine\sqcupinit";
     nl();
     print_line "⊔⊔subroutine⊔final⊔()";
     print_line "\u\u\u\u\all\u\m\final\u()";
     print\_line "\sqcup \sqcupend\sqcupsubroutine\sqcupfinal";
     nl();
     print_line "□□subroutine□update_alpha_s□(alpha_s)";
     print\_line \ ("$\sqcup \sqcup \sqcup \sqcup \square$ real (kind=" ^!kind ^ "), $\sqcup intent(in)$_{\sqcup}::$\sqcup alpha_s");
     print\_line "_{\sqcup\sqcup\sqcup\sqcup\sqcup} call_{\sqcup} model\_update\_alpha\_s_{\sqcup} (alpha\_s) ";
     print\_line " \sqcup \sqcup end \sqcup subroutine \sqcup update\_alpha\_s";
     nl()
  end
  else begin
     print\_line "\sqcup \sqcupsubroutine\sqcupinit\sqcup()";
     print\_line ("_{ \  \  } bytecode\_file_{ \  \  } "" \ ^! bytecode\_file \ ^"");
     print_line "⊔⊔⊔⊔⊔call⊔init_parameters⊔()";
     print_line "____call_initialize_vm_(vm,_bytecode_file)";
     print_line "⊔⊔end⊔subroutine"
  end in
  let print\_lookup\_functions() = begin
```

```
print_line "⊔⊔pure⊔function_number_particles_in⊔()⊔result⊔(n)";
print_line "⊔⊔⊔⊔integer⊔::⊔n";
print\_line "_{\sqcup \sqcup \sqcup \sqcup \sqcup} n_{\sqcup} = _{\sqcup} vm %number\_particles\_in_{\sqcup} () ";
print_line "□□end□function□number_particles_in";
print\_line " \sqcup \sqcup pure \sqcup function \sqcup number\_particles\_out \sqcup () \sqcup result \sqcup (n) ";
print\_line "_{\sqcup \sqcup \sqcup \sqcup \sqcup} integer_{\sqcup} :: _{\sqcup} n";
print\_line "_{\sqcup \sqcup \sqcup \sqcup} n_{\sqcup} = _{\sqcup} vm %number\_particles\_out_{\sqcup} () ";
print_line "⊔⊔end⊔function⊔number_particles_out";
print\_line \verb|"|_{\sqcup \sqcup} pure_{\sqcup} function_{\sqcup} number\_spin\_states_{\sqcup}()_{\sqcup} result_{\sqcup}(n) \verb|"|;
print_line "⊔⊔⊔⊔integer⊔::⊔n";
print\_line "_{\sqcup\sqcup\sqcup\sqcup\sqcup} n_{\sqcup} = _{\sqcup} vm %number\_spin\_states_{\sqcup} () ";
print\_line " \sqcup \sqcup end \sqcup function \sqcup number\_spin\_states";
nl();
print\_line "_{\sqcup\sqcup}pure_{\sqcup}subroutine_{\sqcup}spin\_states_{\sqcup}(a) ";
print\_line "_{\sqcup \sqcup \sqcup \sqcup \sqcup} integer, _{\sqcup} dimension(:,:), _{\sqcup} intent(out)_{\sqcup} ::_{\sqcup} a";
print_line "⊔⊔⊔⊔call⊔vm%spin_states⊔(a)";
print\_line " \sqcup \sqcup end \sqcup subroutine \sqcup spin\_states";
nl();
print_line "□□pure□function□number_flavor_states□()□result□(n)";
print\_line " \_ \sqcup \sqcup \sqcup \sqcup = = : \sqcup n ";
print_line "□□□□□n□=□vm%number_flavor_states□()";
print\_line "\sqcup \sqcup end \sqcup function \sqcup number\_flavor\_states";
print\_line "_{\sqcup \sqcup} pure_{\sqcup} subroutine_{\sqcup} flavor\_states_{\sqcup}(a) ";
print\_line "_{\sqcup \sqcup \sqcup \sqcup} integer,_{\sqcup} dimension(:,:),_{\sqcup} intent(out)_{\sqcup}::_{\sqcup} a";
print_line "⊔⊔⊔⊔call⊔vm%flavor_states⊔(a)";
print_line "⊔⊔end⊔subroutine⊔flavor_states";
nl();
print_line "□□pure□function□number_color_indices□()□result□(n)";
print_line "⊔⊔⊔⊔integer⊔::⊔n";
print\_line " \_ \_ vm %number\_color\_indices \_ () ";
print\_line "_uuend_function_number_color_indices";
nl();
print_line "⊔⊔pure⊔function⊔number_color_flows⊔()⊔result⊔(n)";
print_line "⊔⊔⊔⊔integer⊔::⊔n";
print\_line "_{\sqcup \sqcup \sqcup \sqcup \sqcup} n_{\sqcup} = _{\sqcup} vm %number\_color\_flows_{\sqcup}() ";
print\_line "_uuend_function_number_color_flows";
print\_line "_{\sqcup \sqcup} pure_{\sqcup} subroutine_{\sqcup} color\_flows_{\sqcup}(a,_{\sqcup}g) ";
print\_line "_{\sqcup \sqcup \sqcup \sqcup \sqcup} integer,_{\sqcup} dimension(:,:,:),_{\sqcup} intent(out)_{\sqcup}::_{\sqcup} a";
print\_line "_{\sqcup \sqcup \sqcup \sqcup} logical, \_dimension(:,:), \_intent(out)_{\sqcup} :: _{\sqcup}g";
print_line "⊔⊔⊔⊔call⊔vm%color_flows⊔(a, ug)";
print_line "uuendusubroutineucolor_flows";
nl();
print\_line "_{\sqcup\sqcup}pure_{\sqcup}function_{\sqcup}number\_color\_factors_{\sqcup}()_{\sqcup}result_{\sqcup}(n) ";
print\_line "_{\sqcup\sqcup\sqcup\sqcup\sqcup}integer_{\sqcup}::_{\sqcup}n";
print\_line "_{\sqcup\sqcup\sqcup\sqcup\sqcup} n_{\sqcup} = \vm\number\_color\_factors_{\sqcup}()";
print\_line "\sqcupuend\sqcupfunction\sqcupnumber\_color\_factors";
nl();
print_line "⊔⊔pure⊔subroutine⊔color_factors⊔(cf)";
print_line "⊔⊔⊔⊔use⊔omega_color";
print_line "uuuutype(omega_color_factor),udimension(:),uintent(out)u::ucf";
print_line "____call_vm%color_factors_(cf)";
print_line "□□end□subroutine□color_factors";
print\_line "\sqcup \sqcup! pure\sqcupunless\sqcupOpenMP";
print_line "uu!pureufunctionucolor_sumu(flv,uhel)uresultu(amp2)";
print\_line " \_ \_ function \_ color\_ sum \_ (flv, \_ hel) \_ result \_ (amp2) ";
print\_line "_\uu\uu\use\ukinds";
```

```
print_line "⊔⊔⊔⊔integer, uintent(in) :: uflv, uhel";
  print\_line "_{\sqcup \sqcup \sqcup \sqcup} real(kind=default)_{\sqcup} ::_{\sqcup} amp2";
  print\_line "_{\sqcup \sqcup \sqcup \sqcup} amp2_{\sqcup} = _{\sqcup} vm\%color\_sum_{\sqcup} (flv,_{\sqcup}hel) ";
  print_line "⊔⊔end⊔function⊔color_sum";
  print\_line "_{\sqcup\sqcup} subroutine_{\sqcup} new\_event_{\sqcup}(p) ";
  print\_line "____use_kinds";
  print\_line "_{\sqcup\sqcup\sqcup\sqcup} real(kind=default), _{\sqcup} dimension(0:3,*), _{\sqcup} intent(in)_{\sqcup}: _{\sqcup}p";
  print_line "____call_vm%new_event_(p)";
  print\_line "\sqcupuend\sqcupsubroutine\sqcupnew_event";
  nl();
  print_line "uusubroutineureset_helicity_selectionu(threshold,ucutoff)";
  print\_line "_\uuuuse\ukinds";
  print_line "□□□□□real(kind=default),□intent(in)□::□threshold";
  print_line "□□□□integer,□intent(in)□::□cutoff";
  print\_line "_{\sqcup \sqcup \sqcup \sqcup} call_{\sqcup} vm\%reset\_helicity\_selection_{\sqcup} (threshold,_{\sqcup} cutoff)";
  print_line "⊔⊔end⊔subroutine⊔reset_helicity_selection";
  nl();
  print\_line "_{\sqcup\sqcup}pure_{\sqcup}function_{\sqcup}is\_allowed_{\sqcup}(flv,_{\sqcup}hel,_{\sqcup}col)_{\sqcup}result_{\sqcup}(yorn)";
  print_line "⊔⊔⊔⊔logical⊔::⊔yorn";
  print\_line " \_ \_ \_ integer, \_ intent(in) \_ : \_ flv, \_ hel, \_ col";
  print\_line "_{\sqcup \sqcup \sqcup \sqcup \sqcup} yorn_{\sqcup} =_{\sqcup} vm\% is\_allowed_{\sqcup} (flv, \_hel, \_col)";
  print\_line " \sqcup \sqcup end \sqcup function \sqcup is\_allowed";
  nl();
  print_line "uupureufunctionuget_amplitudeu(flv,uhel,ucol)uresultu(amp_result)";
  print\_line "____use_kinds";
  print\_line "_{\sqcup \sqcup \sqcup \sqcup} complex(kind=default)_{\sqcup} :: _{\sqcup} amp\_result";
  print_line "□□□□□integer,□intent(in)□::□flv,□hel,□col";
  print\_line " \_ \_ \_ m \ get\_amplitude (flv, \_ hel, \_ col) ";
  print_line "⊔⊔end⊔function⊔get_amplitude";
  nl();
end in
print_line ("module<sub>□</sub>" ^ !wrapper_module);
print\_line ("\_use\_" ^ !parameter\_module\_external);
print\_line "_\uuse\uiso_varying_string,\usering_t\u=>\uverying_string";
print_line "⊔⊔use⊔kinds";
print\_line "\sqcup \sqcupuse\sqcupomegavm95";
print_line "□□implicit□none";
print\_line "uprivate";
print_line "□□type(vm_t)□::□vm";
print_line "□□type(string_t)□::□bytecode_file";
print_line ("⊔⊔public_::⊔number_particles_in, unumber_particles_out," ^
     "

_number_spin_states, _&");
print_line ("□□□□spin_states,□number_flavor_states,□flavor_states," ^
     "unumber_color_indices,u&");
print\_line \ ("$\sqcup \sqcup \sqcup \sqcup \square$ number\_color\_flows, $\sqcup color\_flows," \ \hat{}
     "unumber_color_factors,ucolor_factors,u&");
print_line ("uuuucolor_sum,unew_event,ureset_helicity_selection," ^
     "_{is}_{allowed},_{iget}_{amplitude},_{ik}");
print_line ("⊔⊔⊔⊔init,⊔"
     (match !md5sum with Some \_ \rightarrow "md5sum,\_"
                             | None \rightarrow "") \hat{ } "openmp_supported");
if !whizard then
  print\_line (" \sqcup public \sqcup : \sqcup final, \sqcup update\_alpha\_s")
  print\_line (" \sqcup public \sqcup : \sqcup initialize\_vm");
declarations ();
print_line "contains";
print_line "⊔⊔subroutine⊔setup_couplings⊔()";
set_coupl "ovm_coupl_cmplx" "" (fst constants_map);
```

```
if arrays\_to\_set then
       set_coupl "ovm_coupl_cmplx2" ":," (snd constants_map);
    print_line "⊔⊔end⊔subroutine⊔setup_couplings";
    print_line "_□□subroutine□initialize_vm□(vm,□bytecode_file)";
    print\_line ~" \_ \_ \_ class(vm\_t), \_ intent(out) \_ : : \_ vm";
    print_line "□□□□□type(string_t),□intent(in)□::□bytecode_file";
    print_line "□□□□□type(string_t)□::□version";
    print\_line "_{\sqcup\sqcup\sqcup\sqcup\sqcup} type(string\_t)_{\sqcup} :: _{\sqcup} model";
    print_line ("___version_=_',OVM_" ^ version ^ ",");
    print_line ("____model__=_', Model_" ^ model_name ^ ",");
    print_line "□□□□□call□setup_couplings□()";
    print_line "uuuucalluvm%initu(bytecode_file,uversion,umodel,uverbose=.False.,u&";
    print_line "____coupl_cmplx=ovm_coupl_cmplx,_\&";
    if arrays\_to\_set then
       print\_line "_{$\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup} coupl\_cmplx2=ovm\_coupl\_cmplx2,$_{\sqcup}\&";
    print\_line ("_{\square\square\square\square\square\square} mass=mass, \_width=width, \_openmp=" ^ (if !openmp then ))
       ".true." else ".false.") ^ ")");
    print\_line "\sqcupuend\sqcupsubroutine\sqcupinitialize\_vm";
    nl();
    print_md5sum ! md5sum;
    print_inquiry_function_openmp ();
    print_interface !whizard;
    print_lookup_functions ();
    print_line ("end_module_" ^ !wrapper_module)
  let parameters\_to\_channel oc =
     parameters_to_fortran oc (CM.parameters ())
end
```

15.4.2 Fortran 90/95

Dirac Fermions

We factor out the code for fermions so that we can use the simpler implementation for Dirac fermions if the model contains no Majorana fermions.

```
module type Fermions =
  sig
    open Coupling
    val psi\_type : string
    val psibar_type : string
    val chi_type : string
    val grav_type : string
    val psi\_incoming : string
    val brs_psi_incoming : string
    val psibar_incoming : string
    val brs_psibar_incoming : string
    val chi\_incoming : string
    val brs_chi_incoming : string
    val grav_incoming : string
    {\tt val} \ psi\_outgoing \ : \ string
    val brs_psi_outgoing : string
    val psibar_outgoing : string
    val brs_psibar_outgoing : string
    val chi_outgoing : string
    val brs_chi_outgoing : string
    val grav_outgoing : string
    val psi_propagator : string
    val\ psibar\_propagator : string
    val\ chi\_propagator\ :\ string
```

```
val grav_propagator : string
     val psi_projector : string
     val psibar_projector : string
     val chi_projector : string
     val grav_projector : string
     val psi\_gauss : string
     val psibar\_gauss : string
     val chi\_gauss : string
     val\ grav\_gauss: string
     val\ print\_current\ :\ int \times fermionbar\ 	imes\ boson\ 	imes\ fermion\ 	o
        string \rightarrow string \rightarrow string \rightarrow fuse2 \rightarrow unit
     val\ print\_current\_mom\ :\ int \times fermionbar\ 	imes\ boson\ 	imes\ fermion\ 	o
        string \rightarrow string \rightarrow string \rightarrow string \rightarrow string \rightarrow string
        \rightarrow fuse2 \rightarrow unit
     \mathsf{val}\ print\_current\_p\ :\ int\ \times\ fermion\ \times\ boson\ \times\ fermion\ \to
        string \rightarrow string \rightarrow string \rightarrow fuse2 \rightarrow unit
     val\ print\_current\_b\ :\ int\ 	imes\ fermionbar\ 	imes\ boson\ 	imes\ fermionbar\ 	o
        string \rightarrow string \rightarrow string \rightarrow fuse2 \rightarrow unit
     \verb|val|| print\_current\_g : int \times fermionbar \times boson \times fermion \rightarrow \\
        string \rightarrow string \rightarrow string \rightarrow string \rightarrow string \rightarrow string
        \rightarrow fuse2 \rightarrow unit
     \verb|val|| print\_current\_g4|: int \times fermionbar \times boson2| \times fermion \rightarrow
        string \rightarrow string \rightarrow string \rightarrow string \rightarrow fuse 3 \rightarrow unit
     val\ reverse\_braket\ :\ bool 
ightarrow\ lorentz\ 
ightarrow\ lorentz\ list 
ightarrow\ bool
     val use_module : string
     val require_library : string list
module Fortran_Fermions : Fermions =
  struct
     open Coupling
     open Format
     let psi_type = "spinor"
     let psibar_type = "conjspinor"
     \mathsf{let}\ \mathit{chi\_type}\ =\ "???"
     let grav_type = "???"
     let psi\_incoming = "u"
     let brs_psi_incoming = "brs_u"
     let psibar_incoming = "vbar"
     let brs_psibar_incoming = "brs_vbar"
     let chi_incoming = "???"
     let brs_chi_incoming = "???"
     \mathsf{let}\ \mathit{grav\_incoming}\ =\ "???"
     let psi\_outgoing = "v"
     let brs_psi_outgoing = "brs_v"
     let psibar_outgoing = "ubar"
     let brs_psibar_outgoing = "brs_ubar"
     let chi_outgoing = "???"
     let brs_chi_outgoing = "???"
     \mathsf{let}\ \mathit{grav\_outgoing}\ =\ "???"
     let \ psi\_propagator = "pr\_psi"
     let psibar_propagator = "pr_psibar"
     let \ \mathit{chi\_propagator} \ = \ "???"
     let \ grav\_propagator = "???"
     \mathsf{let}\ psi\_projector\ =\ \mathtt{"pj\_psi"}
     let psibar_projector = "pj_psibar"
     let \ \mathit{chi\_projector} \ = \ "???"
     let qrav\_projector = "???"
     \mathsf{let}\ psi\_gauss\ =\ \mathtt{"pg\_psi"}
```

```
let psibar_gauss = "pg_psibar"
let chi_qauss = "???"
let grav\_gauss = "???"
let format\_coupling coeff c =
  match coeff with
    1 \rightarrow c
     -1 \rightarrow "(-" \hat{c} \hat{c})"
  | coeff \rightarrow string\_of\_int coeff ^ "*" ^ c
let format\_coupling\_2 coeff c =
  match coeff with
   1 \rightarrow c
     -1 \rightarrow "-" \hat{c}
  | coeff \rightarrow string\_of\_int coeff ^"*" ^ c
```



JR's coupling constant HACK, necessitated by tho's bad design descition.

```
let fastener \ s \ i \ ?p \ ?q \ () =
  try
     let offset = (String.index s '(')) in
     if ((String.get\ s\ (String.length\ s\ -\ 1))\ \not\equiv\ \ \ \ \ )') then
        failwith "fastener: \_wrong_\_usage\_of\_parentheses"
        let func\_name = (String.sub \ s \ 0 \ offset) and
           (String.sub\ s\ (succ\ offset)\ (String.length\ s\ -\ offset\ -\ 2))\ in
        if (String.contains func\_name ')') \lor
            (String.contains\ tail\ '(')\ \lor
            (String.contains tail ')') then
           failwith "fastener: \_wrong\_usage\_of\_parentheses"
           func\_name ^ "(" ^ string\_of\_int i ^ ", " ^ tail ^ ")"
  with
   \mid Not\_found \rightarrow
        if (String.contains \ s), then
           failwith "fastener: \squarewrong \squareusage \squareof \squareparentheses"
        else
           match p with
             None \rightarrow s ^ "(" ^ string\_of\_int i ^ ")"
             Some \ p \rightarrow
              \mathsf{match}\ q\ \mathsf{with}
              | None \rightarrow s \hat{} "(" \hat{} p \hat{} "*" \hat{} p \hat{} "," \hat{} string_of_int i \hat{} ")"
              Some q \rightarrow s ^{\circ} "(" ^{\circ} p ^{\circ} "," ^{\circ} q ^{\circ} "," ^{\circ} string\_of\_int i ^{\circ} ")"
let print_fermion_current coeff f c wf1 wf2 fusion =
  let c = format\_coupling coeff c in
  match fusion with
     F13 \rightarrow printf "%s_ff(%s,%s,%s)" f c wf1 wf2
     F31 \rightarrow printf "%s_ff(%s,%s,%s)" f c wf2 wf1
     F23 \rightarrow printf \text{ "f\_\%sf(\%s,\%s,\%s)"} f c wf1 wf2
     F32 \rightarrow printf "f_%sf(%s,%s,%s)" f c wf2 wf1
     F12 \rightarrow printf \text{ "f_f%s(%s,%s,%s)"} f c wf1 wf2
     F21 \rightarrow printf "f_f%s(%s,%s,%s)" f c wf2 wf1
```



Using a two element array for the combined vector-axial and scalar-pseudo couplings helps to support HELAS as well. Since we will probable are HELAS as well. Since we will probably never support general boson couplings with HELAS, it might be retired in favor of two separate variables. For this *Model.constant_symbol* has to be generalized.



NB: passing the array instead of two separate constants would be a bad idea, because the support for Majorana spinors below will be a bad idea. Majorana spinors below will have to flip signs!

```
let print_fermion_current2 coeff f c wf1 wf2 fusion =
   let c = format\_coupling\_2 coeff c in
   let c1 = fastener \ c \ 1 \ ()
   and c2 = fastener \ c \ 2 () in
   match fusion with
     F13 \rightarrow printf "%s_ff(%s,%s,%s,%s)" f c1 c2 wf1 wf2
     F31 \rightarrow printf "%s_ff(%s,%s,%s,%s)" f c1 c2 wf2 wf1
     F23 \rightarrow printf "f_%sf(%s,%s,%s,%s)" f c1 c2 wf1 wf2
     F32 \rightarrow printf \text{"f-\%sf(\%s,\%s,\%s,\%s)"} f c1 c2 wf2 wf1
     F12 \rightarrow printf \text{"f_f%s(\%s,\%s,\%s,\%s)"} f c1 c2 wf1 wf2
   F21 \rightarrow printf "f_f%s(%s,%s,%s,%s)" f c1 c2 wf2 wf1
let print\_fermion\_current\_mom\_v1 coeff f c wf1 wf2 p1 p2 p12 fusion =
   let c = format\_coupling coeff c in
   \mathsf{let}\ c1\ =\ \mathit{fastener}\ c\ 1\ \mathsf{and}
        c2 = fastener \ c \ 2 \ in
   match fusion with
     F13 \rightarrow printf "%s_ff(%s,%s,%s,%s)" f(c1 \stackrel{\sim}{p}: p12) ()) (c2 \stackrel{\sim}{p}: p12) ()) wf1 wf2
     F31 \rightarrow printf "%s_ff(%s,%s,%s,%s,%s)" f(c1 \tilde{p}: p12) ()) (c2 \tilde{p}: p12) () wf2 wf1
     F23 \rightarrow printf \text{ "f\_\%sf(\%s,\%s,\%s,\%s)"} f (c1 \tilde{p}:p1 ()) (c2 \tilde{p}:p1 ()) wf1 wf2
     F32 \rightarrow printf \ "f_%sf(%s,%s,%s,%s)" \ f \ (c1 \ \tilde{p}:p2 \ ()) \ (c2 \ \tilde{p}:p2 \ ()) \ wf2 \ wf1 \\ F12 \rightarrow printf \ "f_f%s(%s,%s,%s,%s)" \ f \ (c1 \ \tilde{p}:p2 \ ()) \ (c2 \ \tilde{p}:p2 \ ()) \ wf1 \ wf2
   \mid F21 \rightarrow printf \text{ "f-f%s(%s,%s,%s,%s)" } f (c1 \tilde{p}:p1 ()) (c2 \tilde{p}:p1 ()) wf2 wf1
let print\_fermion\_current\_mom\_v2 coeff f c wf1 wf2 p1 p2 p12 fusion =
   let c = format\_coupling coeff c in
   let c1 = fastener \ c \ 1 and
         c2 \ = \ fastener \ c \ 2 \ \mathsf{in}
   match fusion with
     F13 \rightarrow printf "%s_ff(%s,%s,@,%s,%s,%s)" f(c1 \tilde{p}:p12) (c2 \tilde{p}:p12) wf1 wf2 p12
     F31 \rightarrow printf "%s_ff(%s,%s,@,%s,%s,%s)" f(c1 \stackrel{\sim}{p}: p12) ()) (c2 \stackrel{\sim}{p}: p12) ()) wf2 wf1 p12
     F23 \rightarrow printf \ "f\_\%sf(\%s,\%s,@,\%s,\%s,\%s)" \ f \ (c1 \ "p:p1 \ ()) \ (c2 \ "p:p1 \ ()) \ wf1 \ wf2 \ p1
     F32 \rightarrow printf \text{ "f\_\%sf(\%s,\%s,@,\%s,\%s,\%s)" } f (c1 \tilde{p}:p2 ()) (c2 \tilde{p}:p2 ()) wf2 wf1 p2
     F12 \rightarrow printf \ "f_f%s(%s,%s,@,%s,%s,%s)" \ f \ (c1 \ p:p2 \ ()) \ (c2 \ p:p2 \ ()) \ wf1 \ wf2 \ p2
     F21 \rightarrow printf \text{ "f\_f%s(%s,%s,@,%s,%s,"s)" } f (c1 \tilde{p}:p1 ()) (c2 \tilde{p}:p1 ()) wf2 wf1 p1
let print\_fermion\_current\_mom\_ff coeff f c wf1 wf2 p1 p2 p12 fusion =
   let c = format\_coupling coeff c in
   let c1 = fastener c 1 and
         c2 = fastener \ c \ 2 \ in
   match fusion with
     F13 \rightarrow printf "%s_ff(%s,%s,%s,%s)" f(c1 \tilde{p}: p1 \tilde{q}: p2)) (c2 \tilde{p}: p1 \tilde{q}: p2) (b)
     F31 \rightarrow printf "%s_ff(%s,%s,%s,%s)" f(c1 \tilde{p}: p1 \tilde{q}: p2) (c2 \tilde{p}: p1 \tilde{q}: p2) wf2
     F23 \rightarrow printf \text{ "f\_\%sf (\%s,\%s,\%s,\%s)" } f (c1 \tilde{p}:p12 \tilde{q}:p2 \tilde{q}) (c2 \tilde{p}:p12 \tilde{q}:p2 \tilde{q}) wf1 wf2
     F32 \rightarrow printf \ "f_%sf(%s,%s,%s,%s)" \ f \ (c1 \ \tilde{p}:p12 \ \tilde{q}:p1 \ ()) \ (c2 \ \tilde{p}:p12 \ \tilde{q}:p1 \ ()) \ wf2 \ wf1 \ F12 \rightarrow printf \ "f_f%s(%s,%s,%s,%s)" \ f \ (c1 \ \tilde{p}:p12 \ \tilde{q}:p1 \ ()) \ (c2 \ \tilde{p}:p12 \ \tilde{q}:p1 \ ()) \ wf1 \ wf2
     F21 \rightarrow printf \ \texttt{"f\_f\%s(\%s,\%s,\%s,\%s)"} \ f \ (c1 \ \tilde{\ } p:p12 \ \tilde{\ } q:p2 \ ()) \ (c2 \ \tilde{\ } p:p12 \ \tilde{\ } q:p2 \ ()) \ wf2 \ wf1
let print\_current = function
     coeff, Psibar, VA, Psi \rightarrow print\_fermion\_current2 coeff "va"
     coeff, Psibar, VA2, Psi → print_fermion_current coeff "va2"
     coeff, Psibar, VA3, Psi \rightarrow print\_fermion\_current coeff "va3"
     coeff, Psibar, V, Psi \rightarrow print\_fermion\_current coeff "v"
     coeff, Psibar, A, Psi \rightarrow print\_fermion\_current coeff "a"
     coeff, Psibar, VL, Psi \rightarrow print\_fermion\_current coeff "vl"
     coeff, Psibar, VR, Psi \rightarrow print\_fermion\_current coeff "vr"
     coeff, Psibar, VLR, Psi → print_fermion_current2 coeff "vlr"
     coeff, Psibar, SP, Psi → print_fermion_current2 coeff "sp"
     coeff, Psibar, S, Psi \rightarrow print\_fermion\_current coeff "s"
     coeff, Psibar, P, Psi → print_fermion_current coeff "p"
     coeff, Psibar, SL, Psi \rightarrow print\_fermion\_current coeff "sl"
     coeff, Psibar, SR, Psi \rightarrow print\_fermion\_current coeff "sr"
     coeff, Psibar, SLR, Psi \rightarrow print\_fermion\_current2 coeff "slr"
```

```
|  _, Psibar, _, Psi \rightarrow invalid\_arg
              "Targets.Fortran_Fermions: unousuperpotential here"
       |  _, Chibar, _, _ |  _, _, _, Chi \rightarrow invalid_arg
              "Targets.Fortran_Fermions: \( \text{Majorana} \) spinors \( \text{not} \) handled"
       "Targets.Fortran_Fermions: Gravitinos not handled"
    let print\_current\_mom = function
         coeff, Psibar, VLRM, Psi \rightarrow print\_fermion\_current\_mom\_v1 coeff "vlr"
         coeff, Psibar, VAM, Psi → print_fermion_current_mom_ff coeff "va"
         coeff, Psibar, VA3M, Psi \rightarrow print\_fermion\_current\_mom\_ff coeff "va3"
         coeff, Psibar, SPM, Psi \rightarrow print\_fermion\_current\_mom\_v1 coeff "sp"
         coeff, Psibar, TVA, Psi \rightarrow print\_fermion\_current\_mom\_v1 coeff "tva"
         coeff,\ Psibar,\ TVAM,\ Psi\ 	o\ print\_fermion\_current\_mom\_v2\ coeff\ "tvam"
         coeff, Psibar, TLR, Psi \rightarrow print\_fermion\_current\_mom\_v1 coeff "tlr"
         coeff, Psibar, TLRM, Psi \rightarrow print\_fermion\_current\_mom\_v2 coeff "tlrm"
         coeff, Psibar, TRL, Psi → print_fermion_current_mom_v1 coeff "trl"
         coeff, Psibar, TRLM, Psi → print_fermion_current_mom_v2 coeff "trlm"
        \_, Psibar, \_, Psi \rightarrow invalid\_arg
              \verb"Targets.Fortran\_Fermions: \verb""only \verb""sigma \verb"" tensor \verb"" coupling \verb""here""
       |  _, Chibar, _, _ |  _, _, _, Chi \rightarrow invalid\_arg
              "Targets.Fortran_Fermions: \( \text{Majorana} \) spinors \( \text{not} \) handled"
       |  _, Gravbar, _, _ |  _, _, _, Grav \rightarrow invalid\_arg
              "Targets.Fortran\_Fermions: \_Gravitinos\_not\_handled"
    let print\_current\_p = function
       | -, -, -, - \rightarrow invalid\_arg
              \verb"Targets.Fortran_Fermions: $\sqcup No_{\sqcup} clashing_{\sqcup} arrows_{\sqcup} here"$
    let print\_current\_b = function
       | -, -, -, - \rightarrow invalid\_arg
              "Targets.Fortran\_Fermions: \_No\_clashing\_arrows\_here"
    let print\_current\_g = function
       | -, -, -, - \rightarrow invalid\_arg
              \verb"Targets.Fortran_Fermions: $\sqcup$No$$ operations $\sqcup$ here"
    let print\_current\_g4 = function
       | -, -, -, - \rightarrow invalid\_arg
              "Targets.Fortran_Fermions: "No gravitinos here"
    let reverse_braket vintage braket =
       {\sf match}\ {\it bra}\ {\sf with}
        Spinor 
ightarrow true
       \mid \ \_ \rightarrow \mathsf{false}
    let \ use\_module \ = \ "omega95"
    let require_library =
       ["omega_spinors_2010_01_A"; "omega_spinor_cpls_2010_01_A"]
  end
                                                  Main Functor
module Make_Fortran (Fermions : Fermions)
    (Fusion\_Maker : Fusion.Maker) (P : Momentum.T) (M : Model.T) =
  struct
    let require\_library =
       Fermions.require_library @
       [ "omega_vectors_2010_01_A"; "omega_polarizations_2010_01_A";
         "omega_couplings_2010_01_A"; "omega_color_2010_01_A";
         "omega_utils_2010_01_A" ]
    module CM = Colorize.It(M)
    module F = Fusion\_Maker(P)(M)
```

```
module \ CF = Fusion.Multi(Fusion\_Maker)(P)(M)
type amplitudes = CF.amplitudes
open Coupling
open Format
type output\_mode =
    Single_Function
    Single_Module of int
    Single_File of int
    Multi\_File of int
let line\_length = ref 80
let continuation\_lines = ref (-1) (* 255 *)
let kind = ref "default"
let fortran95 = ref true
let module_name = ref "omega_amplitude"
let \ output\_mode = ref \ (Single\_Module \ 10)
let use\_modules = ref[]
let whizard = ref false
let \ amp\_triv = ref \ false
let \ parameter\_module \ = \ ref \ ""
let md5sum = ref None
let no\_write = ref false
let km_-write = ref false
let km_-pure = ref false
let km_2write = ref false
let km_2pure = ref false
\mathsf{let}\ \mathit{openm} p\ =\ \mathit{ref}\ \mathsf{false}
let \ pure\_unless\_openmp \ = \ false
let options = Options.create
  [ "90", Arg. Clear fortran95,
    "don't \_ use \_ Fortran 95 \_ features \_ that \_ are \_ not \_ in \_ Fortran 90";
     "kind", Arg.String (fun s \rightarrow kind := s),
     "real\_and\_complex\_kind\_(default:\_" ^ !kind ^ ")";
     "width", Arg.Int (fun w \rightarrow line\_length := w), "maximum_{\sqcup}line_{\sqcup}length";
    "continuation", Arg.Int (fun l \rightarrow continuation\_lines := l),
     "maximum_{\sqcup}#_{\sqcup}of_{\sqcup}continuation_{\sqcup}lines";
     "module", Arg.String (fun s \rightarrow module\_name := s), "module_name";
     "single_function", Arg.Unit (fun () \rightarrow output\_mode := Single\_Function),
     "compute\_the\_matrix\_element(s)\_in\_a\_monolithic\_function";
     "split_function", Arg.Int (fun n \rightarrow output\_mode := Single\_Module n),
     "splitutheumatrixuelement(s)uintousmallufunctionsu[default,usizeu=u10]";
     "split_module", Arg.Int (fun n \rightarrow output\_mode := Single\_File n),
     "split the matrix element(s) into small modules";
     "split_file", Arg.Int (fun n \rightarrow output\_mode := Multi\_File n),
     "split_the_matrix_element(s)_into_small_files";
     "use", Arg.String (fun s \rightarrow use\_modules := s :: !use\_modules),
     "use_module";
     "parameter_module", Arg.String (fun s \rightarrow parameter_module := s),
     "parameter_module";
     "md5sum", Arg.String (fun s \rightarrow md5sum := Some s),
     "transfer_MD5_checksum";
     "whizard", Arg. Set whizard, "include_WHIZARD_interface";
     "amp_triv", Arg.Set\ amp\_triv, "only_print_trivial_amplitude";
     "no_write", Arg.Set no_write, "no_'write'_statements";
    "kmatrix_write", Arg.Set \ km\_2\_write, "write_K_matrix_functions";
     "kmatrix_2_write", Arg.Set km_write, "write_K_matrix_2_functions";
     "kmatrix_write_pure", Arg.Set\ km\_pure, "write_K_matrix_pure_functions";
     "kmatrix_2_write_pure", Arg.Set km_2_pure, "write_Kmatrix2pure_functions";
     "openmp", Arg.Set\ openmp, "activate_OpenMP_support_in_generated_code"]
```

Fortran style line continuation:

```
let nl = Format\_Fortran.newline
    let print_list = function
       | [] \rightarrow ()
       | a :: rest \rightarrow
            print\_string \ a;
            List.iter 	ext{ (fun } s 	o printf ", @_{\sqcup} %s" s) 	ext{ } rest
                                               Variables and Declarations
"NC" is already used up in the module "constants":
    \mathsf{let}\ \mathit{nc\_parameter}\ =\ \mathtt{"N\_"}
    let \ omega\_color\_factor\_abbrev \ = \ "OCF"
    let openmp_tld_type = "thread_local_data"
    let \ openmp\_tld = "tld"
    let flavors\_symbol ?(decl = false) flavors =
       (if !openmp \land \neg decl then openmp\_tld \ ^ "%" else "" ) ^ 
       "oks_" ^ String.concat "" (List.map CM.flavor_symbol flavors)
    let p2s p =
       if p \geq 0 \land p \leq 9 then
          string\_of\_int p
       else if p \leq 36 then
          String.make\ 1\ (Char.chr\ (Char.code\ 'A'\ +\ p\ -\ 10))
          "_"
    let format\_momentum p =
       "p" ^ String.concat "" (List.map p2s p)
    let format_p wf =
       String.concat "" (List.map p2s (F.momentum_list wf))
    let ext\_momentum \ wf =
       match F.momentum\_list \ wf with
         [n] \rightarrow n
       oxed{\ } \rightarrow invalid\_arg "Targets.Fortran.ext_momentum"
    module PSet = Set.Make (struct type t = int \ list \ let \ compare = compare \ end)
    module WFSet = Set.Make (struct type t = F.wf let compare = compare end)
    let add_{-}tag \ wf \ name =
       match F.wf\_tag \ wf with
         None \rightarrow name
       | Some tag \rightarrow name \hat{\ } "\_" \hat{\ } tag
    let variable ?(decl = false) wf =
       (if !openmp \land \neg decl then openmp\_tld \ \hat{\ } "%" else "")
         add\_tag\ wf\ ("owf\_"\ ^ CM.flavor\_symbol\ (F.flavor\ wf)\ ^ "\_"\ ^ format\_p\ wf)
    let momentum \ wf = "p" \hat{\ } format\_p \ wf
    let spin \ wf = "s(" \hat string\_of\_int (ext\_momentum \ wf) \hat ")"
    let format\_multiple\_variable\ ?(decl = false)\ wf\ i =
       variable \ \tilde{\ } decl \ wf \ \hat{\ } "\_X" \ \hat{\ } string\_of\_int \ i
    let multiple\_variable\ ?(decl = false)\ amplitude\ dictionary\ wf =
         format_multiple_variable ~decl wf (dictionary amplitude wf)
       with
       \mid Not\_found \rightarrow variable \ wf
    let multiple\_variables?(decl = false) multiplicity wf =
       try
          List.map
            (format\_multiple\_variable \ \tilde{\ } decl \ wf)
```

```
(ThoList.range\ 1\ (multiplicity\ wf))
  with
  | Not\_found \rightarrow [variable \ \ decl \ wf]
let declaration\_chunk\_size = 64
let declare\_list\_chunk \ multiplicity \ t = function
   | [] \rightarrow ()
  | wfs \rightarrow
       printf ~" \_ \_ \_ @[<2>\%s \_ : : \_ " ~t;
        print_list (ThoList.flatmap (multiple_variables ~decl :true multiplicity) wfs); nl ()
let declare\_list multiplicity t = function
   | \ | \ | \rightarrow \ ()
  | wfs \rightarrow
        List.iter
          (declare\_list\_chunk \ multiplicity \ t)
          (ThoList.chopn\ declaration\_chunk\_size\ wfs)
type declarations =
     \{ scalars : F.wf list; \}
        spinors : F.wf \ list;
        conjspinors : F.wf list;
        realspinors : F.wf list;
        ghostspinors : F.wf list;
        vectorspinors : F.wf list;
        vectors : F.wf list;
        ward\_vectors : F.wf list;
        massive\_vectors : F.wf list;
        tensors\_1 : F.wf list;
        tensors \_2 : F.wf list;
        brs\_scalars : F.wf\ list;
        brs\_spinors : F.wf list;
        brs\_conjspinors : F.wf list;
        brs\_realspinors : F.wf list;
        brs\_vectorspinors : F.wf list;
        brs\_vectors : F.wf list;
        brs_massive_vectors : F.wf list }
let rec classify\_wfs' acc = function
   | [] \rightarrow acc
  | wf :: rest \rightarrow
        classify\_wfs'
           (match CM.lorentz (F.flavor wf) with
             Scalar \rightarrow \{acc \text{ with } scalars = wf :: acc.scalars\}
             Spinor \rightarrow \{acc \text{ with } spinors = wf :: acc.spinors\}
             ConjSpinor \rightarrow \{acc \text{ with } conjspinors = wf :: acc.conjspinors\}
             Majorana \rightarrow \{acc \ with \ real spinors = wf :: acc.real spinors\}
             Maj\_Ghost \rightarrow \{acc \text{ with } ghostspinors = wf :: acc.ghostspinors\}
             Vectorspinor \rightarrow
                \{acc \text{ with } vectorspinors = wf :: acc.vectorspinors\}
             Vector \rightarrow \{acc \text{ with } vectors = wf :: acc.vectors\}
             Massive\_Vector \rightarrow
                \{acc\ with\ massive\_vectors = wf :: acc.massive\_vectors\}
             Tensor_1 \rightarrow \{acc \text{ with } tensors_1 = wf :: acc.tensors_1\}
             Tensor_2 \rightarrow \{acc \text{ with } tensors_2 = wf :: acc.tensors_2\}
             BRS\ Scalar \rightarrow \{acc\ with\ brs\_scalars = wf :: acc.brs\_scalars\}
             BRS\ Spinor\ 	o\ \{acc\ with\ brs\_spinors\ =\ wf\ ::\ acc.brs\_spinors\}
            BRS\ ConjSpinor\ 	o\ \{acc\ with\ brs\_conjspinors\ =
                                      wf :: acc.brs\_conjspinors
          BRS\ Majorana \rightarrow \{acc\ with\ brs\_realspinors =
                                   wf :: acc.brs\_realspinors}
          \mid BRS \ Vectorspinor \rightarrow \{acc \ with \ brs\_vectorspinors =
                                         wf :: acc.brs\_vectorspinors
```

```
BRS\ Vector \rightarrow \{acc\ with\ brs\_vectors = wf :: acc.brs\_vectors\}
            BRS\ Massive\_Vector \rightarrow \{acc\ with\ brs\_massive\_vectors =
                                          wf :: acc.brs\_massive\_vectors
          let classify\_wfs \ wfs = classify\_wfs'
     \{ scalars = []; spinors = []; conjspinors = []; realspinors = [];
       ghostspinors = []; vectorspinors = []; vectors = [];
        ward\_vectors = [];
        massive\_vectors = []; tensors\_1 = []; tensors\_2 = [];
        brs_scalars = []; brs_spinors = []; brs_conjspinors = [];
        brs_realspinors = []; brs_vectorspinors = [];
       brs_vectors = []; brs_massive_vectors = []}
     wfs
                                                  Parameters
type \alpha parameters =
     \{ real\_singles : \alpha list; 
       real\_arrays : (\alpha \times int) list;
        complex\_singles : \alpha list;
       complex\_arrays : (\alpha \times int) \ list \}
let rec classify\_singles acc = function
  | [] \rightarrow acc
  | Real p :: rest \rightarrow classify\_singles
          \{ acc \text{ with } real\_singles = p :: acc.real\_singles \} rest
  | Complex p :: rest \rightarrow classify\_singles
          \{ acc \text{ with } complex\_singles = p :: acc.complex\_singles \} rest
let rec classify\_arrays acc = function
   | \ | \ | \rightarrow acc
   | (Real\_Array \ p, \ rhs) :: rest \rightarrow classify\_arrays
          \{ acc \text{ with } real\_arrays = 
            (p, List.length rhs) :: acc.real\_arrays \} rest
  | (Complex\_Array \ p, \ rhs) :: rest \rightarrow classify\_arrays
          \{ acc \text{ with } complex\_arrays = 
            (p, List.length rhs) :: acc.complex_arrays } rest
let classify\_parameters params =
   classify\_arrays
     (classify\_singles
         \{ real\_singles = [];
           real\_arrays = [];
           complex\_singles = [];
           complex\_arrays = [] 
         (List.map fst params.derived)) params.derived_arrays
let \ schisma = ThoList.chopn
let schisma_num i n l =
   ThoList.enumerate \ i \ (schisma \ n \ l)
\mathsf{let}\ \mathit{declare\_parameters'}\ t\ =\ \mathsf{function}
   | [] \rightarrow ()
  | plist \rightarrow
       printf " \sqcup \sqcup @[<2>\%s(kind=\%s), \sqcup public, \sqcup save \sqcup : : \sqcup " t ! kind;
       print_list (List.map CM.constant_symbol plist); nl ()
let \ declare\_parameters \ t \ plist =
   List.iter (declare_parameters' t) plist
let declare\_parameter\_array\ t\ (p,\ n)\ =
  printf " \sqcup \sqcup 0 [<2>\%s(kind=\%s), \sqcup dimension(\%d), \sqcup public, \sqcup save \sqcup :: \sqcup \%s"
```

```
t ! kind n (CM.constant\_symbol p); nl ()
NB: we use string_of_float to make sure that a decimal point is included to make Fortran compilers happy.
     let default\_parameter(x, v) =
        printf \ "@_ \%s_ =_ \%s \ (CM.constant\_symbol \ x) \ (string\_of\_float \ v) \ !kind
     let \ declare\_default\_parameters \ t \ = \ function
        | [] \rightarrow ()
        p :: plist \rightarrow
             printf "_{\sqcup \sqcup} @[<2>\%s(kind=\%s),_{\sqcup}public,_{\sqcup}save_{\sqcup}::" t!kind;
             default\_parameter p;
             List.iter (fun p' \rightarrow printf ","; default_parameter p') plist;
             nl()
     let format\_constant = function
        I \rightarrow "(0,1)"
        Integer c \rightarrow
            if c < 0 then
              sprintf "(%d.0_%s)" c !kind
              sprintf "%d.0_%s" c !kind
        \mid Float \ x \rightarrow
            if x < 0. then
              "(" \hat{string\_of\_float} \ x \ \hat{"} = \hat{kind} \ \hat{"}")"
              string\_of\_float \ x ` "\_" ` !kind
        | _ → invalid_arg "format_constant"
     let rec eval\_parameter' = function
        \mid (I \mid Integer \_ \mid Float \_) as c \rightarrow
            printf "%s" (format_constant c)
          Atom x \rightarrow printf "%s" (CM.constant\_symbol x)
          Sum [] \rightarrow printf "0.0_%s" !kind
          Sum [x] \rightarrow eval\_parameter' x
          Sum (x :: xs) \rightarrow
             printf "@,("; eval_parameter' x;
             List.iter (fun x \rightarrow printf "0, \Box + \Box"; eval\_parameter'(x)(xs);
             printf ")"
        \mid Diff(x, y) \rightarrow
             printf "@,("; eval_parameter' x;
             printf "¬¬¬"; eval¬parameter' y; printf ")"
          Neg \ x \rightarrow printf \ "@,(_- _ "; eval\_parameter' \ x; printf ")"
          Prod [] \rightarrow printf "1.0_%s" !kind
          Prod[x] \rightarrow eval\_parameter' x
        | Prod (x :: xs) \rightarrow
             printf "@,("; eval_parameter' x;
             List.iter (fun x \rightarrow printf "_{\sqcup}*_{\sqcup}"; eval\_parameter' x) xs;
             printf ")"
        | Quot(x, y) \rightarrow
             printf "@,("; eval_parameter' x;
             printf " \_ / \_ "; eval\_parameter' y; printf ") "
        \mid Rec \ x \rightarrow
             printf "@,_{\square}(1.0_{-}%s_{\square}/_{\square}" !kind; eval\_parameter' x; printf ")"
        \mid Pow(x, n) \rightarrow
            printf "@,("; eval_parameter' x;
            if n < 0 then
              printf "**(%d)" n
            else
              printf "**%d" n;
            printf ")"
        \mid PowX(x, y) \rightarrow
             printf "@,("; eval_parameter' x;
              printf "**"; eval_parameter' y; printf ")"
```

```
Sqrt \ x \rightarrow printf \ "@,sqrt_{\sqcup}("; eval\_parameter' \ x; printf ")"
     Sin \ x \rightarrow printf \ "0, sin_{\perp}("; eval\_parameter' \ x; printf ")"
     Cos x \rightarrow printf "0, cos_{\sqcup}("; eval\_parameter' x; printf ")"
     Tan \ x \rightarrow printf \ "O, tan_{\sqcup}("; eval\_parameter' \ x; printf ")"
     Cot x \rightarrow printf "0,cot_("; eval\_parameter' x; printf ")"
     Asin x \rightarrow printf "0,asin("; eval\_parameter' x; printf ")"
     Acos x \rightarrow printf "@,acos_{\sqcup}("; eval\_parameter' x; printf ")"
     Atan x \rightarrow printf "@,atan\sqcup("; eval\_parameter' x; printf ")"
     Atan2(y, x) \rightarrow printf "@,atan2 ("; eval\_parameter' y; 
       printf ", Q_"; eval_parameter' x; printf ")"
     Sinh \ x \rightarrow printf "O,sinh\sqcup("; eval\_parameter' \ x; \ printf ")"
     Cosh \ x \rightarrow printf \ "@, cosh_{\sqcup}("; eval\_parameter' \ x; printf ")"
     Tanh x \rightarrow printf "@,tanh\sqcup("; eval\_parameter' x; printf ")"
     Exp \ x \rightarrow printf "@,exp\sqcup("; eval\_parameter' \ x; \ printf ")"
     Log x \rightarrow printf "@,log_{\sqcup}("; eval\_parameter' x; printf ")"
     Log10 \ x \rightarrow printf "@,log10\sqcup("; eval\_parameter' \ x; \ printf ")"
     Conj (Integer \_ | Float \_ as x) \rightarrow eval\_parameter' x
     Conj \ x \rightarrow printf \ "@,cconjg_{\sqcup}("; eval\_parameter' \ x; printf ")"
     Abs \ x \rightarrow printf "@,abs\sqcup("; eval\_parameter' \ x; \ printf ")"
let strip\_single\_tag = function
     Real \ x \rightarrow x
     Complex x \rightarrow x
let strip\_array\_tag = function
    Real\_Array x \rightarrow x
   | Complex\_Array x \rightarrow x
let \ eval\_parameter \ (lhs, \ rhs) =
  let x = CM.constant\_symbol (strip\_single\_tag lhs) in
  let eval\_para\_list \ n \ l =
  printf "\sqcup \sqcupsubroutine\sqcupsetup\_parameters\_%03d_{\sqcup}()" n; nl ();
  List.iter\ eval\_parameter\ l;
  printf "□□end□subroutine□setup_parameters_%03d" n; nl ()
let eval\_parameter\_pair (lhs, rhs) =
  let x = CM.constant\_symbol (strip\_array\_tag lhs) in
  let \_ = List.fold\_left (fun i \ rhs' \rightarrow
     printf = 2 \%s(%d) = x i; eval\_parameter' rhs'; nl();
     succ i) 1 rhs in
let \ eval\_para\_pair\_list \ n \ l =
  printf "\sqcup \sqcupsubroutine\sqcupsetup\_parameters\_%03d_{\sqcup}()" n; nl ();
  List.iter eval_parameter_pair l;
  printf "□□end□subroutine□setup_parameters_%03d" n; nl ()
let print_echo fmt p =
  let s = CM.constant\_symbol p in
  printf "___write_(unit_=_*,_fmt_=_fmt_%s)_\"%s\",_%s"
     fmt \ s \ s; \ nl \ ()
let print\_echo\_array\ fmt\ (p,\ n) =
  let s = CM.constant\_symbol p in
  for i = 1 to n do
     printf "___write_(unit_=_*,_fmt_=_fmt_%s_array)_" fmt;
     printf "\"%s\", \\\\d,\\\\s(\%d)\" s i s i; nl ()
  done
let contains params couplings =
   List.exists
     (fun (name, \bot) \rightarrow List.mem (CM.constant\_symbol name) params)
     couplings.input
```

```
let rec depends\_on\ params\ =\ function
    I \mid Integer \_ \mid Float \_ \rightarrow false
    Atom\ name\ 	o\ List.mem\ (CM.constant\_symbol\ name)\ params
    Sum\ es\ |\ Prod\ es\ 
ightarrow
      List.exists (depends_on params) es
    Diff(e1, e2) \mid Quot(e1, e2) \mid PowX(e1, e2) \rightarrow
      depends\_on\ params\ e1\ \lor\ depends\_on\ params\ e2
    Neg \ e \mid Rec \ e \mid Pow \ (e, \bot) \rightarrow
      depends_on params e
     Sqrt \ e \mid Exp \ e \mid Log \ e \mid Log 10 \ e
     Sin \ e \mid Cos \ e \mid Tan \ e \mid Cot \ e
     Asin e | Acos e | Atan e
     Sinh \ e \mid Cosh \ e \mid Tanh \ e
    Conj e \mid Abs \ e \rightarrow
      depends\_on\ params\ e
   | Atan2 (e1, e2) \rightarrow
      depends_on params e1 ∨ depends_on params e2
let dependencies params couplings =
  if contains params couplings then
     List.rev
        (fst (List.fold_left
                  (fun (deps, plist) (param, v) \rightarrow
                    match param with
                    \mid Real \ name \mid Complex \ name \rightarrow
                        if depends\_on\ plist\ v then
                          ((param, v) :: deps, CM.constant\_symbol name :: plist)
                        else
                          (deps, plist)
                  ([], params) couplings.derived))
  else
     let dependencies_arrays params couplings =
  if contains params couplings then
     List.rev
        (fst (List.fold_left
                  (fun (deps, plist) (param, vlist) \rightarrow
                    match param with
                    \mid Real\_Array \ name \mid Complex\_Array \ name \rightarrow
                        if List.exists (depends_on plist) vlist then
                          ((param, vlist) :: deps,
                            CM.constant\_symbol\ name\ ::\ plist)
                        else
                          (deps, plist)
                  ([], params) couplings.derived_arrays))
  else
let parameters\_to\_fortran oc params =
   Format_Fortran.set_formatter_out_channel ~width :!line_length oc;
  let declarations = classify\_parameters params in
  printf "module\sqcup%s" !parameter\_module; nl ();
  printf "\sqcup \sqcupuse\sqcupkinds"; nl ();
  printf "\sqcup \sqcupuse\sqcupconstants"; nl ();
  printf "⊔⊔implicit⊔none"; nl ();
  printf "\sqcup \sqcup private"; nl ();
  printf "□□0[<2>public□::□setup_parameters";
  printf ~", @\_{\tt import\_from\_whizard"};
  printf ", @_model_update_alpha_s";
  if !no\_write then begin
     printf "!\\\\\No\\\\print\\\\parameters\\\\;
  end else begin
```

```
printf ", @_print_parameters";
end; nl();
declare_default_parameters "real" params.input;
declare_parameters "real" (schisma 69 declarations.real_singles);
List.iter (declare_parameter_array "real") declarations.real_arrays;
declare_parameters "complex" (schisma 69 declarations.complex_singles);
List.iter (declare_parameter_array "complex") declarations.complex_arrays;
printf "□□interface□cconjg"; nl ();
printf "____module_procedure_cconjg_real,_cconjg_complex"; nl ();
printf "\sqcup \sqcup end \sqcup interface"; nl ();
printf "□□private□::□cconjg_real,□cconjg_complex"; nl ();
printf "contains"; nl ();
printf "___function_cconjg_real_(x)_result_(xc)"; nl();
printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup} real(kind=default), \_intent(in)_{\sqcup}::_{\sqcup}x"; nl();
printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup} real(kind=default)_{\sqcup} : _{\sqcup}xc"; nl ();
printf " \sqcup \sqcup \sqcup \sqcup \mathsf{xc} \sqcup = \sqcup \mathsf{x} "; nl ();
printf "ullendufunctionucconjg_real"; nl ();
printf "_{\sqcup\sqcup} function_{\sqcup} cconjg_{-complex_{\sqcup}}(z)_{\sqcup} result_{\sqcup}(zc)"; nl ();
printf "_{\sqcup\sqcup\sqcup\sqcup} complex(kind=default), _{\sqcup} intent(in)_{\sqcup} ::_{\sqcup} z"; nl();
printf "_{\sqcup \sqcup \sqcup \sqcup} complex(kind=default)_{\sqcup} : _{\sqcup}zc"; nl ();
printf = zc_1 = conig_1(z) = nl(z)
printf "⊔⊔end⊔function⊔cconjg_complex"; nl ();
printf "_{\sqcup \sqcup}!_{\sqcup}derived_{\sqcup}parameters:"; nl ();
let shredded = schisma\_num 1 120 params.derived in
let shredded\_arrays = schisma\_num \ 1 \ 120 \ params.derived\_arrays in
let num\_sub = List.length shredded in
let num\_sub\_arrays = List.length shredded\_arrays in
List.iter (fun (i, l) \rightarrow eval\_para\_list i l) shredded;
List.iter (fun (i, l) \rightarrow eval\_para\_pair\_list (num\_sub + i) l)
  shredded\_arrays;
printf "⊔⊔subroutine⊔setup_parameters⊔()"; nl ();
for i = 1 to num\_sub + num\_sub\_arrays do
  printf "____call_setup_parameters_%03d_()" i; nl();
done;
printf "\sqcup \sqcupend\sqcupsubroutine\sqcupsetup\_parameters"; nl ();
printf "_usubroutine_import_from_whizard_(par_array,uscheme)"; nl ();
printf
  "⊔⊔⊔⊓real(%s), dimension(%d), intent(in) :: par_array"
  !kind (List.length params.input); nl ();
printf "□□□□□integer,□intent(in)□::□scheme"; nl ();
let i = ref 1 in
List.iter
  (fun (p, \_) \rightarrow
     printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup} %s_{\sqcup} = par_{array} (%d) " (CM.constant_symbol p) !i; nl ();
     incr i)
  params.input;
printf "_{\sqcup \sqcup \sqcup \sqcup} call_{\sqcup} setup_parameters_{\sqcup}() "; nl();
printf "⊔⊔end⊔subroutine⊔import_from_whizard"; nl ();
printf "_{\sqcup\sqcup} subroutine_{\sqcup} model\_update\_alpha\_s_{\sqcup} (alpha\_s) "; nl ();
printf "uuuureal(%s),uintent(in)u::ualpha_s" !kind; nl ();
begin match (dependencies ["aS"] params,
                 dependencies_arrays ["aS"] params) with
| [], [] \rightarrow
    printf "____!_'aS',_not_among_the_input_parameters"; nl ();
| deps, deps\_arrays \rightarrow
    printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup}aS_{\sqcup}=_{\sqcup}alpha_s"; nl ();
    List.iter eval_parameter deps;
    List.iter eval_parameter_pair deps_arrays
printf "uuendusubroutineumodelupdate_alpha_s"; nl();
if !no\_write then begin
```

```
printf "!\_No\_print\_parameters"; nl ();
       end else begin
         printf "□□subroutine□print_parameters□()"; nl();
         printf "____@[<2>character(len=*),_parameter_::";
         printf "@\fmt_real\=\\"(A12,4X,'\=\',E25.18)\",";
         printf "Q_fmt_complex_=\\"(A12,4X,'_=\',E25.18,'\_+\_i*',E25.18)\\",";
         printf "@ufmt_real_arrayu=u\"(A12,'(',I2.2,')','u=u',E25.18)\",";
         printf "@□fmt_complex_array□=□";
         printf "\"(A12,'(',I2.2,')','=\',E25.18,'\+\i\*',E25.18)\""; nl ();
         printf "\"default_values_for_the_input_parameters:\""; nl ();
         List.iter (fun (p, \_) \rightarrow print\_echo "real" p) params.input;
         printf "_{\sqcup \sqcup \sqcup \sqcup} @ [<2> write_{\sqcup} (unit_{\sqcup = \sqcup} *, _{\sqcup} fmt_{\sqcup = \sqcup} \setminus "(A) \setminus ")_{\sqcup} @, ";
         printf "\"derived_parameters:\""; nl ();
         List.iter (print_echo "real") declarations.real_singles;
         List.iter (print_echo "complex") declarations.complex_singles;
         List.iter (print_echo_array "real") declarations.real_arrays;
         List.iter (print_echo_array "complex") declarations.complex_arrays;
         printf "⊔⊔end⊔subroutine⊔print_parameters"; nl ();
       end:
       printf "end_module_\%s" !parameter_module; nl ()
                                              Run-Time Diagnostics
    type diagnostic = All | Arguments | Momenta | Gauge
    type diagnostic\_mode = Off \mid Warn \mid Panic
    \mathsf{let} \ warn \ mode \ =
       match !mode with
         \mathit{Off} \ 	o \ \mathsf{false}
         Warn \rightarrow {\sf true}
        Panic \rightarrow \mathsf{true}
    let panic mode =
       match !mode with
         O\!f\!f \ 	o \ \mathsf{false}
         Warn \rightarrow \mathsf{false}
         Panic \rightarrow true
    let suffix mode =
       if panic mode then
          "panic"
       else
          "warn"
    let diagnose\_arguments = ref Off
    let diagnose\_momenta = ref Off
    let diagnose\_gauge = ref Off
    let rec parse\_diagnostic = function
       \mid All, panic \rightarrow
            parse_diagnostic (Arguments, panic);
            parse\_diagnostic\ (Momenta,\ panic);
            parse_diagnostic (Gauge, panic)
       \mid Arguments, panic \rightarrow
            diagnose\_arguments := if panic then Panic else Warn
       | Momenta, panic \rightarrow
            diagnose\_momenta := if panic then Panic else Warn
       \mid Gauge, panic \rightarrow
            diagnose\_gauge := if panic then Panic else Warn
If diagnostics are required, we have to switch off Fortran95 features like pure functions.
    let parse\_diagnostics = function
```

```
| [] \rightarrow ()
    diagnostics \rightarrow
       fortran95 := false;
       List.iter parse_diagnostic diagnostics
                                                  Amplitude
let declare\_momenta\_chunk = function
   | [] \rightarrow ()
    momenta \rightarrow
       printf "_{\sqcup\sqcup\sqcup\sqcup}@[<2>type(momentum)_{\sqcup}::_{\sqcup}";
       print_list (List.map format_momentum momenta); nl ()
let declare\_momenta = function
   [] \rightarrow ()
   \mid momenta \rightarrow
       List.iter
          declare\_momenta\_chunk
          (ThoList.chopn declaration_chunk_size momenta)
let declare_wavefunctions multiplicity wfs =
  let wfs' = classify\_wfs \ wfs in
   declare_list multiplicity ("complex(kind=" ^ !kind ^ ")")
     (wfs'.scalars @ wfs'.brs_scalars);
   declare_list multiplicity ("type(" ^ Fermions.psi_type ^ ")")
     (wfs'.spinors @ wfs'.brs_spinors);
   declare_list multiplicity ("type(" ^ Fermions.psibar_type ^ ")")
     (wfs'.conjspinors @ wfs'.brs\_conjspinors);
   declare_list multiplicity ("type(" ^ Fermions.chi_type ^ ")")
     (wfs'.realspinors @ wfs'.brs_realspinors @ wfs'.ghostspinors);
   declare_list multiplicity ("type(" ^ Fermions.grav_type ^ ")") wfs'.vectorspinors;
   declare_list multiplicity "type(vector)" (wfs'.vectors @ wfs'.massive_vectors @
      wfs'.brs_vectors @ wfs'.brs_massive_vectors @ wfs'.ward_vectors);
   declare_list multiplicity "type(tensor2odd)" wfs'.tensors_1;
   declare_list multiplicity "type(tensor)" wfs'.tensors_2
let flavors \ a = F.incoming \ a @ F.outgoing \ a
let declare\_brakets\_chunk = function
   | [] \rightarrow ()
    amplitudes \rightarrow
       printf " \subseteq 0 << > complex(kind=%s) : : \subseteq "!kind;
       print\_list\ (List.map\ (fun\ a\ 	o\ flavors\_symbol\ 	ilde{} decl\ :true\ (flavors\ a))\ amplitudes);\ nl\ ()
let \ declare\_brakets = function
   | [] \rightarrow ()
   \mid amplitudes \rightarrow
       List.iter
          declare\_brakets\_chunk
          (ThoList.chopn declaration_chunk_size amplitudes)
let print_variable_declarations amplitudes =
  let \ multiplicity = CF.multiplicity \ amplitudes
  and processes = CF.processes amplitudes in
  if \neg !amp\_triv then begin
     declare\_momenta
       (PSet.elements
           (List.fold\_left
               (fun set \ a \rightarrow
                 PSet.union set (List.fold_right)
                                        (\text{fun } wf \rightarrow PSet.add (F.momentum\_list } wf))
                                        (F.externals \ a) \ PSet.empty))
               PSet.empty\ processes));
```

```
declare\_momenta
  (PSet.elements
      (List.fold_left
          (fun set \ a \rightarrow
            PSet.union set (List.fold_right)
                                  (\text{fun } wf \rightarrow PSet.add (F.momentum\_list } wf))
                                  (F.variables\ a)\ PSet.empty))
          PSet.empty\ processes));
if !openmp then begin
  printf "_u_type_\%s@[<2>" openmp_tld_type;
  nl();
end;
declare_wavefunctions multiplicity
  (WFSet.elements
      (List.fold\_left
         (fun set \ a \rightarrow
            WFSet.union set (List.fold_right WFSet.add (F.externals a) WFSet.empty))
          WFSet.empty processes));
declare_wavefunctions multiplicity
  (WFSet.elements
      (List.fold_left
          (fun set \ a \rightarrow
            WFSet.union\ set\ (List.fold\_right\ WFSet.add\ (F.variables\ a)\ WFSet.empty))
          WFSet.empty\ processes));
declare_brakets processes;
if !openmp then begin
  printf "@] ___end__type__%s\n" openmp\_tld\_type;
  printf "⊔⊔type(%s)⊔::⊔%s" openmp_tld_type openmp_tld;
  nl();
end;
```

 $print_current$ is the most important function that has to match the functions in omega95 (see appendix Y). It offers plentiful opportunities for making mistakes, in particular those related to signs. We start with a few auxiliary functions:

```
let children2 \ rhs =  match F.children \ rhs with | \ [wf1; \ wf2] \rightarrow (wf1, \ wf2)  | \ \_ \rightarrow failwith \ "Targets.children2: \_can't_lhappen" let children3 \ rhs =  match F.children \ rhs with | \ [wf1; \ wf2; \ wf3] \rightarrow (wf1, \ wf2, \ wf3)  | \ \_ \rightarrow invalid\_arg \ "Targets.children3: \_can't_lhappen"
```

Note that it is (marginally) faster to multiply the two scalar products with the coupling constant than the four vector components.



This could be part of omegalib as well ...



The following is error prone and should be generated automagically.

```
let print_vector4 c wf1 wf2 wf3 fusion (coeff, contraction) =
  match contraction, fusion with
     C_{-12-34}, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
     C_{-13-42}, (F241 | F421 | F243 | F423 | F132 | F312 | F134 | F314)
     C\_14\_23,\; (F231\mid F321\mid F234\mid F324\mid F142\mid F412\mid F143\mid F413)\;\to\;
       printf "((%s%s)*(%s*%s))*%s" (format\_coeff coeff) c wf1 wf2 wf3
     C_{-12}_{-34}, (F_{134} \mid F_{143} \mid F_{234} \mid F_{243} \mid F_{312} \mid F_{321} \mid F_{412} \mid F_{421})
     C_{-}13_{-}42, (F124 \mid F142 \mid F324 \mid F342 \mid F213 \mid F231 \mid F413 \mid F431)
    C_{-}14_{-}23, (F123 \mid F132 \mid F423 \mid F432 \mid F214 \mid F241 \mid F314 \mid F341) <math>\rightarrow
       printf "((%s%s)*(%s*%s))*%s" (format\_coeff coeff) c wf2 wf3 wf1
    C_{-12-34}, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241)
    C_{-13-42}, (F214 | F412 | F234 | F432 | F123 | F321 | F143 | F341)
   | C_{-}14_{-}23, (F213 | F312 | F243 | F342 | F124 | F421 | F134 | F431) \rightarrow
       printf "((%s%s)*(%s*%s))*%s" (format_coeff coeff) c wf1 wf3 wf2
let print_vector4_t_0 c wf1 p1 wf2 p2 wf3 p3 fusion (coeff, contraction) =
  match contraction, fusion with
     C_{-}12_{-}34, (F234 | F243 | F134 | F143 | F421 | F321 | F412 | F312)
     C_{-13}_{-42}, (F324 | F342 | F124 | F142 | F431 | F231 | F413 | F213)
   \mid C_{-}14_{-}23, (F423 \mid F432 \mid F123 \mid F132 \mid F341 \mid F241 \mid F314 \mid F214) \rightarrow
       printf "g_dim8g3_t_0(%s,%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2 wf3 p3
     C_{-12-34}, (F324 | F314 | F423 | F413 | F142 | F132 | F241 | F231)
    C_{-13}_42, (F234 \mid F214 \mid F432 \mid F412 \mid F143 \mid F123 \mid F341 \mid F321)
   \mid C_14_23, (F243 \mid F213 \mid F342 \mid F312 \mid F134 \mid F124 \mid F431 \mid F421) \rightarrow
       printf "g_dim8g3_t_0(%s,%s,%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1 wf3 p3
    C_{12}_{34}, (F342 | F341 | F432 | F431 | F124 | F123 | F214 | F213)
     C_{-13}_42, (F243 | F241 | F423 | F421 | F134 | F132 | F314 | F312)
    C_{-}14_{-}23, (F234 \mid F231 \mid F324 \mid F321 \mid F143 \mid F142 \mid F413 \mid F412) \rightarrow
       printf "g_dim8g3_t_0(%s,%s,%s,%s,%s,%s,%s)" c wf3 p3 wf1 p1 wf2 p2
let print\_vector 4\_t\_1 c wf1 p1 wf2 p2 wf3 p3 fusion (coeff, contraction) =
  match contraction, fusion with
     C_{-12-34}, (F234 | F243 | F134 | F143 | F421 | F321 | F412 | F312)
     C_{-}13_{-}42, (F324 | F342 | F124 | F142 | F431 | F231 | F413 | F213)
    C_{-}14_{-}23, (F423 \mid F432 \mid F123 \mid F132 \mid F341 \mid F241 \mid F314 \mid F214) <math>\rightarrow
       printf "g_dim8g3_t_1(%s,%s,%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2 wf3 p3
     C_{-12-34}, (F324 | F314 | F423 | F413 | F142 | F132 | F241 | F231)
     C_{-}13_{-}42, (F234 | F214 | F432 | F412 | F143 | F123 | F341 | F321)
    C_{-}14_{-}23, (F243 \mid F213 \mid F342 \mid F312 \mid F134 \mid F124 \mid F431 \mid F421) <math>\rightarrow
       printf "g_dim8g3_t_1(%s,%s,%s,%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1 wf3 p3
    C_{-}12_{-}34, (F342 | F341 | F432 | F431 | F124 | F123 | F214 | F213)
    C_{-}13_{-}42, (F243 | F241 | F423 | F421 | F134 | F132 | F314 | F312)
   \mid C_14_23, (F234 \mid F231 \mid F324 \mid F321 \mid F143 \mid F142 \mid F413 \mid F412) \rightarrow
       printf "g_dim8g3_t_1(%s,%s,%s,%s,%s,%s,%s,%s)" c wf3 p3 wf1 p1 wf2 p2
let print_vector4_t_2 c wf1 p1 wf2 p2 wf3 p3 fusion (coeff, contraction) =
  match contraction, fusion with
    C_{-12-34}, (F234 | F243 | F134 | F143 | F421 | F321 | F412 | F312)
     C_{-}13_{-}42, (F324 | F342 | F124 | F142 | F431 | F231 | F413 | F213)
   | C_{-}14_{-}23, (F_{4}23 | F_{4}32 | F_{1}23 | F_{1}32 | F_{3}41 | F_{2}41 | F_{3}14 | F_{2}14) \rightarrow
       printf "g_dim8g3_t_2(%s,%s,%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2 wf3 p3
    C_{-12-34}, (F324 | F314 | F423 | F413 | F142 | F132 | F241 | F231)
    C_{-}13_{-}42, (F234 | F214 | F432 | F412 | F143 | F123 | F341 | F321)
   \mid C_{-}14_{-}23, (F243 \mid F213 \mid F342 \mid F312 \mid F134 \mid F124 \mid F431 \mid F421) \rightarrow
       printf "g_dim8g3_t_2(%s,%s,%s,%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1 wf3 p3
    C_{-}12_{-}34, (F342 | F341 | F432 | F431 | F124 | F123 | F214 | F213)
    C_{-}13_{-}42, (F243 | F241 | F423 | F421 | F134 | F132 | F314 | F312)
   \mid C_{-}14_{-}23, (F234 \mid F231 \mid F324 \mid F321 \mid F143 \mid F142 \mid F413 \mid F412) \rightarrow
       printf \ \verb"g_dim8g3_t_2(%s,%s,%s,%s,%s,%s,%s)" \ \textit{c wf3 p3 wf1 p1 wf2 p2}
let print\_vector4\_m\_0 c wf1 p1 wf2 p2 wf3 p3 fusion (coeff, contraction) =
  match contraction, fusion with
   | C_12_34, (F234 | F243 | F134 | F143 | F421 | F321 | F412 | F312)
```

```
| C_{13}_{42}, (F324 | F342 | F124 | F142 | F431 | F231 | F413 | F213)
   C_{-}14_{-}23, (F423 \mid F432 \mid F123 \mid F132 \mid F341 \mid F241 \mid F314 \mid F214) <math>\rightarrow
       printf "g_dim8g3_m_0(%s,%s,%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2 wf3 p3
    C_{-12-34}, (F324 | F314 | F423 | F413 | F142 | F132 | F241 | F231)
    C_{-13}_42, (F234 | F214 | F432 | F412 | F143 | F123 | F341 | F321)
    C_{-}14_{-}23, (F243 \mid F213 \mid F342 \mid F312 \mid F134 \mid F124 \mid F431 \mid F421) <math>\rightarrow
       printf "g_dim8g3_m_0(%s,%s,%s,%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1 wf3 p3
    C_{-12-34}, (F342 \mid F341 \mid F432 \mid F431 \mid F124 \mid F123 \mid F214 \mid F213)
    C_{-}13_{-}42, (F243 | F241 | F423 | F421 | F134 | F132 | F314 | F312)
  \mid C_{-}14_{-}23, (F234 \mid F231 \mid F324 \mid F321 \mid F143 \mid F142 \mid F413 \mid F412) \rightarrow
       printf "g_dim8g3_m_0(%s,%s,%s,%s,%s,%s,%s,%s)" c wf3 p3 wf1 p1 wf2 p2
let print\_vector4\_m\_1 c wf1 p1 wf2 p2 wf3 p3 fusion (coeff, contraction) =
  match contraction, fusion with
    C_{12}34, (F234 | F243 | F134 | F143 | F421 | F321 | F412 | F312)
    C_{-13}_{-42}, (F324 | F342 | F124 | F142 | F431 | F231 | F413 | F213)
    C_{-}14_{-}23, (F423 \mid F432 \mid F123 \mid F132 \mid F341 \mid F241 \mid F314 \mid F214) <math>\rightarrow
       printf "g_dim8g3_m_1(%s,%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2 wf3 p3
   C_{-12}_{-34}, (F324 | F314 | F423 | F413 | F142 | F132 | F241 | F231)
    C_{-13-42}, (F234 | F214 | F432 | F412 | F143 | F123 | F341 | F321)
  \mid C_{-}14_{-}23, (F243 \mid F213 \mid F342 \mid F312 \mid F134 \mid F124 \mid F431 \mid F421) \rightarrow
       printf "g_dim8g3_m_1(%s,%s,%s,%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1 wf3 p3
    C_{-12-34}, (F342 \mid F341 \mid F432 \mid F431 \mid F124 \mid F123 \mid F214 \mid F213)
    C_{-}13_{-}42, (F243 \mid F241 \mid F423 \mid F421 \mid F134 \mid F132 \mid F314 \mid F312)
  |C_14_23, (F234 | F231 | F324 | F321 | F143 | F142 | F413 | F412) \rightarrow
       printf "g_dim8g3_m_1(%s,%s,%s,%s,%s,%s,%s,%s)" c wf3 p3 wf1 p1 wf2 p2
 let print_vector4_m_7 c wf1 p1 wf2 p2 wf3 p3 fusion (coeff, contraction) =
  match contraction, fusion with
    C_{-12-34}, (F234 | F243 | F134 | F143 | F421 | F321 | F412 | F312)
    C_{-13}_42, (F324 | F342 | F124 | F142 | F431 | F231 | F413 | F213)
    C_{-}14_{-}23, (F423 \mid F432 \mid F123 \mid F132 \mid F341 \mid F241 \mid F314 \mid F214) <math>\rightarrow
       printf "g_dim8g3_m_7(%s,%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2 wf3 p3
    C_{12}34, (F324 | F314 | F423 | F413 | F142 | F132 | F241 | F231)
    C_{-}13_{-}42, (F234 | F214 | F432 | F412 | F143 | F123 | F341 | F321)
    C_{-}14_{-}23, (F243 \mid F213 \mid F342 \mid F312 \mid F134 \mid F124 \mid F431 \mid F421) <math>\rightarrow
       printf "g_dim8g3_m_7(%s,%s,%s,%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1 wf3 p3
    C_12_34, (F342 | F341 | F432 | F431 | F124 | F123 | F214 | F213)
    C_13_42, (F243 | F241 | F423 | F421 | F134 | F132 | F314 | F312)
  \mid C_{-}14_{-}23, (F234 \mid F231 \mid F324 \mid F321 \mid F143 \mid F142 \mid F413 \mid F412) \rightarrow
       printf "g_dim8g3_m_7(%s,%s,%s,%s,%s,%s,%s,%s)" c wf3 p3 wf1 p1 wf2 p2
let print_add_vector4 c wf1 wf2 wf3 fusion (coeff, contraction) =
  printf "@_{\sqcup}+_{\sqcup}";
  print_vector4 c wf1 wf2 wf3 fusion (coeff, contraction)
let print_vector4_km c pa pb wf1 wf2 wf3 fusion (coeff, contraction) =
  match contraction, fusion with
    C_{-}12_{-}34, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
    C_{-13}_{-42}, (F241 | F421 | F243 | F423 | F132 | F312 | F134 | F314)
    C_{-}14_{-}23, (F231 \mid F321 \mid F234 \mid F324 \mid F142 \mid F412 \mid F143 \mid F413) <math>\rightarrow
       printf "((%s%s%s+%s))*(%s*%s))*%s"
          (format_coeff coeff) c pa pb wf1 wf2 wf3
    C_{-}12_{-}34, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421)
    C_{-}13_{-}42, (F124 | F142 | F324 | F342 | F213 | F231 | F413 | F431)
  | C_{-}14_{-}23, (F123 | F132 | F423 | F432 | F214 | F241 | F314 | F341) \rightarrow
       printf "((%s%s%s+%s))*(%s*%s))*%s"
          (format_coeff coeff) c pa pb wf2 wf3 wf1
  | C_12_34, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241)
   C_{-}13_{-}42, (F214 | F412 | F234 | F432 | F123 | F321 | F143 | F341)
  \mid C_{-}14_{-}23, (F213 \mid F312 \mid F243 \mid F342 \mid F124 \mid F421 \mid F134 \mid F431) \rightarrow
       printf "((%s%s%s+%s))*(%s*%s))*%s"
          (format\_coeff\ coeff)\ c\ pa\ pb\ wf1\ wf3\ wf2
```

```
let print\_vector4\_km\_t\_0 c pa pb wf1 p1 wf2 p2 wf3 p3 fusion (coeff, contraction) =
  match contraction, fusion with
    C_{-12-34}, (F234 | F243 | F134 | F143 | F421 | F321 | F412 | F312)
    C_{-13-42}, (F324 | F342 | F124 | F142 | F431 | F231 | F413 | F213)
    C\_14\_23, (F423 \mid F432 \mid F123 \mid F132 \mid F341 \mid F241 \mid F314 \mid F214) <math>\rightarrow
       printf \ "@[(%s%s%s+%s)*g\_dim8g3\_t\_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
    C_{-}12_{-}34, (F324 \mid F314 \mid F423 \mid F413 \mid F142 \mid F132 \mid F241 \mid F231)
    C-13-42, (F234 | F214 | F432 | F412 | F143 | F123 | F341 | F321)
  \mid C_{-}14_{-}23, (F243 \mid F213 \mid F342 \mid F312 \mid F134 \mid F124 \mid F431 \mid F421) \rightarrow
       printf "@[(%s%s%s+%s)*g_dim8g3_t_0(cmplx(1,kind=default), @_\%s, %s, %s, %s, %s, %s, %s))@]"
          (format_coeff coeff) c pa pb wf2 p2 wf1 p1 wf3 p3
   C_{-12-34}, (F342 | F341 | F432 | F431 | F124 | F123 | F214 | F213)
    C_{-}13_{-}42, (F243 | F241 | F423 | F421 | F134 | F132 | F314 | F312)
  \mid C_{-}14_{-}23, (F234 \mid F231 \mid F324 \mid F321 \mid F143 \mid F142 \mid F413 \mid F412) \rightarrow
       printf "@[(%s%s%s+%s)*g_dim8g3_t_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf3 p3 wf1 p1 wf2 p2
let print\_vector4\_km\_t\_1 c pa pb wf1 p1 wf2 p2 wf3 p3 fusion (coeff, contraction) =
  match contraction, fusion with
    C_{-12-34}, (F234 | F243 | F134 | F143 | F421 | F321 | F412 | F312)
    C_{-13}_42, (F324 | F342 | F124 | F142 | F431 | F231 | F413 | F213)
  \mid C_{-}14_{-}23, (F423 \mid F432 \mid F123 \mid F132 \mid F341 \mid F241 \mid F314 \mid F214) \rightarrow
       printf \ "@[(%s%s%s+%s)*g\_dim8g3\_t_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
          (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
    C_{-12-34}, (F324 \mid F314 \mid F423 \mid F413 \mid F142 \mid F132 \mid F241 \mid F231)
    C_{-}13_{-}42, (F234 | F214 | F432 | F412 | F143 | F123 | F341 | F321)
    C_{-}14_{-}23, (F243 \mid F213 \mid F342 \mid F312 \mid F134 \mid F124 \mid F431 \mid F421) <math>\rightarrow
       printf "@[(%s%s%s+%s)*g_dim8g3_t_1(cmplx(1,kind=default), @_%s, %s, %s, %s, %s, %s))@]"
         (format_coeff coeff) c pa pb wf2 p2 wf1 p1 wf3 p3
    C_{12_{34}}, (F342 | F341 | F432 | F431 | F124 | F123 | F214 | F213)
    C_{-}13_{-}42, (F243 | F241 | F423 | F421 | F134 | F132 | F314 | F312)
  \mid C_{-}14_{-}23, (F234 \mid F231 \mid F324 \mid F321 \mid F143 \mid F142 \mid F413 \mid F412) \rightarrow
       printf "@[(%s%s%s+%s)*g_dim8g3_t_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf3 p3 wf1 p1 wf2 p2
let print\_vector4\_km\_t\_2 c pa pb wf1 p1 wf2 p2 wf3 p3 fusion (coeff, contraction) =
  match contraction, fusion with
    C_{12}34, (F234 | F243 | F134 | F143 | F421 | F321 | F412 | F312)
    C_{-13}_42, (F324 | F342 | F124 | F142 | F431 | F231 | F413 | F213)
    C_{-}14_{-}23, (F423 \mid F432 \mid F123 \mid F132 \mid F341 \mid F241 \mid F314 \mid F214) <math>\rightarrow
       printf "@[(%s%s%s+%s)*g_dim8g3_t_2(cmplx(1,kind=default),0_\%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
   C_12_34, (F324 | F314 | F423 | F413 | F142 | F132 | F241 | F231)
    C_{-}13_{-}42, (F234 | F214 | F432 | F412 | F143 | F123 | F341 | F321)
  \mid C_{-}14_{-}23, (F243 \mid F213 \mid F342 \mid F312 \mid F134 \mid F124 \mid F431 \mid F421) \rightarrow
       printf "@[(%s%s%s+%s)*g_dim8g3_t_2(cmplx(1,kind=default), @_%s, %s, %s, %s, %s, %s))@]"
          (format_coeff coeff) c pa pb wf2 p2 wf1 p1 wf3 p3
  | C_12_34, (F342 | F341 | F432 | F431 | F124 | F123 | F214 | F213)
    C_{-}13_{-}42, (F243 | F241 | F423 | F421 | F134 | F132 | F314 | F312)
  \mid C_{-}14_{-}23, (F234 \mid F231 \mid F324 \mid F321 \mid F143 \mid F142 \mid F413 \mid F412) \rightarrow
       printf "@[(%s%s%s+%s)*g_dim8g3_t_2(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf3 p3 wf1 p1 wf2 p2
let print_vector4_km_t_rsi c pa pb pc wf1 p1 wf2 p2 wf3 p3 fusion (coeff, contraction) =
  match contraction, fusion with
    C_{12}_{34}, (F234 | F243 | F134 | F143 | F421 | F321 | F412 | F312)
    C_{-13}_42, (F324 | F342 | F124 | F142 | F431 | F231 | F413 | F213)
  \mid C_{-}14_{-}23, (F423 \mid F432 \mid F123 \mid F132 \mid F341 \mid F241 \mid F314 \mid F214) \rightarrow
       printf \ "@[(%s%s%s+%s)*g\_dim8g3\_t\_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
          (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
   \mid C_{-}12_{-}34, (F324 \mid F314 \mid F423 \mid F413 \mid F142 \mid F132 \mid F241 \mid F231)
  | C_{-}13_{-}42, (F234 | F214 | F432 | F412 | F143 | F123 | F341 | F321)
```

```
|C_{-}14_{-}23|, (F243 | F213 | F342 | F312 | F134 | F124 | F431 | F421) <math>\rightarrow
             printf "@[(%s%s%s+%s)*g_dim8g3_t_0(cmplx(1,kind=default), @_%s,%s,%s,%s,%s,%s,%s))*((%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s)*(%s+%s+%s)*(%s+%s+%s)*(%s+%s+(s+%s)*(s+%s)*(s+%s)*(%s+(s+%s)*(s+%s)*(s+%s)*(s+%s)*(s+%s+(s+%s)*(s+%s)*(s+%s)*(s+%s)*(s+%
                  (format_coeff coeff) c pa pb wf2 p2 wf1 p1 wf3 p3 pa pb pa pb pc pb pc
        C_{-12} 34, (F342 \mid F341 \mid F432 \mid F431 \mid F124 \mid F123 \mid F214 \mid F213)
        C_{-}13_{-}42, \; (F243 \mid F241 \mid F423 \mid F421 \mid F134 \mid F132 \mid F314 \mid F312)
     \mid C_{-}14_{-}23, (F234 \mid F231 \mid F324 \mid F321 \mid F143 \mid F142 \mid F413 \mid F412) \rightarrow
             printf \ "@[(%s\%s\%s+\%s)*g\_dim8g3\_t\_0(cmplx(1,kind=default), @_\%s,\%s,\%s,\%s,\%s,\%s,\%s))*((%s+\%s)*(\%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s)*(%s+\%s
                  (format_coeff coeff) c pa pb wf3 p3 wf1 p1 wf2 p2 pa pb pa pb pa pc pa pc
let print\_vector4\_km\_m\_0 c pa pb wf1 p1 wf2 p2 wf3 p3 fusion (coeff, contraction) =
    match contraction, fusion with
         C_{-12-34}, (F234 \mid F243 \mid F134 \mid F143 \mid F421 \mid F321 \mid F412 \mid F312)
         C_{-13}_42, (F324 | F342 | F124 | F142 | F431 | F231 | F413 | F213)
       C\_14\_23, (F423 \mid F432 \mid F123 \mid F132 \mid F341 \mid F241 \mid F314 \mid F214) <math>
ightarrow
             if (String.contains \ c \ 'w' \ \lor \ String.contains \ c \ '4') then
                    (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
                    printf "@[((%s%s%s+%s))*g_dim8g3_m_0(cmplx(costhw**(-2),kind=default),cmplx(costhw**2,kind=default)
                            (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
      C_{-}12_{-}34, (F324 | F314 | F423 | F413 | F142 | F132 | F241 | F231)
        C_{-13}_42, (F234 \mid F214 \mid F432 \mid F412 \mid F143 \mid F123 \mid F341 \mid F321)
     \mid C_{-}14_{-}23, (F243 \mid F213 \mid F342 \mid F312 \mid F134 \mid F124 \mid F431 \mid F421) \rightarrow
             if (String.contains \ c \ `w` \lor String.contains \ c \ `4`) then
                    (format_coeff coeff) c pa pb wf2 p2 wf1 p1 wf3 p3
             else
                    printf "@[(%s%s%s+%s)*g_dim8g3_m_0(cmplx(costhw**(-2),kind=default),cmplx(costhw**2,kind=def
                            (format_coeff coeff) c pa pb wf2 p2 wf1 p1 wf3 p3
        C_{-12-34}, (F342 \mid F341 \mid F432 \mid F431 \mid F124 \mid F123 \mid F214 \mid F213)
         C_{-}13_{-}42, (F243 \mid F241 \mid F423 \mid F421 \mid F134 \mid F132 \mid F314 \mid F312)
       C_{-}14_{-}23, (F234 \mid F231 \mid F324 \mid F321 \mid F143 \mid F142 \mid F413 \mid F412) <math>\rightarrow
             if (String.contains \ c \ 'w' \ \lor \ String.contains \ c \ '4') then
                    printf "@[(%s%s%s+%s)*g_dim8g3_m_0(cmplx(1,kind=default),cmplx(1,kind=default),@_%s,%s,%s,%s,%s
                            (format\_coeff\ coeff)\ c\ pa\ pb\ wf3\ p3\ wf1\ p1\ wf2\ p2
             else
                    printf "@[(%s%s%s+%s)*g_dim8g3_m_0(cmplx(costhw**(-2),kind=default),cmplx(costhw**2,kind=def
                            (format_coeff coeff) c pa pb wf3 p3 wf1 p1 wf2 p2
let print_vector4_km_m_1 c pa pb wf1 p1 wf2 p2 wf3 p3 fusion (coeff, contraction) =
    match contraction, fusion with
         C_{-12-34}, (F234 | F243 | F134 | F143 | F421 | F321 | F412 | F312)
         C_{-13}_{-42}, (F324 | F342 | F124 | F142 | F431 | F231 | F413 | F213)
    \mid C_{-}14_{-}23, (F423 \mid F432 \mid F123 \mid F132 \mid F341 \mid F241 \mid F314 \mid F214) \rightarrow
             if (String.contains \ c \ `w' \ \lor \ String.contains \ c \ `4') then
                    printf "@[(%s%s%s+%s)*g_dim8g3_m_1(cmplx(1,kind=default),cmplx(1,kind=default),@_{\square}%s,%s,%s,%s,%s
                            (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
             else
                    printf "@[(%s%s%s+%s)*g_dim8g3_m_1(cmplx(costhw**(-2),kind=default),cmplx(costhw**2,kind=def
                            (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
      C_{-}12_{-}34, (F324 \mid F314 \mid F423 \mid F413 \mid F142 \mid F132 \mid F241 \mid F231)
         C_{-13-42}, (F234 | F214 | F432 | F412 | F143 | F123 | F341 | F321)
      C\_14\_23, (F243 \mid F213 \mid F342 \mid F312 \mid F134 \mid F124 \mid F431 \mid F421) <math>\rightarrow
             if (String.contains \ c \ `w' \ \lor \ String.contains \ c \ `4') then
                    (format_coeff coeff) c pa pb wf2 p2 wf1 p1 wf3 p3
                    printf "@[(%s%s%s+%s)*g_dim8g3_m_1(cmplx(costhw**(-2),kind=default),cmplx(costhw**2,kind=def
                            (format_coeff coeff) c pa pb wf2 p2 wf1 p1 wf3 p3
     | C_{12}_{34}, (F_{342} | F_{341} | F_{432} | F_{431} | F_{124} | F_{123} | F_{214} | F_{213})
        C-13-42, (F243 | F241 | F423 | F421 | F134 | F132 | F314 | F312)
     \mid C_14_23, (F234 \mid F231 \mid F324 \mid F321 \mid F143 \mid F142 \mid F413 \mid F412) \rightarrow
```

```
if (String.contains \ c \ 'w' \ \lor \ String.contains \ c \ '4') then
                printf "@[(%s%s%s+%s)*g_dim8g3_m_1(cmplx(1,kind=default),cmplx(1,kind=default),@_%s,%s,%s,%s,%s
                      (format_coeff coeff) c pa pb wf3 p3 wf1 p1 wf2 p2
          else
                printf "@[(%s%s%s+%s)*g_dim8g3_m_1(cmplx(costhw**(-2),kind=default),cmplx(costhw**2,kind=def
                      (format_coeff coeff) c pa pb wf3 p3 wf1 p1 wf2 p2
let print_vector4_km_m_7 c pa pb wf1 p1 wf2 p2 wf3 p3 fusion (coeff, contraction) =
   match contraction, fusion with
       C_{-}12_{-}34, (F234 | F243 | F134 | F143 | F421 | F321 | F412 | F312)
       C_{-}13_{-}42, (F324 | F342 | F124 | F142 | F431 | F231 | F413 | F213)
      C\_14\_23,\; (F423\mid F432\mid F123\mid F132\mid F341\mid F241\mid F314\mid F214)\;\to\;
          if (String.contains \ c \ `w` \lor String.contains \ c \ `4`) then
                printf "@[(%s%s%s+%s)*@_g_dim8g3_m_7(cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default)
                      (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
          else
                printf "@[(%s%s%s+%s)*@_g_dim8g3_m_7(cmplx(costhw**(-2),kind=default),cmplx(1,kind=default),
                      (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
    | C_12_34, (F324 | F314 | F423 | F413 | F142 | F132 | F241 | F231)
      C_{-13}_42, (F234 \mid F214 \mid F432 \mid F412 \mid F143 \mid F123 \mid F341 \mid F321)
   |C_{-}14_{-}23|, (F243 | F213 | F342 | F312 | F134 | F124 | F431 | F421) <math>\rightarrow
          if (String.contains \ c \ `w' \ \lor \ String.contains \ c \ `4') then
                printf "@[(%s%s%s+%s)*@_g_dim8g3_m_7(cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default)
                      (format_coeff coeff) c pa pb wf2 p2 wf1 p1 wf3 p3
          else
                printf "@[(%s%s%s+%s)*@_g_dim8g3_m_7(cmplx(costhw**(-2),kind=default),cmplx(1,kind=default),
                      (format_coeff coeff) c pa pb wf2 p2 wf1 p1 wf3 p3
      C_{-12-34}, (F342 \mid F341 \mid F432 \mid F431 \mid F124 \mid F123 \mid F214 \mid F213)
       C_{-13}_{-42}, (F243 | F241 | F423 | F421 | F134 | F132 | F314 | F312)
      C_{-}14_{-}23, (F234 \mid F231 \mid F324 \mid F321 \mid F143 \mid F142 \mid F413 \mid F412) <math>\rightarrow
          if (String.contains \ c \ `w' \ \lor \ String.contains \ c \ `4') then
                printf "@[(%s%s%s+%s)*@_g_dim8g3_m_7(cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default)]
                      (format_coeff coeff) c pa pb wf3 p3 wf1 p1 wf2 p2
                printf "@[(%s%s%s+%s)*@_g_dim8g3_m_7(cmplx(costhw**(-2),kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=default),cmplx(1,kind=def
                      (format_coeff coeff) c pa pb wf3 p3 wf1 p1 wf2 p2
let print_add_vector4_km c pa pb wf1 wf2 wf3 fusion (coeff, contraction) =
   printf \ "@_{\sqcup} +_{\sqcup} ";
   print_vector4_km c pa pb wf1 wf2 wf3 fusion (coeff, contraction)
let print_dscalar4 c wf1 wf2 wf3 p1 p2 p3 p123
       fusion (coeff, contraction) =
    match contraction, fusion with
      C_{-}12_{-}34, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
      C_{-}13_{-}42, (F241 \mid F421 \mid F243 \mid F423 \mid F132 \mid F312 \mid F134 \mid F314)
    \mid C_{-}14_{-}23, (F231 \mid F321 \mid F234 \mid F324 \mid F142 \mid F412 \mid F143 \mid F413) \rightarrow
          printf "((%s%s)*(%s*%s)*(%s*%s)*%s*%s*%s)"
              (format_coeff coeff) c p1 p2 p3 p123 wf1 wf2 wf3
    | C_{-}12_{-}34, (F_{1}34 | F_{1}43 | F_{2}34 | F_{2}43 | F_{3}12 | F_{3}21 | F_{4}12 | F_{4}21)
     C_{-}13_{-}42, (F124 | F142 | F324 | F342 | F213 | F231 | F413 | F431)
   \mid C_{-}14_{-}23, (F123 \mid F132 \mid F423 \mid F432 \mid F214 \mid F241 \mid F314 \mid F341) \rightarrow
          printf "((%s%s)*(%s*%s)*(%s*%s)*%s*%s*%s)"
               (format_coeff coeff) c p2 p3 p1 p123 wf1 wf2 wf3
    | C_12_34, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241)
      C_13_42, (F214 | F412 | F234 | F432 | F123 | F321 | F143 | F341)
      C_{-}14_{-}23, (F213 \mid F312 \mid F243 \mid F342 \mid F124 \mid F421 \mid F134 \mid F431) <math>\rightarrow
          printf "((%s%s)*(%s*%s)*(%s*%s)*%s*%s*%s)"
              (format_coeff coeff) c p1 p3 p2 p123 wf1 wf2 wf3
let print_add_dscalar4 c wf1 wf2 wf3 p1 p2 p3 p123
       fusion (coeff, contraction) =
   printf "@_{\sqcup}+_{\sqcup}";
```

```
print_dscalar4 c wf1 wf2 wf3 p1 p2 p3 p123 fusion (coeff, contraction)
let print_dscalar2_vector2 c wf1 wf2 wf3 p1 p2 p3 p123 fusion (coeff, contraction) =
  match contraction, fusion with
  \mid C_{1234}, (F_{123} \mid F_{213} \mid F_{124} \mid F_{214}) \rightarrow
       printf "(%s%s)*(%s*%s)*(%s*%s)*%s"
          (format_coeff coeff) c p1 p2 wf1 wf2 wf3
  \mid C_{-}12_{-}34, (F134 \mid F143 \mid F234 \mid F243) \rightarrow
       printf "(%s%s)*(%s*%s)*(%s*%s)*%s"
          (format_coeff coeff) c p1 p123 wf2 wf3 wf1
  | C_{-}12_{-}34, (F132 | F231 | F142 | F241) \rightarrow
       printf "(%s%s)*(%s*%s)*(%s*%s)*%s"
          (format_coeff coeff) c p1 p3 wf1 wf3 wf2
  \mid C_{-}12_{-}34, (F312 \mid F321 \mid F412 \mid F421) \rightarrow
       printf "(%s%s)*(%s*%s)*(%s*%s)*%s"
          (format_coeff coeff) c p2 p3 wf2 wf3 wf1
  | C_{-}12_{-}34, (F314 | F413 | F324 | F423) \rightarrow
       printf "(%s%s)*(%s*%s)*(%s*%s)*%s"
          (format_coeff coeff) c p2 p123 wf1 wf3 wf2
  \mid \ C\_12\_34, \ (F341 \mid F431 \mid F342 \mid F432) \ \rightarrow
       printf "(%s%s)*(%s*%s)*(%s*%s)*%s"
          (format_coeff coeff) c p3 p123 wf1 wf2 wf3
    C_{-}13_{-}42, (F123 \mid F214)
  \mid C_114_23, (F124 \mid F213) \rightarrow
       printf "((%s%s)*(%s*%s*%s)*%s*%s)"
          (format_coeff coeff) c wf1 p1 wf3 wf2 p2
    C_{-}13_{-}42, (F124 | F213)
    C_{-}14_{-}23, (F123 \mid F214) \rightarrow
       printf "((%s%s)*(%s*%s*%s)*%s*%s)"
          (format_coeff coeff) c wf2 p2 wf3 wf1 p1
    C_{-}13_{-}42, (F132 \mid F241)
  \mid C_{-}14_{-}23, (F142 \mid F231) \rightarrow
       printf "((%s%s)*(%s*%s*%s)*%s*%s)"
          (format_coeff coeff) c wf1 p1 wf2 wf3 p3
  | C_{13}_{42}, (F_{142} | F_{231})
  | C_{-}14_{-}23, (F132 | F241) \rightarrow
       printf "((%s%s)*(%s*%s*%s)*%s*%s)"
          (format_coeff coeff) c wf3 p3 wf2 wf1 p1
  | C_13_42, (F312 | F421)
  | C_{-}14_{-}23, (F412 | F321) \rightarrow
       printf "((%s%s)*(%s*%s*%s)*%s*%s)"
          (format\_coeff\ coeff)\ c\ wf2\ p2\ wf1\ wf3\ p3
   C_{13}_{42}, (F321 | F412)
   \mid C_{-}14_{-}23, (F421 \mid F312) \rightarrow
       printf "((%s%s)*(%s*%s*%s)*%s*%s)"
          (format_coeff coeff) c wf3 p3 wf1 wf2 p2
   C_{13}_{42}, (F_{134} | F_{243})
  | C_{-}14_{-}23, (F143 | F234) \rightarrow
       printf "((%s%s)*(%s*%s)*(%s*%s*%s))"
          (format_coeff coeff) c wf3 p123 wf1 p1 wf2
  | C_{-}13_{-}42, (F143 | F234)
  | C_{14}_{23}, (F134 | F243) \rightarrow
       printf "((%s%s)*(%s*%s)*(%s*%s*%s))"
          (format_coeff coeff) c wf2 p123 wf1 p1 wf3
  | C_{13}_{42}, (F314 | F423)
  | C_{-}14_{-}23, (F413 | F324) \rightarrow
       printf "((%s%s)*(%s*%s)*(%s*%s*%s))"
          (format_coeff coeff) c wf3 p123 wf2 p2 wf1
   C_{-}13_{-}42, (F324 \mid F413)
  \mid C_{-}14_{-}23, (F423 \mid F314) \rightarrow
       printf "((%s%s)*(%s*%s)*(%s*%s*%s))"
```

```
(format_coeff coeff) c wf1 p123 wf2 p2 wf3
  | C_{-}13_{-}42, (F341 | F432)
  \mid C_{-}14_{-}23, (F431 \mid F342) \rightarrow
       printf "((%s%s)*(%s*%s)*(%s*%s*%s))"
         (format_coeff coeff) c wf2 p123 wf3 p3 wf1
    C_{13}_{42}, (F342 \mid F431)
  | C_{-}14_{-}23, (F432 | F341) \rightarrow
       printf "((%s%s)*(%s*%s)*(%s*%s*%s))"
         (format_coeff coeff) c wf1 p123 wf3 p3 wf2
let print_add_dscalar2_vector2 c wf1 wf2 wf3 p1 p2 p3 p123
    fusion (coeff, contraction) =
  printf "@_{\sqcup}+_{\sqcup}";
  print_dscalar2_vector2 c wf1 wf2 wf3 p1 p2 p3 p123
    fusion (coeff, contraction)
let print_dscalar2_vector2_km c pa pb wf1 wf2 wf3 p1 p2 p3 p123 fusion (coeff, contraction) =
  match contraction, fusion with
    C_{-}12_{-}34, (F123 | F213 | F124 | F214) \rightarrow
       printf "(%s%s%s+%s))*(%s*%s)*(%s*%s)*%s"
         (format_coeff coeff) c pa pb p1 p2 wf1 wf2 wf3
  | C_{-}12_{-}34, (F134 | F143 | F234 | F243) \rightarrow
       printf "(%s%s%s+%s))*(%s*%s)*(%s*%s)*%s"
         (format_coeff coeff) c pa pb p1 p123 wf2 wf3 wf1
  \mid C_{-}12_{-}34, (F132 \mid F231 \mid F142 \mid F241) \rightarrow
       printf "(%s%s%s+%s))*(%s*%s)*(%s*%s)*%s"
         (format_coeff coeff) c pa pb p1 p3 wf1 wf3 wf2
  | C_{-}12_{-}34, (F312 | F321 | F412 | F421) \rightarrow
       printf "(%s%s%s+%s))*(%s*%s)*(%s*%s)*%s"
         (format\_coeff\ coeff)\ c\ pa\ pb\ p2\ p3\ wf2\ wf3\ wf1
  \mid C_{-}12_{-}34, (F314 \mid F413 \mid F324 \mid F423) \rightarrow
       printf "(%s%s%s+%s))*(%s*%s)*(%s*%s)*%s"
         (format_coeff coeff) c pa pb p2 p123 wf1 wf3 wf2
  | C_{12}_{34}, (F_{341} | F_{431} | F_{342} | F_{432}) \rightarrow
       printf "(%s%s%s+%s))*(%s*%s)*(%s*%s)*%s"
         (format_coeff coeff) c pa pb p3 p123 wf1 wf2 wf3
    C_{-}13_{-}42, (F123 \mid F214)
  \mid C_{-}14_{-}23, (F124 \mid F213) \rightarrow
       printf \ "((\%s\%s\%s+\%s))*(\%s*\%s*\%s)*\%s*\%s)"
         (format_coeff coeff) c pa pb wf1 p1 wf3 wf2 p2
  | C_{13}_{42}, (F124 | F213)
  | C_{-}14_{-}23, (F123 | F214) \rightarrow
       printf "((%s%s%s+%s))*(%s*%s*%s)*%s*%s)"
          (format_coeff coeff) c pa pb wf2 p2 wf3 wf1 p1
  | C_13_42, (F132 | F241)
  \mid C_{-}14_{-}23, (F142 \mid F231) \rightarrow
       printf "((%s%s%s+%s))*(%s*%s*%s)*%s*%s)"
         (format_coeff coeff) c pa pb wf1 p1 wf2 wf3 p3
  | C_{-}13_{-}42, (F142 | F231)
   \mid C_{-}14_{-}23, (F132 \mid F241) \rightarrow
       printf "((%s%s%s+%s))*(%s*%s*%s)*%s*%s)"
         (format_coeff coeff) c pa pb wf3 p3 wf2 wf1 p1
   C_{-}13_{-}42, (F312 \mid F421)
  | C_{14}_{23}, (F412 | F321) \rightarrow
       printf "((%s%s%s+%s))*(%s*%s*%s)*%s*%s)"
         (format_coeff coeff) c pa pb wf2 p2 wf1 wf3 p3
  | C_{13}_{42}, (F321 | F412)
  | C_{-}14_{-}23, (F421 | F312) \rightarrow
       printf "((%s%s%s+%s))*(%s*%s*%s)*%s*%s)"
         (format_coeff coeff) c pa pb wf3 p3 wf1 wf2 p2
    C_{-}13_{-}42, (F134 | F243)
  \mid C_{-}14_{-}23, (F143 \mid F234) \rightarrow
```

```
printf "((%s%s%s+%s))*(%s*%s)*(%s*%s*%s))"
         (format_coeff coeff) c pa pb wf3 p123 wf1 p1 wf2
  | C_{-}13_{-}42, (F_{1}43 | F_{2}34)
  C_{-}14_{-}23, (F134 \mid F243) \rightarrow
       printf "((%s%s%s+%s))*(%s*%s)*(%s*%s*%s))"
         (format_coeff coeff) c pa pb wf2 p123 wf1 p1 wf3
    C_{-}13_{-}42, (F314 \mid F423)
  | C_{-}14_{-}23, (F413 | F324) \rightarrow
       printf "((%s%s%s+%s))*(%s*%s)*(%s*%s*%s))"
         (format_coeff coeff) c pa pb wf3 p123 wf2 p2 wf1
  C_{13}_{42}, (F324 | F413)
  | C_{-}14_{-}23, (F423 | F314) \rightarrow
       printf "((%s%s%s+%s))*(%s*%s)*(%s*%s*%s))"
         (format_coeff coeff) c pa pb wf1 p123 wf2 p2 wf3
  | C_13_42, (F341 | F432)
  \mid C_{-}14_{-}23, (F431 \mid F342) \rightarrow
       printf "((%s%s%s+%s))*(%s*%s)*(%s*%s*%s))"
         (format_coeff coeff) c pa pb wf2 p123 wf3 p3 wf1
  | C_{13}_{42}, (F342 | F431)
   C_{-}14_{-}23, (F432 \mid F341) \rightarrow
       printf "((%s%s%s+%s))*(%s*%s)*(%s*%s*%s))"
         (format_coeff coeff) c pa pb wf1 p123 wf3 p3 wf2
let print_add_dscalar2_vector2_km c pa pb wf1 wf2 wf3 p1 p2 p3 p123 fusion (coeff, contraction) =
  printf "@<sub>□</sub>+<sub>□</sub>";
  print_dscalar2_vector2_km c pa pb wf1 wf2 wf3 p1 p2 p3 p123 fusion (coeff, contraction)
let print\_dscalar2\_vector2\_m\_0\_km c pa pb wf1 wf2 wf3 p1 p2 p3 fusion (coeff, contraction) =
  match contraction, fusion with
  \mid C_{-}12_{-}34, (F123 \mid F213 \mid F124 \mid F214) \rightarrow
       printf "@[((%s%s%s+%s))*v_phi2v_m_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
  | C_{-}12_{-}34, (F134 | F143 | F234 | F243) \rightarrow
       printf "@[((%s%s%s+%s))*phi_phi2v_m_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
  | C_{12_{34}}, (F_{132} | F_{231} | F_{142} | F_{241}) \rightarrow
       printf "@[((%s%s%s+%s))*v_phi2v_m_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf1 p1 wf3 p3 wf2 p2
  | C_{-}12_{-}34, (F312 | F321 | F412 | F421) \rightarrow
       printf "@[((%s%s%s+%s))*v_phi2v_m_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf3 p3 wf2 p2 wf1 p1
  | C_{-}12_{-}34, (F314 | F413 | F324 | F423) \rightarrow
       printf "@[((%s%s%s+%s))*phi_phi2v_m_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf2 p2 wf1 p1 wf3 p3
  | C_{12}_{34}, (F341 | F431 | F342 | F432) \rightarrow
       printf "@[((%s%s%s+%s))*phi_phi2v_m_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf3 p3 wf2 p2 wf1 p1
    C_{-}13_{-}42, (F123 \mid F214)
   C_{-}14_{-}23, (F124 \mid F213) \rightarrow
       printf "@[((%s%s%s+%s))*v_phi2v_m_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf1 p1 wf2 p3 wf3 p2
    C_{-}13_{-}42, (F124 \mid F213)
  C_14_23, (F123 \mid F214) \rightarrow
       printf "@[((%s%s%s+%s))*v_phi2v_m_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf2 p2 wf1 p3 wf3 p1
  | C_13_42, (F132 | F241)
  | C_{-}14_{-}23, (F_{1}42 | F_{2}31) \rightarrow
       printf "@[((%s%s%s+%s))*v_phi2v_m_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf1 p1 wf3 p2 wf2 p3
  | C_{13}_{42}, (F_{142} | F_{231})
  | C_{-}14_{-}23, (F132 | F241) \rightarrow
       printf \ "@[((\%s\%s\%s+\%s))*v\_phi2v\_m\_0(cmplx(1,kind=default),@_{\square}\%s,\%s,\%s,\%s,\%s,\%s))@]"
```

```
(format_coeff coeff) c pa pb wf3 p3 wf1 p2 wf2 p1
      C_{-}13_{-}42, (F312 \mid F421)
      \mid C_{-}14_{-}23, (F412 \mid F321) \rightarrow
            printf "@[((%s%s%s+%s))*v_phi2v_m_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
                (format_coeff coeff) c pa pb wf2 p2 wf3 p1 wf1 p3
       C_{-}13_{-}42, (F321 \mid F412)
     | C_{-}14_{-}23, (F421 | F312) \rightarrow
            printf "@[((%s%s%s+%s))*v_phi2v_m_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
                (format_coeff coeff) c pa pb wf3 p3 wf2 p1 wf1 p2
     | C_{-}13_{-}42, (F134 | F243)
     | C_{-}14_{-}23, (F143 | F234) \rightarrow
            printf \ "@[((\%s\%s\%s+\%s))*phi\_phi2v\_m\_0(cmplx(1,kind=default),@_{\sqcup}\%s,\%s,\%s,\%s,\%s,\%s))@]"
                (format_coeff coeff) c pa pb wf1 p3 wf3 p1 wf2 p2
     | C_13_42, (F143 | F234)
     \mid C_{14}_{23}, (F134 \mid F243) \rightarrow
            printf "@[((%s%s%s+%s))*phi_phi2v_m_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
                (format_coeff coeff) c pa pb wf1 p2 wf2 p1 wf3 p3
      C_{-}13_{-}42, (F314 \mid F423)
      C_{-}14_{-}23, (F413 \mid F324) \rightarrow
            printf "@[((%s%s%s+%s))*phi_phi2v_m_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
                (format_coeff coeff) c pa pb wf2 p3 wf3 p2 wf1 p1
       C_{-}13_{-}42, (F324 \mid F413)
     \mid C_114_23, (F423 \mid F314) \rightarrow
            printf "@[((%s%s%s+%s))*phi_phi2v_m_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
                (format_coeff coeff) c pa pb wf2 p1 wf1 p2 wf3 p3
     | C_{13}_{42}, (F_{341} | F_{432})|
     | C_{-}14_{-}23, (F431 | F342) \rightarrow
            printf "@[((%s%s%s+%s))*phi_phi2v_m_0(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
                (format_coeff coeff) c pa pb wf3 p2 wf2 p3 wf1 p1
      C_{13}_{42}, (F342 \mid F431)
     | C_{-}14_{-}23, (F432 | F341) \rightarrow
            printf "@[((%s%s%s+%s))*phi_phi2v_m_0(cmplx(1,kind=default),@_\%s,\%s,\%s,\%s,\%s,\%s))@]"
                (format_coeff coeff) c pa pb wf3 p1 wf1 p3 wf2 p2
 let \ print\_add\_dscalar2\_vector2\_m\_0\_km \ c \ pa \ pb \ wf1 \ wf2 \ wf3 \ p1 \ p2 \ p3 \ fusion \ (coeff, \ contraction) \ = \ (coeff, \ coeff, \ co
     printf "@_+_";
     print_dscalar2_vector2_m_0_km c pa pb wf1 wf2 wf3 p1 p2 p3 fusion (coeff, contraction)
let print\_dscalar2\_vector2\_m\_1\_km c pa pb wf1 wf2 wf3 p1 p2 p3 fusion (coeff, contraction) =
     match contraction, fusion with
     | C_{-}12_{-}34, (F123 | F213 | F124 | F214) \rightarrow
            printf \ "@[((\%s\%s\%s+\%s))*v\_phi2v\_m\_1(cmplx(1,kind=default),@_{\square}\%s,\%s,\%s,\%s,\%s,\%s))@]"
                (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
     \mid C_{12}_{34}, (F_{134} \mid F_{143} \mid F_{234} \mid F_{243}) \rightarrow
            printf "@[((%s%s%s+%s))*phi_phi2v_m_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
                (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
     | C_{-}12_{-}34, (F132 | F231 | F142 | F241) \rightarrow
            printf "@[((%s%s%s+%s))*v_phi2v_m_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
                (format_coeff coeff) c pa pb wf1 p1 wf3 p3 wf2 p2
     | C_{-}12_{-}34, (F312 | F321 | F412 | F421) \rightarrow
            printf "@[((%s%s%s+%s))*v_phi2v_m_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
                (format_coeff coeff) c pa pb wf3 p3 wf2 p2 wf1 p1
     | C_{12}_{34}, (F314 | F413 | F324 | F423) \rightarrow
            printf "@[((%s%s%s+%s))*phi_phi2v_m_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
                (format_coeff coeff) c pa pb wf2 p2 wf1 p1 wf3 p3
     | C_{-}12_{-}34, (F341 | F431 | F342 | F432) \rightarrow
            printf "@[((%s%s%s+%s))*phi_phi2v_m_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
                (format_coeff coeff) c pa pb wf3 p3 wf2 p2 wf1 p1
     | C_{-}13_{-}42, (F123 | F214)
     | C_{-}14_{-}23, (F124 | F213) \rightarrow
            printf \ "@[((%s%s%s+%s))*v_phi2v_m_1(cmplx(1,kind=default),@_\%s,%s,%s,%s,%s,%s,%s))@]"
                (format\_coeff\ coeff)\ c\ pa\ pb\ wf1\ p1\ wf2\ p3\ wf3\ p2
```

```
C_{13}_{42}, (F124 | F213)
   | C_{-}14_{-}23, (F123 | F214) \rightarrow
       printf "@[((%s%s%s+%s))*v_phi2v_m_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
          (format_coeff coeff) c pa pb wf2 p2 wf1 p3 wf3 p1
    C_{13}_{42}, (F_{132} | F_{241})
    C_14_23, (F142 \mid F231) \rightarrow
        printf "@[((%s%s%s+%s))*v_phi2v_m_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
          (format_coeff coeff) c pa pb wf1 p1 wf3 p2 wf2 p3
    C_{-}13_{-}42, (F142 \mid F231)
   | C_{-}14_{-}23, (F132 | F241) \rightarrow
        printf "@[((%s%s%s+%s))*v_phi2v_m_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
          (format\_coeff\ coeff)\ c\ pa\ pb\ wf3\ p3\ wf1\ p2\ wf2\ p1
   | C_{-}13_{-}42, (F312 | F421)
   | C_{-}14_{-}23, (F412 | F321) \rightarrow
        printf "@[((%s%s%s+%s))*v_phi2v_m_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
          (format_coeff coeff) c pa pb wf2 p2 wf3 p1 wf1 p3
   | C_{13}_{42}, (F321 | F412)
    C_{-}14_{-}23, (F421 \mid F312) \rightarrow
        printf \ "@[((%s%s%s+%s))*v_phi2v_m_1(cmplx(1,kind=default),@_\%s,%s,%s,%s,%s,%s,%s))@]"
          (format_coeff coeff) c pa pb wf3 p3 wf2 p1 wf1 p2
   | C_{13}_{42}, (F_{134} | F_{243})
   | C_{-}14_{-}23, (F_{1}43 | F_{2}34) \rightarrow
        printf "@[((%s%s%s+%s))*phi_phi2v_m_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
          (format_coeff coeff) c pa pb wf1 p3 wf3 p1 wf2 p2
   | C_13_42, (F_{143} | F_{234})|
   | C_{-}14_{-}23, (F134 | F243) \rightarrow
        printf "@[((%s%s%s+%s))*phi_phi2v_m_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
          (format_coeff coeff) c pa pb wf1 p2 wf2 p1 wf3 p3
   | C_{13}_{42}, (F314 | F423)
   | C_{-}14_{-}23, (F413 | F324) \rightarrow
        printf "@[((%s%s%s+%s))*phi_phi2v_m_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
          (format_coeff coeff) c pa pb wf2 p3 wf3 p2 wf1 p1
   C_{13}_{42}, (F324 | F413)
    C_14_23, (F423 \mid F314) \rightarrow
        printf "@[((%s%s%s+%s))*phi_phi2v_m_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
          (format_coeff coeff) c pa pb wf2 p1 wf1 p2 wf3 p3
    C_{-}13_{-}42, (F341 \mid F432)
   | C_{-}14_{-}23, (F431 | F342) \rightarrow
        printf "@[((%s%s%s+%s))*phi_phi2v_m_1(cmplx(1,kind=default), @_\%s, %s, %s, %s, %s, %s, %s))@]"
          (format_coeff coeff) c pa pb wf3 p2 wf2 p3 wf1 p1
   | C_{-}13_{-}42, (F342 | F431)
   | C_{14}_{23}, (F_{432} | F_{341}) \rightarrow
        printf "@[((%s%s%s+%s))*phi_phi2v_m_1(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
          (format_coeff coeff) c pa pb wf3 p1 wf1 p3 wf2 p2
 let print_add_dscalar2_vector2_m_1_km c pa pb wf1 wf2 wf3 p1 p2 p3 fusion (coeff, contraction) =
   printf "@_+_";
   print_dscalar2_vector2_m_1_km c pa pb wf1 wf2 wf3 p1 p2 p3 fusion (coeff, contraction)
let print\_dscalar2\_vector2\_m\_7\_km c pa pb wf1 wf2 wf3 p1 p2 p3 fusion (coeff, contraction) =
   match contraction, fusion with
   \mid C_{-}12_{-}34, (F123 \mid F213 \mid F124 \mid F214) \rightarrow
        printf "@[((%s%s%s+%s))*v_phi2v_m_7(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
          (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
   | C_{12_{34}}, (F_{134} | F_{143} | F_{234} | F_{243}) \rightarrow
        printf "@[((%s%s%s+%s))*phi_phi2v_m_7(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
          (format_coeff coeff) c pa pb wf1 p1 wf2 p2 wf3 p3
   | C_{-}12_{-}34, (F132 | F231 | F142 | F241) \rightarrow
        printf "@[((%s%s%s+%s))*v_phi2v_m_7(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
          (format_coeff coeff) c pa pb wf1 p1 wf3 p3 wf2 p2
   | C_{-}12_{-}34, (F312 | F321 | F412 | F421) \rightarrow
        printf \ "@[((\%s\%s\%s+\%s))*v\_phi2v\_m\_7(cmplx(1,kind=default),@_{\square}\%s,\%s,\%s,\%s,\%s,\%s))@]"
```

```
(format_coeff coeff) c pa pb wf3 p3 wf2 p2 wf1 p1
  | C_{-}12_{-}34, (F314 | F413 | F324 | F423) \rightarrow
      printf "@[((%s%s%s+%s))*phi_phi2v_m_7(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf2 p2 wf1 p1 wf3 p3
  \mid C_{-}12_{-}34, (F341 \mid F431 \mid F342 \mid F432) \rightarrow
      printf "@[((%s%s%s+%s))*phi_phi2v_m_7(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf3 p3 wf2 p2 wf1 p1
  | C_13_42, (F123 | F214)
  | C_{-}14_{-}23, (F124 | F213) \rightarrow
      printf "@[((%s%s%s+%s))*v_phi2v_m_7(cmplx(1,kind=default),@_\%s,%s,\%s,\%s,\%s,\%s,\%s))@]"
         (format_coeff coeff) c pa pb wf1 p1 wf2 p3 wf3 p2
  C_{13}_{42}, (F124 | F213)
  | C_{-}14_{-}23, (F123 | F214) \rightarrow
      printf "@[((%s%s%s+%s))*v_phi2v_m_7(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf2 p2 wf1 p3 wf3 p1
  C_{13}_{42}, (F_{132} | F_{241})
  | C_{-}14_{-}23, (F142 | F231) \rightarrow
      printf "@[((%s%s%s+%s))*v_phi2v_m_7(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format\_coeff\ coeff)\ c\ pa\ pb\ wf1\ p1\ wf3\ p2\ wf2\ p3
  C_{-}13_{-}42, (F142 \mid F231)
  | C_{14}_{23}, (F_{132} | F_{241}) \rightarrow
      printf "@[((%s%s%s+%s))*v_phi2v_m_7(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf3 p3 wf1 p2 wf2 p1
   C_{13}_{42}, (F312 \mid F421)
  | C_{-}14_{-}23, (F412 | F321) \rightarrow
      printf "@[((%s%s%s+%s))*v_phi2v_m_7(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf2 p2 wf3 p1 wf1 p3
  C_{13}_{42}, (F321 | F412)
  | C_{-}14_{-}23, (F421 | F312) \rightarrow
      printf \ "@[((\%s\%s\%s+\%s))*v\_phi2v\_m\_7(cmplx(1,kind=default),@_{\square}\%s,\%s,\%s,\%s,\%s,\%s))@]"
         (format_coeff coeff) c pa pb wf3 p3 wf2 p1 wf1 p2
   C_{-}13_{-}42, (F134 \mid F243)
   C_{-}14_{-}23, (F143 \mid F234) \rightarrow
      printf "@[((%s%s%s+%s))*phi_phi2v_m_7(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf1 p3 wf3 p1 wf2 p2
  | C_{13}_{42}, (F_{143} | F_{234})
  \mid C_{-}14_{-}23, (F134 \mid F243) \rightarrow
      printf "@[((%s%s%s+%s))*phi_phi2v_m_7(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf1 p2 wf2 p1 wf3 p3
  | C_{-}13_{-}42, (F314 | F423)
  | C_{-}14_{-}23, (F413 | F324) \rightarrow
      printf "@[((%s%s%s+%s))*phi_phi2v_m_7(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf2 p3 wf3 p2 wf1 p1
  C_{13}_{42}, (F324 | F413)
  | C_{-}14_{-}23, (F423 | F314) \rightarrow
      printf "@[((%s%s%s+%s))*phi_phi2v_m_7(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf2 p1 wf1 p2 wf3 p3
  | C_{-}13_{-}42, (F341 | F432)
  \mid C_{-}14_{-}23, (F431 \mid F342) \rightarrow
      printf "@[((%s%s%s+%s))*phi_phi2v_m_7(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf3 p2 wf2 p3 wf1 p1
  | C_{13}_{42}, (F342 | F431)
  \mid C_{14}_{23}, (F_{432} \mid F_{341}) \rightarrow
      printf "@[((%s%s%s+%s))*phi_phi2v_m_7(cmplx(1,kind=default),@_%s,%s,%s,%s,%s,%s,%s))@]"
         (format_coeff coeff) c pa pb wf3 p1 wf1 p3 wf2 p2
let print_add_dscalar2_vector2_m_7_km c pa pb wf1 wf2 wf3 p1 p2 p3 fusion (coeff, contraction) =
  printf "@_+_";
  print_dscalar2_vector2_m_7_km c pa pb wf1 wf2 wf3 p1 p2 p3 fusion (coeff, contraction)
let print_dscalar4_km c pa pb wf1 wf2 wf3 p1 p2 p3 p123 fusion (coeff, contraction) =
  match contraction, fusion with
```

```
C_12_34, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214)
    C_{-}13_{-}42, (F241 | F421 | F243 | F423 | F132 | F312 | F134 | F314)
    C_{-}14_{-}23, (F231 \mid F321 \mid F234 \mid F324 \mid F142 \mid F412 \mid F412 \mid F413 \mid F413) <math>\rightarrow
       printf "((%s%s%s+%s))*(%s*%s)*(%s*%s)*%s*%s*%s)"
         (format_coeff coeff) c pa pb p1 p2 p3 p123 wf1 wf2 wf3
    C_{-}12_{-}34, (F_{1}34 | F_{1}43 | F_{2}34 | F_{2}43 | F_{3}12 | F_{3}21 | F_{4}12 | F_{4}21)
    C_{-}13_{-}42, (F124 | F142 | F324 | F342 | F213 | F231 | F413 | F431)
  \mid C_{-}14_{-}23, (F123 \mid F132 \mid F423 \mid F432 \mid F214 \mid F241 \mid F314 \mid F341) \rightarrow
       printf "((%s%s%s+%s))*(%s*%s)*(%s*%s)*%s*%s*%s)"
         (format_coeff coeff) c pa pb p2 p3 p1 p123 wf1 wf2 wf3
  | C_12_34, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241)
    C_{-13-42}, (F214 | F412 | F234 | F432 | F123 | F321 | F143 | F341)
  \mid C_{-}14_{-}23, (F213 \mid F312 \mid F243 \mid F342 \mid F124 \mid F421 \mid F134 \mid F431) \rightarrow
       printf "((%s%s%s+%s))*(%s*%s)*(%s*%s)*%s*%s*%s)"
         (format_coeff coeff) c pa pb p1 p3 p2 p123 wf1 wf2 wf3
let print_add_dscalar4_km c pa pb wf1 wf2 wf3 p1 p2 p3 p123 fusion (coeff, contraction) =
  printf "@__+__";
  print_dscalar4_km c pa pb wf1 wf2 wf3 p1 p2 p3 p123 fusion (coeff, contraction)
let print_current amplitude dictionary rhs =
  match F.coupling rhs with
  V3 (vertex, fusion, constant) \rightarrow
       let ch1, ch2 = children2 \ rhs in
       let wf1 = multiple\_variable amplitude dictionary ch1
       and wf2 = multiple\_variable amplitude dictionary ch2
       and p1 = momentum ch1
       and p2 = momentum \ ch2
       and m1 = CM.mass\_symbol (F.flavor ch1)
       and m2 = CM.mass\_symbol (F.flavor ch2) in
       let c = CM.constant\_symbol\ constant in
       printf "0, "s" (if (F.sign\ rhs) < 0 then "-" else "+");
       begin match vertex with
```

Fermionic currents $\bar{\psi} \not A \psi$ and $\bar{\psi} \phi \psi$ are handled by the *Fermions* module, since they depend on the choice of Feynman rules: Dirac or Majorana.

```
\mid FBF (coeff, fb, b, f) \rightarrow
    begin match coeff, fb, b, f with
     \mid _, _, (VLRM \mid SPM \mid VAM \mid VA3M \mid TVA \mid TVAM \mid TLR \mid TLRM \mid TRL \mid TRLM), _ <math>\rightarrow
          let p12 = Printf.sprintf "(-%s-%s)" p1 p2 in
          Fermions.print_current_mom (coeff, fb, b, f) c wf1 wf2 p1 p2
               p12 fusion
     | \quad \_, \quad \_, \quad \_, \quad \_ \quad \rightarrow
          Fermions.print_current (coeff, fb, b, f) c wf1 wf2 fusion
     end
\mid PBP (coeff, f1, b, f2) \rightarrow
     Fermions.print\_current\_p (coeff, f1, b, f2) c wf1 wf2 fusion
 BBB \ (coeff, fb1, b, fb2) \rightarrow
     Fermions.print_current_b (coeff, fb1, b, fb2) c wf1 wf2 fusion
 GBG (coeff, fb, b, f) \rightarrow let p12 =
     Printf.sprintf "(-%s-%s)" p1 p2 in
     Fermions.print_current_g (coeff, fb, b, f) c wf1 wf2 p1 p2
           p12 fusion
```

Table 9.13 is a bit misleading, since if includes totally antisymmetric structure constants. The space-time part alone is also totally antisymmetric:

```
| Gauge\_Gauge\_Gauge\_coeff \rightarrow

let c = format\_coupling\ coeff\ c in

begin match fusion\ with

| (F23 \mid F31 \mid F12) \rightarrow

printf\ "g\_gg(%s,%s,%s,%s,%s)" c\ wf1\ p1\ wf2\ p2

| (F32 \mid F13 \mid F21) \rightarrow
```

```
\begin{array}{c} printf \  \, \text{"g-gg(\%s,\%s,\%s,\%s,\%s)"} \  \, c \  \, wf2 \  \, p2 \  \, wf1 \  \, p1 \\ \text{end} \\ \\ \mid I\_Gauge\_Gauge\_Gauge\_coeff} \  \, \rightarrow \\ \quad \text{let} \  \, c \  \, = \  \, format\_coupling \  \, coeff \  \, c \  \, \text{in} \\ \quad \text{begin match } fusion \  \, \text{with} \\ \mid (F23 \mid F31 \mid F12) \  \, \rightarrow \\ \quad \quad printf \  \, \text{"g-gg((0,1)*(\%s),\%s,\%s,\%s,\%s)"} \  \, c \  \, wf1 \  \, p1 \  \, wf2 \  \, p2 \\ \mid (F32 \mid F13 \mid F21) \  \, \rightarrow \\ \quad \quad printf \  \, \text{"g-gg((0,1)*(\%s),\%s,\%s,\%s,\%s,\%s)"} \  \, c \  \, wf2 \  \, p2 \  \, wf1 \  \, p1 \\ \quad \text{end} \\ \end{array}
```

In Aux_Gauge_Gauge , we can not rely on antisymmetry alone, because of the different Lorentz representations of the auxialiary and the gauge field. Instead we have to provide the sign in

$$(V_2 \wedge V_3) \cdot T_1 = \begin{cases} V_2 \cdot (T_1 \cdot V_3) = -V_2 \cdot (V_3 \cdot T_1) \\ V_3 \cdot (V_2 \cdot T_1) = -V_3 \cdot (T_1 \cdot V_2) \end{cases}$$
(15.2)

ourselves. Alternatively, one could provide g_xg mirroring g_gx.

```
 \begin{array}{l} |\; Aux\_Gauge\_Gauge\; coeff \; \to \\ |\; \text{let}\; c \; = \; format\_coupling\; coeff}\; c \; \text{in} \\ |\; begin\; \text{match}\; fusion\; \text{with} \\ |\; F23 \; \to \; printf \; "x\_gg(\%s,\%s,\%s)" \; c \; wf1 \; wf2 \\ |\; F32 \; \to \; printf \; "x\_gg(\%s,\%s,\%s)" \; c \; wf2 \; wf1 \\ |\; F12 \; \to \; printf \; "g\_gx(\%s,\%s,\%s)" \; c \; wf2 \; wf1 \\ |\; F21 \; \to \; printf \; "g\_gx(\%s,\%s,\%s)" \; c \; wf1 \; wf2 \\ |\; F13 \; \to \; printf \; "(-1)*g\_gx(\%s,\%s,\%s)" \; c \; wf2 \; wf1 \\ |\; F31 \; \to \; printf \; "(-1)*g\_gx(\%s,\%s,\%s)" \; c \; wf1 \; wf2 \\ |\; end \end{array}
```

These cases are symmetric and we just have to juxtapose the correct fields and provide parentheses to minimize the number of multiplications.

```
| Scalar\_Vector\_Vector\ coeff \rightarrow | let c=format\_coupling\ coeff\ c in | begin match fusion with | (F23\mid F32) \rightarrow printf "%s*(%s*%s)" c wf1 wf2 | (F12\mid F13) \rightarrow printf "(%s*%s)*%s" c wf1 wf2 | (F21\mid F31) \rightarrow printf "(%s*%s)*%s" c wf2 wf1 end | Aux\_Vector\_Vector\ coeff \rightarrow | let c=format\_coupling\ coeff\ c in | begin match fusion with | (F23\mid F32) \rightarrow printf "%s*(%s*%s)" c wf1 wf2 | (F12\mid F13) \rightarrow printf "(%s*%s)*%s" c wf1 wf2 | (F21\mid F31) \rightarrow printf "(%s*%s)*%s" c wf2 wf1 end
```

Even simpler:

```
 | Scalar\_Scalar\_Scalar coeff \rightarrow \\ printf "(\%s*\%s*\%s)" (format\_coupling coeff c) wf1 wf2 \\ | Aux\_Scalar\_Scalar coeff \rightarrow \\ printf "(\%s*\%s*\%s)" (format\_coupling coeff c) wf1 wf2 \\ | Aux\_Scalar\_Vector coeff \rightarrow \\ | let c = format\_coupling coeff c in \\ | begin match fusion with \\ | (F13 | F31) \rightarrow printf "\%s*(\%s*\%s)" c wf1 wf2 \\ | (F23 | F21) \rightarrow printf "(\%s*\%s)*\%s" c wf1 wf2 \\ | (F32 | F12) \rightarrow printf "(\%s*\%s)*\%s" c wf2 wf1 \\ | end \\ | Vector\_Scalar\_Scalar coeff \rightarrow \\ | Vector\_Scalar\_Scalar coeff \rightarrow | Vector\_Scalar\_Scalar coeff | Vector\_Scalar\_Sc
```

```
let c = format\_coupling coeff c in
    begin match fusion with
      F23 \rightarrow printf \text{"v_ss(\%s,\%s,\%s,\%s,\%s)"} c wf1 p1 wf2 p2
       F32 \rightarrow printf \text{"v_ss(\%s,\%s,\%s,\%s,\%s)"} c wf2 p2 wf1 p1
       F12 \rightarrow printf "s_vs(%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
       F21 \rightarrow printf \text{ "s_vs(%s,%s,%s,%s,%s)"} c wf2 p2 wf1 p1
       F13 \rightarrow printf "(-1)*s_vs(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
     F31 \rightarrow printf "(-1)*s_vs(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     end
| Graviton\_Scalar\_Scalar coeff \rightarrow
    \mathsf{let}\ c\ =\ \mathit{format\_coupling}\ \mathit{coeff}\ c\ \mathsf{in}
     begin match fusion with
       F12 \rightarrow printf "s_gravs(%s,%s,-(%s+%s),%s,%s,%s,"s)" c m2 p1 p2 p2 wf1 wf2
       F21 \rightarrow printf "s_gravs(%s,%s,-(%s+%s),%s,%s,%s,"s)" c m1 p1 p2 p1 wf2 wf1
       F13 \rightarrow printf "s_gravs(%s,%s,%s,-(%s+%s),%s,%s)" c m2 p2 p1 p2 wf1 wf2
       F31 \rightarrow printf "s_gravs(%s,%s,%s,-(%s+%s),%s,%s)" c m1 p1 p2 wf2 wf1
      F23 \rightarrow printf \, "grav_ss(%s,%s,%s,%s,%s,%s,%s)" \, c \, m1 \, p1 \, p2 \, wf1 \, wf2
     | F32 \rightarrow printf "grav_ss(%s,%s,%s,%s,%s,%s)" c m1 p2 p1 wf2 wf1
     end
```

In producing a vector in the fusion we always contract the rightmost index with the vector wavefunction from rhs. So the first momentum is always the one of the vector boson produced in the fusion, while the second one is that from the rhs. This makes the cases F12 and F13 as well as F21 and F31 equal. In principle, we could have already done this for the $Graviton_Scalar_Scalar$ case.

```
| Graviton\_Vector\_Vector coeff \rightarrow
    \mathsf{let}\ c\ =\ format\_coupling\ coef\!\!f\ c\ \mathsf{in}
    begin match fusion with
      (F12 \mid F13) \rightarrow printf "v_gravv(%s,%s,-(%s+%s),%s,%s,%s)" c m2 p1 p2 p2 wf1 wf2
      (F21 \mid F31) \rightarrow printf "v_gravv(%s,%s,-(%s+%s),%s,%s,%s)" c m1 p1 p2 p1 wf2 wf1
      F23 \rightarrow printf \ "grav_vv(%s,%s,%s,%s,%s,%s,%s)" \ c \ m1 \ p1 \ p2 \ wf1 \ wf2
     | F32 \rightarrow printf "grav_vv(%s,%s,%s,%s,%s,%s)" c m1 p2 p1 wf2 wf1
| Graviton\_Spinor\_Spinor coeff \rightarrow
    let c = format\_coupling coeff c in
     begin match fusion with
      F23 \rightarrow printf \text{"f-gravf(%s,%s,-(%s+%s),(-%s),%s,%s)"} c m2 p1 p2 p2 wf1 wf2
      F32 \rightarrow printf \text{"f\_gravf(%s,%s,-(%s+%s),(-%s),%s,%s)"} c m1 p1 p2 p1 wf2 wf1
      F12 \rightarrow printf \text{"f-fgrav(\%s,\%s,\%s,\%s,\%s,\%s,\%s)"} c m1 p1 p1 p2 wf1 wf2
      F21 \rightarrow printf \text{"f\_fgrav(\%s,\%s,\%s,\%s+\%s,\%s,\%s)"} c m2 p2 p1 p2 wf2 wf1
      F13 \rightarrow printf \, "grav_ff(%s,%s,%s,(-%s),%s,%s)" \, c \, m1 \, p1 \, p2 \, wf1 \, wf2
     F31 \rightarrow printf "grav_ff(%s,%s,%s,(-%s),%s,%s)" c m1 p2 p1 wf2 wf1
     end
Dim4\_Vector\_Vector\_Vector\_T\ coeff \rightarrow
    let c = format\_coupling coeff c in
     begin match fusion with
      F23 \rightarrow printf "tkv_vv(%s,%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
      F32 \rightarrow printf "tkv_vv(%s,%s,%s,%s,%s,%s)" c \ wf2 \ p2 \ wf1 \ p1
      F12 \rightarrow printf "tv_kvv(%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
      F21 \rightarrow printf "tv_kvv(%s,%s,%s,%s,%s)" c \ wf2 \ p2 \ wf1 \ p1
      F13 \rightarrow printf "(-1)*tv_kvv(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
     F31 \rightarrow printf "(-1)*tv_kvv(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     end
| Dim4\_Vector\_Vector\_L coeff \rightarrow
    let c = format\_coupling coeff c in
    begin match fusion with
      F23 \rightarrow printf "lkv_vv(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
      F32 \rightarrow printf "lkv_vv(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
       F12 \mid F13 \rightarrow printf "lv_kvv(%s,%s,%s,%s)" c wf1 p1 wf2
     \mid F21 \mid F31 \rightarrow printf "lv_kvv(%s,%s,%s,%s)" c wf2 p2 wf1
```

```
end
```

```
| Dim6\_Gauge\_Gauge\_Gauge\_coeff \rightarrow
    let c = format\_coupling coeff c in
    begin match fusion with
     \mid F23 \mid F31 \mid F12 \rightarrow
          printf "kg_kgkg(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
     \mid F32 \mid F13 \mid F21 \rightarrow
          printf "kg_kgkg(%s,%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
    end
Dim4\_Vector\_Vector\_Vector\_T5\ coeff \rightarrow
    let c = format\_coupling coeff c in
     begin match fusion with
       F23 \rightarrow printf "t5kv_vv(%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
       F32 \rightarrow printf \text{ "t5kv_vv(%s,%s,%s,%s,%s)"} c wf2 p2 wf1 p1
       F12 \mid F13 \rightarrow printf \text{ "t5v_kvv(%s,%s,%s,%s,%s)"} c wf1 p1 wf2 p2
     \mid F21 \mid F31 \rightarrow printf "t5v_kvv(%s,%s,%s,%s,%s)" c \ wf2 \ p2 \ wf1 \ p1
     end
| Dim4_Vector_Vector_L5 coeff \rightarrow
    let c = format\_coupling coeff c in
    begin match fusion with
       F23 \rightarrow printf "15kv_vv(%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
       F32 \rightarrow printf "15kv_vv(%s,%s,%s,%s,%s,%s)" c \ wf2 \ p2 \ wf1 \ p1
       F12 \rightarrow printf "15v_kvv(%s,%s,%s,%s)" c wf1 p1 wf2
       F21 \rightarrow printf "15v_kvv(%s,%s,%s,%s)" c \ wf2 \ p2 \ wf1
       F13 \rightarrow printf "(-1)*15v_kvv(%s,%s,%s,%s,%s)" c wf1 p1 wf2
     |F31 \rightarrow printf "(-1)*15v_kvv(%s,%s,%s,%s)" c wf2 p2 wf1
     end
| \ \mathit{Dim6\_Gauge\_Gauge\_Gauge\_5} \ \mathit{coeff} \ \rightarrow
    let c = format\_coupling coeff c in
     begin match fusion with
       F23 \rightarrow printf \text{ "kg5\_kgkg(%s,%s,%s,%s,%s)" } c wf1 p1 wf2 p2
       F32 \rightarrow printf \text{ "kg5\_kgkg(%s,%s,%s,%s,%s)" } c wf2 p2 wf1 p1
      F12 \rightarrow printf \text{ "kg_kg5kg(\%s,\%s,\%s,\%s,\%s)" } c wf1 p1 wf2 p2
      F21 \rightarrow printf \text{ "kg_kg5kg(\%s,\%s,\%s,\%s,\%s)"} c wf2 p2 wf1 p1
       F13 \rightarrow printf "(-1)*kg_kg5kg(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
      F31 \rightarrow printf "(-1)*kg_kg5kg(%s,%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     end
Aux\_DScalar\_DScalar\ coeff \rightarrow
    \mathsf{let}\ c\ =\ format\_coupling\ coef\!\!f\ c\ \mathsf{in}
    begin match fusion with
     | (F23 | F32) \rightarrow
         printf "%s*(%s*%s)*(%s*%s)" c p1 p2 wf1 wf2
     \mid (F12 \mid F13) \rightarrow
          printf "%s*(-((%s+%s)*%s))*(%s*%s)" c p1 p2 p2 wf1 wf2
     | (F21 | F31) \rightarrow
          printf "%s*(-((%s+%s)*%s))*(%s*%s)" c p1 p2 p1 wf1 wf2
\mid Aux\_Vector\_DScalar\ coeff \rightarrow
    let c = format\_coupling coeff c in
    begin match fusion with
     | F23 \rightarrow printf "%s*(%s*%s)*%s" c \ wf1 \ p2 \ wf2
      F32 \rightarrow printf "%s*(%s*%s)*%s" c \ wf2 \ p1 \ wf1
       F12 \rightarrow printf "%s*(-((%s+%s)*%s))*%s" c p1 p2 wf2 wf1
       F21 \rightarrow printf "%s*(-((%s+%s)*%s))*%s" c p1 p2 wf1 wf2
     \mid (F13 \mid F31) \rightarrow printf "(-(%s+%s))*(%s*%s*%s)" p1 p2 c wf1 wf2
     end
| Dim5\_Scalar\_Gauge2 \ coeff \rightarrow
    let c = format\_coupling coeff c in
```

```
begin match fusion with
     | (F23 | F32) \rightarrow printf "(\%s)*((\%s*\%s)*(\%s*\%s)_{-}(\%s*\%s)*(\%s*\%s))"
             c p1 wf2 p2 wf1 p1 p2 wf2 wf1
     | (F12 | F13) \rightarrow printf "(%s)*%s*((-((%s+%s)*%s))*%s))*%s_{-}((-(%s+%s)*%s))*%s)"
             c wf1 p1 p2 wf2 p2 p1 p2 p2 wf2
     \mid (F21 \mid F31) \rightarrow printf "(%s)*%s*((-((%s+%s)*%s))*%s_{-}((-(%s+%s)*%s))*%s)"
             c wf2 p2 p1 wf1 p1 p1 p2 p1 wf1
     end
Dim5\_Scalar\_Gauge2\_Skew\ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     | (F23 \mid F32) \rightarrow printf "(-uphi_vv_u(%s,u%s,u%s,u%s,u%s,u%s))" c p1 p2 wf1 wf2
     \mid (F12 \mid F13) \rightarrow printf "(-uv_phiv_u(%s,u%s,u%s,u%s,u%s))" c wf1 p1 p2 wf2
     | (F21 \mid F31) \rightarrow printf \text{"v_phiv_(%s,_,%s,_,%s,_,%s,_,%s,_,,%s)"} c wf2 p1 p2 wf1
     end
Dim5\_Scalar\_Vector\_Vector\_T\ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     | (F23 \mid F32) \rightarrow printf "(\%s)*(\%s*\%s)*(\%s*\%s)" c p1 wf2 p2 wf1
     \mid (F12 \mid F13) \rightarrow printf "(%s)*%s*(-((%s+%s)*%s))*%s" c wf1 p1 p2 wf2 p2
     \mid (F21 \mid F31) \rightarrow printf "(%s)*%s*(-((%s+%s)*%s))*%s" c \ wf2 \ p2 \ p1 \ wf1 \ p1
     end
Dim5\_Scalar\_Vector\_Vector\_U\ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     \mid (F23 \mid F32) \rightarrow printf "phi_u_vv_u(%s,u%s,u%s,u%s,u%s,u%s)" c p1 p2 wf1 wf2
     (F12 \mid F13) \rightarrow printf "v_u_phiv_(%s,\\\\s,\\\s,\\\s,\\\s,\\\\s,\\\\s)" c \ wf1 \ p1 \ p2 \ wf2
     \mid (F21 \mid F31) \rightarrow printf \text{"v_u_phiv_u(%s,u%s,u%s,u%s,u%s)"} c wf2 p2 p1 wf1
     end
Dim5\_Scalar\_Vector\_Vector\_TU\ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     |F23 \rightarrow printf "(%s)*((%s*%s)*(-(%s+%s)*%s)_{\sqcup}-_{\sqcup}(-(%s+%s)*%s)*(%s*%s))"
            c p1 wf2 p1 p2 wf1 p1 p2 p1 wf1 wf2
     |F32\rangle \rightarrow printf "(%s)*((%s*%s)*(-(%s+%s)*%s)_{\square}-_{\square}(-(%s+%s)*%s)*(%s*%s))"
            c p2 wf1 p1 p2 wf2 p1 p2 p2 wf1 wf2
     |F12\rangle \rightarrow printf "(%s)*%s*((%s*%s)*%s_{\square}-_{\square}(%s*%s)*%s)"
            c wf1 p1 wf2 p2 p1 p2 wf2
     F21 \rightarrow printf "(%s)*%s*((%s*%s)*%s_{\square}-_{\square}(%s*%s)*%s)"
             c wf2 p2 wf1 p1 p1 p2 wf1
     | F13 \rightarrow printf "(%s)*%s*((-(%s+%s)*%s)*%s_{\square}-_{\square}(-(%s+%s)*%s)*%s)"
             c wf1 p1 p2 wf2 p1 p1 p2 p1 wf2
     | F31 \rightarrow printf "(%s)*%s*((-(%s+%s)*%s)*%s)-_\(-(%s+%s)*%s)*%s)"
             c wf2 p1 p2 wf1 p2 p1 p2 p2 wf1
     end
| Dim5\_Scalar\_Scalar2 \ coeff \rightarrow
     \mathsf{let}\ c\ =\ format\_coupling\ coef\!\!f\ c\ \mathsf{in}
     begin match fusion with
     |(F23 \mid F32) \rightarrow printf "phi_dim5s2(%s, \u00bb/ks, \u00bb/ks, \u00bb/ks, \u00bb/ks, \u00bb/ks)"
          c wf1 p1 wf2 p2
     |(F12 \mid F13) \rightarrow \text{let } p12 = Printf.sprintf "(-\%s-\%s)" p1 p2 in
          printf "phi_dim5s2(%s,%s,%s,%s,%s)" c wf1 p12 wf2 p2
     |(F21 | F31) \rightarrow \text{let } p12 = Printf.sprintf "(-%s-%s)" p1 p2 in
          printf "phi_dim5s2(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p12
     end
| Scalar\_Vector\_Vector\_t coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
```

```
\mid (F23 \mid F32) \rightarrow printf \ "s_vv_t(\%s,\%s,\%s,\%s,\%s)" \ c \ wf1 \ p1 \ wf2 \ p2
      |(F12 \mid F13) \rightarrow printf \text{"v_sv_t(%s,\%s,\%s,\%s,\%s,\%s)"} c wf1 p1 wf2 p2
      |(F21 \mid F31) \rightarrow printf \text{"v_sv_t(%s,%s,%s,%s,%s,%s)"} c wf2 p2 wf1 p1
      end
 Dim6\_Vector\_Vector\_Vector\_T\ coeff \rightarrow
      let c = format\_coupling coeff c in
      begin match fusion with
      F23 \rightarrow printf "(%s)*(%s*%s)*(%s*%s)*(%s-%s)" c p2 wf1 p1 wf2 p1 p2
       F32 \rightarrow printf "(%s)*(%s*%s)*(%s*%s)*(%s-%s)" c p1 wf2 p2 wf1 p2 p1
      | (F12 | F13) \rightarrow printf "(\%s)*((\%s+2*\%s)*\%s)*(-((\%s+\%s)*\%s))*\%s"
              c p1 p2 wf1 p1 p2 wf2 p2
      | (F21 | F31) \rightarrow printf "(%s)*((-((%s+%s)*%s))*(%s+2*%s)*%s)*%s"
              c p2 p1 wf1 p2 p1 wf2 p1
      end
 | Tensor_2 Vector_Vector coeff \rightarrow
      let c = format\_coupling coeff c in
      begin match fusion with
      \mid (F23 \mid F32) \rightarrow printf "t2_vv(%s,%s,%s)" c wf1 wf2
      \mid (F12 \mid F13) \rightarrow printf "v_t2v(%s,%s,%s)" c \ wf1 \ wf2
      | (F21 | F31) \rightarrow printf "v_t2v(%s,%s,%s)" c wf2 wf1
      end
 | Tensor_2_Scalar_Scalar coeff \rightarrow
      let c = format\_coupling coeff c in
      begin match fusion with
      | (F23 | F32) \rightarrow printf "t2_phi2(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
       (F12 \mid F13) \rightarrow printf "phi_t2phi(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
      | (F21 | F31) \rightarrow printf "phi_t2phi(%s,%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
      end
 | Tensor_2 - Vector_1 \ coeff \rightarrow
      let c = format\_coupling coeff c in
      begin match fusion with
      | (F23 \mid F32) \rightarrow printf "t2_vv_1(%s,%s,%s)" c wf1 wf2
      \mid (F12 \mid F13) \rightarrow printf \text{"v_t2v_1(%s,%s,%s)"} c wf1 wf2
      | (F21 \mid F31) \rightarrow printf "v_t2v_1(%s,%s,%s)" c wf2 wf1
      end
 | Tensor_2 - Vector_Vector_cf \ coeff \rightarrow
      let c = format\_coupling coeff c in
      begin match fusion with
      | (F23 \mid F32) \rightarrow printf "t2_vv_cf(%s,%s,%s)" c wf1 wf2
       (F12 \mid F13) \rightarrow printf \text{"v_t2v_cf(%s,%s,%s)"} c wf1 wf2
      | (F21 | F31) \rightarrow printf "v_t2v_cf(%s,%s,%s)" c wf2 wf1
      end
 | Tensor_2\_Scalar\_Scalar\_cf \ coeff \rightarrow
      let c = format\_coupling coeff c in
      begin match fusion with
      | (F23 \mid F32) \rightarrow printf "t2\_phi2\_cf(%s,%s,%s,%s,u%s)" c wf1 p1 wf2 p2
      \mid (F12 \mid F13) \rightarrow printf \ "phi_t2phi_cf(%s,%s,%s,%s,%s)" \ c \ wf1 \ p1 \ wf2 \ p2
      |(F21 | F31) \rightarrow printf "phi_t2phi_cf(%s,%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
      end
 | Dim5\_Tensor\_2\_Vector\_Vector\_1 \ coeff \rightarrow
      let c = format\_coupling coeff c in
      begin match fusion with
      | (F23 \mid F32) \rightarrow printf "t2\_vv\_d5\_1(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
      | (F12 \mid F13) \rightarrow printf \text{"v_t2v_d5_1(%s,%s,%s,%s,%s,%s)"} c wf1 p1 wf2 p2
      |(F21 \mid F31) \rightarrow printf \text{"v_t2v_d5_1(%s,%s,%s,%s,%s)"} c wf2 p2 wf1 p1
      end
| Tensor_2 \_Vector_Vector_t coeff \rightarrow
```

```
let c = format\_coupling coeff c in
     begin match fusion with
     | (F23 \mid F32) \rightarrow printf "t2_vv_t(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
     \mid (F12 \mid F13) \rightarrow printf \text{"v_t2v_t(%s,%s,%s,%s,%s)"} c wf1 p1 wf2 p2
     | (F21 | F31) \rightarrow printf "v_t2v_t(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     end
| Dim5\_Tensor\_2\_Vector\_Vector\_2 coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     \mid F23 \rightarrow printf  "t2_vv_d5_2(%s,%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
      F32 \rightarrow printf \text{ "t2\_vv\_d5\_2(%s,%s,%s,%s,%s)" } c wf2 p2 wf1 p1
      (F12 \mid F13) \rightarrow printf \text{"v_t2v_d5_2(%s,%s,%s,%s,%s,%s)"} c wf1 p1 wf2 p2
     \mid (F21 \mid F31) \rightarrow printf \text{ "v_t2v_d5_2(%s,%s,%s,%s,%s)" } c wf2 p2 wf1 p1
     end
TensorVector\_Vector\_Vector\ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     \mid (F23 \mid F32) \rightarrow printf \text{"dv_vv(\%s,\%s,\%s,\%s,\%s)"} c wf1 p1 wf2 p2
     | (F12 \mid F13) \rightarrow printf \text{"v_dvv(%s,%s,%s,%s)"} c \text{ wf1 p1 wf2}
     | (F21 \mid F31) \rightarrow printf \text{"v_dvv(%s,%s,%s,%s)"} c wf2 p2 wf1
     end
| TensorVector\_Vector\_Vector\_cf \ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     | (F23 \mid F32) \rightarrow printf "dv_vv_cf(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
      (F12 \mid F13) \rightarrow printf \text{"v_dvv_cf(%s,%s,%s,%s)"} c wf1 p1 wf2
     | (F21 | F31) \rightarrow printf "v_dvv_cf(%s,%s,%s,%s)" c wf2 p2 wf1
     end
| TensorVector\_Scalar\_Scalar coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     | (F23 | F32) \rightarrow printf "dv_phi2(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
     \mid (F12 \mid F13) \rightarrow printf \; "phi_dvphi(%s,%s,%s,%s,%s)" \; c \; wf1 \; p1 \; wf2 \; p2
     |(F21 | F31) \rightarrow printf "phi_dvphi(%s,%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     end
TensorVector\_Scalar\_Scalar\_cf\ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     \mid (F23 \mid F32) \rightarrow printf \text{"dv_phi2_cf(%s,%s,%s,%s,%s,%s)"} c wf1 p1 wf2 p2
      (F12 \mid F13) \rightarrow printf "phi_dvphi_cf(%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
     | (F21 | F31) \rightarrow printf "phi_dvphi_cf(%s,%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     end
| TensorScalar\_Vector\_Vector\ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     | (F23 | F32) \rightarrow printf "tphi_vv(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
     \mid (F12 \mid F13) \rightarrow printf \text{"v\_tphiv(%s,%s,%s,%s,%s)"} c wf1 p1 wf2 p2
     | (F21 | F31) \rightarrow printf "v_tphiv(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     end
| TensorScalar\_Vector\_Vector\_cf \ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     | (F23 | F32) \rightarrow printf "tphi_vv_cf(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
     | (F12 | F13) \rightarrow printf "v_tphiv_cf(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
     | (F21 | F31) \rightarrow printf "v_tphiv_cf(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     end
| TensorScalar\_Scalar\_Scalar coeff \rightarrow
```

```
let c = format\_coupling coeff c in
     begin match fusion with
     | (F23 | F32) \rightarrow printf "tphi_ss(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
      (F12 \mid F13) \rightarrow printf "s_tphis(%s,%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
     | (F21 | F31) \rightarrow printf "s_tphis(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     end
TensorScalar\_Scalar\_Scalar\_cf\ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     |(F23 \mid F32) \rightarrow printf "tphi_ss_cf(%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
     | (F12 | F13) \rightarrow printf "s_tphis_cf(%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
     |(F21 | F31) \rightarrow printf "s_tphis_cf(%s,%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     end
| Dim7\_Tensor\_2\_Vector\_Vector\_T coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     \mid F23 \rightarrow printf "t2\_vv\_d7(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
     \mid F32 
ightarrow printf "t2_vv_d7(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     \mid (F12 \mid F13) \rightarrow printf \text{"v_t2v_d7(%s,%s,%s,%s,%s)"} c wf1 p1 wf2 p2
     \mid (F21 \mid F31) \rightarrow printf "v_t2v_d7(%s,%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     end
Dim6\_Scalar\_Vector\_Vector\_D\ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     | (F23 \mid F32) \rightarrow printf "s_vv_6D(%s,%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
      (F12 \mid F13) \rightarrow printf \text{"v_sv_6D(%s,%s,%s,%s,%s)"} c wf1 p1 wf2 p2
     | (F21 | F31) \rightarrow printf \text{"v_sv_6D(\%s,\%s,\%s,\%s,\%s)"} c \text{ } wf2 \text{ } p2 \text{ } wf1 \text{ } p1
     end
Dim6\_Scalar\_Vector\_Vector\_DP\ coeff \rightarrow
     \mathsf{let}\ c\ =\ \mathit{format\_coupling}\ \mathit{coeff}\ c\ \mathsf{in}
     begin match fusion with
     | (F23 \mid F32) \rightarrow printf \text{"s_vv_6DP(%s,%s,%s,%s,%s,%s)"} c wf1 p1 wf2 p2
      (F12 \mid F13) \rightarrow printf \text{"v_sv_6DP(\%s,\%s,\%s,\%s,\%s)"} c wf1 p1 wf2 p2
     | (F21 | F31) \rightarrow printf \text{"v_sv_6DP(%s,%s,%s,%s,%s)"} c wf2 p2 wf1 p1
     end
\mid Dim6\_HAZ\_D \ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
       F23 \rightarrow printf \text{ "h_az_D(\%s,\%s,\%s,\%s,\%s)"} c wf1 p1 wf2 p2
       F32 \rightarrow printf \text{ "h_az_D(%s,%s,%s,%s,%s)"} c wf2 p2 wf1 p1
       F13 \rightarrow printf "a_hz_D(%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
       F31 \rightarrow printf "a_hz_D(%s,%s,%s,%s,%s)" c \ wf2 \ p2 \ wf1 \ p1
       F12 \rightarrow printf "z_ah_D(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     F21 \rightarrow printf "z_ah_D(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
     end
\mid Dim6\_HAZ\_DP \ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
       F23 \rightarrow printf \text{ "h_az_DP(\%s,\%s,\%s,\%s,\%s)"} c wf1 p1 wf2 p2
       F32 \rightarrow printf \text{ "h_az_DP(%s,\%s,\%s,\%s,\%s)"} c wf2 p2 wf1 p1
      F13 \rightarrow printf "a_hz_DP(%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
      F31 \rightarrow printf "a_hz_DP(%s,%s,%s,%s,%s)" c \ wf2 \ p2 \ wf1 \ p1
      F12 \rightarrow printf "z_ah_DP(%s,%s,%s,%s,%s)" c \ wf2 \ p2 \ wf1 \ p1
     F21 \rightarrow printf "z_ah_DP(%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
| Gauge\_Gauge\_i coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     F23 \rightarrow printf \text{ "g-gg-23(%s,%s,%s,%s,%s)"} c wf1 p1 wf2 p2
```

```
F32 \rightarrow printf \text{ "g-gg-23(%s,\%s,\%s,\%s,\%s)"} c wf2 p2 wf1 p1
       F13 \rightarrow printf \text{ "g-gg-13(%s,\%s,\%s,\%s,\%s)"} c wf1 p1 wf2 p2
       F31 \rightarrow printf \ "g_gg_13(%s,%s,%s,%s,%s)" \ c \ wf2 \ p2 \ wf1 \ p1
       F12 \rightarrow printf "(-1)_{\perp}*_{\perp}g_{-}gg_{-}13(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
      \mid F21 \rightarrow printf "(-1)\sqcup*\sqcupg\_gg\_13(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     end
\mid Dim6\_GGG \ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
       F23 \rightarrow printf \text{ "g-gg-6(%s,%s,%s,%s,%s)"} c wf1 p1 wf2 p2
       F32 \rightarrow printf \text{"g-gg-6(%s,%s,%s,%s,%s)"} c wf2 p2 wf1 p1
       F12 \rightarrow printf \text{"g-gg-6(%s,%s,%s,%s,%s)"} c wf1 p1 wf2 p2
       F21 \rightarrow printf \ "g_gg_6(%s,%s,%s,%s,%s)" \ c \ wf2 \ p2 \ wf1 \ p1
       F13 \rightarrow printf "(-1)_{\perp}*_{\perp}g_{-}gg_{-}6(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
     |F31 \rightarrow printf "(-1) + g_gg_6(%s, %s, %s, %s, %s) " c wf2 p2 wf1 p1
     end
\mid Dim6\_AWW\_DP \ coeff \rightarrow
     \mathsf{let}\ c\ =\ format\_coupling\ coef\!\!f\ c\ \mathsf{in}
     begin match fusion with
       F23 \rightarrow printf "a_ww_DP(%s,%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
       F32 \rightarrow printf "a_ww_DP(%s,%s,%s,%s,%s)" c \ wf2 \ p2 \ wf1 \ p1
       F13 \rightarrow printf \text{"w_aw_DP(\%s,\%s,\%s,\%s,\%s)"} c wf1 p1 wf2 p2
      F31 \rightarrow printf "w_aw_DP(%s,%s,%s,%s,%s)" c \ wf2 \ p2 \ wf1 \ p1
      F12 \rightarrow printf "(-1)_{\square}*_{\square}w_aw_DP(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
     F21 \rightarrow printf "(-1)_{\bot}*_{\bot}w_aw_DP(%s,%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     end
\mid Dim6\_AWW\_DW \ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
       F23 \rightarrow printf "a_ww_DW(%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
       F32 \rightarrow printf "a_ww_DW(%s,%s,%s,%s,%s)" c \ wf2 \ p2 \ wf1 \ p1
      F13 \rightarrow printf "(-1)_{\perp}*_{\perp}a_{ww}DW(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
      F31 \rightarrow printf "(-1)_{\perp}*_{\perp}a_{ww}DW(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
       F12 \rightarrow printf "a_ww_DW(%s,%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
      \mid F21 \rightarrow printf "a_ww_DW(%s,%s,%s,%s,%s)" c \ wf2 \ p2 \ wf1 \ p1
     end
Dim6\_Gauge\_Gauge\_i\ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     \mid F23 \mid F31 \mid F12 \rightarrow
          printf "kg_kgkg_i(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
     \mid F32 \mid F13 \mid F21 \rightarrow
          printf "kg_kgkg_i(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
     end
\mid Dim6\_HHH \ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
     \mid (F23 \mid F32 \mid F12 \mid F21 \mid F13 \mid F31) \rightarrow
        printf "h_hh_6(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
     end
| Dim6\_WWZ\_DPWDW \ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
      \mid F23 \rightarrow printf "w_wz_DPW(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
       F32 \rightarrow printf \text{"w_wz_DPW(%s,%s,%s,%s,%s,%s)"} c wf2 p2 wf1 p1
       F13 \rightarrow printf "(-1)_{\perp}*_{\perp}w_{-}wz_{-}DPW(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
       F31 \rightarrow printf "(-1)_\_*\_w_wz_DP\(%s,%s,%s,%s,%s,%s)\" c \ wf2 \ p2 \ wf1 \ p1
      F12 \rightarrow printf "z_ww_DPW(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
```

```
F21 \rightarrow printf "z_ww_DPW(%s,%s,%s,%s,%s,%s)" c \ wf2 \ p2 \ wf1 \ p1
     end
\mid Dim6\_WWZ\_DW \ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
       F23 \rightarrow printf "w_wz_DW(%s,%s,%s,%s,%s,%s)" c \ wf1 \ p1 \ wf2 \ p2
       F32 \rightarrow printf \text{ "w_wz_DW(%s,%s,%s,%s,%s)"} c wf2 p2 wf1 p1
       F13 \rightarrow printf "(-1)_{\sqcup}*_{\sqcup}w_{-}wz_{-}DW(%s,%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
       F31 \rightarrow printf "(-1)_{\bot}*_{\bot}w_{\bot}wz_{\bot}DW(\%s,\%s,\%s,\%s,\%s,\%s)" c wf2 p2 wf1 p1
       F12 \rightarrow printf "z_ww_DW(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
     F21 \rightarrow printf "z_ww_DW(%s,%s,%s,%s,%s,%s)" c \ wf2 \ p2 \ wf1 \ p1
     end
\mid Dim6\_WWZ\_D \ coeff \rightarrow
     let c = format\_coupling coeff c in
     begin match fusion with
       F23 \rightarrow printf \text{ "w_wz_D(%s,%s,%s,%s,%s)"} c wf1 p1 wf2 p2
       F32 \rightarrow printf \text{ "w_wz_D(%s,%s,%s,%s,%s)"} c wf2 p2 wf1 p1
       F13 \rightarrow printf "(-1)_{\sqcup}*_{\sqcup}w_{wz}_{D}(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
       F31 \rightarrow printf "(-1)_{\square}*_{\square}w_{wz}D(%s,%s,%s,%s,%s)" c wf2 p2 wf1 p1
       F12 \rightarrow printf "z_ww_D(%s,%s,%s,%s,%s)" c wf1 p1 wf2 p2
       F21 \rightarrow printf \ "z_ww_D(%s,%s,%s,%s,%s)" \ c \ wf2 \ p2 \ wf1 \ p1
     end
end
```

Flip the sign to account for the i² relative to diagrams with only cubic couplings.

\$

That's an *slightly dangerous* hack!!! How do we account for such signs when treating n-ary vertices uniformly?

```
V4 (vertex, fusion, constant) \rightarrow
     \mathsf{let}\ c\ =\ CM.constant\_symbol\ constant
     and ch1, ch2, ch3 = children3 rhs in
     let wf1 = multiple\_variable amplitude dictionary ch1
     and wf2 = multiple\_variable amplitude dictionary ch2
     and wf3 = multiple\_variable amplitude dictionary ch3
     and p1 = momentum ch1
     and p2 = momentum \ ch2
     and p3 = momentum \ ch3 in
     printf "0, "%s" (if (F.sign\ rhs) < 0 then "+" else "-");
     begin match vertex with
     \mid Scalar4 coeff \rightarrow
         printf "(%s*%s*%s*%s)" (format_coupling coeff c) wf1 wf2 wf3
     \mid Scalar2\_Vector2 \ coeff \rightarrow
         let c = format\_coupling coeff c in
         begin match fusion with
          \mid F134 \mid F143 \mid F234 \mid F243 \rightarrow
              printf "%s*%s*(%s*%s)" c wf1 wf2 wf3
          \mid F314 \mid F413 \mid F324 \mid F423 \rightarrow
              printf "%s*%s*(%s*%s)" c wf2 wf1 wf3
          \mid F341 \mid F431 \mid F342 \mid F432 \rightarrow
              printf "%s*%s*(%s*%s)" c wf3 wf1 wf2
           F312 \mid F321 \mid F412 \mid F421 \rightarrow
              printf "(%s*%s*%s)*%s" c wf2 wf3 wf1
           F231 \mid F132 \mid F241 \mid F142 \rightarrow
              printf "(%s*%s*%s)*%s" c wf1 wf3 wf2
           F123 \mid F213 \mid F124 \mid F214 \rightarrow
               printf "(%s*%s*%s)*%s" c wf1 wf2 wf3
          end
     | Vector4 contractions \rightarrow
         begin match contractions with
          [] 
ightarrow invalid\_arg "Targets.print_current:_{\sqcup}Vector4_{\sqcup}[]"
```

```
\mid \ head \ :: \ tail \ \rightarrow
          printf "(";
          print_vector4 c wf1 wf2 wf3 fusion head;
          List.iter (print_add_vector4 c wf1 wf2 wf3 fusion) tail;
          printf ")"
     end
Dim8\_Vector4\_t\_0 contractions \rightarrow
     begin match contractions with
     [] \rightarrow invalid\_arg "Targets.print_current:_Vector4_[]"
     \mid head :: tail \rightarrow
          print_vector4_t_0 c wf1 p1 wf2 p2 wf3 p3 fusion head;
          List.iter (print_add_vector4 c wf1 wf2 wf3 fusion) tail;
     end
Dim8\_Vector4\_t\_1 \ contractions \rightarrow
     begin match contractions with
     [] 
ightarrow invalid\_arg "Targets.print_current:_{\sqcup}Vector4_{\sqcup}[]"
     \mid head :: tail \rightarrow
          print\_vector4\_t\_1 c wf1 p1 wf2 p2 wf3 p3 fusion head;
          List.iter\ (print\_add\_vector \rlap/\ c\ wf1\ wf2\ wf3\ fusion)\ tail;
     end
Dim8\_Vector4\_t\_2\ contractions \rightarrow
     begin match contractions with
     |\hspace{.06cm}|\hspace{.08cm}|\hspace{.08cm}| \rightarrow invalid\_arg "Targets.print_current:_{\sqcup}Vector4_{\sqcup}[]"
     \mid head :: tail \rightarrow
          print_vector4_t_2 c wf1 p1 wf2 p2 wf3 p3 fusion head;
          List.iter (print_add_vector4 c wf1 wf2 wf3 fusion) tail;
     end
Dim8\_Vector4\_m\_0 \ contractions \rightarrow
     begin match contractions with
     [] 
ightarrow invalid\_arg "Targets.print\_current: \_Vector4 \_[]"
     \mid head :: tail \rightarrow
          print\_vector4\_m\_0 c wf1 p1 wf2 p2 wf3 p3 fusion head;
          List.iter (print_add_vector4 c wf1 wf2 wf3 fusion) tail;
     end
Dim8\_Vector4\_m\_1 \ contractions \rightarrow
     begin match contractions with
     |\hspace{.06cm}[\hspace{.08cm}]\hspace{.08cm}
ightarrow \hspace{.08cm} invalid\_arg "Targets.print_current:_{\sqcup}Vector4_{\sqcup}[]"
     \mid head :: tail \rightarrow
          print_vector4_m_1 c wf1 p1 wf2 p2 wf3 p3 fusion head;
          List.iter (print_add_vector4 c wf1 wf2 wf3 fusion) tail;
     end
Dim8\_Vector4\_m\_7 \ contractions \rightarrow
     begin match contractions with
     | [] → invalid_arg "Targets.print_current: Uector4 []"
     \mid head :: tail \rightarrow
          print\_vector4\_m\_7 c wf1 p1 wf2 p2 wf3 p3 fusion head;
          List.iter (print_add_vector4 c wf1 wf2 wf3 fusion) tail;
  Vector4\_K\_Matrix\_tho\ (\_,\ poles) \rightarrow
     let pa, pb =
       begin match fusion with
        | (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214) \rightarrow (p1, p2)
         (F134 \mid F143 \mid F234 \mid F243 \mid F312 \mid F321 \mid F412 \mid F421) \rightarrow (p2, p3)
       | (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241) \rightarrow (p1, p3)
     printf "(%s*(%s*%s)*(%s*%s)*(%s*%s)@,*("
        c p1 wf1 p2 wf2 p3 wf3;
     List.iter (fun (coeff, pole) \rightarrow
       printf "+%s/((%s+%s)*(%s+%s)-%s)"
          (CM.constant_symbol coeff) pa pb pa pb
          (CM.constant\_symbol\ pole))
```

```
poles;
     printf ")*(-%s-%s-%s))" p1 p2 p3
  Vector4\_K\_Matrix\_jr\ (disc,\ contractions) \rightarrow
     let pa, pb =
        begin match disc, fusion with
          3, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234) \rightarrow (p1, p2)
          3, (F314 \mid F341 \mid F214 \mid F241 \mid F132 \mid F123 \mid F432 \mid F423) \rightarrow (p2, p3)
          3, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow (p1, p3)
          -, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214) <math>\rightarrow (p1, p2)
          \_, (F134 \mid F143 \mid F234 \mid F243 \mid F312 \mid F321 \mid F412 \mid F421) \rightarrow (p2, p3)
        | -, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241) \rightarrow (p1, p3)
       end in
     begin match contractions with
     [] 
ightarrow invalid\_arg "Targets.print_current: \( \text{Vector4_K_Matrix_jr} \( [] \) "
     \mid head :: tail \rightarrow
          printf "(";
          print_vector4_km c pa pb wf1 wf2 wf3 fusion head;
          List.iter (print_add_vector4_km c pa pb wf1 wf2 wf3 fusion)
             tail:
          printf ")"
     end
 Vector 4 \_K \_Matrix \_cf \_t0 \ (disc, \ contractions) \rightarrow
     let pa, pb, pc =
        begin match disc, fusion with
          3, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F321 \mid F324 \mid F234) \rightarrow (p1, p2, p3)
          3, (F314 \mid F341 \mid F214 \mid F241 \mid F132 \mid F123 \mid F432 \mid F423) \rightarrow (p2, p3, p1)
          3, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow (p1, p3, p2)
          -, (F341 \mid F431 \mid F342 \mid F432 \mid F123 \mid F213 \mid F124 \mid F214) <math>\rightarrow (p1, p2, p3)
         -, \; (F134 \mid F143 \mid F234 \mid F243 \mid F312 \mid F321 \mid F412 \mid F421) \; \rightarrow \; (p2, \; p3, \; p1)
        | -, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241) \rightarrow (p1, p3, p2)
       end in
     begin match contractions with
     |\ [\ ] 
ightarrow invalid_arg "Targets.print_current:_{\sqcup}Vector4_K_Matrix_cf_t0_{\sqcup}[]"
     \mid head :: tail \rightarrow
          printf "(";
          print_vector4_km_t_0 c pa pb wf1 p1 wf2 p2 wf3 p3 fusion head;
          List.iter (print_add_vector4_km c pa pb wf1 wf2 wf3 fusion)
             tail:
          printf ")"
     end
| Vector 4 \_K \_Matrix \_cf \_t1 (disc, contractions) \rightarrow
     let pa, pb =
        begin match disc, fusion with
          3, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F321 \mid F324 \mid F234) \rightarrow (p1, p2)
          3, (F314 \mid F341 \mid F214 \mid F241 \mid F132 \mid F123 \mid F432 \mid F423) \rightarrow (p2, p3)
          3, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow (p1, p3)
          -, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214) <math>\rightarrow (p1, p2)
          -, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421) <math>\rightarrow (p2, p3)
          -, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241) <math>\rightarrow (p1, p3)
       end in
     begin match contractions with
     |\ [\ ] 
ightarrow \ invalid\_arg "Targets.print_current:_{\sqcup}Vector4_K_Matrix_cf_t1_{\sqcup}[]"
     \mid head :: tail \rightarrow
          printf "(";
          print_vector4_km_t_1 c pa pb wf1 p1 wf2 p2 wf3 p3 fusion head;
          List.iter (print_add_vector4_km c pa pb wf1 wf2 wf3 fusion)
             tail:
          printf ")"
| Vector 4 \_K \_Matrix \_cf \_t2 (disc, contractions) \rightarrow
     let pa, pb =
```

```
begin match \mathit{disc}, \ \mathit{fusion} with
          3, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234) \rightarrow (p1, p2)
          3, (F314 \mid F341 \mid F214 \mid F241 \mid F132 \mid F123 \mid F432 \mid F423) \rightarrow (p2, p3)
          3, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow (p1, p3)
          \_, (F341 \mid F431 \mid F342 \mid F432 \mid F123 \mid F213 \mid F124 \mid F214) <math>\rightarrow (p1, p2)
          -, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421) <math>\rightarrow (p2, p3)
         -, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241) \rightarrow (p1, p3)
       end in
     begin match contractions with
     | [] 
ightarrow invalid\_arg "Targets.print_current:_{	t U}Vector4_K_Matrix_cf_t2_{	t U}[]"
     \mid head :: tail \rightarrow
          printf "(";
          print_vector4_km_t_2 c pa pb wf1 p1 wf2 p2 wf3 p3 fusion head;
          List.iter (print_add_vector4_km c pa pb wf1 wf2 wf3 fusion)
             tail:
          printf ")"
     end
 Vector 4\_K\_Matrix\_cf\_t\_rsi\ (disc,\ contractions) \rightarrow
     let pa, pb, pc =
       begin match disc, fusion with
          3, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234) \rightarrow (p1, p2, p3)
          3, (F314 \mid F341 \mid F214 \mid F241 \mid F132 \mid F123 \mid F432 \mid F423) \rightarrow (p2, p3, p1)
          3, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow (p1, p3, p2)
          \_, (F341 \mid F431 \mid F342 \mid F432 \mid F123 \mid F213 \mid F124 \mid F214) <math>\rightarrow (p1, p2, p3)
          \_, (F134 \mid F143 \mid F234 \mid F243 \mid F312 \mid F321 \mid F412 \mid F421) <math>\rightarrow (p2, p3, p1)
         \_, \; (F314 \mid F413 \mid F324 \mid F423 \mid F132 \mid F231 \mid F142 \mid F241) \; \rightarrow \; (p1, \; p3, \; p2)
       end in
     begin match contractions with
     [] 
ightharpoonup invalid\_arg "Targets.print\_current:_\UVector4_K_Matrix_cf_t_rsi_\[]"
     \mid head :: tail \rightarrow
          printf "(";
          print_vector4_km_t_rsi c pa pb pc wf1 p1 wf2 p2 wf3 p3 fusion head;
          List.iter (print_add_vector4_km c pa pb wf1 wf2 wf3 fusion)
             tail:
          printf ")"
     end
 Vector4\_K\_Matrix\_cf\_m0 \ (disc, \ contractions) \rightarrow
     let pa, pb =
       begin match disc, fusion with
         3, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234) \rightarrow (p1, p2)
          3, (F314 \mid F341 \mid F214 \mid F241 \mid F132 \mid F123 \mid F432 \mid F423) \rightarrow (p2, p3)
          3, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow (p1, p3)
          \_, (F341 \mid F431 \mid F342 \mid F432 \mid F123 \mid F213 \mid F124 \mid F214) \rightarrow (p1, p2)
          \_, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421) <math>\rightarrow (p2, p3)
         -, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241) <math>\rightarrow (p1, p3)
       end in
     begin match contractions with
     [] 
ightarrow invalid_arg "Targets.print_current: _{\sqcup}Vector4_K_Matrix_cf_m0_{\sqcup}[]"
     \mid head :: tail \rightarrow
          printf "(";
          print_vector4_km_m_0 c pa pb wf1 p1 wf2 p2 wf3 p3 fusion head;
          List.iter (print_add_vector4_km c pa pb wf1 wf2 wf3 fusion)
             tail:
          printf ")"
     end
Vector4\_K\_Matrix\_cf\_m1 (disc, contractions) \rightarrow
     let pa, pb =
       begin match disc, fusion with
         3, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F321 \mid F324 \mid F324 \mid F234) \rightarrow (p1, p2)
         3, (F314 \mid F341 \mid F214 \mid F241 \mid F132 \mid F123 \mid F432 \mid F423) \rightarrow (p2, p3)
       | 3, (F134 | F431 | F124 | F421 | F312 | F213 | F342 | F243) \rightarrow (p1, p3)
```

```
| -, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214) \rightarrow (p1, p2)
                -, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421) <math>\rightarrow (p2, p3)
            | -, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241) \rightarrow (p1, p3)
            end in
        begin match contractions with
         ] 
ightharpoonup invalid\_arg "Targets.print_current:\_Vector4_K_Matrix_cf_m1\_[]"
        \mid head :: tail \rightarrow
                printf "(";
                print\_vector4\_km\_m\_1 c pa pb wf1 p1 wf2 p2 wf3 p3 fusion head;
                List.iter (print_add_vector4_km c pa pb wf1 wf2 wf3 fusion)
                     tail:
                 printf ")"
        end
| Vector 4 \_K \_Matrix \_cf \_m7 (disc, contractions) \rightarrow
        \mathsf{let}\ pa,\ pb\ =
            begin match disc, fusion with
                3, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234) \rightarrow (p1, p2)
                3, (F314 \mid F341 \mid F214 \mid F241 \mid F132 \mid F123 \mid F432 \mid F423) \rightarrow (p2, p3)
                3, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow (p1, p3)
                -, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214) <math>\rightarrow (p1, p2)
                -, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421) <math>\rightarrow (p2, p3)
               -, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241) <math>\rightarrow (p1, p3)
             end in
        begin match contractions with
          [] 
ightarrow invalid\_arg "Targets.print_current:_{\sqcup}Vector4_K_Matrix_cf_m7_{\sqcup}[]"
        \mid head :: tail \rightarrow
                printf "(";
                print_vector4_km_m_7 c pa pb wf1 p1 wf2 p2 wf3 p3 fusion head;
                 List.iter (print_add_vector4_km c pa pb wf1 wf2 wf3 fusion)
                     tail:
                 printf ")"
        end
DScalar2\_Vector2\_K\_Matrix\_ms\ (disc,\ contractions) \rightarrow
        let p123 = Printf.sprintf "(-%s-%s-%s)" p1 p2 p3 in
        let pa, pb =
            begin match disc, fusion with
               3, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234) \rightarrow (p1, p2)
                3, (F314 \mid F341 \mid F214 \mid F241 \mid F132 \mid F123 \mid F432 \mid F423) \rightarrow (p2, p3)
               3, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow (p1, p3)
               4, \; (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F321 \mid F324 \mid F324 \mid F234) \; \rightarrow \; (p1, \; p2)
                4, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423) \rightarrow (p2, p3)
                4, (F134 | F431 | F124 | F421 | F312 | F213 | F342 | F243) \rightarrow
                                                                                                                                          (p1, p3)
                5, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234) \rightarrow
                                                                                                                                          (p1, p2)
                5, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423) \rightarrow
                                                                                                                                          (p2, p3)
                5, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow
                                                                                                                                          (p1, p3)
                6, \; (F134 \mid F132 \mid F314 \mid F312 \mid F241 \mid F243 \mid F421 \mid F423) \; \rightarrow \;
                                                                                                                                         (p1, p2)
                6, \; (F213 \mid F413 \mid F231 \mid F431 \mid F124 \mid F324 \mid F142 \mid F342) \; \rightarrow \;
                                                                                                                                          (p2, p3)
                6, (F143 \mid F123 \mid F341 \mid F321 \mid F412 \mid F214 \mid F432 \mid F234) \rightarrow (p1, p3)
                                                                                                                    F423) \rightarrow (p1, p2)
                7, (F134 | F132 | F314 | F312 | F241 | F243 | F421 |
                7, (F213 \mid F413 \mid F231 \mid F431 \mid F124 \mid F324 \mid F142 \mid F342) \rightarrow
                                                                                                                                         (p2, p3)
                7, (F143 | F123 | F341 | F321 | F412 | F214 | F432 |
                                                                                                                    F234) \rightarrow
                                                                                                                                          (p1, p3)
                                    F132 | F314 | F312 | F241 | F243 | F421 |
                                                                                                                    F423) \rightarrow
                8, (F134
                                                                                                                                         (p1, p2)
                8, (F213 \mid F413 \mid F231 \mid F431 \mid F124 \mid F324 \mid F142 \mid F342) \rightarrow (p2, p3)
               8, (F143 \mid F123 \mid F341 \mid F321 \mid F412 \mid F214 \mid F432 \mid F234) \rightarrow (p1, p3)
                \_, (F341 | F431 | F342 | F432 | F123 | F213 | F124 | F214) <math>\rightarrow (p1, p2)
                \_, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421) <math>\rightarrow (p2, p3)
               \_, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241) <math>\rightarrow (p1, p3)
            end in
        begin match contractions with
           [] 
ightarrow invalid\_arg \ "Targets.print\_current: \_DScalar2\_Vector4\_K\_Matrix\_ms_ [] \ "Targets.print_current: \_DScalar3\_Vector4\_K\_Matrix\_ms_ [] \ "Targets.print_current: \_DScalar3\_Ve
         \mid head :: tail \rightarrow
```

```
printf "(";
          print\_dscalar2\_vector2\_km
             c pa pb wf1 wf2 wf3 p1 p2 p3 p123 fusion head;
          List.iter\ (print\_add\_dscalar2\_vector2\_km
                                c pa pb wf1 wf2 wf3 p1 p2 p3 p123 fusion)
             tail;
          printf ")"
     end
DScalar2\_Vector2\_m\_0\_K\_Matrix\_cf (disc, contractions) \rightarrow
     let pa, pb =
        begin match disc, fusion with
          3, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234) \rightarrow (p1, p2)
                       F341 | F214 | F241 | F132 | F123 | F432 |
                                                                         F423) \rightarrow
                                                                                       (p2, p3)
                               F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow
                                                                                      (p1, p3)
          3, (F134
                      F431
          4, (F143 | F413 | F142 | F412 | F321 | F231 | F324 | F234) \rightarrow
                                                                                      (p1, p2)
          4, (F314)
                      F341 \mid F214 \mid F241 \mid F132 \mid F123 \mid F432 \mid F423) \rightarrow
                                                                                       (p2, p3)
          4, (F134)
                      F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow
                                                                                       (p1, p3)
          5, (F143 | F413 | F142 | F412 | F321 | F231 | F324 |
                                                                         F234) \rightarrow
                                                                                       (p1, p2)
          5, (F314 \mid F341 \mid F214 \mid F241 \mid F132 \mid F123 \mid F432 \mid F423) \rightarrow
                                                                                       (p2, p3)
          5, (F134
                      F431 | F124 | F421 | F312 | F213 | F342 |
                                                                         F243) \rightarrow
                                                                                       (p1, p3)
          6. (F134
                      F132 | F314 | F312 | F241 | F243 | F421
                                                                         F423) \rightarrow
                                                                                       (p1, p2)
          6. (F213
                      F413
                               F231 | F431 | F124 | F324
                                                                F142
                                                                         F342) \rightarrow
                                                                                       (p2, p3)
                      F123 \mid F341 \mid F321 \mid F412 \mid F214 \mid F432 \mid F234) \rightarrow
          6, (F143 |
                                                                                      (p1, p3)
          7, (F134 | F132 | F314 | F312 | F241 | F243 | F421 | F423) \rightarrow
                                                                                      (p1, p2)
          7, (F213 | F413 | F231 | F431 | F124 | F324 | F142 | F342) \rightarrow
                                                                                       (p2, p3)
          7, (F143 | F123 | F341 | F321 | F412 | F214 | F432 | F234) \rightarrow
                                                                                       (p1, p3)
          8, (F134 |
                      F132 | F314 | F312 | F241 | F243 | F421 |
                                                                         F423) \rightarrow
                                                                                       (p1, p2)
          8, (F213 |
                      F413 \mid F231 \mid F431 \mid F124 \mid F324 \mid F142 \mid F342) \rightarrow
                                                                                       (p2, p3)
                               F341 \mid F321 \mid F412 \mid F214 \mid F432 \mid F234) \rightarrow
          8, (F143 |
                      F123
                                                                                       (p1, p3)
          \_, (F341 \mid F431 \mid F342 \mid F432 \mid F123 \mid F213 \mid F124 \mid F214) <math>\rightarrow
                                                                                      (p1, p2)
          \_, (F134 \mid F143 \mid F234 \mid F243 \mid F312 \mid F321 \mid F412 \mid F421) <math>\rightarrow (p2, p3)
          -, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241) <math>\rightarrow (p1, p3)
       end in
     begin match contractions with
       [] → invalid_arg "Targets.print_current: DScalar2_Vector4_K_Matrix_cf_m0_[]"
      head :: tail \rightarrow
          printf "(";
          print\_dscalar2\_vector2\_m\_0\_km
             c pa pb wf1 wf2 wf3 p1 p2 p3 fusion head;
          List.iter\ (print\_add\_dscalar2\_vector2\_m\_0\_km
                                c pa pb wf1 wf2 wf3 p1 p2 p3 fusion)
             tail:
          printf ")"
     end
|DScalar2\_Vector2\_m\_1\_K\_Matrix\_cf (disc, contractions)| \rightarrow
     let pa, pb =
        begin match disc, fusion with
          3, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234) \rightarrow (p1, p2)
          3, (F314 \mid F341 \mid F214 \mid F241 \mid F132 \mid F123 \mid F432 \mid F423) \rightarrow (p2, p3)
          3, (F134 | F431 | F124 | F421 | F312 | F213 | F342 |
                                                                         F243) \rightarrow
                                                                                       (p1, p3)
                      F413 | F142 | F412 | F321
          4, (F143)
                                                        F231
                                                                 F324
                                                                         F234) \rightarrow
                                                                                       (p1, p2)
          4, (F314
                      F341
                               F214 | F241 | F132
                                                        F123
                                                                F432
                                                                         F423) \rightarrow
                                                                                       (p2, p3)
          4. (F134
                      F431 | F124 | F421 | F312 | F213 | F342 |
                                                                         F243) \rightarrow
                                                                                      (p1, p3)
          5, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234) \rightarrow
                                                                                      (p1, p2)
          5, (F314 \mid F341 \mid F214 \mid F241 \mid F132 \mid F123 \mid F432 \mid F423) \rightarrow
                                                                                      (p2, p3)
          5, (F134 | F431 | F124 | F421 | F312 | F213 | F342 | F243) \rightarrow
                                                                                       (p1, p3)
          6, (F134 | F132 | F314 | F312 | F241 | F243 | F421 | F423) \rightarrow
                                                                                      (p1, p2)
          6, (F213 \mid F413 \mid F231 \mid F431 \mid F124 \mid F324 \mid F142 \mid F342) \rightarrow
                                                                                       (p2, p3)
          6, (F143 \mid F123 \mid F341 \mid F321 \mid F412 \mid F214 \mid F432 \mid F234) \rightarrow
                                                                                       (p1, p3)
                      F132 \mid F314 \mid F312 \mid F241 \mid F243 \mid F421 \mid F423) \rightarrow
          7, (F134)
                                                                                      (p1, p2)
         7, (F213 \mid F413 \mid F231 \mid F431 \mid F124 \mid F324 \mid F142 \mid F342) \rightarrow (p2, p3)
```

```
7, (F143 \mid F123 \mid F341 \mid F321 \mid F412 \mid F214 \mid F432 \mid F234) \rightarrow (p1, p3)
          8, (F134 \mid F132 \mid F314 \mid F312 \mid F241 \mid F243 \mid F421 \mid F423) \rightarrow (p1, p2)
          8, (F213 \mid F413 \mid F231 \mid F431 \mid F124 \mid F324 \mid F142 \mid F342) \rightarrow (p2, p3)
          8, (F143 \mid F123 \mid F341 \mid F321 \mid F412 \mid F214 \mid F432 \mid F234) \rightarrow
                                                                                           (p1, p3)
          \_, (F341 \mid F431 \mid F342 \mid F432 \mid F123 \mid F213 \mid F124 \mid F214) <math>\rightarrow
                                                                                           (p1, p2)
          -, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421) <math>\rightarrow (p2, p3)
          -, \; (F314 \; | \; F413 \; | \; F324 \; | \; F423 \; | \; F132 \; | \; F231 \; | \; F142 \; | \; F241) \; \rightarrow \; (p1, \; p3)
        end in
     begin match contractions with
      [] 
ightarrow invalid\_arg "Targets.print_current:\\\\DScalar2\\\Vector4\\\K\\Matrix\\\cf_\m1\\\\[]\\"
     \mid head :: tail \rightarrow
          printf "(";
          print\_dscalar2\_vector2\_m\_1\_km
              c pa pb wf1 wf2 wf3 p1 p2 p3 fusion head;
           List.iter\ (print\_add\_dscalar2\_vector2\_m\_1\_km
                                 c pa pb wf1 wf2 wf3 p1 p2 p3 fusion)
              tail;
           printf ")"
     end
DScalar2\_Vector2\_m\_7\_K\_Matrix\_cf (disc, contractions) \rightarrow
     let pa, pb =
        begin match disc, fusion with
          3, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234) \rightarrow (p1, p2)
          3, (F314 \mid F341 \mid F214 \mid F241 \mid F132 \mid F123 \mid F432 \mid F423) \rightarrow (p2, p3)
          3, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow (p1, p3)
          4, (F143 | F413 | F142 | F412 | F321 | F231 | F324 | F234) \rightarrow
                                                                                           (p1, p2)
          4, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423) \rightarrow
                                                                                          (p2, p3)
          4, (F134 | F431 | F124 | F421 | F312 | F213 | F342 | F243) \rightarrow
                                                                                           (p1, p3)
          5, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234) \rightarrow
                                                                                           (p1, p2)
          5, (F314 \mid F341 \mid F214 \mid F241 \mid F132 \mid F123 \mid F432 \mid F423) \rightarrow
                                                                                           (p2, p3)
          5, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow
                                                                                           (p1, p3)
          6. (F134
                      | F132 | F314 | F312 | F241 | F243 | F421 |
                                                                            F423) \rightarrow
                                                                                           (p1, p2)
          6, (F213 | F413 | F231 | F431 | F124 | F324 | F142 |
                                                                            F342) \rightarrow (p2, p3)
          6, (F143 \mid F123 \mid F341 \mid F321 \mid F412 \mid F214 \mid F432 \mid F234) \rightarrow (p1, p3)
          7, (F134 | F132 | F314 | F312 | F241 | F243 | F421 |
                                                                            F423) \rightarrow
                                                                                           (p1, p2)
                                                                            F342) \rightarrow
          7, (F213 | F413 | F231 | F431 | F124 | F324 | F142 |
                                                                                           (p2, p3)
          7, (F143 \mid F123 \mid F341 \mid F321 \mid F412 \mid F214 \mid F432 \mid F234) \rightarrow (p1, p3)
          8, (F134 \mid F132 \mid F314 \mid F312 \mid F241 \mid F243 \mid F421 \mid F423) \rightarrow (p1, p2)
          8, (F213 \mid F413 \mid F231 \mid F431 \mid F124 \mid F324 \mid F142 \mid F342) \rightarrow (p2, p3)
          8, (F143 \mid F123 \mid F341 \mid F321 \mid F412 \mid F214 \mid F432 \mid F234) \rightarrow (p1, p3)
          \_, (F341 \mid F431 \mid F342 \mid F432 \mid F123 \mid F213 \mid F124 \mid F214) <math>\rightarrow (p1, p2)
          \_, (F134 \mid F143 \mid F234 \mid F243 \mid F312 \mid F321 \mid F412 \mid F421) <math>\rightarrow (p2, p3)
          \_, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241) <math>\rightarrow (p1, p3)
        end in
     begin match contractions with
     | \ [ \ ] 
ightarrow invalid\_arg "Targets.print_current:_{\sqcup}DScalar2_Vector4_K_Matrix_cf_m7_{\sqcup}[]"
      head :: tail \rightarrow
          printf "(";
          print\_dscalar2\_vector2\_m\_7\_km
              c pa pb wf1 wf2 wf3 p1 p2 p3 fusion head;
           List.iter\ (print\_add\_dscalar2\_vector2\_m\_7\_km
                                 c pa pb wf1 wf2 wf3 p1 p2 p3 fusion)
              tail:
           printf ")"
     end
DScalar4\_K\_Matrix\_ms (disc, contractions) \rightarrow
     let p123 = Printf.sprintf "(-%s-%s-%s)" p1 p2 p3 in
     let pa, pb =
        begin match disc, fusion with
          3, (F143 \mid F413 \mid F142 \mid F412 \mid F321 \mid F231 \mid F324 \mid F234) \rightarrow (p1, p2)
        | 3, (F314 | F341 | F214 | F241 | F132 | F123 | F432 | F423) \rightarrow (p2, p3)
```

```
3, (F134 \mid F431 \mid F124 \mid F421 \mid F312 \mid F213 \mid F342 \mid F243) \rightarrow (p1, p3)
         \_, (F341 \mid F431 \mid F342 \mid F432 \mid F123 \mid F213 \mid F124 \mid F214) <math>\rightarrow (p1, p2)
         -, (F134 | F143 | F234 | F243 | F312 | F321 | F412 | F421) <math>\rightarrow (p2, p3)
       | -, (F314 | F413 | F324 | F423 | F132 | F231 | F142 | F241) \rightarrow (p1, p3)
       end in
     begin match contractions with
     |\ [\ ] 
ightarrow invalid\_arg "Targets.print_current:_{\sqcup}DScalar4_K_Matrix_ms_{\sqcup}[]"
     \mid head :: tail \rightarrow
          printf "(";
          print_dscalar4_km
             c pa pb wf1 wf2 wf3 p1 p2 p3 p123 fusion head;
          List.iter\ (print\_add\_dscalar4\_km
                               c pa pb wf1 wf2 wf3 p1 p2 p3 p123 fusion)
             tail;
          printf ")"
     end
Dim8\_Scalar2\_Vector2\_1 coeff \rightarrow
     let c = format\_coupling coeff c in
          begin match \mathit{fusion} with
          \mid F134 \mid F143 \mid F234 \mid F243 \rightarrow
               printf "phi_phi2v_1(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
          \mid F314 \mid F413 \mid F324 \mid F423 \rightarrow
               printf "phi_phi2v_1(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf1 p1 wf3 p3
          \mid F341 \mid F431 \mid F342 \mid F432 \rightarrow
               printf "phi_phi2v_1(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
          \mid F312 \mid F321 \mid F412 \mid F421 \rightarrow
               printf "v_phi2v_1(%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1
          \mid F231 \mid F132 \mid F241 \mid F142 \rightarrow
               printf "v_phi2v_1(%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf3 p3 wf2
          \mid F123 \mid F213 \mid F124 \mid F214 \rightarrow
               printf "v_phi2v_1(%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3
          end
| Dim8\_Scalar2\_Vector2\_2 \ coeff \rightarrow
     let c = format\_coupling coeff c in
          begin match fusion with
          \mid F134 \mid F143 \mid F234 \mid F243 \rightarrow
               printf "phi_phi2v_2(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
          \mid F314 \mid F413 \mid F324 \mid F423 \rightarrow
               printf "phi_phi2v_2(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf1 p1 wf3 p3
          \mid F341 \mid F431 \mid F342 \mid F432 \rightarrow
               printf "phi_phi2v_2(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
          \mid F312 \mid F321 \mid F412 \mid F421 \rightarrow
               printf "v_phi2v_2(%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1
          \mid F231 \mid F132 \mid F241 \mid F142 \rightarrow
               printf "v_phi2v_2(%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf3 p3 wf2
          \mid F123 \mid F213 \mid F124 \mid F214 \rightarrow
               printf "v_phi2v_2(%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3
          end
| Dim8\_Scalar2\_Vector2\_m\_0 coeff \rightarrow
```

```
let c = format\_coupling coeff c in
          begin match fusion with
          \mid F134 \mid F143 \mid F234 \mid F243 \rightarrow
               printf "phi_phi2v_m_0(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
          \mid F314 \mid F413 \mid F324 \mid F423 \rightarrow
               printf "phi_phi2v_m_0(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c\ \textit{wf2}\ \textit{p2}\ \textit{wf1}\ \textit{p1}\ \textit{wf3}\ \textit{p3}
          \mid F341 \mid F431 \mid F342 \mid F432 \rightarrow
               printf "phi_phi2v_m_0(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
          \mid F312 \mid F321 \mid F412 \mid F421 \rightarrow
               printf "v_phi2v_m_0(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
          \mid F231 \mid F132 \mid F241 \mid F142 \rightarrow
               printf "v_phi2v_m_0(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf3 p3 wf2 p2
          \mid F123 \mid F213 \mid F124 \mid F214 \rightarrow
               printf "v_phi2v_m_0(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
Dim8\_Scalar2\_Vector2\_m\_1 coeff \rightarrow
    let c = format\_coupling coeff c in
          begin match fusion with
          \mid F134 \mid F143 \mid F234 \mid F243 \rightarrow
               printf "phi_phi2v_m_1(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
          \mid F314 \mid F413 \mid F324 \mid F423 \rightarrow
               printf "phi_phi2v_m_1(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf1 p1 wf3 p3
          \mid F341 \mid F431 \mid F342 \mid F432 \rightarrow
               printf "phi_phi2v_m_1(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c\ wf3\ p3\ wf2\ p2\ wf1\ p1
          \mid F312 \mid F321 \mid F412 \mid F421 \rightarrow
               printf "v_phi2v_m_1(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
          \mid F231 \mid F132 \mid F241 \mid F142 \rightarrow
               printf \ "v_phi2v_m_1(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf3 p3 wf2 p2
           F123 \mid F213 \mid F124 \mid F214 \rightarrow
               printf "v_phi2v_m_1(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
          end
Dim8\_Scalar2\_Vector2\_m\_7 coeff \rightarrow
    let c = format\_coupling coeff c in
          begin match fusion with
          \mid F134 \mid F143 \mid F234 \mid F243 \rightarrow
               printf "phi_phi2v_m_7(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
          \mid F314 \mid F413 \mid F324 \mid F423 \rightarrow
               printf "phi_phi2v_m_7(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf1 p1 wf3 p3
          \mid F341 \mid F431 \mid F342 \mid F432 \rightarrow
               printf "phi_phi2v_m_7(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
          \mid F312 \mid F321 \mid F412 \mid F421 \rightarrow
               printf "v_phi2v_m_7(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
          \mid F231 \mid F132 \mid F241 \mid F142 \rightarrow
               printf "v_phi2v_m_7(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf3 p3 wf2 p2
```

```
\mid F123 \mid F213 \mid F124 \mid F214 \rightarrow
               printf "v_phi2v_m_7(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
          end
\mid Dim8\_Scalar4 \ coeff \rightarrow
    \mathsf{let}\ c\ =\ format\_coupling\ coef\!\!f\ c\ \mathsf{in}
          begin match fusion with
                 F134 | F143 | F234 | F243 | F314 | F413 | F324 | F423
                 F341 | F431 | F342 | F432 | F312 | F321 | F412 | F421
                 F231 \mid F132 \mid F241 \mid F142 \mid F123 \mid F213 \mid F124 \mid F214 \rightarrow
                    printf "s_dim8s3_(%s,%s,%s,%s,%s,%s,%s,%s)"
                         c\ \textit{wf1}\ \textit{p1}\ \textit{wf2}\ \textit{p2}\ \textit{wf3}\ \textit{p3}
          end
\mid GBBG \ (coeff, fb, b, f) \rightarrow
     Fermions.print\_current\_g4 (coeff, fb, b, f) c wf1 wf2 wf3
           fusion
\mid Dim6\_H4\_P2 \ coeff \rightarrow
    let c = format\_coupling coeff c in
          begin match fusion with
                F134 | F143 | F234 | F243 | F314 | F413 | F324 | F423
                 F341 | F431 | F342 | F432 | F312 | F321
                                                                          F412 \mid F421
               \mid F231 \mid F132 \mid F241 \mid F142 \mid F123 \mid F213 \mid F124 \mid F214 \rightarrow
                    printf "hhhh_p2_\(\lambda\s,\%\s,\%\s,\%\s,\%\s,\%\s,\%\s\)"
                         c wf1 p1 wf2 p2 wf3 p3
          end
\mid Dim6\_AHWW\_DPB \ coeff \rightarrow
    let c = format\_coupling coeff c in
          begin match fusion with
          \mid F234 \rightarrow
               printf "a_hww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
           F243 \rightarrow
               printf "a_hww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf3 p3 wf2 p2
          \mid F342 \rightarrow
               printf "a_hww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf1 p1 wf2 p2
          \mid F324 \rightarrow
               printf "a_hww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf1 p1 wf3 p3
          \mid F423 \rightarrow
               printf "a_hww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf3 p3 wf1 p1
          \mid F432 \rightarrow
               printf "a_hww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
           F134 \rightarrow
               printf "h_aww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
          \mid F143 \rightarrow
               printf "h_aww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf3 p3 wf2 p2
          \mid F341 \rightarrow
               printf "h_aww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf1 p1 wf2 p2
          \mid F314 \rightarrow
               printf "h_aww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf1 p1 wf3 p3
          \mid F413 \rightarrow
               printf "h_aww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
```

```
c wf2 p2 wf3 p3 wf1 p1
         \mid F431 \rightarrow
              printf "h_aww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
          \mid F124 \rightarrow
              printf "w_ahw_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf1 p1 wf2 p2 wf3 p3
          \mid F142 \rightarrow
              printf "w_ahw_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf1 p1 wf3 p3 wf2 p2
          \mid F241 \rightarrow
              printf "w_ahw_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf3 p3 wf1 p1 wf2 p2
          \mid F214 \rightarrow
              printf "w_ahw_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf2 p2 wf1 p1 wf3 p3
         \mid F412 \rightarrow
              printf "w_ahw_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf2 p2 wf3 p3 wf1 p1
          \mid F421 \rightarrow
              printf "w_ahw_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf3 p3 wf2 p2 wf1 p1
          \mid F123 \rightarrow
              printf "(-1)*w_ahw_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf1 p1 wf2 p2 wf3 p3
          \mid F132 \rightarrow
              printf "(-1)*w_ahw_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf1 p1 wf3 p3 wf2 p2
          \mid F231 \rightarrow
              printf "(-1)*w_ahw_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf3 p3 wf1 p1 wf2 p2
          \mid F213 \rightarrow
              printf "(-1)*w_ahw_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf2 p2 wf1 p1 wf3 p3
          \mid F312 \rightarrow
              printf "(-1)*w_ahw_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf2 p2 wf3 p3 wf1 p1
           F321 \rightarrow
              printf "(-1)*w_ahw_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
         end
\mid Dim6\_AHWW\_DPW \ coeff \rightarrow
    \mathsf{let}\ c\ =\ format\_coupling\ coef\!\!f\ c\ \mathsf{in}
         begin match fusion with
          \mid F234 \rightarrow
              printf "a_hww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf1 p1 wf2 p2 wf3 p3
          \mid F243 \rightarrow
              printf "a_hww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf1 p1 wf3 p3 wf2 p2
          \mid F342 \rightarrow
              printf "a_hww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf3 p3 wf1 p1 wf2 p2
          \mid F324 \rightarrow
              printf "a_hww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf2 p2 wf1 p1 wf3 p3
              printf "a_hww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf2 p2 wf3 p3 wf1 p1
          \mid F432 \rightarrow
              printf "a_hww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
```

```
c wf3 p3 wf2 p2 wf1 p1
         \mid F134 \rightarrow
              printf "h_aww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf1 p1 wf2 p2 wf3 p3
         \mid F143 \rightarrow
              printf "h_aww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf1 p1 wf3 p3 wf2 p2
         \mid F341 \rightarrow
              printf "h_aww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf3 p3 wf1 p1 wf2 p2
         \mid F314 \rightarrow
              printf "h_aww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf2 p2 wf1 p1 wf3 p3
         \mid F413 \rightarrow
              printf "h_aww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf2 p2 wf3 p3 wf1 p1
         \mid F431 \rightarrow
              printf "h_aww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf3 p3 wf2 p2 wf1 p1
         \mid F124 \rightarrow
              printf "w_ahw_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf1 p1 wf2 p2 wf3 p3
         \mid F142 \rightarrow
              printf "w_ahw_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c\ wf1\ p1\ wf3\ p3\ wf2\ p2
         \mid F241 \rightarrow
              printf "w_ahw_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf3 p3 wf1 p1 wf2 p2
         \mid F214 \rightarrow
              printf "w_ahw_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf2 p2 wf1 p1 wf3 p3
         \mid F412 \rightarrow
              printf "w_ahw_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf2 p2 wf3 p3 wf1 p1
         \mid F421 \rightarrow
              printf "w_ahw_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf3 p3 wf2 p2 wf1 p1
         \mid F123 \rightarrow
              printf "(-1)*w_ahw_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf1 p1 wf2 p2 wf3 p3
         \mid F132 \rightarrow
              printf "(-1)*w_ahw_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf1 p1 wf3 p3 wf2 p2
         F231 \rightarrow
              printf "(-1)*w_ahw_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf3 p3 wf1 p1 wf2 p2
         \mid F213 \rightarrow
              printf "(-1)*w_ahw_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf2 p2 wf1 p1 wf3 p3
         \mid F312 \rightarrow
              printf "(-1)*w_ahw_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf2 p2 wf3 p3 wf1 p1
           F321 \rightarrow
              printf "(-1)*w_ahw_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                   c wf3 p3 wf2 p2 wf1 p1
         end
| Dim6\_AHWW\_DW coeff \rightarrow
    let c = format\_coupling coeff c in
        begin match fusion with
         \mid F234 \rightarrow
              printf "a_hww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
```

```
c wf1 p1 wf2 p2 wf3 p3
\mid F243 \rightarrow
     printf "a_hww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf1 p1 wf3 p3 wf2 p2
 F342 \rightarrow
    printf "a_hww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf3 p3 wf1 p1 wf2 p2
\mid F324 \rightarrow
     printf "a_hww_DW(%s, %s, %s, %s, %s, %s, %s, %s)"
          c wf2 p2 wf1 p1 wf3 p3
\mid F423 \rightarrow
     printf "a_hww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf2 p2 wf3 p3 wf1 p1
\mid F432 \rightarrow
     printf "a_hww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf3 p3 wf2 p2 wf1 p1
\mid F134 \rightarrow
     printf "h_aww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf1 p1 wf2 p2 wf3 p3
\mid F143 \rightarrow
    printf "h_aww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf1 p1 wf3 p3 wf2 p2
\mid F341 \rightarrow
     printf \ "h_aww_DW(%s, %s, %s, %s, %s, %s, %s, %s)"
          c wf3 p3 wf1 p1 wf2 p2
\mid F314 \rightarrow
     printf "h_aww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf2 p2 wf1 p1 wf3 p3
\mid F413 \rightarrow
     printf "h_aww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf2 p2 wf3 p3 wf1 p1
\mid F431 \rightarrow
     printf "h_aww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf3 p3 wf2 p2 wf1 p1
\mid F124 \rightarrow
     printf "w3_ahw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf1 p1 wf2 p2 wf3 p3
\mid F142 \rightarrow
     printf "w3_ahw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf1 p1 wf3 p3 wf2 p2
\mid F241 \rightarrow
     printf "w3_ahw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf3 p3 wf1 p1 wf2 p2
     printf "w3_ahw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf2 p2 wf1 p1 wf3 p3
\mid F412 \rightarrow
     printf "w3_ahw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf2 p2 wf3 p3 wf1 p1
\mid F421 \rightarrow
     printf "w3_ahw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c\ wf3\ p3\ wf2\ p2\ wf1\ p1
\mid F123 \rightarrow
     printf "w4_ahw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf1 p1 wf2 p2 wf3 p3
\mid F132 \rightarrow
     printf "w4_ahw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf1 p1 wf3 p3 wf2 p2
\mid F231 \rightarrow
     printf "w4_ahw_DW(%s,%s,%s,%s,%s,%s,%s)"
          c wf3 p3 wf1 p1 wf2 p2
```

```
\mid F213 \rightarrow
            printf "w4_ahw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F312 \rightarrow
            printf "w4_ahw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F321 \rightarrow
            printf "w4_ahw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       end
| Dim6\_Scalar2\_Vector2\_D coeff \rightarrow
  \mathsf{let}\ c\ =\ format\_coupling\ coef\!\!f\ c\ \mathsf{in}
       begin match fusion with
       \mid F234 \mid F134 \rightarrow
            printf "h_hww_D(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F243 \mid F143 \rightarrow
            printf "h_hww_D(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F342 \mid F341 \rightarrow
            printf "h_hww_D(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F324 \mid F314 \rightarrow
            printf "h_hww_D(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F423 \mid F413 \rightarrow
            printf "h_hww_D(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F432 \mid F431 \rightarrow
            printf "h_hww_D(%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       \mid F124 \mid F123 \rightarrow
            printf "w_hhw_D(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F142 \mid F132 \rightarrow
            printf "w_hhw_D(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
         F241 \mid F231 \rightarrow
            printf "w_hhw_D(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F214 \mid F213 \rightarrow
            printf "w_hhw_D(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F412 \mid F312 \rightarrow
            printf "w_hhw_D(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F421 \mid F321 \rightarrow
            printf "w_hhw_D(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       end
 Dim6\_Scalar2\_Vector2\_DP\ coeff\ 
ightarrow
  let c = format\_coupling coeff c in
       begin match fusion with
       \mid F234 \mid F134 \rightarrow
            printf "h_hww_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F342 \mid F341 \rightarrow
            printf "h_hww_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F423 \mid F413 \rightarrow
            printf "h_hww_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
```

```
c wf2 p2 wf3 p3 wf1 p1
          \mid F243 \mid F143 \rightarrow
               printf "h_hww_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf3 p3 wf2 p2
          \mid F324 \mid F314 \rightarrow
               printf "h_hww_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf1 p1 wf3 p3
          \mid F432 \mid F431 \rightarrow
               printf "h_hww_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
          \mid F123 \mid F124 \rightarrow
               printf "w_hhw_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
          \mid F231 \mid F241 \rightarrow
               printf "w_hhw_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf1 p1 wf2 p2
          \mid F312 \mid F412 \rightarrow
               printf "w_hhw_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf3 p3 wf1 p1
          \mid F132 \mid F142 \rightarrow
               printf "w_hhw_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf3 p3 wf2 p2
          \mid F213 \mid F214 \rightarrow
               printf "w_hhw_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf1 p1 wf3 p3
          \mid F321 \mid F421 \rightarrow
               printf "w_hhw_DP(%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
| Dim6\_Scalar2\_Vector2\_PB \ coeff \rightarrow
     let c = format\_coupling coeff c in
          begin match fusion with
          \mid F234 \mid F134 \rightarrow
               printf "h_hvv_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
          \mid F342 \mid F341 \rightarrow
               printf "h_hvv_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf1 p1 wf2 p2
          \mid F423 \mid F413 \rightarrow
               printf "h_hvv_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf3 p3 wf1 p1
          \mid F243 \mid F143 \rightarrow
               printf "h_hvv_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf3 p3 wf2 p2
          \mid F324 \mid F314 \rightarrow
               printf "h_hvv_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf1 p1 wf3 p3
          \mid F432 \mid F431 \rightarrow
               printf "h_hvv_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
          \mid F123 \mid F124 \rightarrow
               printf "v_hhv_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c\ \mathit{wf1}\ \mathit{p1}\ \mathit{wf2}\ \mathit{p2}\ \mathit{wf3}\ \mathit{p3}
          \mid F231 \mid F241 \rightarrow
               printf "v_hhv_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf1 p1 wf2 p2
          \mid F312 \mid F412 \rightarrow
               printf "v_hhv_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf3 p3 wf1 p1
          \mid F132 \mid F142 \rightarrow
               printf "v_hhv_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
```

```
c wf1 p1 wf3 p3 wf2 p2
          \mid F213 \mid F214 \rightarrow
               printf "v_hhv_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf1 p1 wf3 p3
          \mid F321 \mid F421 \rightarrow
               printf "v_hhv_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
         end
\mid Dim6\_HHZZ\_T \ coeff \rightarrow
    let c = format\_coupling coeff c in
         begin match fusion with
          \mid F234 \mid F134 \rightarrow
               printf "(%s)*(%s)*(%s)*(%s)" c wf1 wf2 wf3
           F342 \mid F341 \rightarrow
               printf "(%s)*(%s)*(%s)*(%s)" c wf3 wf1 wf2
          \mid F423 \mid F413 \rightarrow
               printf "(%s)*(%s)*(%s)*(%s)" c wf2 wf3 wf1
          \mid F243 \mid F143 \rightarrow
               printf "(%s)*(%s)*(%s)*(%s)" c wf1 wf3 wf2
          \mid F324 \mid F314 \rightarrow
               printf "(%s)*(%s)*(%s)*(%s)" c wf2 wf1 wf3
          \mid F432 \mid F431 \rightarrow
              printf "(%s)*(%s)*(%s)*(%s)" c wf3 wf2 wf1
           F123 \mid F124 \mid F231 \mid F241 \mid F312 \mid F412 \rightarrow
              printf "(%s)*(%s)*(%s)*(%s)" c wf1 wf2 wf3
          \mid F132 \mid F142 \mid F213 \mid F214 \mid F321 \mid F421 \rightarrow
               printf "(%s)*(%s)*(%s)*(%s)" c wf1 wf2 wf3
          end
| Dim6\_Vector4\_DW coeff \rightarrow
    let c = format\_coupling coeff c in
         begin match fusion with
          \mid F234 \mid F134 \rightarrow
               printf "a_aww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
         \mid F342 \mid F341 \rightarrow
               printf "a_aww_DW(%s, %s, %s, %s, %s, %s, %s, %s)"
                    c wf3 p3 wf1 p1 wf2 p2
          \mid F423 \mid F413 \rightarrow
               printf "a_aww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf3 p3 wf1 p1
          \mid F243 \mid F143 \rightarrow
               printf "a_aww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf3 p3 wf2 p2
          \mid F324 \mid F314 \rightarrow
               printf "a_aww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf1 p1 wf3 p3
          \mid F432 \mid F431 \rightarrow
               printf "a_aww_DW(%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
         \mid F124 \mid F123 \rightarrow
               printf "w_aaw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
          \mid F241 \mid F231 \rightarrow
               printf "w_aaw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf1 p1 wf2 p2
          \mid F412 \mid F312 \rightarrow
               printf "w_aaw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf3 p3 wf1 p1
          \mid F142 \mid F132 \rightarrow
               printf "w_aaw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
```

```
c wf1 p1 wf3 p3 wf2 p2
          \mid F214 \mid F213 \rightarrow
               printf "w_aaw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf1 p1 wf3 p3
          \mid F421 \mid F321 \rightarrow
               printf "w_aaw_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
         end
| Dim6\_Vector4\_W coeff \rightarrow
    let c = format\_coupling coeff c in
         begin match fusion with
          \mid F234 \mid F134 \rightarrow
               printf "a_aww_W(%s, %s, %s, %s, %s, %s, %s)"
                    c wf1 p1 wf2 p2 wf3 p3
          \mid F342 \mid F341 \rightarrow
               printf "a_aww_W(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf1 p1 wf2 p2
          \mid F423 \mid F413 \rightarrow
               printf "a_aww_W(%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf3 p3 wf1 p1
          \mid F243 \mid F143 \rightarrow
               printf "a_aww_W(%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf3 p3 wf2 p2
          \mid F324 \mid F314 \rightarrow
               printf "a_aww_W(%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf1 p1 wf3 p3
          \mid F432 \mid F431 \rightarrow
               printf "a_aww_W(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
          \mid F123 \mid F124 \rightarrow
               printf "w_aaw_W(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
          \mid F231 \mid F241 \rightarrow
               printf "w_aaw_W(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf1 p1 wf2 p2
          \mid F312 \mid F412 \rightarrow
               printf "w_aaw_W(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf3 p3 wf1 p1
          \mid F132 \mid F142 \rightarrow
               printf "w_aaw_W(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf3 p3 wf2 p2
          \mid F213 \mid F214 \rightarrow
               printf "w_aaw_W(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf2 p2 wf1 p1 wf3 p3
          \mid F321 \mid F421 \rightarrow
               printf "w_aaw_W(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf3 p3 wf2 p2 wf1 p1
  \mid Dim6\_HWWZ\_DW \ coeff \rightarrow
    let c = format\_coupling coeff c in
         begin match fusion with
          \mid F234 \rightarrow
               printf "h_wwz_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf2 p2 wf3 p3
          \mid F243 \rightarrow
               printf "h_wwz_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
                    c wf1 p1 wf3 p3 wf2 p2
         \mid F342 \rightarrow
               printf "h_wwz_DW(%s, %s, %s, %s, %s, %s, %s, %s)"
                    c wf3 p3 wf1 p1 wf2 p2
          \mid F324 \rightarrow
```

```
printf "h_wwz_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf1 p1 wf3 p3
F423 \rightarrow
    printf "h_wwz_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf3 p3 wf1 p1
\mid F432 \rightarrow
    printf "h_wwz_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf3 p3 wf2 p2 wf1 p1
 F124 \rightarrow
    printf "(-1)*w_hwz_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf1 p1 wf2 p2 wf3 p3
\mid F142 \rightarrow
    printf "(-1)*w_hwz_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf1 p1 wf3 p3 wf2 p2
\mid F241 \rightarrow
    printf "(-1)*w_hwz_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf3 p3 wf1 p1 wf2 p2
\mid F214 \rightarrow
    printf "(-1)*w_hwz_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf1 p1 wf3 p3
\mid F412 \rightarrow
    printf "(-1)*w_hwz_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf3 p3 wf1 p1
\mid F421 \rightarrow
    printf "(-1)*w_hwz_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf3 p3 wf2 p2 wf1 p1
\mid F134 \rightarrow
    printf "w_hwz_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf1 p1 wf2 p2 wf3 p3
\mid F143 \rightarrow
    printf "w_hwz_DW(%s,%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf1 p1 wf3 p3 wf2 p2
\mid F341 \rightarrow
    printf "w_hwz_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf3 p3 wf1 p1 wf2 p2
\mid F314 \rightarrow
    printf "w_hwz_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf1 p1 wf3 p3
\mid F413 \rightarrow
    printf "w_hwz_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf3 p3 wf1 p1
\mid F431 \rightarrow
    printf \ "w_hwz_DW(\%s,\%s,\%s,\%s,\%s,\%s,\%s)"
         c wf3 p3 wf2 p2 wf1 p1
\mid F123 \rightarrow
    printf "z_hww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf1 p1 wf2 p2 wf3 p3
\mid F132 \rightarrow
    printf "z_hww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf1 p1 wf3 p3 wf2 p2
\mid F231 \rightarrow
    printf "z_hww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf3 p3 wf1 p1 wf2 p2
\mid F213 \rightarrow
    printf "z_hww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf1 p1 wf3 p3
\mid F312 \rightarrow
    printf "z_hww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf3 p3 wf1 p1
\mid F321 \rightarrow
    printf "z_hww_DW(%s,%s,%s,%s,%s,%s,%s,%s)"
```

```
c wf3 p3 wf2 p2 wf1 p1
       end
\mid Dim6\_HWWZ\_DPB \ coeff \rightarrow
  let c = format\_coupling coeff c in
       begin match fusion with
       \mid F234 \rightarrow
            printf "h_wwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
         F243 \rightarrow
            printf "h_wwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c\ \textit{wf1}\ \textit{p1}\ \textit{wf3}\ \textit{p3}\ \textit{wf2}\ \textit{p2}
       \mid F342 \rightarrow
            printf "h_wwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F324 \rightarrow
            printf "h_wwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F423 \rightarrow
            printf "h_wwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F432 \rightarrow
            printf "h_wwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
         F124 \rightarrow
            printf "(-1)*w_hwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F142 \rightarrow
            printf "(-1)*w_hwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F241 \rightarrow
            printf "(-1)*w_hwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F214 \rightarrow
            printf "(-1)*w_hwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F412 \rightarrow
            printf "(-1)*w_hwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F421 \rightarrow
            printf "(-1)*w_hwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       \mid F134 \rightarrow
            printf "w_hwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F143 \rightarrow
            printf "w_hwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F341 \rightarrow
            printf "w_hwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F314 \rightarrow
            printf "w_hwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F413 \rightarrow
            printf "w_hwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F431 \rightarrow
            printf "w_hwz_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       \mid F123 \rightarrow
            printf "z_hww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
```

```
c wf1 p1 wf2 p2 wf3 p3
       \mid F132 \rightarrow
            printf "z_hww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
         F231 \rightarrow
            printf "z_hww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F213 \rightarrow
            printf "z_hww_DPB(%s, %s, %s, %s, %s, %s, %s, %s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F312 \rightarrow
            printf "z_hww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F321 \rightarrow
            printf "z_hww_DPB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       end
\mid Dim6\_HWWZ\_DDPW \ coeff \rightarrow
  let c = format\_coupling coeff c in
       begin match fusion with
       \mid F234 \rightarrow
            printf "h_wwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F243 \rightarrow
            printf "h_wwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F342 \rightarrow
            printf "h_wwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F324 \rightarrow
            printf "h_wwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F423 \rightarrow
            printf "h_wwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F432 \rightarrow
            printf "h_wwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       \mid F124 \rightarrow
            printf "(-1)*w_hwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F142 \rightarrow
            printf "(-1)*w_hwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F241 \rightarrow
            printf "(-1)*w_hwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F214 \rightarrow
            printf "(-1)*w_hwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F412 \rightarrow
            printf "(-1)*w_hwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F421 \rightarrow
            printf "(-1)*w_hwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       \mid F134 \rightarrow
            printf "w_hwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F143 \rightarrow
            printf "w_hwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
```

```
c wf1 p1 wf3 p3 wf2 p2
       \mid F341 \rightarrow
            printf "w_hwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F314 \rightarrow
           printf "w_hwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F413 \rightarrow
            printf "w_hwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F431 \rightarrow
            printf "w_hwz_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       \mid F123 \rightarrow
            printf "z_hww_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F132 \rightarrow
            printf "z_hww_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F231 \rightarrow
            printf "z_hww_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                c wf3 p3 wf1 p1 wf2 p2
       \mid F213 \rightarrow
            printf "z_hww_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F312 \rightarrow
            printf "z_hww_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
            printf "z_hww_DDPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       end
Dim6\_HWWZ\_DPW \ coeff \rightarrow
  let c = format\_coupling coeff c in
       begin match fusion with
       \mid F234 \rightarrow
            printf "h_wwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F243 \rightarrow
           printf "h_wwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F342 \rightarrow
            printf "h_wwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F324 \rightarrow
            printf "h_wwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F423 \rightarrow
            printf "h_wwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F432 \rightarrow
            printf "h_wwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       \mid F124 \rightarrow
           printf "(-1)*w_hwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F142 \rightarrow
            printf "(-1)*w_hwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F241 \rightarrow
           printf "(-1)*w_hwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
```

```
c wf3 p3 wf1 p1 wf2 p2
       \mid F214 \rightarrow
            printf "(-1)*w_hwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F412 \rightarrow
            printf "(-1)*w_hwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F421 \rightarrow
            printf "(-1)*w_hwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       \mid F134 \rightarrow
            printf "w_hwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F143 \rightarrow
            printf "w_hwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F341 \rightarrow
            printf "w_hwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F314 \rightarrow
            printf "w_hwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F413 \rightarrow
            printf "w_hwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F431 \rightarrow
            printf "w_hwz_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       \mid F123 \rightarrow
            printf "z_hww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F132 \rightarrow
            printf "z_hww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F231 \rightarrow
            printf "z_hww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
        F213 \rightarrow
            printf "z_hww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F312 \rightarrow
            printf "z_hww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
            printf "z_hww_DPW(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       end
| Dim6\_AHHZ\_D \ coeff \rightarrow
  \mathsf{let}\ c\ =\ format\_coupling\ coef\!\!f\ c\ \mathsf{in}
       begin match fusion with
       \mid F234 \rightarrow
            printf "a_hhz_D(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F243 \rightarrow
            printf "a_hhz_D(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F342 \rightarrow
            printf "a_hhz_D(%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F324 \rightarrow
            printf "a_hhz_D(%s,%s,%s,%s,%s,%s,%s,%s)"
```

```
c wf2 p2 wf1 p1 wf3 p3
\mid F423 \rightarrow
    printf "a_hhz_D(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf3 p3 wf1 p1
\mid F432 \rightarrow
    printf "a_hhz_D(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf3 p3 wf2 p2 wf1 p1
\mid F124 \rightarrow
    printf "h_ahz_D(%s, %s, %s, %s, %s, %s, %s)"
         c wf1 p1 wf2 p2 wf3 p3
\mid F142 \rightarrow
    printf "h_ahz_D(%s,%s,%s,%s,%s,%s,%s)"
         c wf1 p1 wf3 p3 wf2 p2
\mid F241 \rightarrow
    printf "h_ahz_D(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf3 p3 wf1 p1 wf2 p2
\mid F214 \rightarrow
    printf "h_ahz_D(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf1 p1 wf3 p3
\mid F412 \rightarrow
    printf "h_ahz_D(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf3 p3 wf1 p1
\mid F421 \rightarrow
    printf "h_ahz_D(%s,%s,%s,%s,%s,%s,%s)"
         c wf3 p3 wf2 p2 wf1 p1
\mid F134 \rightarrow
    printf "h_ahz_D(%s,%s,%s,%s,%s,%s,%s)"
         c wf1 p1 wf2 p2 wf3 p3
\mid F143 \rightarrow
    printf "h_ahz_D(%s,%s,%s,%s,%s,%s,%s)"
         c wf1 p1 wf3 p3 wf2 p2
\mid F341 \rightarrow
    printf "h_ahz_D(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf3 p3 wf1 p1 wf2 p2
\mid F314 \rightarrow
    printf "h_ahz_D(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf1 p1 wf3 p3
\mid F413 \rightarrow
    printf "h_ahz_D(%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf3 p3 wf1 p1
\mid F431 \rightarrow
    printf "h_ahz_D(%s,%s,%s,%s,%s,%s,%s)"
         c wf3 p3 wf2 p2 wf1 p1
\mid F123 \rightarrow
    printf "z_ahh_D(%s,%s,%s,%s,%s,%s,%s)"
         c wf1 p1 wf2 p2 wf3 p3
\mid F132 \rightarrow
    printf "z_ahh_D(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf1 p1 wf3 p3 wf2 p2
\mid F231 \rightarrow
    printf "z_ahh_D(%s,%s,%s,%s,%s,%s,%s,%s)"
         c\ wf3\ p3\ wf1\ p1\ wf2\ p2
 F213 \rightarrow
    printf "z_ahh_D(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf1 p1 wf3 p3
\mid F312 \rightarrow
    printf "z_ahh_D(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf2 p2 wf3 p3 wf1 p1
    printf "z_ahh_D(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf3 p3 wf2 p2 wf1 p1
```

```
end
| Dim6\_AHHZ\_DP \ coeff \rightarrow
  let c = format\_coupling coeff c in
       begin match \mathit{fusion} with
       \mid F234 \rightarrow
            printf "a_hhz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c\ \mathit{wf1}\ \mathit{p1}\ \mathit{wf2}\ \mathit{p2}\ \mathit{wf3}\ \mathit{p3}
       \mid F243 \rightarrow
            printf "a_hhz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F342 \rightarrow
            printf "a_hhz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F324 \rightarrow
            printf "a_hhz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F423 \rightarrow
            printf "a_hhz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F432 \rightarrow
            printf "a_hhz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       \mid F124 \rightarrow
            printf "h_ahz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F142 \rightarrow
            printf "h_ahz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F241 \rightarrow
            printf "h_ahz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F214 \rightarrow
            printf "h_ahz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F412 \rightarrow
            printf "h_ahz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F421 \rightarrow
            printf "h_ahz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       \mid F134 \rightarrow
            printf "h_ahz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F143 \rightarrow
            printf "h_ahz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F341 \rightarrow
            printf "h_ahz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F314 \rightarrow
            printf "h_ahz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F413 \rightarrow
            printf "h_ahz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F431 \rightarrow
            printf "h_ahz_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                  c wf3 p3 wf2 p2 wf1 p1
       \mid F123 \rightarrow
            printf "z_ahh_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                  c wf1 p1 wf2 p2 wf3 p3
```

```
\mid F132 \rightarrow
            printf "z_ahh_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F231 \rightarrow
            printf "z_ahh_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
         F213 \rightarrow
            printf "z_ahh_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F312 \rightarrow
            printf "z_ahh_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F321 \rightarrow
            printf "z_ahh_DP(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       end
\mid Dim6\_AHHZ\_PB \ coeff \rightarrow
  let c = format\_coupling coeff c in
       begin match fusion with
       \mid F234 \rightarrow
            printf "a_hhz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F243 \rightarrow
            printf "a_hhz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c\ wf1\ p1\ wf3\ p3\ wf2\ p2
       \mid F342 \rightarrow
            printf "a_hhz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F324 \rightarrow
            printf "a_hhz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F423 \rightarrow
            printf "a_hhz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F432 \rightarrow
            printf "a_hhz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       \mid F124 \rightarrow
            printf "h_ahz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F142 \rightarrow
            printf "h_ahz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
       \mid F241 \rightarrow
            printf "h_ahz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf1 p1 wf2 p2
       \mid F214 \rightarrow
            printf "h_ahz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf1 p1 wf3 p3
       \mid F412 \rightarrow
            printf "h_ahz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf2 p2 wf3 p3 wf1 p1
       \mid F421 \rightarrow
            printf "h_ahz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf3 p3 wf2 p2 wf1 p1
       \mid F134 \rightarrow
            printf "h_ahz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf2 p2 wf3 p3
       \mid F143 \rightarrow
            printf "h_ahz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
                 c wf1 p1 wf3 p3 wf2 p2
```

```
\mid F341 \rightarrow
    printf "h_ahz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf3 p3 wf1 p1 wf2 p2
\mid F314 \rightarrow
    printf "h_ahz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf1 p1 wf3 p3
\mid F413 \rightarrow
    printf "h_ahz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf3 p3 wf1 p1
\mid F431 \rightarrow
    printf "h_ahz_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf3 p3 wf2 p2 wf1 p1
\mid F123 \rightarrow
    printf "z_ahh_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf1 p1 wf2 p2 wf3 p3
\mid F132 \rightarrow
    printf "z_ahh_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf1 p1 wf3 p3 wf2 p2
\mid F231 \rightarrow
    printf "z_ahh_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf3 p3 wf1 p1 wf2 p2
 F213 \rightarrow
    printf "z_ahh_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
         c wf2 p2 wf1 p1 wf3 p3
 F312 \rightarrow
    printf "z_ahh_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf2 p2 wf3 p3 wf1 p1
\mid F321 \rightarrow
    printf "z_ahh_PB(%s,%s,%s,%s,%s,%s,%s,%s)"
          c wf3 p3 wf2 p2 wf1 p1
end
```



In principle, p4 could be obtained from the left hand side ...

```
\mid DScalar 4 \ contractions \rightarrow
     let p123 = Printf.sprintf "(-%s-%s-%s)" p1 p2 p3 in
     begin match contractions with
      [] \rightarrow invalid\_arg "Targets.print_current:_\DScalar4\\[]"
     \mid head :: tail \rightarrow
          printf "(";
          print\_dscalar4 c wf1 wf2 wf3 p1 p2 p3 p123 fusion head;
          List.iter (print_add_dscalar4
                          c wf1 wf2 wf3 p1 p2 p3 p123 fusion) tail;
          printf ")"
     end
DScalar2\_Vector2\ contractions \rightarrow
     let p123 = Printf.sprintf "(-%s-%s-%s)" p1 p2 p3 in
     begin match contractions with
     | \ | \ | \rightarrow invalid\_arg  "Targets.print_current:_{\sqcup}DScalar4_{\sqcup}[]"
     \mid head :: tail \rightarrow
          printf "(";
          print\_dscalar2\_vector2
            c wf1 wf2 wf3 p1 p2 p3 p123 fusion head;
          List.iter (print_add_dscalar2_vector2
                         c wf1 wf2 wf3 p1 p2 p3 p123 fusion) tail;
          printf ")"
     end
end
```



This reproduces the hack on page 510 and gives the correct results up to quartic vertices. Make sure that it is also correct in light of (15.3), i.e.

```
iT = i^{\text{#vertices}} i^{\text{#propagators}} \cdots = i^{n-2} i^{n-3} \cdots = -i(-1)^n \cdots
```

```
| Vn (UFO (c, v, s, fl, color), fusion, constant) \rightarrow
               if Color. Vertex.trivial color then
                     let g = CM.constant\_symbol\ constant
                     and chn = F.children \ rhs in
                     let wfs = List.map (multiple\_variable amplitude dictionary) chn
                     and ps = List.map momentum chn in
                     let n = List.length fusion in
                     let eps = if n \mod 2 = 0 then -1 else 1 in
                     printf "0," (if (eps \times F.sign\ rhs) < 0 then "-" else "+");
                      UFO. Targets. Fortran. fuse c v s fl g wfs ps fusion
               else
                     failwith "print_current: | nontrivial | color | structure"
let print\_propagator f p m gamma =
      let minus\_third = "(-1.0\_" ^!kind ^ "/3.0\_" ^!kind ^ ")" in
      let w =
            begin match CM.width f with
                         Vanishing \mid Fudged \rightarrow "0.0\_" ^!kind
                        Constant \mid Complex\_Mass \rightarrow gamma
                        Timelike \rightarrow "wd_tl(" ^ p ^ "," ^ gamma ^ ")"
                        Running → "wd_run(" ^ p ^ ", " ^ m ^ ", " ^ gamma ^ ")"
                        Custom \ f \rightarrow f ` "(" ` p ` ", " ` gamma ` ")"
            end in
      let \ cms =
            begin match CM.width \ f with
                       Complex\_Mass \rightarrow ".true."
                       _{\text{-}} \rightarrow ".false."
            end in
      match CM.propagator f with
            \mid Prop\_Scalar \rightarrow
                  printf "pr_phi(%s,%s,%s," p m w
            | Prop\_Col\_Scalar \rightarrow
                  printf "%s<sub>\uper_</sub>*pr_phi(%s,%s,%s," minus_third p m w
                Prop\_Ghost \rightarrow printf "(0,1)_{\sqcup}*_{\sqcup}pr\_phi(%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{\sqcup}%s,_{
            | Prop\_Spinor \rightarrow
                  printf "%s(%s,%s,%s,%s," Fermions.psi_propagator p m w cms
            \mid Prop\_ConjSpinor \rightarrow
                  printf "%s (%s ,%s ,%s ,%s ," Fermions.psibar\_propagator p m w cms
            \mid Prop\_Majorana \rightarrow
                  printf "%s(%s,%s,%s,%s," Fermions.chi_propagator p m w cms
            | Prop\_Col\_Majorana \rightarrow
                  printf "%s<sub>\underline*s</sub>(%s,%s,%s,%s," minus_third Fermions.chi_propagator p m w cms
            \mid Prop\_Unitarity \rightarrow
                  printf "pr_unitarity(%s,%s,%s,%s," p m w cms
            | Prop\_Col\_Unitarity \rightarrow
                  \mid Prop\_Feynman \rightarrow
                  printf "pr_feynman(%s," p
            | Prop\_Col\_Feynman \rightarrow
                  printf "%s<sub>\upper</sub>* \upperprese printf "%s<sub>\upper</sub> third p
            \mid Prop\_Gauge \ xi \rightarrow
                  printf "pr_gauge(%s,%s," p (CM.gauge_symbol xi)
            | Prop_Rxi xi \rightarrow
                  printf "pr_rxi(%s,%s,%s,%s," p m w (CM.gauge_symbol xi)
            | Prop_Tensor_2 \rightarrow
                  printf "pr_tensor(%s,%s,%s," p m w
```

```
\mid Prop\_Tensor\_pure \rightarrow
       printf "pr_tensor_pure(%s,%s,%s," p m w
     | Prop_Vector_pure \rightarrow
       printf "pr_vector_pure(%s,%s,%s," p m w
     Prop\_Vectorspinor \rightarrow
       printf "pr_grav(%s,%s,%s," p\ m\ w
       Aux\_Scalar \mid Aux\_Spinor \mid Aux\_ConjSpinor \mid Aux\_Majorana
       Aux\_Vector \mid Aux\_Tensor\_1 \rightarrow printf "("
       Aux\_Col\_Scalar \mid Aux\_Col\_Vector \mid Aux\_Col\_Tensor\_1 \rightarrow printf "%s_\*\_(" minus\_third
      Only\_Insertion \rightarrow printf "("
      Prop_-UFO \ name \rightarrow
       printf "pr_U_%s(%s,%s,%s," name\ p\ m\ w
let print\_projector f p m gamma =
  let minus\_third = "(-1.0\_" ^!kind ^ "/3.0\_" ^!kind ^ ")" in
  \mathsf{match}\ \mathit{CM}.\mathit{propagator}\ f\ \mathsf{with}
  | Prop\_Scalar \rightarrow
       printf "pj_phi(%s,%s," m gamma
  | Prop\_Col\_Scalar \rightarrow
       printf "%s<sub>□</sub>*<sub>□</sub>pj_phi(%s,%s," minus_third m gamma
  \mid Prop\_Ghost \rightarrow
       printf "(0,1)<sub>□</sub>*<sub>□</sub>pj_phi(%s,%s," m gamma
  | Prop\_Spinor \rightarrow
       printf "%s(%s,%s," Fermions.psi_projector p m gamma
  \mid Prop\_ConjSpinor \rightarrow
       printf "%s(%s,%s,%s," Fermions.psibar_projector p m gamma
   \mid Prop\_Majorana \rightarrow
       printf \ \verb"%s(%s,%s,"s") \ Fermions.chi\_projector \ p \ m \ gamma
    Prop\_Col\_Majorana \rightarrow
       Prop\_Unitarity \rightarrow
       printf "pj_unitarity(%s,%s,%s," p m gamma
    Prop\_Col\_Unitarity \rightarrow
       printf "%s<sub>\\\*\\</sub>pj_unitarity(%s,%s,\\'s,\'\ minus_third p m gamma
  | Prop\_Feynman | Prop\_Col\_Feynman \rightarrow
       invalid_arg "no⊔on-shell⊔Feynman⊔propagator!"
  | Prop_Gauge_{\rightarrow} \rightarrow
       invalid\_arg "no\sqcupon-shell\sqcupmassless\sqcupgauge\sqcuppropagator!"
  \mid Prop\_Rxi\_ \rightarrow
       invalid_arg "no⊔on-shell_Rxi_propagator!"
   | Prop_{-}Vectorspinor \rightarrow
       printf "pj_grav(%s,%s,%s," p m gamma
    Prop\_Tensor\_2 \rightarrow
       printf "pj_tensor(%s,%s,%s," p m gamma
   Prop\_Tensor\_pure \rightarrow
       invalid\_arg "no\sqcupon-shell\sqcuppure\sqcupTensor\sqcuppropagator!"
    Prop\_Vector\_pure \rightarrow
       invalid_arg "no⊔on-shellupure⊔Vector⊔propagator!"
    Aux\_Scalar \mid Aux\_Spinor \mid Aux\_ConjSpinor \mid Aux\_Majorana
    Aux\_Vector \mid Aux\_Tensor\_1 \rightarrow printf "("
    Aux\_Col\_Scalar \mid Aux\_Col\_Vector \mid Aux\_Col\_Tensor\_1 \rightarrow printf "%s_{\sqcup}*_{\sqcup}(" minus_third
    Only\_Insertion \rightarrow printf "("
    Prop\_UFO \ name \rightarrow
      invalid\_arg "no\sqcupon\sqcupshell\sqcupUFO\sqcuppropagator"
let print\_gauss \ f \ p \ m \ gamma =
  let minus\_third = "(-1.0\_" ^!kind ^ "/3.0\_" ^!kind ^ ")" in
  match CM.propagator f with
  | Prop\_Scalar \rightarrow
       printf "pg_phi(%s,%s,%s," p m gamma
  | Prop\_Ghost \rightarrow
       printf "(0,1)_{\square}*_{\square}pg_{phi}(%s,%s,%s," p m gamma)
```

```
| Prop\_Spinor \rightarrow
       printf "%s(%s,%s,%s," Fermions.psi_projector p m gamma
    Prop\_ConjSpinor \rightarrow
       printf "%s(%s,%s,%s," Fermions.psibar_projector p m gamma
    Prop\_Majorana \rightarrow
       printf "%s(%s,%s," Fermions.chi_projector p m gamma
    Prop\_Col\_Majorana \rightarrow
       printf "%s<sub>□</sub>*<sub>□</sub>%s (%s, %s, "s, "minus_third Fermions.chi_projector p m gamma
    Prop\_Unitarity \rightarrow
       printf "pg_unitarity(%s,%s,%s," p m gamma
  | Prop\_Feynman | Prop\_Col\_Feynman \rightarrow
       invalid\_arg "no\sqcupon-shell\sqcupFeynman\sqcuppropagator!"
  | Prop_Gauge_{-} \rightarrow
       invalid\_arg "no\sqcupon-shell\sqcupmassless\sqcupgauge\sqcuppropagator!"
  | Prop_Rxi_{-} \rightarrow
       invalid\_arg "no\sqcupon-shell\sqcupRxi\sqcuppropagator!"
    Prop\_Tensor\_2 \rightarrow
       printf "pg_tensor(%s,%s,%s," p m gamma
    Prop\_Tensor\_pure \rightarrow
       invalid\_arg "no_pure_tensor_propagator!"
    Prop\_Vector\_pure \rightarrow
        invalid_arg "no_pure_vector_propagator!"
     Aux\_Scalar \mid Aux\_Spinor \mid Aux\_ConjSpinor \mid Aux\_Majorana
     Aux\_Vector \mid Aux\_Tensor\_1 \rightarrow printf "("
     Only\_Insertion \rightarrow printf "("
    Prop\_UFO \ name \rightarrow
      invalid\_arg "no\sqcupUFO\sqcupgauss\sqcupinsertion"
  \mid \rightarrow invalid\_arg "targets:print_gauss:\sqcupnot\sqcupavailable"
let print_fusion_diagnostics amplitude dictionary fusion =
  if warn diagnose_gauge then begin
     let lhs = F.lhs fusion in
     let f = F.flavor lhs
     and v = variable lhs
     and p = momentum \ lhs in
     let mass = CM.mass\_symbol f in
     match CM.propagator f with
      Prop_Gauge _ | Prop_Feynman
     | Prop_Rxi_{-} | Prop_Unitarity \rightarrow
          printf "_____@[<2>%s_=" v;
          List.iter (print_current amplitude dictionary) (F.rhs fusion); nl ();
          begin match CM.goldstone\ f with
           None \rightarrow
               printf "____call_omega_ward_%s(\"%s\",%s,%s,%s)"
                 (suffix \ diagnose\_gauge) \ v \ mass \ p \ v; \ nl \ ()
          | Some (g, phase) \rightarrow
               let gv = add\_tag \ lhs \ (CM.flavor\_symbol \ g ` "\_" ` format\_p \ lhs) in
               printf "____call_omega_slavnov_%s"
                  (suffix \ diagnose\_gauge);
               printf "(@[\"%s\",%s,%s,%s,@,%s*%s)"
                  v mass p v (format_constant phase) gv; nl ()
          end
      \rightarrow ()
  end
let print_fusion amplitude dictionary fusion =
  let lhs = F.lhs fusion in
  \mathsf{let}\; f \; = \; F.\mathit{flavor}\; \mathit{lhs} \; \mathsf{in}
  printf "_{ \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} @[<2>\%s_{ \sqcup} = @,_{ \sqcup}" (multiple\_variable \ amplitude \ dictionary \ lhs);
  if F.on\_shell amplitude lhs then
     print_projector f (momentum lhs)
       (CM.mass\_symbol\ f)\ (CM.width\_symbol\ f)
```

```
else
     if F.is\_gauss amplitude lhs then
       print\_gauss\ f\ (momentum\ lhs)
          (CM.mass\_symbol\ f)\ (CM.width\_symbol\ f)
     else
       print\_propagator\ f\ (momentum\ lhs)
          (CM.mass\_symbol\ f)\ (CM.width\_symbol\ f);
  List.iter (print_current amplitude dictionary) (F.rhs fusion);
  printf ")"; nl ()
let print_momenta seen_momenta amplitude =
  List.fold\_left (fun seen f \rightarrow
    let wf = F.lhs f in
    let p = F.momentum\_list \ wf in
    if \neg (PSet.mem \ p \ seen) then begin
       let rhs1 = List.hd (F.rhs f) in
       printf "_{ \sqcup \sqcup \sqcup \sqcup \sqcup } %s = _{ \sqcup } %s " (momentum wf)
          (String.concat " \bot + \bot "
              (List.map momentum (F.children rhs1))); nl ()
     end;
     PSet.add \ p \ seen)
     seen\_momenta (F. fusions amplitude)
let print_fusions dictionary fusions =
  List.iter
     (fun (f, amplitude) \rightarrow
       print\_fusion\_diagnostics amplitude dictionary f;
       print\_fusion \ amplitude \ dictionary \ f)
    fusions
```

\$

The following will need a bit more work, because the decision when to $reverse_braket$ for UFO models with Majorana fermions needs collaboration from UFO.Targets.Fortran.fuse which is called by $print_current$. See the function $UFO_targets.Fortran.jrr_print_majorana_current_transposing$ for illustration (the function is never used and only for documentation).

```
let spins\_of\_rhs \ rhs =
  List.map (fun wf \rightarrow CM.lorentz (F.flavor wf)) (F.children rhs)
let spins\_of\_ket ket =
  match ThoList.uniq (List.map spins_of_rhs ket) with
    [spins] \rightarrow spins
    [\ ] \ 	o \ failwith \ "Targets.Fortran.spins_of_ket: \_empty"
       \rightarrow [] (* HACK! *)
{\tt let} \ print\_braket \ amplitude \ dictionary \ name \ braket \ =
  let bra = F.bra braket
  and ket = F.ket \ braket in
  let spin_bra = CM.lorentz (F.flavor bra)
  and spins\_ket = spins\_of\_ket \ ket \ in
  let \ vintage = true (* F.vintage *) in
  printf "_____@[<2>%s_=_%s@,_+_" name\ name;
  if Fermions.reverse_braket vintage spin_bra spins_ket then
     begin
       printf "@,(";
       List.iter (print_current amplitude dictionary) ket;
       printf ")*%s" (multiple_variable amplitude dictionary bra)
     end
  else
       printf "%s*@,(" (multiple_variable amplitude dictionary bra);
       List.iter (print_current amplitude dictionary) ket;
       printf ")"
     end;
```

nl()

$$iT = i^{\text{#vertices}} i^{\text{#propagators}} \cdots = i^{n-2} i^{n-3} \cdots = -i(-1)^n \cdots$$
(15.3)



tho: we write some brakets twice using different names. Is it useful to cache them?

```
let print_brakets dictionary amplitude =
  let name = flavors\_symbol (flavors amplitude) in
  printf "_____%s__=_0" name; \ nl \ ();
  List.iter (print_braket amplitude dictionary name) (F.brakets amplitude);
  let n = List.length (F.externals amplitude) in
  if n \mod 2 = 0 then begin
     printf "_____@[<2>%s_=@,__-_\%s__!_\%d_vertices,_\%d_propagators"
        name name (n-2)(n-3); nl()
     printf "_____!_\%s__=_\%s__!_\%d_vertices,_\%d_propagators"
        name name (n-2)(n-3); nl()
  end:
  let s = F.symmetry amplitude in
  if s > 1 then
     printf "______@[<2>%s_=@,__%s@,__/_sqrt(%d.0_%s)__!_symmetry_factor" name\ name\ s\ !kind
  else
     printf "_____!_unit_symmetry_factor";
  nl()
let print\_incoming \ wf =
  let p = momentum wf
  and s = spin wf
  and f = F.flavor wf in
  let m = CM.mass\_symbol f in
  match CM.lorentz f with
     Scalar \rightarrow printf "1"
     BRS\ Scalar\ 	o\ printf "(0,-1)_{\sqcup}*_{\sqcup}(\sl_{\boxtimes}*_{\sqcup}\sl_{\boxtimes}-_{\sqcup}\sl_{\boxtimes}**2)" p\ p\ m
     Spinor \rightarrow
        printf "%s_{\sqcup}(%s,_{\sqcup}-_{\sqcup}%s,_{\sqcup}%s)" Fermions.psi\_incoming\ m\ p\ s
   \mid BRS \ Spinor \rightarrow
        printf "%s_{\sqcup}(%s,_{\sqcup}-_{\sqcup}%s,_{\sqcup}%s)" Fermions.brs\_psi\_incoming\ m\ p\ s
   | ConjSpinor \rightarrow
        printf "%s_{\sqcup}(%s,_{\sqcup}-_{\sqcup}%s,_{\sqcup}%s)" Fermions.psibar\_incoming\ m\ p\ s
   \mid BRS\ ConjSpinor \rightarrow
        printf "%s_{\sqcup}(%s,_{\sqcup}-_{\sqcup}%s,_{\sqcup}%s)" Fermions.brs\_psibar\_incoming\ m\ p\ s
        printf "%s_\(\%s,\(\_\_\%s,\\_\%s)\)" Fermions.chi\_incoming\ m\ p\ s
     Maj\_Ghost \rightarrow printf "ghost\_(%s,\_-\_%s,\_%s)" m p s
     BRS\ Majorana\ 
ightarrow
        printf "%s_{\sqcup}(%s,_{\sqcup}-_{\sqcup}%s,_{\sqcup}%s)" Fermions.brs\_chi\_incoming\ m\ p\ s
     Vector \mid Massive\_Vector \rightarrow
        printf "eps_{\sqcup}(%s,_{\sqcup}-_{\sqcup}%s,_{\sqcup}%s)" m\ p\ s
     BRS\ Vector\ |\ BRS\ Massive\_Vector\ 	o\ printf
           "(0,1)_{\sqcup}*_{\sqcup}(%s_{\sqcup}*_{\sqcup}%s_{\sqcup}-_{\sqcup}%s**2)_{\sqcup}*_{\sqcup}eps_{\sqcup}(%s,_{\sqcup}-%s,_{\sqcup}%s)" p p m m p s
     Vectorspinor \mid BRS \ Vectorspinor \rightarrow
        printf "%s_{\sqcup}(%s,_{\sqcup}-_{\sqcup}%s,_{\sqcup}%s)" Fermions.grav\_incoming\ m\ p\ s
     Tensor\_1 \rightarrow invalid\_arg "Tensor\_1\sqcuponly\sqcupinternal"
     Tensor_2 \rightarrow printf "eps2_{\sqcup}(%s,_{\sqcup}-_{\sqcup}%s,_{\sqcup}%s)" m p s
   \mid \ \_ \ 
ightarrow \ invalid\_arg "no\sqcupsuch\sqcupBRST\sqcuptransformations"
let print\_outgoing \ wf =
  let p = momentum wf
  and s = spin wf
  and f = F.flavor wf in
  let m = CM.mass\_symbol f in
```

```
match CM.lorentz f with
     Scalar \rightarrow printf "1"
     BRS\ Scalar\ 	o\ printf\ "(0,-1) \sqcup * \sqcup (%s \sqcup * \sqcup %s \sqcup - \sqcup %s**2) "\ p\ p\ m
     Spinor \rightarrow
        printf "%s_{\sqcup}(%s,_{\sqcup}%s,_{\sqcup}%s)" Fermions.psi\_outgoing\ m\ p\ s
     BRS\ Spinor\ 	o
        ConjSpinor \rightarrow
        printf "%s_{\sqcup}(%s,_{\sqcup}%s,_{\sqcup}%s)" Fermions.psibar\_outgoing\ m\ p\ s
    BRS\ ConjSpinor \rightarrow
        printf "%s_{\sqcup}(%s,_{\sqcup}%s)" Fermions.brs\_psibar\_outgoing\ m\ p\ s
   Majorana \rightarrow
        printf "%s_{\sqcup}(%s,_{\sqcup}%s,_{\sqcup}%s)" Fermions.chi\_outgoing\ m\ p\ s
   \mid BRS \ Majorana \rightarrow
        printf "\sl_{\sl}(\sl_{\sl}s,\sl_{\sl}s)" Fermions.brs\_chi\_outgoing\ m\ p\ s
     Maj\_Ghost \rightarrow printf "ghost (%s, \, \, %s, \, \, %s) " m p s
     Vector \mid Massive\_Vector \rightarrow
        printf "conjg_(eps_(%s,_%s,_%s))" m p s
     BRS\ Vector\ |\ BRS\ Massive\_Vector\ 	o\ printf
           "(0,1)_{\sqcup}*_{\sqcup}(%s*%s-%s**2)_{\sqcup}*_{\sqcup}(conjg_{\sqcup}(eps_{\sqcup}(%s,_{\sqcup}%s,_{\sqcup}%s)))" p p m m p s
     Vectorspinor \mid BRS \ Vectorspinor \rightarrow
        printf "%s_{\sqcup}(%s,_{\sqcup}%s,_{\sqcup}%s)" Fermions.grav\_incoming\ m\ p\ s
     Tensor\_1 \rightarrow invalid\_arg "Tensor\_1\sqcuponly\sqcupinternal"
     Tensor_2 \rightarrow printf "conjg_{\sqcup}(eps2_{\sqcup}(%s,_{\sqcup}%s,_{\sqcup}%s))" m p s
     BRS \_ \rightarrow invalid\_arg "no\_such\_BRST\_transformations"
let print_external_momenta amplitude =
  let externals =
     List.combine
        (F.externals amplitude)
        (List.map (fun \_ \rightarrow true) (F.incoming amplitude) @
         List.map \; (\mathsf{fun} \; \_ \; \to \; \mathsf{false}) \; (F.outgoing \; amplitude)) \; \mathsf{in}
   List.iter (fun (wf, incoming) \rightarrow
     if incoming then
        printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup} %s_{\sqcup} =_{\sqcup} -_{\sqcup} k(:, %d)_{\sqcup} !_{\sqcup} incoming"
           (momentum \ wf) \ (ext\_momentum \ wf)
     else
        printf "_____%s_=___k(:,%d)__!_outgoing"
           (momentum \ wf) \ (ext\_momentum \ wf); \ nl \ ()) \ externals
let print_externals seen_wfs amplitude =
  let externals =
     List.combine \\
        (F.externals amplitude)
        (List.map (fun _{\perp} \rightarrow \text{true}) (F.incoming amplitude) @
          List.map (fun \_ \rightarrow false) (F.outgoing \ amplitude)) in
   List.fold\_left (fun seen (wf, incoming) \rightarrow
     if \neg (WFSet.mem wf seen) then begin
        printf "_____@[<2>%s_=@,_" (variable\ wf);
        (if incoming then print_incoming else print_outgoing) wf; nl ()
      WFSet.add wf seen) seen_wfs externals
let flavors_sans_color_to_string flavors =
   String.concat "\Box" (List.map M.flavor\_to\_string flavors)
let process_sans_color_to_string (fin, fout) =
   flavors\_sans\_color\_to\_string\ fin\ ^ "_->_ "
  flavors_sans_color_to_string fout
let print\_fudge\_factor amplitude =
  let name = flavors\_symbol (flavors amplitude) in
   List.iter (fun wf \rightarrow
     let p = momentum wf
```

```
and f = F.flavor\ wf in match CM.width\ f with  |\ Fudged \rightarrow \\ |\ let\ m = CM.mass\_symbol\ f \\ |\ and\ w = CM.width\_symbol\ f \ in \\ |\ printf\ "$_{\cuplum} = 1_{\cuplum} (\%s_{\cuplum})_{\cuplum} (\%s_{\cuplum})_{\cup
```

Spin, Flavor & Color Tables

The following abomination is required to keep the number of continuation lines as low as possible. FORTRAN77-style DATA statements are actually a bit nicer here, but they are nor available for *constant* arrays.



We used to have a more elegant design with a sentinel 0 added to each initializer, but some revisions of the Compaq/Digital Compiler have a bug that causes it to reject this variant.



The actual table writing code using reshape should be factored, since it's the same algorithm every time.

```
let print_integer_parameter name value =
  printf " \sqcup \sqcup @[<2> integer, \sqcup parameter \sqcup : : \sqcup \%s \sqcup = \sqcup \%d" name value; nl ()
let print_real_parameter name value =
  printf "_{\sqcup\sqcup} @ [<2> real(kind=%s),_{\sqcup} parameter_{\sqcup}::_{\sqcup} %s_{\sqcup} =_{\sqcup} %d"
     !kind name value; nl ()
let print_logical_parameter name value =
  printf " \sqcup \sqcup @[<2> logical, \sqcup parameter \sqcup : : \sqcup %s \sqcup = \sqcup . %s."
     name (if value then "true" else "false"); nl ()
let num\_particles\_in amplitudes =
  match CF.flavors amplitudes with
    [] \rightarrow 0
   | (fin, \_) :: \_ \rightarrow List.length fin
let num\_particles\_out amplitudes =
  match CF.flavors amplitudes with
   | [] \rightarrow 0
  | (\_, fout) :: \_ \rightarrow List.length fout
let num\_particles amplitudes =
  match CF.flavors amplitudes with
   [] \rightarrow 0
   | (fin, fout) :: \_ \rightarrow List.length fin + List.length fout
module \ CFlow = Color.Flow
let num\_color\_flows amplitudes =
  if !amp\_triv then
     1
  else
     List.length (CF.color\_flows amplitudes)
let num\_color\_indices\_default = 2 (* Standard model *)
let num\_color\_indices amplitudes =
  try CFlow.rank (List.hd (CF.color\_flows amplitudes)) with \_ \rightarrow num\_color\_indices\_default
let color\_to\_string \ c =
```

```
"(" ^{(String.concat} "," (List.map\ (Printf.sprintf\ "%3d")\ c)) ^{(String.concat)}")"
let cflow\_to\_string \ cflow =
  String.concat "\Box" (List.map\ color\_to\_string\ (CFlow.in\_to\_lists\ cflow)) ^{\ }"_{\Box}->_{\Box}" ^{\ }
  String.concat "" (List.map color_to_string (CFlow.out_to_lists cflow))
let protected = ", protected" (* Fortran 2003! *)
let print\_spin\_table name tuples =
  printf "_{\sqcup\sqcup} @[<2> integer,_{\sqcup} dimension(n_prt,n_hel),_{\sqcup} save%s_{\sqcup} ::_{\sqcup} table_spin_%s"
     protected name; nl();
  match tuples with
   [] \rightarrow ()
        ignore\ (List.fold\_left\ (fun\ i\ (tuple1,\ tuple2)\ 	o
          printf "_{\sqcup \sqcup} @[<2> data_{\sqcup}table\_spin_%s(:,%4d)_{\sqcup}/_{\sqcup}%s_{\sqcup}/" name i
             (String.concat ", " (List.map (Printf.sprintf "%2d") (tuple1 @ tuple2)));
          nl(); succ(i) 1 tuples)
let print\_spin\_tables amplitudes =
  (* print_spin_table_old "s" "states_old" (CF.helicities amplitudes); *)
  print_spin_table "states" (CF.helicities amplitudes);
  nl()
let print_flavor_table name tuples =
  printf "uu0[<2>integer,udimension(n_prt,n_flv),usave%su::utable_flavor_%s"]
     protected name; nl ();
  match tuples with
   | [] \rightarrow ()
        ignore\ (List.fold\_left\ (fun\ i\ tuple\ 
ightarrow
          printf " \sqcup \sqcup @[<2> data \sqcup table_flavor_%s(:,%4d) \sqcup / \sqcup %s \sqcup / \sqcup ! \sqcup %s" name i 
             (String.concat ", "
                 (List.map (fun f \rightarrow Printf.sprintf "%3d" (M.pdg f)) tuple))
             (String.concat `` \ ' \ (List.map M.flavor_to_string tuple));
          nl(); succ(i) 1 tuples)
let print_flavor_tables amplitudes =
   (* let n = num\_particles amplitudes in *)
  (*print\_flavor\_table\_old\ n "f" "states\_old" (List.map (fun (fin, fout) \rightarrow fin @ fout) (CF.flavors amplitudes));
  print_flavor_table "states"
     (List.map (fun (fin, fout) \rightarrow fin @ fout) (CF.flavors amplitudes));
  nl()
let num\_flavors amplitudes =
   List.length (CF.flavors amplitudes)
let print_color_flows_table tuples =
  if !amp\_triv then begin
     printf
        "_{\sqcup\sqcup}@[<2> integer, _{\sqcup}dimension(n_{cindex},n_{prt},n_{cflow}), _{\sqcup}save%s_{\sqcup}: _{\sqcup}table_{color_flows} =_{\sqcup}0"
        protected; nl ();
     end
  else begin
        "uu@[<2>integer,udimension(n_cindex,n_prt,n_cflow),usave%su::utable_color_flows"
        protected; nl();
  end;
  if \neg !amp\_triv then begin
     match tuples with
     | [] \rightarrow ()
     \mid \_ :: \_ as tuples \rightarrow
          ignore\ (List.fold\_left\ (fun\ i\ tuple\ 
ightarrow
             begin match CFlow.to_lists tuple with
```

*)

```
| [] \rightarrow ()
              | cf1 :: cfn \rightarrow
                    printf "_{\sqcup \sqcup} @[<2> data_{\sqcup} table_color_flows(:,:,%4d)_{\sqcup}/" i;
                    printf "Q_{\bot}%s" (String.concat "," (List.map\ string\_of\_int\ cf1));
                    List.iter (function cf \rightarrow
                       printf ", Q_{\sqcup\sqcup}%s" (String.concat "," (List.map\ string\_of\_int\ cf))) cfn;
                    printf "@_{\sqcup}/"; nl ()
              end:
              succ i) 1 tuples)
   end
let print\_ghost\_flags\_table tuples =
   if !amp\_triv then begin
     printf
        "\cup_{\cup}0[<2>logical,_{\cup}dimension(n_prt,n_cflow),_{\cup}save%s_{\cup}::_{\cup}table_ghost_flags_{\cup}=_{\cup}F"
        protected; nl();
      end
   else begin
     printf
         "_{\sqcup\sqcup}@[<2>\logical,_{\sqcup}dimension(n_prt,n_cflow),_{\sqcup}save%s_{\sqcup}::_{\sqcup}table_ghost_flags"
        protected; nl ();
     match tuples with
     | [] \rightarrow ()
      -
           ignore~(List.fold\_left~(fun~i~tuple~\rightarrow
              begin match CFlow.ghost_flags tuple with
              | [] \rightarrow ()
              \mid gf1 :: gfn \rightarrow
                    printf "_{\sqcup \sqcup} @[<2> data_{\sqcup} table_ghost_flags(:, %4d)_{\sqcup}/" i;
                    printf "@_{\sqcup}%s" (if gf1 then "T" else "F");
                    List.iter (function gf \rightarrow printf ", Q_{\sqcup\sqcup}%s" (if gf then "T" else "F")) gfn;
                    printf " \_ / ";
                    nl()
              end;
              succ i) 1 tuples)
   end
let format_power_of x
      \{ Color.Flow.num = num; Color.Flow.den = den; Color.Flow.power = pwr \} =
   match num,\ den,\ pwr with
     \_, 0, \_ \rightarrow invalid\_arg "format_power_of:\_zero\_denominator"
     0, -, - \rightarrow "+zero"
     1, 1, 0 \mid -1, -1, 0 \rightarrow "+one"
     -1, 1, 0 \mid 1, -1, 0 \rightarrow "-one"
     1,\ 1,\ 1\ \mid\ -1,\ -1,\ 1\ \rightarrow\ "+"\ \hat{\ } x
     -1, 1, 1 \mid 1, -1, 1 \rightarrow "-" ^ x
     1, \ 1, \ -1 \ | \ -1, \ -1, \ -1 \ 
ightarrow "+1/" \ \hat{} \ x
     -1, 1, -1 \mid 1, -1, -1 \rightarrow "-1/" \hat{x}
   |\  \  1,\ 1,\ p\ |\  \  -1,\ -1,\ p\ \to
        "+" \hat{} (if p > 0 then "" else "1/") \hat{} \hat{} \hat{} "**" \hat{} string\_of\_int (abs\ p)
   | -1, 1, p | 1, -1, p \rightarrow
        "-" \hat{} (if p > 0 then "" else "1/") \hat{} \hat{} \hat{} \hat{} **" \hat{} string\_of\_int (abs\ p)
   \mid n, 1, 0 \rightarrow
        (if n < 0 then "-" else "+") \hat{s}tring\_of\_int (abs n) \hat{"}.0_" \hat{s}!kind
   \mid n, d, 0 \rightarrow
        (if n \times d < 0 then "-" else "+") \hat{}
        string\_of\_int (abs n) ^ ".0_" ^ !kind ^ "/" ^
        string\_of\_int (abs d)
   \mid n, 1, 1 \rightarrow
         (if n < 0 then "-" else "+") \hat{s}tring\_of\_int (abs n) \hat{s}"*" \hat{s}
   \mid n, 1, -1 \rightarrow
        (if n < 0 then "-" else "+") \hat{s}tring\_of\_int (abs n) \hat{"}" \hat{x}
```

```
\mid n, d, 1 \rightarrow
        (if n \times d < 0 then "-" else "+") \hat{}
        string\_of\_int (abs n) ^ ".0_" ^ !kind ^ "/" ^
        string\_of\_int\ (abs\ d)\ ^{^{^{\ast}}}"*"\ ^{^{\ast}}x
  | n, d, -1 \rightarrow
        (if n \times d < 0 then "-" else "+") \hat{}
        string\_of\_int (abs n) ^ ".0_" ^ !kind ^ "/" ^
        string\_of\_int (abs d) ^ "/" ^ x
  \mid n, 1, p \rightarrow
        (if n < 0 then "-" else "+") \hat{string\_of\_int} (abs n) \hat{string\_of\_int}
        (if p > 0 then "*" else "/") \hat{x} "**" \hat{string\_of\_int} (abs p)
  \mid n, d, p \rightarrow
        (if n \times d < 0 then "-" else "+") \hat{}
        string\_of\_int (abs n) ^ ".0_" ^!kind ^ "/" ^
        string\_of\_int (abs d)
        (if p > 0 then "*" else "/") \hat{x} "**" \hat{string\_of\_int} (abs p)
let format\_powers\_of x = function
   | \ [] 
ightarrow "zero"
  | powers \rightarrow String.concat "" (List.map (format_power_of x) powers)
```



We can optimize the following slightly by reusing common color factor parameters.

```
let \ print\_color\_factor\_table \ table =
  let n\_cflow = Array.length \ table in
  let n\_cfactors = ref 0 in
  for c1 = 0 to pred n\_cflow do
    for c2 = 0 to pred n\_cflow do
      match table.(c1).(c2) with
      | \ | \ | \rightarrow \ ()
      \mid _ \rightarrow incr n_cfactors
    done
  done;
  print_integer_parameter "n_cfactors" !n_cfactors;
  printf "___0[<2>type(%s),_dimension(n_cfactors),_save%s_::"
    omega_color_factor_abbrev protected;
  printf "@_table_color_factors"; nl ();
  if \neg !amp\_triv then begin
    let i = ref 1 in
    if n_-cflow > 0 then begin
      for c1 = 0 to pred n\_cflow do
        for c2 = 0 to pred n\_cflow do
          match table.(c1).(c2) with
            [] \rightarrow ()
           | cf \rightarrow
               printf "___0[<2>real(kind=%s),_parameter,_private_::_color_factor_%06d_=_%s"
                 !kind !i (format_powers_of nc_parameter cf);
               !i\ omega\_color\_factor\_abbrev\ (succ\ c1)\ (succ\ c2)\ !i;
               incr i;
               nl();
        done
      done
    end;
  end
let print\_color\_tables amplitudes =
  let \ cflows = CF.color\_flows \ amplitudes
  and cfactors = CF.color_factors amplitudes in
  (* print_color_flows_table_old "c" cflows; nl (); *)
```

print_color_flows_table cflows; nl ();

```
(* print_ghost_flags_table_old "g" cflows; nl (); *)
              print_qhost_flags_table cflows; nl ();
              (* print_color_factor_table_old cfactors; nl (); *)
              print_color_factor_table cfactors; nl ()
         let option\_to\_logical = function
                  Some \_ \rightarrow "T"
                  None \rightarrow "F"
         let print\_flavor\_color\_table n\_flv n\_cflow table =
              if !amp\_triv then begin
                   printf
                         "\verb|u|| @ [<2> logical, \verb|u|| dimension(n_flv, \verb|u|n_cflow), \verb|u|save%s|| : : \verb|u|| @ | flv_col_is_allowed_u = \verb|u|T|| | flv_col_is_allowed_u = |
                   protected; nl();
                   end
              else begin
                   printf
                         "\sqcup_\sqcup @[<2> logical, \sqcup dimension(n_flv, \sqcup n_cflow), \sqcup save \%s \sqcup :: \sqcup @ \sqcup flv_col_is_allowed"
                   protected; nl();
                   if n_-flv > 0 then begin
                        for c = 0 to pred n\_cflow do
                             printf
                                   "\square0[<2>data\squareflv_col_is_allowed(:,%4d)\square/" (succ c);
                             printf "@_{\sqcup}%s" (option\_to\_logical \ table.(0).(c));
                             for f = 1 to pred n_{-}flv do
                                  printf ", Q_{\square}%s" (option\_to\_logical\ table.(f).(c))
                             done:
                             printf "@□/"; nl ()
                        done;
                   end;
              end
         let print\_amplitude\_table a =
               (*print\_flavor\_color\_table\_old "a" (num\_flavors\ a)\ (List.length\ (CF.color\_flows\ a))\ (CF.process\_table\ a);\ nl\ ();
*)
              print_flavor_color_table
                   (num_flavors a) (List.length (CF.color_flows a)) (CF.process_table a);
              nl();
              printf
                    "\sqcup\sqcup @[<2>complex(kind=%s), \sqcup dimension(n_flv, \sqcup n_cflow, \sqcup n_hel), \sqcup save_\sqcup: \sqcup amp" ! kind;
              nl();
              nl()
         let print_helicity_selection_table () =
              printf "□□□@[<2>logical, dimension(n_hel), saveu:::u";
              printf "hel_is_allowed_=\T"; nl();
              printf "_{\sqcup \sqcup} @[<2>real(kind=%s),_{\sqcup} dimension(n_hel),_{\sqcup} save_{\sqcup} ::_{\sqcup} "!kind;
              printf "hel_max_abs_=_0"; nl();
              printf "_{\sqcup \sqcup} @ [<2> real(kind=%s), _{\sqcup} save_{\sqcup} : :_{\sqcup}" !kind;
              printf "hel_sum_abs_=_0,_";
              printf "hel_threshold_=\lefta1E10_%s" !kind; nl ();
              printf "_{\sqcup \sqcup} @[<2> integer,_{\sqcup} save_{\sqcup}::_{\sqcup}";
              printf "hel_count_=_0, ";
              printf "hel_cutoff_=_100"; nl();
              printf "□□@[<2>integer□::□";
              printf "i"; nl ();
              printf "___@[<2>integer,_save,_dimension(n_hel)_::__";
              printf "hel_map_=(/(i,_i,_i,_n_hel)/)"; nl();
              printf ~" \_ \_ @[<2> integer, \_ save \_ : : \_ hel\_finite \_ = \_ n\_hel"; ~nl~();
              nl()
```

Optional MD5 sum function

```
let print\_md5sum\_functions = function
  \mid Some \ s \rightarrow
        printf "⊔⊔@[<5>"; if !fortran95 then printf "pure⊔";
        printf "function_md5sum_()"; nl();
        printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup} character(len=32)_{\sqcup}: _{\sqcup}md5sum"; nl ();
        printf "____md5sum__=_\\"%s\"" s; nl ();
       printf "□□end□function□md5sum"; nl();
        nl()
  | None \rightarrow ()
                                       Maintenance & Inquiry Functions
let print\_maintenance\_functions () =
  if !whizard then begin
     printf "_{\sqcup \sqcup} subroutine_{\sqcup} init_{\sqcup} (par,_{\sqcup} scheme) "; nl ();
     printf "_{\square\square\square\square}real(kind=%s), _dimension(*), _intent(in)_::_par" !kind; nl ();
     printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup} integer,_{\sqcup} intent(in)_{\sqcup} ::_{\sqcup} scheme"; nl();
     printf "____call_import_from_whizard_(par,_scheme)"; nl ();
     printf "_{\sqcup\sqcup} end_{\sqcup} subroutine_{\sqcup} init"; nl ();
     nl();
     printf "_usubroutine_final_()"; nl();
     printf "_{\sqcup \sqcup} end_{\sqcup} subroutine_{\sqcup} final"; nl ();
     nl();
     printf ~" \verb| u u broutine | u p date = alpha = s u (alpha = s) "; ~nl ~();
     printf "_{\sqcup\sqcup\sqcup\sqcup} real(kind=%s), \_intent(in)_{\sqcup}::_{\sqcup} alpha_s" !kind; nl ();
     printf "uuuucallumodelupdate_alpha_su(alpha_s)"; nl ();
     printf \verb|"_uuend_usubroutine_update_alpha_s"; nl ();
     nl()
  end
let print_inquiry_function_openmp() = begin
  printf "_{\sqcup \sqcup} pure_{\sqcup} function_{\sqcup} openmp_supported_{\sqcup}()_{\sqcup} result_{\sqcup}(status) "; nl ();
  printf "⊔⊔⊔⊔logical⊔::⊔status"; nl ();
  printf "_\underset status_\underset = \underset \sigma " (if !openmp then ".true." else ".false."); nl ();
  printf "⊔⊔end⊔function⊔openmp_supported"; nl ();
  nl()
let print_external_mass_case flv (fin, fout) =
  printf "_{\square\square\square\square} case_{\square} (\%3d) " (succ flv); nl ();
  List.iteri
     (fun i f \rightarrow
        (fin @ fout)
let \ print\_external\_masses \ amplitudes =
  printf "⊔∪@[<5>"; if !fortran95 then printf "pure⊔";
  printf "subroutine_external_masses_(m, _flv)"; nl ();
  printf "_{\sqcup \sqcup \sqcup \sqcup} real(kind=\%s), _{\sqcup} dimension(:), _{\sqcup} intent(out)_{\sqcup} : _{\sqcup} m" ! kind; nl ();
  printf "□□□□integer,□intent(in)□::□flv"; nl ();
  printf "____select_case_(flv)"; nl ();
  List.iteri print_external_mass_case (CF.flavors amplitudes);
  printf "\sqcup \sqcup \sqcup \sqcup \sqcup end\sqcupselect"; nl ();
  printf "\sqcup \sqcupend\sqcupsubroutine\sqcupexternal\_masses"; nl ();
  nl()
let print\_numeric\_inquiry\_functions (f, v) =
  printf "___0[<5>"; if !fortran95 then printf "pure_";
  printf "function_\%s_\()\(\text{result}\(\text{n}\)\(); nl();
```

```
printf " \sqcup \sqcup \sqcup \sqcup \sqcup integer \sqcup : : \sqcup n "; nl ();
  printf "_{\sqcup \sqcup} end_{\sqcup} function_{\sqcup} %s" f; nl ();
   nl()
let print\_inquiry\_functions name =
  printf "___@[<5>"; if !fortran95 then printf "pure_";
  printf "function_number_%s_()_result_(n) " name; nl ();
  printf "_{\sqcup \sqcup \sqcup \sqcup} integer_{\sqcup} :: _{\sqcup} n"; nl ();
  printf "_{\square\square\square\square} n_{\square} = size_{\square} (table_{s,\square} dim=2) " name; nl ();
  printf "⊔⊔end⊔function⊔number_%s" name; nl();
  nl();
  printf "⊔∪@[<5>"; if !fortran95 then printf "pure⊔";
  printf "subroutine_\%s_\(a)" name; nl ();
  printf " \sqcup \sqcup \sqcup \sqcup \sqcup integer, \sqcup dimension(:,:), \sqcup intent(out) \sqcup :: \sqcup a"; nl();
  printf "_{\square\square\square\square}a_{\square}=_{\square}table_{s}" name; nl();
  printf "⊔⊔end⊔subroutine⊔%s" name; nl ();
   nl()
let print\_color\_flows() =
  printf "⊔⊔@[<5>"; if !fortran95 then printf "pure⊔";
  printf "function_number_color_indices_()_result_(n)"; nl();
  printf "_{\sqcup \sqcup \sqcup \sqcup} integer_{\sqcup} :: _{\sqcup} n"; nl ();
  if !amp\_triv then begin
     printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup} n_{\sqcup} = _{\sqcup} n_{-} cindex"; nl ();
     end
  else begin
     printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup} n_{\sqcup} = \sqcup size_{\sqcup} (table\_color\_flows, \sqcup dim=1) "; nl ();
  end;
  printf "□□end□function□number_color_indices"; nl ();
  nl();
  printf "_\u0[<5>"; if !fortran95 then printf "pure\u0";
  printf "function_number_color_flows_()_result_(n)"; nl();
  printf " \sqcup \sqcup \sqcup \sqcup \sqcup integer \sqcup : : \sqcup n"; nl ();
  if !amp\_triv then begin
     end
  else begin
     printf "_{\square \square \square \square} n_{\square} = size_{\square} (table\_color\_flows,_{\square} dim=3) "; nl ();
  end;
  printf "_{\sqcup \sqcup} end_{\sqcup} function_{\sqcup} number_color_flows"; nl ();
  nl();
  printf "⊔⊔@[<5>"; if !fortran95 then printf "pure⊔";
  printf "subroutine_color_flows_(a,_g)"; nl ();
  printf " \sqcup \sqcup \sqcup \sqcup \sqcup integer, \sqcup dimension(:,:,:), \sqcup intent(out) \sqcup :: \sqcup a "; nl();
  printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup} logical, _{\sqcup} dimension(:,:), _{\sqcup} intent(out)_{\sqcup}::_{\sqcup} g"; nl ();
  printf "____a_=_table_color_flows"; nl ();
  printf "____g_=_table_ghost_flags"; nl ();
   printf "\sqcup \sqcupend\sqcupsubroutine\sqcupcolor\_flows"; nl ();
   nl()
let print\_color\_factors() =
  printf "⊔∪@[<5>"; if !fortran95 then printf "pure⊔";
  printf "function_number_color_factors_()_result_(n)"; nl ();
  printf " \sqcup \sqcup \sqcup \sqcup \sqcup integer \sqcup : : \sqcup n "; nl ();
  printf "____ni=_size_(table_color_factors)"; nl();
  printf "⊔⊔end⊔function⊔number_color_factors"; nl ();
  nl();
  printf "\sqcup \sqcup @[<5>"; if !fortran95 then printf "pure\sqcup";
  printf "subroutine color factors (cf)"; nl ();
  printf "\u\u\u\type(%s),\u\dimension(:),\u\intent(out)\u\::\u\cf"
      omega_color_factor_abbrev; nl ();
  printf "____cf_=_table_color_factors"; nl ();
```

```
printf "\sqcup \sqcupend\sqcupsubroutine\sqcupcolor\botfactors"; nl ();
     nl();
     printf "⊔⊔@[<5>"; if !fortran95 ∧ pure_unless_openmp then printf "pure⊔";
     printf "function color sum (flv, hel) result (amp2)"; nl ();
     printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup} integer,_{\sqcup} intent(in)_{\sqcup} ::_{\sqcup} flv,_{\sqcup} hel"; nl();
     printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup} real(kind=%s)_{\sqcup}::_{\sqcup} amp2" !kind; nl ();
     printf "_\uu_\upprimamp2\upprimamp2\upprimamp2\upprimamp2\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\upprimamp\uppri
     printf " \sqcup \sqcup end \sqcup function \sqcup color \_ sum"; nl ();
      nl()
let print\_dispatch\_functions () =
     printf "⊔⊔@[<5>";
     printf "subroutine_new_event_(p)"; nl();
     printf "\u|\u|\u|real(kind=\%s),\u|dimension(0:3,*),\u|intent(in)\u|:\u|p\u|!kind; nl();
     printf "____logical_::_mask_dirty"; nl();
     printf "⊔⊔⊔⊔integer⊔::⊔hel"; nl();
     printf "LULULCalculate_amplitudes_(amp,_p,_hel_is_allowed)"; nl ();
     printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup} if_{\sqcup}((hel\_threshold_{\sqcup}.gt._{\sqcup}0)_{\sqcup}.and._{\sqcup}(hel\_count_{\sqcup}.le._{\sqcup}hel\_cutoff))_{\sqcup}then"; nl ();
     printf ~ \verb|'| \verb|| \verb|| \verb|| \verb|| call | \verb|| @ [<3 > omega\_update\_helicity\_selection@| (hel\_count,@|| amp,@||"; | left | count, end | cou
     printf "hel_max_abs, Q_hel_sum_abs, Q_hel_is_allowed, Q_hel_threshold, Q_hel_cutoff, Q_hmask_dirty) "; nl(n)
     printf "____if_ (mask_dirty)_then"; nl ();
     printf "____hel_finite_=_0"; nl ();
     printf "____do_hel_=_1,_n_hel"; nl ();
     printf "_____if__(hel_is_allowed(hel))_then"; nl();
     printf "____hel_finite__=_hel_finite__+_1"; nl();
     printf "____hel_"; nl ();
     printf "____end_if"; nl();
     printf "_____end_do"; nl ();
      printf " \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup = nd \sqcup if "; nl ();
     printf "_{\sqcup\sqcup\sqcup\sqcup} end_{\sqcup}if"; nl ();
     printf "\sqcup \sqcupend\sqcupsubroutine\sqcupnew\_event"; nl ();
     nl();
     printf "⊔⊔@[<5>";
     printf "subroutine_reset_helicity_selection_(threshold, cutoff)"; nl ();
     printf "_{\square \square \square \square}real(kind=%s),_{\square}intent(in)_{\square}::_{\square}threshold" !kind; nl ();
     printf "____integer,_intent(in)_::_cutoff"; nl ();
     printf "□□□□□integer□::□i"; nl ();
     printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup}hel_is_allowed_{\sqcup}=_{\sqcup}T"; nl();
     printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup}hel_{\max\_abs_{\sqcup}=\sqcup}0"; nl ();
     printf " \sqcup \sqcup \sqcup \sqcup \sqcup hel = sum_abs = \sqcup 0"; nl ();
     printf "_{\square \square \square \square} hel_count_{\square} = _{\square} 0"; nl ();
     printf "___hel_threshold_=_threshold"; nl ();
     printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup}hel\_cutoff_{\sqcup}=_{\sqcup}cutoff"; nl ();
     printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup}hel_map_{\sqcup}=_{\sqcup}(/(i,_{\sqcup}i_{\sqcup}=_{\sqcup}1,_{\sqcup}n_hel)/)"; nl();
     printf "_{\sqcup \sqcup \sqcup \sqcup} hel_finite_{\sqcup = \sqcup} n_hel"; nl ();
     printf "□□end□subroutine□reset_helicity_selection"; nl ();
     nl();
     printf "⊔⊔@[<5>"; if !fortran95 then printf "pure⊔";
     printf "function_is_allowed_(flv,_hel,_col)_result_(yorn)"; nl ();
     printf " \sqcup \sqcup \sqcup \sqcup \logical \sqcup : \sqcup yorn "; nl ();
     printf "____integer,_intent(in)_::_flv,_hel,_col"; nl ();
     if !amp\_triv then begin
               printf "□□□□!□print□*,□'inside□is_allowed'"; nl ();
     end;
     if \neg !amp\_triv then begin
               printf "_{\sqcup \sqcup \sqcup \sqcup} yorn_{\sqcup} = _{\sqcup} hel_{is\_allowed(hel)_{\sqcup}}.and._{\sqcup}";
               printf "flv_col_is_allowed(flv,col)"; nl ();
              end
     else begin
               printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup} yorn_{\sqcup} =_{\sqcup} .false."; nl ();
     end:
```

```
printf "\sqcup \sqcupend\sqcupfunction\sqcupis_allowed"; nl ();
  nl();
  printf "⊔∪@[<5>"; if !fortran95 then printf "pure⊔";
  printf "function_get_amplitude_(flv,_hel,_col)_result_(amp_result)"; nl ();
  printf " \subseteq complex(kind=%s) \subseteq : \subseteq amp\_result" ! kind; nl();
  printf "uuuuinteger,uintent(in)u::uflv,uhel,ucol"; nl ();
  printf "_{\sqcup\sqcup\sqcup\sqcup} amp\_result_{\sqcup} =_{\sqcup} amp(flv,_{\sqcup}col,_{\sqcup}hel) "; nl();
  printf "_{\sqcup\sqcup} end_{\sqcup} function_{\sqcup} get_amplitude"; nl ();
   nl()
                                                    Main Function
let format_power_of_nc
      \{ Color.Flow.num = num; Color.Flow.den = den; Color.Flow.power = pwr \} =
   match num, den, pwr with
     -, 0, - \rightarrow invalid\_arg "format_power_of_nc:\Boxzero\Boxdenominator"
     0, -, - \rightarrow ""
     1, 1, 0 \mid -1, -1, 0 \rightarrow "+ \bot 1"
     -1, 1, 0 \mid 1, -1, 0 \rightarrow "-_{\sqcup}1"
     1, 1, 1 \mid -1, -1, 1 \rightarrow "+ \sqcup N"
     -1, 1, 1 \mid 1, -1, 1 \rightarrow "- \square N"
    1, 1, -1 \mid -1, -1, -1 \rightarrow "+ 1/N"
     -1, 1, -1 \mid 1, -1, -1 \rightarrow "-_{\sqcup}1/N"
    1, 1, p \mid -1, -1, p \rightarrow
        "+\Box" \hat{} (if p>0 then "" else "1/") \hat{} "N\hat{}" \hat{} string\_of\_int (abs\ p)
     -1, 1, p \mid 1, -1, p \rightarrow
        "-\Box" ^ (if p > 0 then "" else "1/") ^ "N^" ^ string\_of\_int (abs p)
   \mid n, 1, 0 \rightarrow
        (if n < 0 then "-\square" else "+\square") \hat{} string\_of\_int (abs n)
   \mid n, d, 0 \rightarrow
        (if n \times d < 0 then "-\square" else "+\square") ^
        string\_of\_int (abs n) ^ "/" ^ string\_of\_int (abs d)
   \mid n, 1, 1 \rightarrow
        (if n < 0 then "-\square" else "+\square") \hat{} string\_of\_int (abs \ n) \hat{} "N"
   n, 1, -1 \rightarrow
        (if n < 0 then "-\square" else "+\square") \hat{} string\_of\_int (abs\ n) \hat{} "/N"
  \mid n, d, 1 \rightarrow
        (if n \times d < 0 then "-\square" else "+\square")
        string\_of\_int (abs n) ^ "/" ^ string\_of\_int (abs d) ^ "N"
  |~n,~d,~-1~\rightarrow
        (if n \times d < 0 then "-\Box" else "+\Box")
        string\_of\_int (abs n) ^ "/" ^ string\_of\_int (abs d) ^ "/N"
        (if n < 0 then "-_{\square}" else "+_{\square}") \hat{} string\_of\_int (abs n) \hat{}
        (if p > 0 then "*" else "/") ^ "N^" ^ string\_of\_int (abs p)
  \mid n, d, p \rightarrow
        (if n \times d < 0 then "-\Box" else "+\Box") ^ string\_of\_int (abs n) ^ "/" ^
        string\_of\_int (abs d) \hat{} (if p > 0 then "*" else "/") \hat{} "N^" \hat{} string\_of\_int (abs p)
let format\_powers\_of\_nc = function
   \mid \; [\;] \; 
ightarrow \; "0"
   powers \rightarrow String.concat "\square" (List.map format_power_of_nc powers)
let print\_description\ cmdline\ amplitudes\ () =
  printf
     "!_File_generated_automatically_by_0'Mega_%s_%s_%s"
      Config.version Config.status Config.date; nl ();
   List.iter (fun s \rightarrow printf "!\_%s" s; nl ()) (M.caveats ());
  printf "!"; nl ();
  printf "!\square\square%s" cmdline; nl();
  printf "!"; nl ();
```

```
printf "!uwithualluscatteringuamplitudesuforutheuprocess(es)"; nl ();
printf "!"; nl ();
printf "!\sqcup\sqcup\sqcupflavor\sqcupcombinations:"; nl ();
printf "!"; nl ();
Tho List.iteri
     (fun i \ process \rightarrow
          printf "!_____%3d:__%s" i (process\_sans\_color\_to\_string process); nl ())
     1 (CF.flavors amplitudes);
printf "!"; nl ();
printf "! LLL color flows: "; nl ();
if \neg !amp\_triv then begin
     printf "!"; nl ();
      Tho List.iteri
          (fun i \ cflow \rightarrow
               1 (CF.color_flows amplitudes);
     printf "!"; nl ();
     printf \; \verb"!_{$\sqcup\sqcup\sqcup\sqcup\sqcup} \verb"NB:$\sqcup i.g.$\sqcup not$\sqcup all$\sqcup color$\sqcup flows$\sqcup contribute$\sqcup to$\sqcup all$\sqcup flavor"; $nl()$; $L(s) = L(s)$ and $L(s) = L(s)$. $L(s) = L(s)$ and $L(s) = L(s)$ and $L(s) = L(s)$. $L(s) = L(s)$ and $L(s) = L(s)$ and $L(s) = L(s)$. $L(s) = L(s)$ and $L(s) = L(s)$ and $L(s) = L(s)$. $L(s) = L(s)$ and $L(s) = L(s)$ and $L(s) = L(s)$. $L(s) = L(s)$ and $L(s) = L(s)$ and $L(s) = L(s)$. $L(s) = L(s)$ and $L(s) = L(s)
     printf "!____combinations.___Consult_the_array_FLV_COL_IS_ALLOWED"; nl();
     printf "!____below_for_the_allowed_combinations."; nl();
end:
printf "!"; nl ();
printf "!□□□Color□Factors:"; nl ();
printf "!"; nl ();
if \neg !amp\_triv then begin
    let cfactors = CF.color\_factors \ amplitudes in
    for c1 = 0 to pred (Array.length cfactors) do
          for c2 = 0 to c1 do
               match cfactors.(c1).(c2) with
               | [] \rightarrow ()
               | cfactor \rightarrow
                       printf "!____(%3d,%3d):__%s"
                            (succ\ c1)\ (succ\ c2)\ (format\_powers\_of\_nc\ cfactor);\ nl\ ()
         done
    done:
end;
if \neg !amp\_triv then begin
       printf "!"; nl ();
       printf "!⊔⊔⊔vanishing⊔or⊔redundant⊔flavor⊔combinations:"; nl ();
       printf "!"; nl ();
       List.iter (fun process \rightarrow
            printf "!_____%s" (process_sans_color_to_string process); nl ())
            (CF.vanishing_flavors amplitudes);
       printf "!"; nl ();
end:
begin
     match CF. constraints amplitudes with
        None \rightarrow ()
        Some \ s \rightarrow
               printf
                    "!_LLLdiagram_selection_(MIGHT_BREAK_GAUGE_INVARIANCE!!!):"; nl ();
               printf "!"; nl ();
               printf "!"; nl ()
end;
printf "!"; nl ()
```

Printing Modules

 $type \ accessibility =$

```
Public
     Private
    Protected (* Fortran 2003 *)
let accessibility\_to\_string = function
     Public 
ightarrow "public"
     Private \rightarrow "private"
    Protected \rightarrow "protected"
type \ used\_symbol =
     As_{-}Is of string
    Aliased of string \times string
let print\_used\_symbol = function
    As\_Is \ name \rightarrow printf "%s" name
    Aliased\ (orig,\ alias) \rightarrow printf "%s_{\sqcup} = >_{\sqcup}%s" alias\ orig
type used\_module =
     Full of string
     Full\_Aliased of string \times (string \times string) list
     Subset of string \times used\_symbol list
let print\_used\_module = function
     Full name
     Full\_Aliased\ (name,\ [\ ])
     Subset (name, []) \rightarrow
        printf "\sqcup \sqcupuse\sqcup%s" name;
        nl()
   \mid Full_Aliased (name, aliases) \rightarrow
        printf "\sqcup \sqcup @[<5>use \sqcup %s" name;
           (fun (orig, alias) \rightarrow printf ", \_%s_\=>_\%s" alias orig)
           aliases;
        nl()
   Subset\ (name,\ used\_symbol\ ::\ used\_symbols)\ 
ightarrow
        printf ~" \_ \_ @[<5> \\ use \_ \%s, \_ only: \_ " ~name;
        print\_used\_symbol\ used\_symbol;
        List.iter (fun s \rightarrow printf ",\square"; print\_used\_symbol s) used\_symbols;
        nl()
\mathsf{type}\; fortran\_module \;\; = \;\;
     \{ module\_name : string; 
        default\_accessibility: accessibility;
        used_modules : used_module list;
        public_symbols : string list;
        print\_declarations : (unit \rightarrow unit) list;
        print\_implementations : (unit \rightarrow unit) list 
let print_public = function
   | name1 :: names \rightarrow
        printf "_{\sqcup \sqcup} @[<2>public_{\sqcup}::_{\sqcup}%s" name1;
        List.iter (fun n \rightarrow printf ", Q_{\square}%s" n) names; nl ()
  | [] \rightarrow ()
let print_module m =
  printf "module_%s" m.module_name; nl ();
   List.iter print_used_module m.used_modules;
  printf "\sqcup \sqcup implicit \sqcup none"; nl();
  printf "_{\sqcup\sqcup}\%s" (accessibility\_to\_string m.default\_accessibility); nl ();
  print_public m.public_symbols; nl ();
  begin match m.print\_declarations with
   | [] \rightarrow ()
   | print\_declarations \rightarrow
        List.iter (fun f \rightarrow f ()) print\_declarations; nl ()
  end;
```

```
begin match m.print\_implementations with
   | [] \rightarrow ()
  | print\_implementations \rightarrow
       printf "contains"; nl(); nl();
       List.iter (fun f \rightarrow f ()) print_implementations; nl ();
  end:
  printf "end_module_%s" m.module_name; nl ()
let print\_modules modules =
  List.iter print_module modules;
  print_flush ()
let module_to_file line_length oc prelude m =
  output\_string\ oc\ (m.module\_name\ ^ "\n");
  let filename = m.module\_name ^ ".f90" in
  \mathsf{let}\ channel\ =\ open\_out\ filename\ \mathsf{in}
  prelude();
  print\_modules [m];
  close\_out\ channel
let modules\_to\_file line\_length oc prelude = function
  | [] \rightarrow ()
  m :: mlist \rightarrow
       module_to_file line_length oc prelude m;
       List.iter\ (module\_to\_file\ line\_length\ oc\ (fun\ ()\ \to\ ()))\ mlist
                                        Chopping Up Amplitudes
let num\_fusions\_brakets \ size \ amplitudes =
  let num_fusions =
     max 1 size in
  let count\_brakets =
     List.fold\_left
       (fun \ sum \ process \rightarrow sum + List.length \ (F.brakets \ process))
       0 (CF.processes amplitudes)
  and count\_processes =
     List.length (CF.processes amplitudes) in
  if count\_brakets > 0 then
    let num\_brakets =
       max \ 1 \ ((num\_fusions \times count\_processes) \ / \ count\_brakets) \ in
     (num\_fusions, num\_brakets)
  else
     (num\_fusions, 1)
let chop_amplitudes size amplitudes =
  let num\_fusions, num\_brakets = num\_fusions\_brakets size amplitudes in
  (ThoList.enumerate 1 (ThoList.chopn num_fusions (CF.fusions amplitudes)),
    ThoList.enumerate \ 1 \ (ThoList.chopn \ num\_brakets \ (CF.processes \ amplitudes)))
let print\_compute\_fusions1 \ dictionary \ (n, \ fusions) =
  if \neg !amp\_triv then begin
    if !openmp then begin
       printf "_{\sqcup\sqcup} subroutine_{\sqcup} compute_fusions_%04d_{\sqcup}(%s)" n openmp_tld; nl ();
       printf "\_\_@[<5>type(%s),\_intent(inout)_::\_%s" openmp\_tld\_type openmp\_tld; nl ();
     end else begin
       printf "_{\sqcup \sqcup} @[<5> subroutine_{\sqcup} compute_fusions_%04d_{\sqcup}() " n; nl ();
     end;
     print_fusions dictionary fusions;
     printf "_{\sqcup\sqcup} end_{\sqcup} subroutine_{\sqcup} compute_fusions_%04d" n; nl ();
  end
and print\_compute\_brakets1 dictionary (n, processes) =
```

```
if \neg !amp\_triv then begin
         if !openmp then begin
            printf "___subroutine_compute_brakets_%04d_(%s)" n openmp_tld; nl ();
                    "\sqcup\sqcup @[<5>type(%s), \sqcup intent(inout)_{\sqcup}:: \sqcup %s" openmp\_tld\_type openmp\_tld; nl();
            printf "_{\sqcup \sqcup} @[<5> subroutine_{\sqcup} compute_brakets_%04d_{\sqcup}() " n; nl ();
         end;
         List.iter (print_brakets dictionary) processes;
         printf "__end_subroutine_compute_brakets_%04d" n; nl();
       end
                                                   Common Stuff
    let omega\_public\_symbols =
       ["number_particles_in"; "number_particles_out";
        "number_color_indices";
        "reset_helicity_selection"; "new_event";
        "is_allowed"; "get_amplitude"; "color_sum";
        "external_masses"; "openmp_supported"] @
       Tho List. flat map
          (\mathsf{fun}\ n\ \to\ [\texttt{"number\_"}\ \hat{}\ n;\ n])
         ["spin_states"; "flavor_states"; "color_flows"; "color_factors"]
    let whizard\_public\_symbols \ md5sum =
       ["init"; "final"; "update_alpha_s"] @
       (match md5sum with Some \, \_ \, \rightarrow \, ["md5sum"] \, | \, None \, \rightarrow \, [])
    let used\_modules() =
       [Full "kinds";
        Full Fermions.use_module;
        Full_Aliased ("omega_color", ["omega_color_factor", omega_color_factor_abbrev])] @
       List.map
          (\mathsf{fun}\ m\ \to\ \mathit{Full}\ m)
          (match !parameter_module with
            "" \rightarrow !use\_modules
           pm \rightarrow pm :: !use\_modules)
    let public\_symbols () =
       if !whizard then
          omega_public_symbols @ (whizard_public_symbols !md5sum)
       else
          omega\_public\_symbols
    {\tt let} \ print\_constants \ amplitudes \ =
       printf "___!_DON'T_EVEN_THINK_of_removing_the_following!"; nl ();
       printf "uu!uIfutheucompilerucomplainsuaboutuundeclared"; nl ();
       printf "_{\sqcup\sqcup}!_{\sqcup}or_{\sqcup}undefined_{\sqcup}variables,_{\sqcup}you_{\sqcup}are_{\sqcup}compiling"; nl ();
       printf "_{\sqcup\sqcup}!_{\sqcup}against_{\sqcup}an_{\sqcup}incompatible_{\sqcup}omega95_{\sqcup}module!"; nl();
       printf "□□0[<2>integer, □dimension(%d), □parameter, □private□::□"
         (List.length\ require\_library);
       printf "require_=0_(/_0[";
       print_list require_library;
       printf "□/)"; nl (); nl ();
Using these parameters makes sense for documentation, but in practice, there is no need to ever change them.
          (function name, value \rightarrow print_integer_parameter name (value amplitudes))
          [ ("n_prt", num_particles);
            ("n_in", num_particles_in);
            ("n_out", num_particles_out);
            ("n_cflow", num_color_flows); (* Number of different color amplitudes. *)
            ("n_cindex", num_color_indices); (* Maximum rank of color tensors. *)
```

```
("n_flv", num_flavors); (* Number of different flavor amplitudes. *)
              ("n_hel", num_helicities) (* Number of different helicty amplitudes. *) ];
        nl();
Abbreviations.
        printf "\sqcup \sqcup ! \sqcup NB : \sqcup you \sqcup MUST \sqcup NOT \sqcup change \sqcup the \sqcup value \sqcup of \sqcup %s \sqcup here!!! " <math>nc\_parameter;
        nl();
        printf "_{\sqcup\sqcup}!_{\;\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup}It_{\sqcup}is_{\sqcup}defined_{\sqcup}here_{\sqcup}for_{\sqcup}convenience_{\sqcup}only_{\sqcup}and_{\sqcup}must_{\sqcup}be"; nl ();
        printf "_{\cup\cup}!_{\cup\cup\cup\cup\cup\cup} compatible_{\cup} with_{\cup} hardcoded_{\cup} values_{\cup} in_{\cup} the_{\cup} amplitude!"; nl ();
        print\_real\_parameter\ nc\_parameter\ (CM.nc\ ());\ (*\ N_C\ *)
        List.iter
           (function name, value \rightarrow print\_logical\_parameter name value)
           [ ("F", false); ("T", true) ]; nl ();
        print_spin_tables amplitudes;
        print_flavor_tables amplitudes;
        print_color_tables amplitudes;
        print_amplitude_table amplitudes;
        print_helicity_selection_table ()
     let print\_interface amplitudes =
        print\_md5sum\_functions \ !md5sum;
        print_maintenance_functions ();
        List.iter print_numeric_inquiry_functions
           [("number_particles_in", "n_in");
            ("number_particles_out", "n_out")];
        List.iter\ print\_inquiry\_functions
           ["spin_states"; "flavor_states"];
        print_external_masses amplitudes;
        print_inquiry_function_openmp ();
        print\_color\_flows ();
        print_color_factors ();
        print_dispatch_functions ();
        nl();
        (* Is this really necessary? *)
        Format_Fortran.switch_line_continuation false;
        if !km\_write \lor !km\_pure then (Targets\_Kmatrix.Fortran.print !km\_pure);
        if !km_2\_write \lor !km_2\_pure then (Targets\_Kmatrix\_2.Fortran.print !km_2\_pure);
        Format_Fortran.switch_line_continuation true;
        nl()
     let\ print\_calculate\_amplitudes\ declarations\ computations\ amplitudes\ =
        printf "_{\sqcup\sqcup}@[<5>subroutine_calculate_amplitudes_{\sqcup}(amp,_{\sqcup}k,_{\sqcup}mask)"; nl ();
        printf "_{\sqcup\sqcup\sqcup\sqcup} complex(kind=%s), _{\sqcup} dimension(:,:,:), _{\sqcup} intent(out)_{\sqcup} ::_{\sqcup} amp" ! kind; nl ();
        printf "_{\square \square \square \square} real(kind=%s),_{\square} dimension(0:3,*),_{\square} intent(in)_{\square}:_{\square} k" !kind; nl ();
        printf "____logical,_dimension(:),_intent(in)_::_mask"; nl ();
        printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup} integer,_{\sqcup} dimension(n_prt)_{\sqcup}::_{\sqcup}s"; nl ();
        printf "_{\sqcup \sqcup \sqcup \sqcup} integer_{\sqcup} :: _{\sqcup} h, _{\sqcup} hi"; nl ();
        declarations();
        if \neg !amp\_triv then begin
           begin match CF.processes amplitudes with
           \mid p :: \_ \rightarrow print\_external\_momenta p
           | - \rightarrow ()
           end;
           ignore (List.fold_left print_momenta PSet.empty (CF.processes amplitudes));
        printf "_{\square \square \square \square} amp_{\square} = _{\square} O"; nl ();
        if \neg !amp\_triv then begin
           if num\_helicities \ amplitudes > 0 then begin
              printf "_{\sqcup\sqcup\sqcup\sqcup} if_{\sqcup} (hel\_finite_{\sqcup} = = \sqcup 0)_{\sqcup} return"; nl ();
              if !openmp then begin
                 printf "!\$0MP_PARALLEL_DO_DEFAULT(SHARED)_PRIVATE(s,_h,_%s)_SCHEDULE(STATIC)" openmp\_tld;\ nl\ ();
              end;
```

```
printf "_{\sqcup \sqcup \sqcup \sqcup} do_{\sqcup} hi_{\sqcup} = _{\sqcup} 1, _{\sqcup} hel_finite"; nl ();
        printf "_{\square \square \square \square \square \square \square} h_{\square} = _{\square} hel_{map}(hi) "; nl();
        printf "____s_=_table_spin_states(:,h)"; nl ();
        ignore (List.fold_left print_externals WFSet.empty (CF.processes amplitudes));
        computations ();
        List.iter print_fudge_factor (CF.processes amplitudes);
     (* This sorting should slightly improve cache locality. *)
        let triple\_snd = fun(\_, x, \_) \rightarrow x
        in let triple\_fst = fun(x, \_, \_) \rightarrow x
            in let rec builder1 flvi flowi flows = match flows with
              (Some\ a)::tl\rightarrow (flvi,\ flowi,\ flavors\_symbol\ (flavors\ a))::(builder1\ flvi\ (flowi\ +\ 1)\ tl)
              None :: tl \rightarrow builder1 \ flvi \ (flowi + 1) \ tl
            | [] \rightarrow []
                in let rec builder2 flvi flvs = match flvs with
                | flv :: tl \rightarrow (builder1 \ flvi \ 1 \ flv) @ (builder2 \ (flvi + 1) \ tl)
                | [] \rightarrow []
                    in let unsorted = builder2 1 (List.map Array.to_list (Array.to_list (CF.process_table amplitudes)))
                        in let sorted = List.sort (fun a \ b \rightarrow
                           if (triple\_snd\ a \not\equiv triple\_snd\ b) then triple\_snd\ a - triple\_snd\ b else (triple\_fst\ a - triple\_fst\ b))
                               unsorted
                            in List.iter (fun (flvi, flowi, flv) \rightarrow
                               (printf "LULULU amp(%d, %d, h) = %s" flvi flowi flv; nl (); )) sorted;
                            printf "ullulendudo"; nl ();
                            if !openmp then begin
                               printf "!$OMP_END_PARALLEL_DO"; nl ();
                            end:
     end:
  end;
  printf "___end__subroutine__calculate_amplitudes"; nl ()
let print_compute_chops chopped_fusions chopped_brakets () =
   List.iter
     (\text{fun } (i, \_) \rightarrow printf " \_ \_ \_ \_ \text{call} \_ \text{compute\_fusions\_\%04d} \_ (\%s) " i
         (if !openmp then openmp_tld else ""); nl ())
     chopped_fusions;
   List.iter
     (\mathsf{fun}\ (i,\ \_)\ \to\ \mathit{printf}\ \verb"$$^{$\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup}\mathsf{call}_{\sqcup}\mathsf{compute\_brakets\_\%04d}_{\sqcup}(\$s)$" $i$
         (if !openmp then openmp_tld else ""); nl ())
     chopped\_brakets
                                                     UFO Fusions
module VSet =
  Set.Make (struct type t = F.constant Coupling.t let compare = compare end)
let ufo\_fusions\_used amplitudes =
  let couplings =
     List.fold\_left
        (fun acc p \rightarrow
          let fusions = ThoList.flatmap F.rhs (F.fusions p)
           and brakets = ThoList.flatmap F.ket (F.brakets p) in
          let couplings =
              VSet.of_list (List.map F.coupling (fusions @ brakets)) in
           VSet.union acc couplings)
        VSet.empty (CF.processes amplitudes) in
   VSet.fold
     (fun v \ acc \rightarrow
        match v with
        | Coupling. Vn (Coupling. UFO (-, v, -, -, -), -, -) \rightarrow
            Sets.String.add\ v\ acc
        | \quad acc \rangle
```

couplings Sets.String.empty

Single Function

```
let \ amplitudes\_to\_channel\_single\_function \ cmdline \ oc \ amplitudes =
  let print_declarations() =
    print_constants amplitudes
  and print\_implementations() =
    print_interface amplitudes;
    print\_calculate\_amplitudes
       (fun () \rightarrow print\_variable\_declarations amplitudes)
       (fun () \rightarrow
         print_fusions (CF.dictionary amplitudes) (CF.fusions amplitudes);
            (print_brakets (CF.dictionary amplitudes))
            (CF.processes amplitudes))
       amplitudes in
  let fortran\_module =
     \{ module\_name = !module\_name; 
       used\_modules = used\_modules ();
       default\_accessibility = Private;
       public\_symbols = public\_symbols ();
       print\_declarations = [print\_declarations];
       print\_implementations = [print\_implementations] } in
  Format_Fortran.set_formatter_out_channel ~width:!line_length oc;
  print_description cmdline amplitudes ();
  print_modules [fortran_module]
                                            Single Module
let amplitudes_to_channel_single_module cmdline oc size amplitudes =
  let print_declarations() =
    print_constants amplitudes;
    print\_variable\_declarations amplitudes
  and print\_implementations() =
    print_interface amplitudes in
  let chopped_fusions, chopped_brakets =
     chop\_amplitudes\ size\ amplitudes\ in
  let \ dictionary = CF. dictionary \ amplitudes \ in
  let print\_compute\_amplitudes() =
    print\_calculate\_amplitudes
       (fun () \rightarrow ())
       (print_compute_chops chopped_fusions chopped_brakets)
       amplitudes
  and print\_compute\_fusions () =
     List.iter (print_compute_fusions1 dictionary) chopped_fusions
  and print\_compute\_brakets () =
     List.iter (print_compute_brakets1 dictionary) chopped_brakets in
  {\tt let} \; fortran\_module \; = \;
     \{ module\_name = !module\_name; 
       used\_modules = used\_modules ();
       default\_accessibility = Private;
       public\_symbols = public\_symbols ();
       print\_declarations = [print\_declarations];
```

```
print\_implementations = [print\_implementations;]
                                   print_compute_amplitudes;
                                   print_compute_fusions;
                                   print_compute_brakets] } in
  Format_Fortran.set_formatter_out_channel ~width :!line_length oc;
  print_description cmdline amplitudes ();
  print_modules [fortran_module]
                                          Multiple Modules
let modules_of_amplitudes _ _ size amplitudes =
  let name = !module\_name in
  let print_declarations() =
    print\_constants\ amplitudes
  and print\_variables() =
    print_variable_declarations amplitudes in
  let constants\_module =
    { module_name = name ^ "_constants";
       used\_modules = used\_modules ();
       default\_accessibility = Public;
      public\_symbols = [];
      print\_declarations = [print\_declarations];
      print\_implementations = []  in
  let \ variables\_module =
    \{ \ module\_name = name ^ "\_variables";
       used\_modules = used\_modules ();
       default\_accessibility = Public;
      public\_symbols = [];
      print\_declarations = [print\_variables];
      print\_implementations = []  in
  let \ dictionary = CF. dictionary \ amplitudes \ in
  let print\_compute\_fusions\ (n, fusions)\ () =
    if \neg !amp\_triv then begin
      if !openmp then begin
         printf "□□subroutine□compute_fusions_%04d□(%s)" n openmp_tld; nl ();
         printf "uu@[<5>type(%s),uintent(inout)u::u%s" openmp_tld_type openmp_tld; nl ();
      end else begin
         printf "_{\sqcup \sqcup} @ [<5> subroutine_{\sqcup} compute_fusions_%04d_{\sqcup}() " n; nl ();
      end:
      print_fusions dictionary fusions;
      printf "\sqcup \sqcupend\sqcupsubroutine\sqcupcompute\_fusions\_%04d" n; nl();
  let print\_compute\_brakets (n, processes) () =
    if \neg !amp\_triv then begin
      if !openmp then begin
         printf "□□subroutine□compute_brakets_%04d□(%s)" n openmp_tld; nl ();
         printf " = (<5>type(%s), intent(inout) : : '%s" openmp_tld_type openmp_tld; nl ();
      end else begin
         printf "_{\sqcup \sqcup} @[<5> subroutine_{\sqcup} compute_brakets_%04d_{\sqcup}() " n; nl ();
      end;
      List.iter (print_brakets dictionary) processes;
      printf "⊔⊔end⊔subroutine⊔compute_brakets_%04d" n; nl ();
    end in
  let fusions\_module (n, \_as fusions) =
    let tag = Printf.sprintf "_fusions_%04d" n in
    \{ module\_name = name ^ tag;
```

```
used\_modules = (used\_modules () @
                         [Full constants_module.module_name;
                          Full variables_module.module_name]);
       default\_accessibility = Private;
       public\_symbols = ["compute" ^ tag];
       print\_declarations = [];
       print\_implementations = [print\_compute\_fusions fusions] } in
  let brakets\_module\ (n, \_ as\ processes) =
    let tag = Printf.sprintf "_brakets_%04d" n in
     \{ module\_name = name ^tag; 
       used\_modules = (used\_modules () @
                         [Full constants_module.module_name;
                          Full variables_module.module_name]);
       default\_accessibility = Private;
       public\_symbols = ["compute" ^ tag];
       print\_declarations = [];
       print\_implementations = [print\_compute\_brakets\ processes] \} in
  let chopped_fusions, chopped_brakets =
     chop_amplitudes size amplitudes in
  let fusions\_modules =
     List.map fusions_module chopped_fusions in
  let brakets\_modules =
     List.map brakets_module chopped_brakets in
  let print\_implementations() =
    print_interface amplitudes;
    print\_calculate\_amplitudes
       (fun () \rightarrow ())
       (print_compute_chops chopped_fusions chopped_brakets)
       amplitudes in
  let public\_module =
     \{ module\_name = name; 
        used\_modules = (used\_modules () @
                           [Full constants_module.module_name;
                            Full\ variables\_module.module\_name\ ]\ @
                          List.map
                             (fun \ m \rightarrow Full \ m.module\_name)
                             (fusions\_modules @ brakets\_modules));
       default\_accessibility = Private;
       public\_symbols = public\_symbols ();
       print\_declarations = [];
       print\_implementations = [print\_implementations]
  and private\_modules =
    [constants_module; variables_module] @
       fusions_modules @ brakets_modules in
  (public_module, private_modules)
let \ amplitudes\_to\_channel\_single\_file \ cmdline \ oc \ size \ amplitudes \ =
  let \ public\_module, \ private\_modules =
    modules_of_amplitudes cmdline oc size amplitudes in
  Format_Fortran.set_formatter_out_channel ~width:!line_length oc;
  print_description cmdline amplitudes ();
  print\_modules \ (private\_modules \ @ [public\_module])
let \ amplitudes\_to\_channel\_multi\_file \ cmdline \ oc \ size \ amplitudes \ =
  let \ public\_module, \ private\_modules =
     modules_of_amplitudes cmdline oc size amplitudes in
  modules_to_file !line_length oc
     (print\_description\ cmdline\ amplitudes)
     (public\_module :: private\_modules)
```

Dispatch

```
let amplitudes_to_channel cmdline oc diagnostics amplitudes =
      parse_diagnostics diagnostics;
      let ufo\_fusions =
         let ufo_fusions_set = ufo_fusions_used amplitudes in
         if Sets.String.is\_empty\ ufo\_fusions\_set then
         else
           Some ufo_fusions_set in
      begin match ufo_fusions with
       | Some only \rightarrow
          let name = !module_name ^ "_ufo"
          and fortran\_module = Fermions.use\_module in
          use\_modules := name :: !use\_modules;
          UFO. Targets. Fortran. lorentz\_module
             \~only \~name \~fortran\_module \~parameter\_module :!parameter\_module
            (Format_Fortran.formatter_of_out_channel oc) ()
      | None \rightarrow ()
      end:
      match !output_mode with
        Single\_Function \rightarrow
           amplitudes\_to\_channel\_single\_function\ cmdline\ oc\ amplitudes
        Single\_Module\ size\ 	o
           amplitudes_to_channel_single_module cmdline oc size amplitudes
       | Single_File size \rightarrow
           amplitudes_to_channel_single_file cmdline oc size amplitudes
        Multi\_File\ size\ 	o
           amplitudes_to_channel_multi_file cmdline oc size amplitudes
    let parameters_to_channel oc =
      parameters_to_fortran oc (CM.parameters ())
  end
module\ Fortran\ =\ Make\_Fortran(Fortran\_Fermions)
```

Majorana Fermions



JR sez' (regarding the Majorana Feynman rules): For this function we need a different approach due to our aim of implementing the fermion vertices with the right line as ingoing (in a calculational sense) and the left line in a fusion as outgoing. In defining all external lines and the fermionic wavefunctions built out of them as ingoing we have to invert the left lines to make them outgoing. This happens by multiplying them with the inverse charge conjugation matrix in an appropriate representation and then transposing it. We must distinguish whether the direction of calculation and the physical direction of the fermion number flow are parallel or antiparallel. In the first case we can use the "normal" Feynman rules for Dirac particles, while in the second, according to the paper of Denner et al., we have to reverse the sign of the vector and antisymmetric bilinears of the Dirac spinors, cf. the Coupling module.

Note the subtlety for the left- and righthanded couplings: Only the vector part of these couplings changes in the appropriate cases its sign, changing the chirality to the negative of the opposite. $(JR's \ probably \ right, but \ I \ need \ to \ check \ myself \dots)$

```
module Fortran_Majorana_Fermions : Fermions =
struct
  open Coupling
  open Format

let psi_type = "bispinor"
  let psibar_type = "bispinor"
  let chi_type = "bispinor"
  let grav_type = "vectorspinor"
```



 $JR\ sez'$ (regarding the Majorana Feynman rules): Because of our rules for fermions we are going to give all incoming fermions a u spinor and all outgoing fermions a v spinor, no matter whether they are Dirac fermions, antifermions or Majorana fermions. (JR's probably right, but I need to check myself ...)

```
let psi_incoming = "u"
let \ \mathit{brs\_psi\_incoming} \ = \ \texttt{"brs\_u"}
let psibar\_incoming = "u"
let brs_psibar_incoming = "brs_u"
let chi_incoming = "u"
let brs_chi_incoming = "brs_u"
\mathsf{let}\ grav\_incoming\ =\ \mathtt{"ueps"}
let psi\_outgoing = "v"
let \ \mathit{brs\_psi\_outgoing} \ = \ \texttt{"brs\_v"}
let psibar\_outgoing = "v"
let brs_psibar_outgoing = "brs_v"
let chi\_outgoing = "v"
let brs_chi_outgoing = "brs_v"
\mathsf{let}\ \mathit{grav\_outgoing}\ =\ \mathsf{"veps"}
let psi_propagator = "pr_psi"
let psibar_propagator = "pr_psi"
let \ chi\_propagator = "pr\_psi"
let grav_propagator = "pr_grav"
let \ psi\_projector = "pj\_psi"
let \ psibar\_projector = "pj\_psi"
\mathsf{let}\ chi\_projector\ =\ \mathtt{"pj\_psi"}
\texttt{let } grav\_projector \ = \ \texttt{"pj\_grav"}
\mathsf{let}\ \mathit{psi\_gauss}\ =\ \mathtt{"pg\_psi"}
let \ psibar\_gauss = "pg\_psi"
let chi_gauss = "pg_psi"
let grav_gauss = "pg_grav"
let format\_coupling coeff c =
  match coeff with
   | 1 \rightarrow c
     -1 \rightarrow "(-" ^{\circ} ^{\circ}")"
   | coeff \rightarrow string\_of\_int coeff ^ "*" ^ c
let format\_coupling\_2 coeff c =
  match coeff with
   1 \rightarrow c
    -1 \rightarrow \text{"-"} \hat{c}
  | coeff \rightarrow string\_of\_int coeff ^"*" ^ c
```



JR's coupling constant HACK, necessitated by tho's bad design descition.

```
let fastener\ s\ i=
try
let offset=(String.index\ s\ '(')\ in
if ((String.get\ s\ (String.length\ s\ -\ 1))\ \not\equiv\ ')\ ') then
failwith\ "fastener: \ _ \ _ \ _ \ wrong \ _ \ usage \ _ \ offset) and
tail=(String.sub\ s\ (succ\ offset)\ (String.length\ s\ -\ offset\ -\ 2))\ in
if (String.contains\ func\_name\ ')\ ')\ \lor
(String.contains\ func\_name\ ')\ ')\ then
failwith\ "fastener: \ _ \ wrong \ _ \ usage \ _ \ of \ _ \ parentheses"
else
func\_name\ ^ \ "("\ ^ \ string\_of\_int\ i\ ^ ", "\ ^ \ tail\ ^ ")\ "
```

```
with
  Not\_found \rightarrow
       if (String.contains s), then
          failwith "fastener: wrong usage of parentheses"
       else
             \hat{\ } "(" \hat{\ } string\_of\_int i \hat{\ } ")"
          s
let print_fermion_current coeff f c wf1 wf2 fusion =
  let c = format\_coupling coeff c in
  match fusion with
    F13 \mid F31 \rightarrow printf "%s_ff(%s,%s,%s)" f c wf1 wf2
    F23 \mid F21 \rightarrow printf \text{"f-\%sf(\%s,\%s,\%s)"} f c wf1 wf2
    F32 \mid F12 \rightarrow printf \text{"f-\%sf(\%s,\%s,\%s)"} f c wf2 wf1
let print_fermion_current2 coeff f c wf1 wf2 fusion =
  let c = format\_coupling\_2 \ coeff \ c in
  \mathsf{let}\ c1\ =\ \mathit{fastener}\ c\ 1\ \mathsf{and}
       c2 = fastener \ c \ 2 \ {\rm in}
  match fusion with
    F13 \mid F31 \rightarrow printf "%s_ff(%s,%s,%s,%s)" f c1 c2 wf1 wf2
          F21 \rightarrow printf "f_%sf(%s,%s,%s,%s)" f c1 c2 wf1 wf2
    F23
    F32 \mid F12 \rightarrow printf \text{"f-\%sf(\%s,\%s,\%s,\%s)"} f c1 c2 wf2 wf1
let print\_fermion\_current\_mom\_v1 coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format\_coupling coeff c in
  let c1 = fastener c 1 and
       c2 = fastener c 2 in
  \mathsf{match}\ \mathit{fusion}\ \mathsf{with}
    F13 \rightarrow printf "%s_ff(%s,%s,%s,%s)" f c1 c2 wf1 wf2
     F31 \rightarrow printf "%s_ff(-(%s),%s,%s,%s)" f c1 c2 wf1 wf2
    F23 \rightarrow printf \text{ "f\_\%sf(\%s,\%s,\%s,\%s)"} f c1 c2 wf1 wf2
    F32 \rightarrow printf \text{ "f\_\%sf(\%s,\%s,\%s,\%s)"} f c1 c2 wf2 wf1
    F12 \rightarrow printf "f_f%s(-(%s),%s,%s,%s)" f c1 c2 wf2 wf1
    F21 \rightarrow printf "f_f%s(-(%s),%s,%s,%s)" f c1 c2 wf1 wf2
let print_fermion_current_mom_v1_chiral\ coeff\ f\ c\ wf1\ wf2\ p1\ p2\ p12\ fusion\ =
  let c = format\_coupling coeff c in
  \mathsf{let}\ c1\ =\ \mathit{fastener}\ c\ 1\ \mathsf{and}
       c2 = fastener \ c \ 2 \ in
  match fusion with
    F13 \rightarrow printf "%s_ff(%s,%s,%s,%s)" f \ c1 \ c2 \ wf1 \ wf2
    F31 \rightarrow printf "%s_ff(-(%s),-(%s),%s,%s)" f \ c2 \ c1 \ wf1 \ wf2
    F23 \rightarrow printf \text{"f-\%sf(\%s,\%s,\%s,\%s)"} f c1 c2 wf1 wf2
    F32 \rightarrow printf \text{ "f\_\%sf(\%s,\%s,\%s,\%s)"} f c1 c2 wf2 wf1
    F12 \rightarrow printf \ "f_f%s(-(%s),-(%s),%s,%s)" f c2 c1 wf2 wf1
   F21 \rightarrow printf "f_f%s(-(%s),-(%s),%s,%s)" f c2 c1 wf2 wf1
let print\_fermion\_current\_mom\_v2 coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format\_coupling coeff c in
  let c1 = fastener c 1 and
       c2 = fastener c 2 in
  match fusion with
    F13 \rightarrow printf "%s_ff(%s,%s,%s,%s,%s)" f c1 c2 wf1 wf2 p12
    F31 \rightarrow printf "%s_ff(-(%s),%s,%s,%s,%s)" f c1 c2 wf1 wf2 p12
    F23 \rightarrow printf \text{ "f\_\%sf(\%s,\%s,\%s,\%s,\%s)"} f c1 c2 wf1 wf2 p1
    F32 \rightarrow printf \text{ "f\_\%sf(\%s,\%s,\%s,\%s,\%s)"} f c1 c2 wf2 wf1 p2
    F12 \rightarrow printf \ "f_f%s(-(%s),%s,%s,%s,%s)" f c1 c2 wf2 wf1 p2
    F21 \rightarrow printf \text{ "f_f%s(-(%s),%s,%s,%s,%s)"} f c1 c2 wf1 wf2 p1
let print\_fermion\_current\_mom\_v2\_chiral coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format\_coupling coeff c in
  let c1 = fastener \ c \ 1 and
       c2 = fastener \ c \ 2 \ in
  match \ fusion \ with
```

```
F13 \rightarrow printf "%s_ff(%s,%s,%s,%s,%s,%s)" f c1 c2 wf1 wf2 p12
     F31 \rightarrow printf "%s_ff(-(%s),-(%s),%s,%s,%s)" f c2 c1 wf2 wf1 p12
     F23 \rightarrow printf \text{ "f\_\%sf(\%s,\%s,\%s,\%s,\%s)"} f c1 c2 wf1 wf2 p1
     F32 \rightarrow printf \text{"f-\%sf(\%s,\%s,\%s,\%s,\%s,\%s)"} f c1 c2 wf2 wf1 p2
     F12 \rightarrow printf \text{ "f_f%s(-(%s),-(%s),%s,%s,%s)" } f c2 c1 wf1 wf2 p2
    F21 \rightarrow printf \text{ "f\_f\%s(-(\%s),-(\%s),\%s,\%s,\%s)" } f c2 c1 wf2 wf1 p1
let \ print\_fermion\_current\_vector \ coeff \ f \ c \ wf1 \ wf2 \ fusion =
  let c = format\_coupling coeff c in
  match fusion with
     F13 \rightarrow printf "%s_ff(%s,%s,%s)" f c wf1 wf2
     F31 \rightarrow printf "%s_ff(-%s,%s,%s)" f c wf1 wf2
     F23 \rightarrow printf "f_%sf(%s,%s,%s)" f c wf1 wf2
     F32 \rightarrow printf "f_%sf(%s,%s,%s)" f c wf2 wf1
     F12 \rightarrow printf "f_%sf(-%s,%s,%s)" f c wf2 wf1
    F21 \rightarrow printf "f_%sf(-%s,%s,%s)" f c wf1 wf2
\label{let:print_fermion_current2_vector} \ coeff \ f \ c \ wf1 \ wf2 \ fusion \ =
  let c = format\_coupling\_2 \ coeff \ c in
  let c1 = fastener \ c \ 1 and
       c2 = fastener c 2 in
  match \ fusion \ with
    F13 \rightarrow printf "%s_ff(%s,%s,%s,%s)" f c1 c2 wf1 wf2
     F31 \rightarrow printf "%s_ff(-(%s),%s,%s,%s)" f c1 c2 wf1 wf2
     F23 \rightarrow printf "f_%sf(%s,%s,%s,%s)" f c1 c2 wf1 wf2
     F32 \rightarrow printf \text{ "f\_\%sf(\%s,\%s,\%s,\%s)"} f c1 c2 wf2 wf1
     F12 \rightarrow printf \text{ "f\_\%sf(-(\%s),\%s,\%s,\%s)"} f c1 c2 wf2 wf1
    F21 \rightarrow printf "f_%sf(-(%s),%s,%s,%s)" f c1 c2 wf1 wf2
let print_fermion_current_chiral coeff f1 f2 c wf1 wf2 fusion =
  let c = format\_coupling coeff c in
  match fusion with
     F13 \rightarrow printf "%s_ff(%s,%s,%s)" f1 c wf1 wf2
     F31 \rightarrow printf "%s_ff(-%s,%s,%s)" f2 c wf1 wf2
     F23 \rightarrow printf "f_%sf(%s,%s,%s)" f1 c wf1 wf2
     F32 \rightarrow printf "f_%sf(%s,%s,%s)" f1 c wf2 wf1
     F12 \rightarrow printf "f\_%sf(-%s,%s,%s)" f2 c wf2 wf1
    F21 \rightarrow printf "f_%sf(-%s,%s,%s)" f2 c wf1 wf2
let print_fermion_current2_chiral coeff f c wf1 wf2 fusion =
  let c = format\_coupling\_2 \ coeff \ c in
  let c1 = fastener c 1 and
       c\mathcal{2} \ = \ fastener \ c \ 2 \ \mathsf{in}
  match fusion with
    F13 \rightarrow printf "%s_ff(%s,%s,%s,%s)" f c1 c2 wf1 wf2
     F31 \rightarrow printf "%s_ff(-(%s),-(%s),%s,%s)" f \ c2 \ c1 \ wf1 \ wf2
    F23 \rightarrow printf \text{"f\_\%sf(\%s,\%s,\%s,\%s)"} f c1 c2 wf1 wf2
    F32 \rightarrow printf \text{ "f-\%sf(\%s,\%s,\%s,\%s)"} f c1 c2 wf2 wf1
     F12 \rightarrow printf \ "f\_\%sf(-(\%s),-(\%s),\%s,\%s)" f c2 c1 wf2 wf1
    F21 \rightarrow printf \text{"f\_\%sf(-(\%s),-(\%s),\%s,\%s)"} f c2 c1 wf1 wf2
let print\_current = function
     coeff, _, VA, _ \rightarrow print\_fermion\_current2\_vector\ coeff "va"
     coeff, \ \_, \ V, \ \_ \ \rightarrow \ print\_fermion\_current\_vector \ coeff \ "v"
     coeff, -, A, - \rightarrow print\_fermion\_current coeff "a"
     coeff, \_, VL, \_ \rightarrow print\_fermion\_current\_chiral\ coeff\ "vl"\ "vr"
     coeff, _, VR, _ \rightarrow print\_fermion\_current\_chiral coeff "vr" "vl"
     coeff, \_, VLR, \_ \rightarrow print\_fermion\_current2\_chiral\ coeff\ "vlr"
     coeff, _, SP, _ \rightarrow print\_fermion\_current2 coeff "sp"
     coeff, \_, S, \_ \rightarrow print\_fermion\_current coeff "s"
     coeff, _, P, _ \rightarrow print\_fermion\_current coeff "p"
     coeff, _, SL, _ \rightarrow print\_fermion\_current coeff "sl"
     coeff, _, SR, _ \rightarrow print\_fermion\_current coeff "sr"
     coeff, _, SLR, _ \rightarrow print\_fermion\_current2 coeff "slr"
```

```
|\hspace{.08cm} coeff, -, POT, - \rightarrow print\_fermion\_current\_vector \hspace{.08cm} coeff \hspace{.1cm} "pot" \\ |\hspace{.08cm} -, -, -, - \rightarrow invalid\_arg \\ |\hspace{.08cm} "Targets.Fortran\_Majorana\_Fermions: \_Not\_needed\_in\_the\_models" \\ |\hspace{.08cm} let \hspace{.08cm} print\_current\_p \hspace{.1cm} = \hspace{.08cm} function \\ |\hspace{.08cm} coeff, \hspace{.08cm} Psi, \hspace{.08cm} SL, \hspace{.08cm} Psi \hspace{.08cm} \rightarrow print\_fermion\_current \hspace{.08cm} coeff \hspace{.08cm} "sr" \\ |\hspace{.08cm} coeff, \hspace{.08cm} Psi, \hspace{.08cm} SLR, \hspace{.08cm} Psi \hspace{.08cm} \rightarrow \hspace{.08cm} print\_fermion\_current2 \hspace{.08cm} coeff \hspace{.08cm} "slr" \\ |\hspace{.08cm} -, -, -, - \rightarrow invalid\_arg \\ |\hspace{.08cm} "Targets.Fortran\_Majorana\_Fermions: \_Not\_needed\_in\_the\_used\_models" \\ |\hspace{.08cm} let \hspace{.08cm} print\_fermion\_current \hspace{.08cm} coeff \hspace{.08cm} "sl" \\ |\hspace{.08cm} coeff, \hspace{.08cm} Psibar, \hspace{.08cm} SL, \hspace{.08cm} Psibar \hspace{.08cm} \rightarrow \hspace{.08cm} print\_fermion\_current \hspace{.08cm} coeff \hspace{.08cm} "sr" \\ |\hspace{.08cm} coeff, \hspace{.08cm} Psibar, \hspace{.08cm} SLR, \hspace{.08cm} Psibar \hspace{.08cm} \rightarrow \hspace{.08cm} print\_fermion\_current2 \hspace{.08cm} coeff \hspace{.08cm} "slr" \\ |\hspace{.08cm} -, -, -, - \rightarrow invalid\_arg \\ |\hspace{.08cm} "Targets.Fortran\_Majorana\_Fermions: \_Not\_needed\_in\_the\_used\_models" \\ |\hspace{.08cm} "Targets.Fortran\_Majorana\_Fermions: \_Not\_need
```

This function is for the vertices with three particles including two fermions but also a momentum, therefore with a dimensionful coupling constant, e.g. the gravitino vertices. One has to dinstinguish between the two kinds of canonical orders in the string of gamma matrices. Of course, the direction of the string of gamma matrices is reversed if one goes from the Gravbar, _, Psi to the Psibar, _, Grav vertices, and the same is true for the couplings of the gravitino to the Majorana fermions. For more details see the tables in the coupling implementation.

We now have to fix the directions of the momenta. For making the compiler happy and because we don't want to make constructions of infinite complexity we list the momentum including vertices without gravitinos here; the pattern matching says that's better. Perhaps we have to find a better name now.

For the cases of MOM, MOM5, MOML and MOMR which arise only in BRST transformations we take the mass as a coupling constant. For VMOM we don't need a mass either. These vertices are like kinetic terms and so need not have a coupling constant. By this we avoid a strange and awful construction with a new variable. But be careful with a generalization if you want to use these vertices for other purposes.

```
let format\_coupling\_mom coeff c =
  match coeff with
    1 \rightarrow c
     -1 \rightarrow "(-" \hat{c} \hat{c})"
   | coeff \rightarrow string\_of\_int coeff ^ "*" ^ c
let \ commute\_proj \ f =
  match f with
     "mom1" \rightarrow "lmom"
     "momr" \rightarrow "rmom"
     "lmom" \rightarrow "moml"
     "rmom" \rightarrow "momr"
     "svl" \rightarrow "svr"
     "svr" \rightarrow "svl"
     "sl" 
ightarrow "sr"
     "sr" \rightarrow "sl"
     "s" 
ightarrow "s"
     "p" \rightarrow "p"
     \_ \rightarrow invalid\_arg "Targets:Fortran_Majorana_Fermions:\_wrong\_case"
let print_fermion_current_mom coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format\_coupling\_mom\ coeff\ c in
  let c1 = fastener \ c \ 1 and
        c2 = fastener \ c \ 2 \ in
  match fusion with
    F13 \rightarrow printf "%s_ff(%s,%s,%s,%s,%s)" f c1 c2 wf1 wf2 p12
     F31 \rightarrow printf "%s_ff(%s,%s,%s,%s,%s)" f c1 c2 wf1 wf2 p12
     F23 \rightarrow printf "f_%sf(%s,%s,%s,%s,%s)" f c1 c2 wf1 wf2 p1
     F32 \rightarrow printf "f_%sf(%s,%s,%s,%s,%s)" f c1 c2 wf2 wf1 p2
     F12 \rightarrow printf \text{"f\_\%sf(\%s,\%s,\%s,\%s,\%s)"} f c1 c2 wf2 wf1 p2
    F21 \rightarrow printf \text{ "f\_\%sf(\%s,\%s,\%s,\%s,\%s)"} f c1 c2 wf1 wf2 p1
let print_fermion_current_mom_sign coeff f c wf1 wf2 p1 p2 p12 fusion =
```

```
let c = format\_coupling\_mom\ coeff\ c in
  let c1 = fastener c 1 and
       c2 = fastener \ c \ 2 \ {\rm in}
  match fusion with
    F13 \rightarrow printf "%s_ff(%s,%s,%s,%s,%s)" f c1 c2 wf1 wf2 p12
    F31 \rightarrow printf "%s_ff(%s,%s,%s,%s,-(%s))" f c1 c2 wf1 wf2 p12
    F23 \rightarrow printf \text{ "f\_\%sf(\%s,\%s,\%s,\%s,\%s)"} f c1 c2 wf1 wf2 p1
    F32 \rightarrow printf "f_%sf(%s,%s,%s,%s,%s)" f c1 c2 wf2 wf1 p2
    F12 \rightarrow printf \text{"f\_\%sf(\%s,\%s,\%s,-(\%s))"} f c1 c2 wf2 wf1 p2
   F21 \rightarrow printf "f_%sf(%s,%s,%s,%s,-(%s))" f c1 c2 wf1 wf2 p1
let print\_fermion\_current\_mom\_sign\_1 coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format\_coupling coeff c in
  match fusion with
    F13 \rightarrow printf "%s_ff(%s,%s,%s,%s)" f c wf1 wf2 p12
    F31 \rightarrow printf "%s_ff(%s,%s,%s,-(%s))" f c wf1 wf2 p12
    F23 \rightarrow printf "f_%sf(%s,%s,%s,%s)" f c wf1 wf2 p1
    F32 \rightarrow printf \text{ "f-\%sf(\%s,\%s,\%s,\%s)" } f c wf2 wf1 p2
    F12 \rightarrow printf \text{ "f\_\%sf(\%s,\%s,\%s,-(\%s))} \text{ "} f c wf2 wf1 p2
  F21 \rightarrow printf "f_%sf(%s,%s,%s,-(%s))" f c wf1 wf2 p1
let print_fermion_current_mom_chiral coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format\_coupling\_mom\ coeff\ c and
       cf = commute\_proj f in
  let c1 = fastener \ c \ 1 and
       c2 = fastener \ c \ 2 \ in
  match fusion with
    F13 \rightarrow printf "%s_ff(%s,%s,%s,%s,%s)" f c1 c2 wf1 wf2 p12
    F31 \rightarrow printf "%s_ff(%s,%s,%s,\u00ab,s,-(%s))" cf c1 c2 wf1 wf2 p12
    F23 \rightarrow printf \text{ "f\_\%sf(\%s,\%s,\%s,\%s,\%s)"} f c1 c2 wf1 wf2 p1
    F32 \rightarrow printf "f_%sf(%s,%s,%s,%s,%s)" f c1 c2 wf2 wf1 p2
    F12 \rightarrow printf \text{"f-\%sf(\%s,\%s,\%s,-(\%s))"} cf c1 c2 wf2 wf1 p2
   | F21 \rightarrow printf "f_%sf(%s,%s,%s,%s,-(%s))" cf c1 c2 wf1 wf2 p1
let print\_fermion\_g\_current coeff f c wf1 wf2 p1 p2 p12 fusion =
  \mathsf{let}\ c\ =\ format\_coupling\ coe\!f\!f\ c\ \mathsf{in}
  match fusion with
    F13 \rightarrow printf "%s_grf(%s,%s,%s,%s,%s)" f c wf1 wf2 p12
     F31 \rightarrow printf "%s_fgr(%s,%s,%s,%s)" f c wf1 wf2 p12
    F23 \rightarrow printf \ "gr_%sf(%s,%s,%s,%s)" f c wf1 wf2 p1
    F32 \rightarrow printf "gr_%sf(%s,%s,%s,%s)" f c wf2 wf1 p2
    F12 \rightarrow printf \text{"f-\%sgr(\%s,\%s,\%s,\%s)"} f c wf2 wf1 p2
    F21 \rightarrow printf \text{"f\_\%sgr(\%s,\%s,\%s,\%s)"} f c wf1 wf2 p1
let print\_fermion\_g\_2\_current coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format\_coupling coeff c in
  match fusion with
    F13 \rightarrow printf "%s_grf(%s(1),%s(2),%s,%s,%s)" f c c wf1 wf2 p12
    F31 \rightarrow printf "%s_fgr(%s(1),%s(2),%s,%s,%s)" f c c wf1 wf2 p12
    F23 \rightarrow printf \ "gr\_\%sf(\%s(1),\%s(2),\%s,\%s,\%s)" \ f \ c \ c \ wf1 \ wf2 \ p1
    F32 \rightarrow printf \ "gr\_\%sf(\%s(1),\%s(2),\%s,\%s,\%s)" \ f \ c \ wf2 \ wf1 \ p2
    F12 \rightarrow printf \text{"f\_\%sgr(\%s(1),\%s(2),\%s,\%s,\%s)"} f c c wf2 wf1 p2
    F21 \rightarrow printf "f_%sgr(%s(1),%s(2),%s,%s,%s)" f c c wf1 wf2 p1
let print_fermion_g_current_rev coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format\_coupling coeff c in
  match fusion with
    F13 \rightarrow printf "%s_fgr(%s,%s,%s,%s)" f c wf1 wf2 p12
    F31 \rightarrow printf "%s_grf(%s,%s,%s,%s)" f c wf1 wf2 p12
    F23 \rightarrow printf "f_%sgr(%s,%s,%s,%s)" f c wf1 wf2 p1
    F32 \rightarrow printf "f_%sgr(%s,%s,%s,%s)" f c wf2 wf1 p2
    F12 \rightarrow printf "gr_%sf(%s,%s,%s,%s)" f c wf2 wf1 p2
    F21 \rightarrow printf \ "gr_%sf(%s,%s,%s,%s)" f c wf1 wf2 p1
```

```
let print\_fermion\_g\_2\_current\_rev coeff f c wf1 wf2 p1 p2 p12 fusion =
  let c = format\_coupling coeff c in
  match fusion with
    F13 \rightarrow printf \text{ "%s-fgr(%s(1),%s(2),%s,%s,%s)"} f c c wf1 wf2 p12
    F31 \rightarrow printf "%s_grf(%s(1),%s(2),%s,%s,%s)" f c c wf1 wf2 p12
    F23 \rightarrow printf \ "f_%sgr(%s(1),%s(2),%s,%s,%s)" f c c wf1 wf2 p1
    F32 \rightarrow printf \text{"f\_\%sgr(\%s(1),\%s(2),\%s,\%s,\%s)"} f c c wf2 wf1 p2
    F12 \rightarrow printf \ "gr_%sf(%s(1),%s(2),%s,%s,%s)" \ f \ c \ wf2 \ wf1 \ p2
    F21 \rightarrow printf "gr_%sf(%s(1),%s(2),%s,%s,%s)" f c c wf1 wf2 p1
\label{let:print_fermion_g_current_vector} \ coeff \ f \ c \ wf1 \ wf2 \ \_ \ \_ \ fusion \ =
  let c = format\_coupling coeff c in
  match fusion with
    F13 \rightarrow printf \text{ "%s\_grf (%s,%s,%s)} \text{ "} f c wf1 wf2
    F31 \rightarrow printf "%s_fgr(-%s,%s,%s)" f c wf1 wf2
    F23 \rightarrow printf "gr_%sf(%s,%s,%s)" f c wf1 wf2
    F32 \rightarrow printf "gr_%sf(%s,%s,%s)" f c wf2 wf1
    F12 \rightarrow printf "f_%sgr(-%s,%s,%s)" f c wf2 wf1
   F21 \rightarrow printf \ "f_%sgr(-%s,%s,%s)" f \ c \ wf1 \ wf2
let print_fermion_g_current_vector_rev coeff f c wf1 wf2 _ _ _ fusion =
  let c = format\_coupling coeff c in
  match fusion with
    F13 \rightarrow printf \text{ "%s_fgr(%s,%s,%s)} \text{ "} f c wf1 wf2
    F31 \rightarrow printf "%s_grf(-%s,%s,%s)" f \ c \ wf1 \ wf2
    F23 \rightarrow printf "f_%sgr(%s,%s,%s)" f c wf1 wf2
    F32 \rightarrow printf "f_%sgr(%s,%s,%s)" f c wf2 wf1
    F12 \rightarrow printf "gr_%sf(-%s,%s,%s)" f c wf2 wf1
    F21 \rightarrow printf "gr_%sf(-%s,%s,%s)" f c wf1 wf2
let print\_current\_g = function
    coeff, _, MOM, _ → print_fermion_current_mom_sign coeff "mom"
    coeff, _, MOM5, _ \rightarrow print\_fermion\_current\_mom\ coeff "mom5"
    coeff, \_, MOML, \_ \rightarrow print\_fermion\_current\_mom\_chiral coeff "moml"
    coeff, _, MOMR, _ → print_fermion_current_mom_chiral coeff "momr"
    coeff, \_, LMOM, \_ \rightarrow print\_fermion\_current\_mom\_chiral\ coeff\ "lmom"
    coeff, _, RMOM, _ → print_fermion_current_mom_chiral coeff "rmom"
    coeff, _, VMOM, _ → print_fermion_current_mom_sign_1 coeff "vmom"
    coeff, Gravbar, S, \_ \rightarrow print\_fermion\_g\_current coeff "s"
    coeff, Gravbar, SL, \rightarrow print\_fermion\_g\_current coeff "sl"
    coeff, Gravbar, SR, \_ \rightarrow print\_fermion\_g\_current coeff "sr"
    coeff, Gravbar, SLR, \rightarrow print\_fermion\_g\_2\_current coeff "slr"
    coeff, Gravbar, P, \_ \rightarrow print\_fermion\_g\_current coeff "p"
    coeff, Gravbar, V, \_ \rightarrow print\_fermion\_g\_current coeff "v"
    coe\!f\!f,\ Gravbar,\ VLR,\ \_\ \rightarrow\ print\_fermion\_g\_2\_current\ coe\!f\!f\ "vlr"
    coeff, Gravbar, POT, \_ \rightarrow print\_fermion\_g\_current\_vector coeff "pot"
    coeff, \ \_, \ S, \ Grav \ 	o \ print\_fermion\_g\_current\_rev \ coeff \ "s"
    coeff, _, SL, Grav \rightarrow print\_fermion\_g\_current\_rev coeff "sl"
    coeff, _, SR, Grav \rightarrow print\_fermion\_g\_current\_rev coeff "sr"
    coeff, _, SLR, Grav \rightarrow print\_fermion\_g\_2\_current\_rev coeff "slr"
    coeff, -, P, Grav \rightarrow print\_fermion\_g\_current\_rev (-coeff) "p"
    coeff, -, V, Grav \rightarrow print\_fermion\_g\_current\_rev coeff "v"
    coeff, \_, VLR, Grav \rightarrow print\_fermion\_g\_2\_current\_rev coeff "vlr"
    coeff, _, POT, Grav → print_fermion_g_current_vector_rev coeff "pot"
    \_, \_, \_, \_ \rightarrow invalid\_arg
       "Targets.Fortran\_Majorana\_Fermions: \_not\_used\_in\_the\_models"
let print\_current\_mom = function
    coeff, _, TVA, _ \rightarrow print\_fermion\_current\_mom\_v1 coeff "tva"
    coeff, _, TVAM, _ \rightarrow print\_fermion\_current\_mom\_v2 coeff "tvam"
    coeff, \_, TLR, \_ \rightarrow print\_fermion\_current\_mom\_v1\_chiral\ coeff\ "tlr"
    coeff, -, TLRM, - \rightarrow print\_fermion\_current\_mom\_v2\_chiral\ coeff "tlrm"
    \_, \ \_, \ \_, \ \_ \ \rightarrow \ invalid\_arg
```

"Targets.Fortran_Majorana_Fermions: \(\)\Not \(\) needed \(\) in \(\)the \(\)models"

We need support for dimension-5 vertices with two fermions and two bosons, appearing in theories of supergravity and also together with in insertions of the supersymmetric current. There is a canonical order fermionbar, $boson_2$, fermion, so what one has to do is a mapping from the fusions F123 etc. to the order of the three wave functions wf1, wf2 and wf3.

The function $d_{-}p$ (for distinct the particle) distinguishes which particle (scalar or vector) must be fused to in the special functions.

```
let d_p = function
    1, ("sv" | "pv" | "svl" | "svr" | "slrv" ) \rightarrow "1"
     1, - \rightarrow ""
     2, ("sv"| "pv"| "svl"| "svr"| "slrv") \rightarrow "2"
     2, \rightarrow ""
   \mid _, _ 
ightarrow invalid\_arg "Targets.Fortran_Majorana_Fermions:\sqcupnot\sqcupused"
let wf \circ of \circ f wf1 wf2 wf3 f =
   \mathsf{match}\; f \; \mathsf{with} \\
     (F123 \mid F423) \rightarrow [wf2; wf3; wf1]
     (F213 \mid F243 \mid F143 \mid F142 \mid F413 \mid F412) \rightarrow [wf1; wf3; wf2]
     (F132 \mid F432) \rightarrow [wf3; wf2; wf1]
    (F231 \mid F234 \mid F134 \mid F124 \mid F431 \mid F421) \rightarrow [wf1; wf2; wf3]
    (F312 \mid F342) \rightarrow [wf3; wf1; wf2]
   \mid (F321 \mid F324 \mid F314 \mid F214 \mid F341 \mid F241) \rightarrow [wf2; wf1; wf3]
\mbox{let } print\_fermion\_g4\_brs\_vector\_current \ coeff \ f \ c \ wf1 \ wf2 \ wf3 \ fusion \ =
  let cf = commute\_proj f and
        cp = format\_coupling coeff c and
        cm = if f = "pv" then
          format_coupling coeff c
        else
          format\_coupling\ (-coeff)\ c
  and
        d1 = d_p(1,f) and
        d2 = d_p(2, f) and
        f1 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 0) and
        f2 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 1) and
        f3 = (List.nth (wf of f wf1 wf2 wf3 fusion) 2) in
  \mathsf{match}\ \mathit{fusion}\ \mathsf{with}
    (F123 \mid F213 \mid F132 \mid F231 \mid F312 \mid F321) \rightarrow
        printf "f_%sf(%s,%s,%s,%s)" cf cm f1 f2 f3
    (F423 \mid F243 \mid F432 \mid F234 \mid F342 \mid F324) \rightarrow
        printf "f_%sf(%s,%s,%s,%s)" f cp f1 f2 f3
     (F134 \mid F143 \mid F314) \rightarrow printf "%s%s_ff(%s,%s,%s,%s)" f d1 cp f1 f2 f3
     (F124 \mid F142 \mid F214) \rightarrow printf "%s%s_ff(%s,%s,%s,%s)" f \ d2 \ cp \ f1 \ f2 \ f3
     (F413 \mid F431 \mid F341) \rightarrow printf "%s%s_ff(%s,%s,%s,%s)" cf \ d1 \ cm \ f1 \ f2 \ f3
    (F241 \mid F412 \mid F421) \rightarrow printf "%s%s_ff(%s,%s,%s,%s,%s)" cf d2 cm f1 f2 f3
let print_fermion_g4_svlr_current coeff _ c wf1 wf2 wf3 fusion =
  let c = format\_coupling\_2 \ coeff \ c and
        f1 = (List.nth (wf\_of\_f wf1 wf2 wf3 fusion) 0) and
        f2 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 1) and
        f3 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 2) in
  let c1 = fastener c 1 and
        c2 = fastener c 2 in
  match fusion with
   \mid (F123 \mid F213 \mid F132 \mid F231 \mid F312 \mid F321) \rightarrow
        printf "f_svlrf(-(%s),-(%s),%s,%s,%s)" c2 c1 f1 f2 f3
  \mid (F423 \mid F243 \mid F432 \mid F234 \mid F342 \mid F324) \rightarrow
        printf "f_svlrf(%s,%s,%s,%s,%s)" c1 c2 f1 f2 f3
   \mid (F134 \mid F143 \mid F314) \rightarrow
        printf "svlr2_ff(%s,%s,%s,%s,%s,%s)" c1 c2 f1 f2 f3
  \mid (F124 \mid F142 \mid F214) \rightarrow
        printf "svlr1_ff(%s,%s,%s,%s,%s,%s)" c1 c2 f1 f2 f3
```

```
\mid (F413 \mid F431 \mid F341) \rightarrow
         printf "svlr2_ff(-(%s),-(%s),%s,%s,%s)" c2 c1 f1 f2 f3
   | (F241 | F412 | F421) \rightarrow
         printf "svlr1_ff(-(%s),-(%s),%s,%s,%s,%s)" c2 c1 f1 f2 f3
let print_fermion_s2_current coeff f c wf1 wf2 wf3 fusion =
   let cp = format\_coupling coeff c and
         cm = if f = "p" then
            format\_coupling\ (-coeff)\ c
         else
            format_coupling coeff c
   and
         cf = commute\_proj f and
         f1 = (List.nth (wf\_of\_f wf1 wf2 wf3 fusion) 0) and
         f2 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 1) and
         f3 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 2) in
   match fusion with
   \mid (F123 \mid F213 \mid F132 \mid F231 \mid F312 \mid F321) <math>\rightarrow
         printf "%s<sub>\\\</sub>*<sub>\\\</sub>f_\%sf(%s,%s,%s)" f1 cf cm f2 f3
   \mid (F423 \mid F243 \mid F432 \mid F234 \mid F342 \mid F324) \rightarrow
         printf "%s<sub>\\\\</sub>*<sub>\\\\</sub>f<sub>\\\\</sub>%sf(%s,%s,\\\\s)" f1 f cp f2 f3
   \mid (F134 \mid F143 \mid F314) \rightarrow
         printf "%s<sub>□</sub>*<sub>□</sub>%s<sub>-</sub>ff(%s,%s,%s)" f2 f cp f1 f3
   \mid (F124 \mid F142 \mid F214) \rightarrow
         printf "%s<sub>\\\\</sub>*<sub>\\\\</sub>%s<sub>\\\\</sub>ff(%s,\\\\s,\\\\s)" f2 f cp f1 f3
   \mid (F413 \mid F431 \mid F341) \rightarrow
         printf "%s<sub>□</sub>*<sub>□</sub>%s<sub>-</sub>ff(%s,%s,%s)" f2 cf cm f1 f3
   \mid (F241 \mid F412 \mid F421) \rightarrow
         printf "%s<sub>\\\\</sub>*<sub>\\\\</sub>%s<sub>\\\</sub>ff(%s,\\\\s,\\\\s)" f2 cf cm f1 f3
let print_fermion_s2p_current coeff f c wf1 wf2 wf3 fusion =
   \mathsf{let}\ c\ =\ \mathit{format\_coupling\_2}\ \mathit{coeff}\ c\ \mathsf{and}
         f1 = (List.nth (wf\_of\_f wf1 wf2 wf3 fusion) 0) and
         f2 = (List.nth (wf \circ of f wf1 wf2 wf3 fusion) 1) and
         f3 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 2) in
   let c1 = fastener \ c \ 1 and
         c2 = fastener c 2 in
   match fusion with
     (F123 \mid F213 \mid F132 \mid F231 \mid F312 \mid F321) \rightarrow
         printf "%s<sub>\\\\</sub>*<sub>\\\\</sub>f<sub>\\\\</sub>%sf(%s,-(\\\\s),\\\\s,\\\\s)" f1 f c1 c2 f2 f3
   \mid (F423 \mid F243 \mid F432 \mid F234 \mid F342 \mid F324) \rightarrow
         printf "%s<sub>□</sub>*<sub>□</sub>f<sub>_</sub>%sf(%s,%s,%s,%s,%s)" f1 f c1 c2 f2 f3
   \mid (F134 \mid F143 \mid F314) \rightarrow
         printf "%s<sub>\\\\</sub>*<sub>\\\\</sub>%s<sub>\\\\</sub>ff(%s,\\\\s,\\\\s,\\\\s)" f2 f c1 c2 f1 f3
   \mid (F124 \mid F142 \mid F214) \rightarrow
         \mid (F413 \mid F431 \mid F341) \rightarrow
         printf "%s<sub>\\\\</sub>*<sub>\\\\</sub>%s<sub>\\\</sub>ff(%s,-(\\\\s),\\\\s,\\\\s)" f2 f c1 c2 f1 f3
   \mid (F241 \mid F412 \mid F421) \rightarrow
         printf "%s<sub>\\\\</sub>*<sub>\\\\</sub>%s<sub>\\\\</sub>ff(%s,-(\\\\s),\\\\s,\\\\s)" f2 f c1 c2 f1 f3
let print_fermion_s2lr_current coeff f c wf1 wf2 wf3 fusion =
   let c = format\_coupling\_2 coeff c and
         f1 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 0) and
         f2 = (List.nth (wf \circ of f wf1 wf2 wf3 fusion) 1) and
         f3 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 2) in
   let c1 = fastener \ c \ 1 and
         c2 = fastener c 2 in
   match fusion with
   \mid (F123 \mid F213 \mid F132 \mid F231 \mid F312 \mid F321) \rightarrow
         printf "%s<sub>□</sub>*<sub>□</sub>f<sub>_</sub>%sf(%s,%s,%s,%s,%s)" f1 f c2 c1 f2 f3
   \mid (F423 \mid F243 \mid F432 \mid F234 \mid F342 \mid F324) \rightarrow
         printf "%s<sub>□</sub>*<sub>□</sub>f<sub>-</sub>%sf(%s,%s,%s,%s,%s)" f1 f c1 c2 f2 f3
```

```
\mid (F134 \mid F143 \mid F314) \rightarrow
        printf "%s<sub>\\\\</sub>*<sub>\\\\</sub>%s<sub>\\\\</sub>ff(\%s,\%s,\%s,\\\\s)" f2 f c1 c2 f1 f3
   (F124 \mid F142 \mid F214) \rightarrow
        printf "%s<sub>□</sub>*<sub>□</sub>%s<sub>−</sub>ff(%s,%s,%s,%s,%s)" f2 f c1 c2 f1 f3
   (F413 \mid F431 \mid F341) \rightarrow
        printf "%s<sub>□</sub>*<sub>□</sub>%s<sub>−</sub>ff(%s,%s,%s,%s,%s)" f2 f c2 c1 f1 f3
   \mid (F241 \mid F412 \mid F421) \rightarrow
        printf "%s<sub>\\\\</sub>*<sub>\\\\</sub>%s<sub>\\\\</sub>ff(%s,\\\\s,\\\\s,\\\\s)" f2 f c2 c1 f1 f3
let print\_fermion\_g4\_current coeff f c wf1 wf2 wf3 fusion =
  let c = format\_coupling coeff c and
       f1 = (List.nth (wf \circ f f wf1 wf2 wf3 fusion) 0) and
       f2 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 1) and
       f3 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 2) in
  match fusion with
   \mid (F123 \mid F213 \mid F132 \mid F231 \mid F312 \mid F321) \rightarrow
        printf "f_%sgr(-%s,%s,%s,%s)" f c f1 f2 f3
  \mid (F423 \mid F243 \mid F432 \mid F234 \mid F342 \mid F324) \rightarrow
        printf "gr_%sf(%s,%s,%s,%s)" f c f1 f2 f3
  \mid (F134 \mid F143 \mid F314 \mid F124 \mid F142 \mid F214) \rightarrow
        printf "%s_grf(%s,%s,%s,%s,%s)" f c f1 f2 f3
  | (F413 \mid F431 \mid F341 \mid F241 \mid F412 \mid F421) \rightarrow
        printf "%s_fgr(-%s,%s,%s,%s)" f c f1 f2 f3
let print_fermion_2_g4_current coeff f c wf1 wf2 wf3 fusion =
  let f1 = (List.nth (wf \circ f - f wf1 wf2 wf3 fusion) 0) and
       f2 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 1) and
       f3 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 2) in
  let c = format\_coupling\_2 \ coeff \ c in
  let c1 = fastener c 1 and
        c2 = fastener c 2 in
  match fusion with
  \mid (F123 \mid F213 \mid F132 \mid F231 \mid F312 \mid F321) \rightarrow
        printf "f_%sgr(-(%s),-(%s),%s,%s,%s)" f c2 c1 f1 f2 f3
  \mid (F423 \mid F243 \mid F432 \mid F234 \mid F342 \mid F324) \rightarrow
        printf "gr_%sf(%s,%s,%s,%s,%s)" f c1 c2 f1 f2 f3
   \mid (F134 \mid F143 \mid F314 \mid F124 \mid F142 \mid F214) \rightarrow
        printf "%s_grf(%s,%s,%s,%s,%s)" f c1 c2 f1 f2 f3
   \mid (F413 \mid F431 \mid F341 \mid F241 \mid F412 \mid F421) \rightarrow
        printf "%s_fgr(-(%s),-(%s),%s,%s,%s)" f c2 c1 f1 f2 f3
let print_fermion_g4_current_rev coeff f c wf1 wf2 wf3 fusion =
  let c = format\_coupling coeff c and
        f1 = (List.nth (wf \circ of f wf1 wf2 wf3 fusion) 0) and
       f2 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 1) and
       f3 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 2) in
  match fusion with
    (F123 \mid F213 \mid F132 \mid F231 \mid F312 \mid F321) \rightarrow
        printf "f_%sgr(%s,%s,%s,%s)" f c f1 f2 f3
   \mid (F423 \mid F243 \mid F432 \mid F234 \mid F342 \mid F324) \rightarrow
        printf "gr_%sf(-%s,%s,%s,%s)" f c f1 f2 f3
  | (F134 \mid F143 \mid F314 \mid F124 \mid F142 \mid F214) \rightarrow
        printf "%s_grf(-%s,%s,%s,%s)" f c f1 f2 f3
  \mid (F413 \mid F431 \mid F341 \mid F241 \mid F412 \mid F421) \rightarrow
        printf "%s_fgr(%s,%s,%s,%s)" f c f1 f2 f3
```

Here we have to distinguish which of the two bosons is produced in the fusion of three particles which include both fermions.

```
let print\_fermion\_g4\_vector\_current coeff\ f\ c\ wf1\ wf2\ wf3\ fusion = let c=format\_coupling\ coeff\ c and d1=d\_p\ (1,f) and d2=d\_p\ (2,f) and f1=(List.nth\ (wf\_of\_f\ wf1\ wf2\ wf3\ fusion)\ 0) and
```

```
f2 = (List.nth (wf \circ of f wf1 wf2 wf3 fusion) 1) and
       f3 = (List.nth (wf \circ of f wf1 wf2 wf3 fusion) 2) in
  match fusion with
  \mid (F123 \mid F213 \mid F132 \mid F231 \mid F312 \mid F321) \rightarrow
       printf "f_%sgr(%s,%s,%s,%s,%s)" f c f1 f2 f3
  \mid (F423 \mid F243 \mid F432 \mid F234 \mid F342 \mid F324) \rightarrow
       printf "gr_%sf(%s,%s,%s,%s)" f c f1 f2 f3
  | (F134 \mid F143 \mid F314) \rightarrow printf  "%s%s_grf(%s,%s,%s,%s)" f d1 c f1 f2 f3
   (F124 \mid F142 \mid F214) \rightarrow printf "%s%s_grf(%s,%s,%s,%s,%s)" f d2 c f1 f2 f3
   (F413 \mid F431 \mid F341) \rightarrow printf "%s%s_fgr(%s,%s,%s,%s,%s)" f d1 c f1 f2 f3
  | (F241 \mid F412 \mid F421) \rightarrow printf \text{ "%s%s_fgr(%s,%s,%s,%s)"} f d2 c f1 f2 f3
let print_fermion_2_g4_vector_current coeff f c wf1 wf2 wf3 fusion =
  let d1 = d_p(1,f) and
       d2 = d_p(2, f) and
       f1 = (List.nth (wf \circ of f wf1 wf2 wf3 fusion) 0) and
       f2 = (List.nth (wf \circ of f wf1 wf2 wf3 fusion) 1) and
       f3 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 2) in
  let c = format\_coupling\_2 \ coeff \ c in
  let c1 = fastener c 1 and
       c2 = fastener \ c \ 2 \ in
  match fusion with
  \mid (F123 \mid F213 \mid F132 \mid F231 \mid F312 \mid F321) \rightarrow
       printf "f_%sgr(%s,%s,%s,%s,%s)" f c1 c2 f1 f2 f3
  \mid (F423 \mid F243 \mid F432 \mid F234 \mid F342 \mid F324) \rightarrow
       printf "gr_%sf(%s,%s,%s,%s,%s)" f c1 c2 f1 f2 f3
    (F134 \mid F143 \mid F314) \rightarrow printf "%s%s_grf(%s,%s,%s,%s,%s,%s)" f d1 c1 c2 f1 f2 f3
    (F413 \mid F431 \mid F341) \rightarrow printf "%s%s_fgr(%s,%s,%s,%s,%s,%s)" f d1 c1 c2 f1 f2 f3
    (F241 \mid F412 \mid F421) \rightarrow printf "%s%s_fgr(%s,%s,%s,%s,%s,%s)" f d2 c1 c2 f1 f2 f3
\label{let:print_fermion_g4_vector_current_rev} \ \ coeff \ f \ \ c \ \ wf1 \ \ wf2 \ \ wf3 \ \ fusion \ =
  let c = format\_coupling coeff c and
       d1 = d_p(1,f) and
       d2 = d_p(2, f) and
       f1 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 0) and
       f2 = (List.nth (wf \circ of f wf1 wf2 wf3 fusion) 1) and
       f3 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 2) in
  match fusion with
  \mid (F123 \mid F213 \mid F132 \mid F231 \mid F312 \mid F321) \rightarrow
       printf "gr_%sf(%s,%s,%s,%s)" f c f1 f2 f3
  \mid (F423 \mid F243 \mid F432 \mid F234 \mid F342 \mid F324) \rightarrow
       printf "f_%sgr(%s,%s,%s,%s)" f c f1 f2 f3
  | (F134 \mid F143 \mid F314) \rightarrow printf \text{ "%s%s_fgr(%s,%s,%s,%s)"} f d1 c f1 f2 f3
   (F124 \mid F142 \mid F214) \rightarrow printf "%s%s_fgr(%s,%s,%s,%s,%s)" f d2 c f1 f2 f3
   (F413 \mid F431 \mid F341) \rightarrow printf "%s%s_grf(%s,%s,%s,%s,%s)" f d1 c f1 f2 f3
  | (F241 \mid F412 \mid F421) \rightarrow printf  "%s%s_grf(%s,%s,%s,%s)" f d2 c f1 f2 f3
let print_fermion_2_g4_current_rev coeff f c wf1 wf2 wf3 fusion =
  let c = format\_coupling\_2 \ coeff \ c in
  let c1 = fastener c 1 and
       c2 = fastener \ c \ 2 and
       d1 = d_{-}p(1, f) and
       d2 = d_{-}p(2,f) in
  let f1 = (List.nth (wf \circ f \circ f wf1 wf2 wf3 fusion) 0) and
       f2 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 1) and
       f3 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 2) in
  match fusion with
  \mid (F123 \mid F213 \mid F132 \mid F231 \mid F312 \mid F321) \rightarrow
       printf "gr_%sf(%s,%s,%s,%s,%s)" f c1 c2 f1 f2 f3
  \mid (F423 \mid F243 \mid F432 \mid F234 \mid F342 \mid F324) \rightarrow
       printf "f_%sgr(-(%s),-(%s),%s,%s,%s)" f c1 c2 f1 f2 f3
  \mid (F134 \mid F143 \mid F314) \rightarrow
```

```
printf "%s%s_fgr(-(%s),-(%s),%s,%s,%s)" f d1 c1 c2 f1 f2 f3
  \mid (F124 \mid F142 \mid F214) \rightarrow
       printf "%s%s_fgr(-(%s),-(%s),%s,%s,%s)" f d2 c1 c2 f1 f2 f3
    (F413 \mid F431 \mid F341) \rightarrow
       printf "%s%s_grf(%s,%s,%s,%s,%s)" f d1 c1 c2 f1 f2 f3
  | (F241 | F412 | F421) \rightarrow
       printf "%s%s_grf(%s,%s,%s,%s,%s)" f d2 c1 c2 f1 f2 f3
let print\_fermion\_2\_g4\_vector\_current\_rev coeff f c wf1 wf2 wf3 fusion =
  (* Here we put in the extra minus sign from the coeff. *)
  let c = format\_coupling coeff c in
  let c1 = fastener \ c \ 1 and
       c2 = fastener \ c \ 2 \ in
  let d1 = d_p(1,f) and
       d2 = d_p(2, f) and
       f1 = (List.nth (wf \circ of f wf1 wf2 wf3 fusion) 0) and
       f2 = (List.nth (wf \circ of f wf1 wf2 wf3 fusion) 1) and
       f3 = (List.nth (wf \_of \_f wf1 wf2 wf3 fusion) 2) in
  match fusion with
  \mid (F123 \mid F213 \mid F132 \mid F231 \mid F312 \mid F321) \rightarrow
       printf "gr_%sf(%s,%s,%s,%s,%s)" f c1 c2 f1 f2 f3
  \mid (F423 \mid F243 \mid F432 \mid F234 \mid F342 \mid F324) \rightarrow
       printf "f_%sgr(%s,%s,%s,%s,%s)" f c1 c2 f1 f2 f3
    (F134 \mid F143 \mid F314) \rightarrow printf "%s%s_fgr(%s,%s,%s,%s,%s,%s)" f d1 c1 c2 f1 f2 f3
    (F124 \mid F142 \mid F214) \rightarrow printf "%s%s_fgr(%s,%s,%s,%s,%s,%s)" f d2 c1 c2 f1 f2 f3
    (\mathit{F413} \mid \mathit{F431} \mid \mathit{F341}) \ \rightarrow \ \mathit{printf} \ \texttt{"%s\%s\_grf(\%s,\%s,\%s,\%s,\%s,\%s)"} \ \mathit{f} \ \mathit{d1} \ \mathit{c1} \ \mathit{c2} \ \mathit{f1} \ \mathit{f2} \ \mathit{f3}
    \mathsf{let}\ \mathit{print\_current\_g4}\ =\ \mathsf{function}
     coeff, Gravbar, S2, \_ \rightarrow print\_fermion\_g4\_current coeff "s2"
     coeff, Gravbar, SV, \_ \rightarrow print\_fermion\_g4\_vector\_current coeff "sv"
     coeff, Gravbar, SLV, \_ \rightarrow print\_fermion\_g4\_vector\_current coeff "slv"
     coeff, Gravbar, SRV, \_ \rightarrow print\_fermion\_g4\_vector\_current coeff "srv"
     coeff, Gravbar, SLRV, _ → print_fermion_2_g4_vector_current coeff "slrv"
     coeff,\ Gravbar,\ PV,\ \_\ 	o \ print\_fermion\_g4\_vector\_current\ coeff\ "pv"
     coeff, Gravbar, V2, \_ \rightarrow print\_fermion\_g4\_current coeff "v2"
     coeff, Gravbar, V2LR, \rightarrow print\_fermion\_2\_g4\_current coeff "v2lr"
     _, Gravbar, _, _ → invalid_arg "print_current_g4: |not| implemented"
     coeff, _, S2, Grav \rightarrow print\_fermion\_g4\_current\_rev coeff "s2"
     coeff, \_, SV, Grav \rightarrow print\_fermion\_g4\_vector\_current\_rev (-coeff) "sv"
     coeff, _, SLV, Grav \rightarrow print\_fermion\_g4\_vector\_current\_rev (-coeff) "slv"
     coeff, _, SRV, Grav \rightarrow print\_fermion\_g4\_vector\_current\_rev (-coeff) "srv"
     coeff, _, SLRV, Grav \rightarrow print\_fermion\_2\_g4\_vector\_current\_rev coeff "slrv"
     coeff, \_, PV, Grav \rightarrow print\_fermion\_g4\_vector\_current\_rev coeff "pv"
     coeff, _, V2, Grav \rightarrow print\_fermion\_g4\_vector\_current\_rev coeff "v2"
     coeff, _, V2LR, Grav → print_fermion_2_g4_current_rev coeff "v2lr"
     coeff, _, S2, _ \rightarrow print\_fermion\_s2\_current coeff "s"
     coeff, _, P2, _ \rightarrow print\_fermion\_s2\_current coeff "p"
     coeff, _, S2P, _ \rightarrow print\_fermion\_s2p\_current coeff "sp"
     coeff, \_, S2L, \_ \rightarrow print\_fermion\_s2\_current coeff "sl"
     coeff, _, S2R, _ \rightarrow print\_fermion\_s2\_current coeff "sr"
     coeff, _, S2LR, _ \rightarrow print\_fermion\_s2lr\_current coeff "slr"
     coeff, \_, V2, \_ \rightarrow print\_fermion\_g4\_brs\_vector\_current coeff "v2"
     coeff, \_, SV, \_ \rightarrow print\_fermion\_g4\_brs\_vector\_current coeff "sv"
     coeff, \_, PV, \_ \rightarrow print\_fermion\_g4\_brs\_vector\_current coeff "pv"
     coeff, _, SLV, _ \rightarrow print\_fermion\_q4\_brs\_vector\_current coeff "svl"
     coeff, \_, SRV, \_ \rightarrow print\_fermion\_g4\_brs\_vector\_current coeff "svr"
     coeff, _, SLRV, _ \rightarrow print\_fermion\_g4\_svlr\_current coeff "svlr"
    \_, \_, V2LR, \_ \rightarrow invalid\_arg "Targets.print_current:\_not\_available"
let reverse\_braket vintage braket =
  if vintage then
```

```
false
      else
         match bra, ket with
          {\it Majorana}, {\it Majorana} :: \_ \to {\sf true}
         \mid \ \_, \ \_ \ 	o \ \mathsf{false}
    let use_module = "omega95_bispinors"
    let require\_library =
       ["omega_bispinors_2010_01_A"; "omega_bispinor_cpls_2010_01_A"]
module\ Fortran\_Majorana\ =\ Make\_Fortran(Fortran\_Majorana\_Fermions)
                                                 FORTRAN 77
module Fortran 77 = Dummy
                                                 15.4.3 C
module C = Dummy
                                                     C++
module Cpp = Dummy
                                                     Java
module Java = Dummy
                                             15.4.4 O'Caml
module \ Ocaml = Dummy
                                              15.4.5 ₽T<sub>E</sub>X
module LaTeX = Dummy
                             15.5 Interface of Targets_Kmatrix
\mathsf{module}\ Fortran\ :\ \mathsf{sig}\ \mathsf{val}\ print\ :\ bool \to\ unit\ \mathsf{end}
                         15.6 Implementation of Targets_Kmatrix
module Fortran =
  struct
    open Format
    let nl = print\_newline
Special functions for the K matrix approach. This might be generalized to other functions that have to have
access to the parameters and coupling constants. At the moment, this is hardcoded.
    let print pure_functions =
      let pure =
         if pure\_functions then
           "pure<sub>□</sub>"
```

```
else
     "" in
printf "_{\sqcup \sqcup}! !! !_{\sqcup}Special_{\sqcup}K_{\sqcup}matrix_{\sqcup}functions"; nl ();
nl();
printf "_{\sqcup \sqcup} \% sfunction_{\sqcup} width_{res_{\sqcup}} (z,res,w_wkm,m,g)_{\sqcup} result_{\sqcup}(w) " pure; nl ();
printf "____real(kind=default),_intent(in)_::__z,_w_wkm,_m,_g"; nl ();
printf "____integer,_intent(in)_::_res"; nl ();
printf "____real(kind=default)_::_w"; nl();
printf "_{\square \square \square \square \square \square} if_{\square}(z.eq.0_{\square}.AND._{\square}w_{wkm}.eq.0_{\square})_{\square}then"; nl ();
printf "_____w_=_0"; nl ();
printf "____else"; nl ();
printf "_____if__(w_wkm.eq.0)_then"; nl ();
printf "_____select_case_(res)"; nl ();
printf "_____case__(1)__!!!__Scalar__isosinglet"; nl ();
printf "_____w_=_3.*g**2/32./Pi_*_m**3/vev**2"; nl ();
printf "____case__(2)_!!!_Scalar_isoquintet"; nl ();
printf "____w=_g**2/64./Pi_*_m**3/vev**2"; nl ();
printf "_____case__(3)__!!!__Vector__isotriplet"; nl();
printf "____w_=_g**2/48./Pi_*_m"; nl ();
printf ~\texttt{"uuuuuuuuucase}_{\square}(4)_{\square} \texttt{!!!}_{\square} \texttt{Tensor}_{\square} \texttt{isosinglet"}; ~nl~();
printf "____w=_g**2/320./Pi_*_m**3/vev**2"; nl ();
printf "_____case__(5)_!!!_Tensor__isoquintet"; nl();
printf "____w=_g**2/1920./Pi_*_m**3/vev**2"; nl ();
printf "_____case_default"; nl ();
printf "_____w_=_0"; nl();
printf "_____end_select"; nl();
printf "_____else"; nl ();
printf "_____w_wkm"; nl();
printf "____end__if"; nl ();
printf "LULULUL end if"; nl();
printf "⊔⊔end⊔function⊔width_res"; nl ();
nl();
printf "___%sfunction_s0stu_(s,_m)_result_(s0)" pure; nl();
printf " \_ \_ \_ \_ \_ \_ \_ \_ = 1  (kind=default), \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ = 1  ();
printf "____real(kind=default)_::_\s0"; nl();
printf "_____if__(m.ge.1.0e08)_then"; nl ();
printf "_____s0_=_0"; nl ();
printf "____else"; nl ();
printf "_{\Box \Box \Box \Box \Box \Box \Box \Box S} 0_{\Box = \Box m * * 2_{\Box} - \Box S} / 2_{\Box} +_{\Box} m * * 4 / s_{\Box} *_{\Box} \log(m * * 2 / (s + m * * 2)) "; nl ();
printf " \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup = nd \sqcup if "; nl ();
printf "\sqcup \sqcupend\sqcupfunction\sqcups0stu"; nl();
nl();
printf "_{\sqcup \sqcup} \%sfunction_{\sqcup} s1stu_{\sqcup} (s,_{\sqcup} m)_{\sqcup} result_{\sqcup} (s1) " pure; nl ();
printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup}real(kind=default)_{\sqcup}::_{\sqcup}s1"; nl();
printf "_\_\i\frac{1}{\lumber} if_\(\mu\) (m.ge.1.0e08)\(\lumber\) then"; nl ();
printf "____s1_=_0"; nl ();
printf "ullullelse"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box\Box} s1_{\Box} = _{\Box} 2*m**4/s_{\Box} + _{\Box} s/6_{\Box} + _{\Box} m**4/s**2*(2*m**2+s)_{\Box} \&"; nl();
printf "_____*_log(m**2/(s+m**2))"; nl ();
printf "____end__if"; nl();
printf "\sqcup \sqcupend\sqcupfunction\sqcups1stu"; nl();
printf "___%sfunction_s2stu_(s,_m)_result_(s2)" pure; nl();
printf "____real(kind=default),_intent(in)_::__s,_m"; nl ();
printf "_\( \subset \) real(kind=default)\( \subset : \( \subset \) s2"; nl ();
printf "_____if_(m.ge.1.0e08)_then"; nl();
printf "_____s2__=_0"; nl ();
printf "ullullelse"; nl ();
```

```
printf "_____s2__=_m**4/s**2_*_(6*m**2_+_3*s)__+_&"; nl();
printf "_____m**4/s**3_*_(6*m**4_+_6*m**2*s_+_s**2)_&"; nl();
printf "_____*_log(m**2/(s+m**2))"; nl ();
printf "____end__if"; nl ();
printf "_{\sqcup \sqcup} end_{\sqcup} function_{\sqcup} s2stu"; nl();
nl();
printf "_{\sqcup}!!_{\sqcup}%sfunction_{\sqcup}s3stu_{\sqcup}(s,_{\sqcup}m)_{\sqcup}result_{\sqcup}(s3)" pure; nl ();
printf "u!!!uuuureal(kind=default),uintent(in)u::us,um"; nl ();
printf "_{\sqcup}!!_{\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup} real(kind=default)_{\sqcup}::_{\sqcup} s3"; nl ();
printf "_{\sqcup}!!_{\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup} if_{\sqcup} (m.ge.1.0e08)_{\sqcup} then"; nl ();
printf "¬!!¬¬¬¬, nl ();
printf " "!! " = else"; nl ();
printf "_{\sqcup}!!_{\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup}3_{\sqcup}=_{m}**4/s**3_{\sqcup}*_{\sqcup}(60*m**4_{\sqcup}+_{\sqcup}60*m**2*s+11*s**2)_{\sqcup}+_{\sqcup}\&"; nl();
printf ""!!
printf "_{\sqcup}!!_{\sqcup \sqcup \sqcup} *_{\sqcup} log(m**2/(s+m**2))"; nl ();
printf "_{\sqcup}!!_{\sqcup\sqcup\sqcup\sqcup\sqcup} end_{\sqcup}if"; nl ();
printf "_{\sqcup}!!_{\sqcup\sqcup}end_{\sqcup}function_{\sqcup}s3stu"; nl();
nl();
printf "□□\%sfunction\p0stu\(s,\m)\presult\(p0)\)" pure; nl ();
printf " \_ \_ \_ \_ \_ \_ \_ \_ = 1  (kind=default), \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ = 1  ();
printf "□□□□□□real(kind=default)□::□p0"; nl ();
printf "_____if__(m.ge.1.0e08)_then"; nl ();
printf "_____p0_=_0"; nl ();
printf "____else"; nl ();
printf " \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup = nd \sqcup if "; nl ();
printf "\sqcup \sqcup end \sqcup function \sqcup p0stu"; nl();
nl();
printf "⊔⊔%sfunction⊔p1stu⊔(s,⊔m)⊔result⊔(p1)" pure; nl ();
printf "____real(kind=default),_intent(in)_::__s,_m"; nl ();
printf "____real(kind=default)_::_p1"; nl ();
printf "_____if__(m.ge.1.0e08)_then"; nl ();
printf "____p1_=_0"; nl ();
printf "____else"; nl ();
printf "_{\Box \Box \Box \Box \Box \Box \Box} p1_{\Box = \Box} (m**2_{\Box} + _{\Box}2*s)/s**2_{\Box}*_{\Box} (2*s + (2*m**2 + s)_{\Box}\&"; nl();
printf "______*_log(m**2/(s+m**2)))"; nl ();
printf " uuuuuuu end uif"; nl ();
printf "\sqcup \sqcupend\sqcupfunction\sqcupp1stu"; nl();
nl();
printf "___%sfunction_d0stu_(s,_m)_result_(d0)" pure; nl();
printf "____real(kind=default),_intent(in)_::__s,_m"; nl ();
printf "□□□□□□real(kind=default)□::□d0"; nl ();
printf "_____d0_=_0"; nl ();
printf "____else"; nl ();
printf "_____d0__=_(2*m**2+11*s)/2_+_(m**4+6*m**2*s+6*s**2)_&"; nl();
printf "_____/s_*_log(m**2/(s+m**2))"; nl ();
printf "____end__if"; nl ();
printf "_{\sqcup \sqcup} end_{\sqcup} function_{\sqcup} d0stu"; nl();
nl();
printf "_{\sqcup\sqcup} %sfunction_{\sqcup} d1stu_{\sqcup} (s,_{\sqcup} m)_{\sqcup} result_{\sqcup} (d1) " pure; nl ();
printf " \_ \_ \_ \_ \_ \_ \_ \_ = 1  (kind=default), \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ = 1  ();
printf "____real(kind=default)_::_d1"; nl();
printf "_____if_ (m.ge.1.0e08)_then"; nl ();
printf "____d1_=_0"; nl ();
printf "____else"; nl ();
printf "_____d1_=_(s*(12*m**4_+_72*m**2*s_+_73*s**2)_&"; nl();
printf "_{\Box\Box\Box\Box\Box\Box\Box\Box\Box\Box\Box\Box} +_{\Box} 6*(2*m**2_{\Box} +_{\Box} 8)*(m**4_{\Box} +_{\Box} 6*m**2*s_{\Box} +_{\Box} 6*s**2)_{\triangle}"; nl();
printf "_____*_log(m**2/(s+m**2)))/6/s**2"; nl ();
printf " \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup = nd \sqcup if "; nl ();
printf "\sqcup \sqcupend\sqcupfunction\sqcupd1stu"; nl();
```

```
nl();
printf "_{\sqcup\sqcup}\%sfunction_{\sqcup}da00_{\sqcup}(cc,_{\sqcup}s,_{\sqcup}m)_{\sqcup}result_{\sqcup}(amp_{\perp}00)" pure; nl();
printf "⊔⊔⊔⊔⊔real(kind=default), intent(in) :: us"; nl ();
printf "_____real(kind=default),_dimension(1:12),_intent(in)_::_cc"; nl ();
printf "\( \sum \) complex(\( \) kind=\( \) default\( \) \( \) : \( \) a00_0, \( \) a00_1, \( \) a00_a, \( \) a00_f"; \( nl \) ();
printf "_\undersigned complex(kind=default),\undersigned dimension(1:7)\undersigned::\undersigned a00"; nl ();
printf "____complex(kind=default)_::_ii,_jj,_amp_00"; nl ();
printf "____real(kind=default)_::_kappal,_kappam,_kappat"; nl();
printf "_____ii_=_cmplx(0.0,1.0/32.0/Pi,default)"; nl ();
printf "____jj_=_s**2/vev**4*ii"; nl ();
printf "_{ \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup } kappal_{ \sqcup = \sqcup } cc(12)*((mass(23)**2+mass(24)**2)/m(4)**2-2*mass(23)**2*mass(24)**2/m(4)**4)
printf "_{ \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup } kappam_{ \sqcup = \sqcup } cc(12)*((mass(23)**4+mass(24)**4)/m(4)**2/(mass(23)**2+mass(24)**2)_{ \& "}; nl();
printf ~~ \verb|"ullull| \verb| kappat| = \verb| ucc(12)*mass(23)**2*mass(24)**2/m(4)**4"; ~nl~(); \\
printf "____!!!_Longitudinal"; nl ();
printf "____!!!_Scalar_isosinglet"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box} a00(1)_{\Box} =_{\Box} -2.0_{\Box} *_{\Box} cc(1) **2/vev **2_{\Box} *_{\Box} s0stu(s,m(1))_{\Box}"; nl();
printf "_____if_ (cc(1)_/=_0)_then"; nl ();
printf "\verb| "uuuuuuuuuuus**2/cmplx(s-m(1)**2,m(1)*wkm(1),default)| "|; nl(); 
printf "____end__if"; nl ();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup}! !!_{\sqcup}Scalar_{\sqcup}isoquintet"; nl ();
printf "_____a00(2)_=_-5.0*cc(2)**2/vev**2_*_s0stu(s,m(2))_/_3.0"; nl();
printf "⊔⊔⊔⊔⊔⊔!!!⊔Vector⊔isotriplet"; nl ();
printf "_____!!!__Tensor__isosinglet"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box\Box} a00(4)_{\Box} =_{\Box} -cc(4)**2/vev**2/3_{\Box}*_{\Box} (d0stu(s,m(4))_{\Box}\&"; nl(); 
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} if_{\sqcup}(_{\sqcup}(cc(4)_{\sqcup}/=_{\sqcup}0).and.(kappal_{\sqcup}/=_{\sqcup}0))_{\sqcup}then"; nl();
printf "_{\Box\Box\Box\Box\Box\Box\Box\Box}a00(4)_{\Box} =_{\Box}a00(4)_{\Box} -_{\Box}cc(4)**2/vev**2*kappal_{\Box}*_{\Box}\&"; nl();
printf "______s**2/cmplx(s-m(4)**2,m(4)_*_wkm(4),default)"; nl();
printf "____end__if"; nl ();
printf "____!!!_Tensor_isoquintet"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box}a00(5)_{\Box} =_{\Box} -5.0*cc(5)**2/vev**2*(d0stu(s,m(5))_{\Box}\&"; nl();
printf "_____/3.0)/6.0"; nl ();
printf "_{\square \square \square \square \square \square}!!!_{\square} Transversal"; nl ();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup}! !!_{\sqcup} Tensor_{\sqcup} isosinglet"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box\Box} a00(6)_{\Box} = -\Box cc(9)**2/Pi/vev**6*mass(23)**2*mass(24)**2/4*_\s**2_\&"; nl();
printf "_______*__((2-2*s/m(4)**2+s**2/m(4)**4)+kappat/2__)"; nl ();
printf "_\underset \( \alpha \) (a00(6)\underset /=\underset 0)\underset \( \text{then''}; \ nl \( \);
printf "____a00(6)/cmplx(s-m(4)**2,_-\w_res/32/Pi_*\real(a00(6),default),default)_"; nl "..."; nl "..."
printf "____end__if"; nl ();
printf "______*_(3*(1+2*s/m(4)**2+2*s**2/m(4)**4)+kappat__))"; nl ();
printf " \square \square \square \square \square ! ! ! \square Mixed"; nl ();
printf "____!!!_Tensor_isosinglet"; nl ();
printf "\verb||| uu || uu 
printf "______*\text{u_((1-4*s/m(4)**2+2*s**2/m(4)**4)+kappam__)"}; nl ();
printf "_\_\frac{1}{2}(a00(7)_\frac{1}{2}=0)_\frac{1}{2}then"; nl ();
printf ~\texttt{"}_{\square\square\square\square\square\square\square\square\square} a00(7) = \texttt{"}_{2}a00(7) / \texttt{cmplx} (\texttt{s-m}(4)**2, \texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt{"}_{2}\texttt
printf " \cup \cup \cup \cup \cup \cup end \cup if"; nl ();
printf ~\texttt{"}_{ \square \square \square \square \square \square \square} a00(7)_{ \square} = \texttt{\_a00}(7)_{ \square} - \texttt{\_cc}(11) * \texttt{cc}(9) * \texttt{cc}(4) / \texttt{Pi/vev} * * 4 * (mass(23) * * 2 + mass(24) * * 2) / 12_{ \square} *_{ \square} (s0stu(3) + s2)_{ \square} (s
printf "_____*_(12*s/m(4)**2+12*s**2/m(4)**4+2*kappam_))"; nl ();
printf "____!!!_Fudge-Higgs"; nl ();
printf "\( \subseteq \text{1.1}\) a00_f \( \subseteq \text{1.2}\) .*fudge_higgs*s/vev**2"; nl();
printf "____!!!_Low_energy_theory_alphas"; nl ();
printf "____a00_0_=_8.*(7.*a4_+_11.*a5)/3.*s**2/vev**4"; nl ();
printf "____a00_1_=_(25.*log(lam_reg**2/s)/9_+_11./54.0_default)*s**2/vev**4"; nl ();
printf "____a00_a_=__a00_0_!!!!_+a00_1/16./Pi**2"; nl ();
```

```
printf " \cup \cup \cup \cup \cup ! !! \cup Unitarize "; nl ();
printf "_{\square \square \square \square \square \square} if_{\square} (fudge_km_{\square}/=_{\square}0)_{\square} then"; nl ();
printf "____amp_00_=_sum(a00)+a00_f+a00_a"; nl();
printf "_____if__(amp_00_\/=\_0)_\then"; nl ();
printf "_____end__if"; nl();
\textit{printf "$$} \verb""lullelese"; nl ();
printf "_\underset = \underset amp_00\underset = \underset (1-part_r)\underset *\underset sum (a00)\underset +\underset part_r\underset *\underset a00(3)"; nl ();
printf "____end__if"; nl();
printf "____amp_00_=_vev**4/s**2_*amp_00"; nl ();
printf "\sqcup \sqcupend\sqcupfunction\sqcupda00"; nl();
nl();
printf "_{\sqcup \sqcup} %sfunction_{\sqcup} da02_{\sqcup} (cc,_{\sqcup}s,_{\sqcup}m)_{\sqcup} result_{\sqcup} (amp_02) " pure; nl ();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} real(kind=default), \_intent(in)_{\sqcup} :: \_s"; nl();
printf "_____real(kind=default),_dimension(1:12),_intent(in)_::_cc"; nl ();
printf "\( \subset \) real(\( \) kind=default \), \( \) dimension(1:5), \( \) intent(\( \) in \( \); \( nl \) ();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} complex(kind=default), _ dimension(1:7)_ :: _ \( a02"; nl (); 
printf "____complex(kind=default)_::_ii,_jj,_amp_02"; nl ();
printf "____real(kind=default)_::_kappal,_kappam,_kappat"; nl();
printf "____ii_=_cmplx(0.0,1.0/32.0/Pi,default)"; nl();
printf "____jj_=_s**2/vev**4*ii"; nl ();
printf "_{UUUUUU} kappal_{U} = _{U} cc(12) * ((mass(23)**2+mass(24)**2)/m(4)**2-2*mass(23)**2*mass(24)**2/m(4)**4)
printf "_{UUUUUU}kappam_{U}=_{U}cc(12)*((mass(23)**4+mass(24)**4)/m(4)**2/(mass(23)**2+mass(24)**2)_{U}%"; nl();
printf "_____2*mass(23)**2*mass(24)**2/m(4)**4)"; nl ();
printf "\( \subseteq \text{LILILICK} \) kappat\( \subseteq \subseteq \text{c(12)*mass(23)**2*mass(24)**2/m(4)**4"}; \ nl\( \);
printf "____!!!_Longitudinal"; nl ();
printf "____!!!_Scalar___isosinglet"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box} a02(1)_{\Box} =_{\Box} -2.0*cc(1)**2/vev**2_{\Box}*_{\Box}s2stu(s,m(1))"; nl();
printf "⊔⊔⊔⊔⊔!!!!⊔Scalar⊔isoquintet"; nl ();
printf = -5.0*cc(2)**2/vev**2_*s2stu(s,m(2))_1/s3.0"; nl();
printf "____!!!_Vector_isotriplet"; nl();
printf "_{\Box\Box\Box\Box\Box\Box\Box} a02(3)_{\Box = \Box} -4.0*cc(3)**2*(2*s+m(3)**2)*s2stu(s,m(3))/m(3)**4"; nl();
printf "____!!!_Tensor_isosinglet"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box} a02(4)_{\Box=\Box} -_{\Box} cc(4)**2/vev**2/3_{\Box}*_{\Box\Box} \&"; nl();
printf "_______(1.+6.*s/m(4)**2+6.*s**2/m(4)**4)-2*kappal)_*_s2stu(s,m(4))"; nl();
printf "_____if__(cc(4)_/=_0)_then"; nl();
printf "_\underset end\underset if"; nl ();
printf "____!!!_Tensor_isoquintet"; nl ();
printf "_____)/6.0"; nl ();
printf ~" \verb| " \verb| | \verb| | \verb| | \verb| | \verb| | Transversal"; ~ nl ~ ();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup}! !!_{\sqcup} Tensor_{\sqcup} isosinglet"; nl ();
printf "_{\cup\cup\cup\cup\cup\cup\cup} a02(6)_{\cup} =_{\cup} -_{\cup} cc(9) **2/Pi/vev **6*mass(23) **2*mass(24) **2/40*_{\cup} s **2"; nl ();
printf "____end__if"; nl ();
printf ~" \verb| | \verb| | \verb| | \verb| | (3*(1+2*s/m(4)**2+2*s**2/m(4)**4) + \verb| | kappat_{\square}))"; ~ nl ~ (); \\
printf "____!!!_Mixed"; nl ();
printf "____!!!_Tensor_isosinglet"; nl ();
printf "_{UUUUUU} = 02(7)_{U} = _{U} -_{U}cc(11)*cc(9)*cc(4)/Pi/vev**4*(mass(23)**2+mass(24)**2)/20_{u}"; nl();
printf "_____*_s**2"; nl ();
printf "_\underset \( \alpha \) (a02(7)\underset /=\underset 0)\underset \( \text{then} \); nl ();
printf "____a02(7)/cmplx(s-m(4)**2,_-\w_res/32/Pi_*\real(a02(7),default),default)_"; nl nl
printf "_____end__if"; nl();
printf ~ \texttt{"}_{ \square \square \square \square \square \square \square} a02(7)_{ \square} = \texttt{\_a}02(7)_{ \square} - \texttt{\_c}c(11) * cc(9) * cc(4) / \texttt{Pi/vev} * 4 * (mass(23) * * 2 + mass(24) * * 2) / 12_{ \square} *_{ \square} (s2stu(3) * 2) / (s2stu(3) * 2) 
printf "_____*_(12*s/m(4)**2+12*s**2/m(4)**4+2*kappam__))"; nl ();
```

```
printf "____!!!_Low_energy_theory_alphas"; nl();
printf "____a02_0_=_(8.*(2.*a4_+a5)/15.)_*_s**2/vev**4"; nl ();
printf "⊔⊔⊔⊔⊔⊔a02_1⊔=⊔(log(lam_reg**2/s)/9.⊔-⊔7./135.0_default)⊔*⊔∪s**2/vev**4"; nl ();
printf "____a02_a_=_a02_0_!!!!_+a02_1/16/Pi**2"; nl ();
printf "_{\square \square \square \square \square \square}!!!_{\square}Unitarize"; nl ();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} if_{\sqcup} (fudge_km_{\sqcup}/=_{\sqcup} 0)_{\sqcup} then"; nl ();
printf "_\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd\u00cd
printf "____amp_02_=_-_a02_a_-_part_r_*_(sum(a02)_-_a02(3))_+_1/(real(1/amp_02,default)-ii)";
printf "____end__if"; nl ();
printf "____else"; nl ();
printf ~\texttt{"} {$\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} \\ \texttt{amp} {$\sqcup 02 \sqcup = \sqcup (1-part\_r) \sqcup * \sqcup sum(a02) \sqcup + \sqcup part\_r \sqcup * \sqcup a02(3)$"}; ~ nl ~ (); \\ \texttt{and} ~\texttt{and} 
printf "____end__if"; nl ();
printf "_\uu_\uap_02\u=\uvev**4/s**2\u*\amp_02\up"; nl ();
printf "\sqcup \sqcup end \sqcup function \sqcup da02"; nl();
printf "_{\sqcup \sqcup}%sfunction_da11_(cc,_\s,_\mu\)_result_\(\text{amp}_11\)" pure; nl();
printf "⊔⊔⊔⊔⊔real(kind=default), intent(in) :: us"; nl ();
printf "_____real(kind=default),_dimension(1:12),_intent(in)_::__cc"; nl ();
printf "_{\square \square \square \square \square \square}real(kind=default), _{\square}dimension(1:5), _{\square}intent(in)_{\square}::_{\square}m"; nl();
printf "____complex(kind=default)_::_a11_0,_a11_1,_a11_a,_a11_f"; nl();
printf "____complex(kind=default)_::_ii,_jj,_amp_11"; nl();
printf "_\underset real(kind=default)\underset : \underset kappal,\underset kappam,\underset kappat "; nl ();
printf "____jj_=_s**2/vev**4*ii"; nl ();
printf ~" \_ \_ \_ cc(12) * ((mass(23)**2+mass(24)**2)/m(4)**2-2*mass(23)**2*mass(24)**2/m(4)**4) + (mass(23)**2+mass(24)**2)/m(4)**2-2*mass(23)**2*mass(24)**2/m(4)**4) + (mass(23)**2+mass(24)**2)/m(4)**2-2*mass(23)**2*mass(24)**2/m(4)**2-2*mass(24)**2/m(4)**2-2*mass(24)**2/m(4)**2-2*mass(24)**2/m(4)**2-2*mass(24)**2/m(4)**2-2*mass(24)**2/m(4)**2-2*mass(24)**2/m(4)**2-2*mass(24)**2/m(4)**2/m(4)**2-2*mass(24)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/m(4)**2/
printf "_{UUUUUU}kappam_{U}=_{U}cc(12)*((mass(23)**4+mass(24)**4)/m(4)**2/(mass(23)**2+mass(24)**2)_{U}"; nl();
printf " = cc(12) * mass(23) * 2* mass(24) * 2/m(4) * 4"; nl();
printf "_{\square \square \square \square \square \square}!!!_{\square}Longitudinal"; nl();
printf "____!!!_Scalar_isosinglet"; nl ();
printf "_{ \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} a11(1)_{ \sqcup = \sqcup - \sqcup} 2.0*cc(1)**2/vev**2_{ \sqcup * \sqcup} s1stu(s,m(1))"; nl();
printf "____!!!_Scalar_isoquintet"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box} a11(2)_{\Box} = _{\Box} 5.0*cc(2)**2/vev**2_{\Box} *_{\Box} s1stu(s,m(2))_{\Box}/_{\Box} 6.0"; nl();
printf "____!!!_Vector_isotriplet"; nl();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} a11(3)_{\sqcup = \sqcup - \sqcup} cc(3)**2_{\sqcup} *_{\sqcup} \&"; nl();
printf "______(s/m(3)**2_+__2.__*_p1stu(s,m(3)))"; nl();
printf "_{\square \square \square \square \square \square} if_{\square}(cc(3)_{\square}/=_{\square}0)_{\square} then"; nl ();
printf "____a11(3)__=_a11(3)__-2./3.__*_cc(3)**2_*_&"; nl ();
printf "_____s/cmplx(s-m(3)**2,m(3)*wkm(3),default)_"; nl();
printf "_\underset end\underset if"; nl ();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup}! !!_{\sqcup} Tensor_{\sqcup} isosinglet"; nl ();
printf "_____/3.0)"; nl ();
printf "_{\square \square \square \square \square \square}! !!_{\square} Tensor_{\square} isoquintet"; nl ();
printf " = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 11(5) = 1
printf "______)/36.0"; nl ();
printf "____!!!uTransversal"; nl ();
printf "____!!!._Tensor_isosinglet"; nl ();
printf ~"\_\_\_\_cc(9)**2/Pi/vev**6*mass(23)**2*mass(24)**2/12\_*\_(s1stu(s,m(4))\_*\_\&"; nl(); 
printf "_{ \cup \cup} (3*(1+2*s/m(4)**2+2*s**2/m(4)**4) + kappat_{ \cup ) \cup \neg \cup} (s/m(4)**2+s**2/m(4)**4) *s) "; nl() = (s/m(4)**2+s**2/m(4)**4) + kappat_{ \cup } (s/m(4)**4) + kappat_{ \cup }
printf "____!!!_Mixed"; nl ();
printf "____!!!_Tensor_isosinglet"; nl ();
printf = 11(7) = -cc(11)*cc(9)*cc(4)/Pi/vev**4*(mass(23)**2+mass(24)**2)/12 = -cc(11)*cc(9)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*cc(11)*c
printf "\verb|||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - ||| - 
printf "____!!!_Fudge-Higgs"; nl ();
printf "\( \subseteq \text{11_l} \) a11_f \( \subseteq \text{11_l} \) fudge_higgs*s/3./vev**2"; nl();
printf "_____!!!_Low_energy_theory_alphas"; nl ();
printf "_{ \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} a11_{ \sqcup \sqcup \sqcup} 4.*(a4_{ \sqcup \sqcup \sqcup} 2*a5)/3._{ \sqcup} *_{ \sqcup} s**2/vev**4_{ \sqcup}"; nl ();
printf "____a11_1_=_-1.0/54.0_default_*s**2/vev**4"; nl ();
```

```
printf "_{\cup \cup \cup \cup \cup \cup} a11_a_{\cup} =_{\cup} a11_0_{\cup}!!!_{\cup} +_{\cup} a11_1/16/Pi**2"; nl ();
printf " \cup \cup \cup \cup \cup ! !! \cup Unitarize "; nl ();
printf "_{\square \square \square \square \square \square} if_{\square} (fudge_km_{\square}/=_{\square}0)_{\square} then"; nl ();
printf "_{\square\square\square\square\square\square\square\square\square} amp_11_{\square} = sum(a11) + a11_f + a11_a"; nl();
printf "_____if__(amp_11__/=_0)_then"; nl();
printf "____end__if"; nl ();
printf ~"\verb||lu|| \verb||else||; ~nl ~();
printf "$_{\square\square\square\square\square\square\square\square\square\square}$ amp_11$_{\square}=$_{\square}(1-part_r)_{\square}*_{\square}sum(a11)_{\square}+_{\square}part_r_{\square}*_{\square}a11(3)"; nl ();
printf "____end__if"; nl ();
printf "\underset \underset \amp_11\underset = \underset vev**4/s**2\underset \underset \unde
printf "_{\sqcup\sqcup}end_{\sqcup}function_{\sqcup}da11"; nl();
nl();
printf ~"_{ \sqcup \sqcup} \% sfunction ~ da 20 ~ (cc, \_s, \_m) ~ Lresult ~ (amp\_20) ~" ~ pure; ~ nl ~ (); \\
printf ~" \verb| ulululu| real(kind=default), \verb| ulintent(in)| | | : | us"; ~nl~();
printf "_____real(kind=default),_dimension(1:12),_intent(in)_::_cc"; nl ();
printf "_{\square \square \square \square \square \square}real(kind=default), _{\square}dimension(1:5), _{\square}intent(in)_{\square}:: _{\square}m"; nl ();
printf "\( \subset \) \( \text{complex(kind=default)} \( \subset \) : \( \subset \) \( \alpha \) = 20_1, \( \subset \) = 20_1, \( \subset \) \( \alpha \) = 20_1, \( \subset \) \( \alpha \) = 20_1, \( \subset \) = 20_1, \( \subse
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} complex(kind=default), _ dimension(1:7)_ :: _ a20"; nl ();
printf "____complex(kind=default)_::_ii,_jj,_amp_20"; nl();
printf "_\underset real(kind=default)\underset : \underset kappal,\underset kappam,\underset kappat "; nl ();
printf ~"\verb|u|u|u|u|i|u=u|cmplx(0.0,1.0/32.0/Pi,default)";~nl~();
printf "____jj_=_s**2/vev**4*ii"; nl ();
printf "____!!!_Scalar_isosinglet"; nl ();
printf "_{UUUUUU}kappal_{U}=_{U}cc(12)*((mass(23)**2+mass(24)**2)/m(4)**2-2*mass(23)**2*mass(24)**2/m(4)**4)
printf "$_{$\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup}$ kappam$_{$\sqcup=\sqcup$cc(12)*((mass(23)**4+mass(24)**4)/m(4)**2/(mass(23)**2+mass(24)**2)$_{$\sqcup\&";\ nl\ ();$_{$\sqcup\&"$}$_{$\sqcup\&"$}.}
printf "_____2*mass(23)**2*mass(24)**2/m(4)**4)"; nl();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} kappat_{\sqcup} = _{\sqcup} cc(12) * mass(23) * * 2 * mass(24) * * 2/m(4) * * 4"; nl();
printf "____!!!_Longitudinal"; nl ();
printf ~" \_ \_ \_ = \_-2.0*cc(1)**2/vev**2 \_ * \_ s0stu(s,m(1))"; ~nl~(); ~nl~()
printf "____!!!.\Scalar_isoquintet"; nl();
printf "_{\square \square \square \square \square \square} if_{\square}(cc(2)_{\square}/=_{\square}0)_{\square} then"; nl();
printf "____a20(2)__-a20(2)__-cc(2)**2/vev**2/2._*&"; nl ();
printf "______s**2/cmplx(s-m(2)**2,m(2)*wkm(2),default)"; nl ();
printf "____end__if"; nl();
printf "____!!!_Vector_isotriplet"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box} a20(3)_{\Box} = _{\Box}cc(3)**2*(2.0*p0stu(s,m(3))_{\Box} + _{\Box}3.0*s/m(3)**2)"; nl ();
printf "____!!!__Tensor__isosinglet"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box\Box} 20(4)_{\Box=\Box-\Box} cc(4)**2/vev**2*(d0stu(s,m(4)-2*kappal*s0stu(s,m(4)))_&"; nl();
printf "_____/3.0)"; nl ();
printf "⊔⊔⊔⊔⊔!!!!⊔Tensor⊔isoquintet"; nl ();
printf "_{$\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup} a20(5)_{$\sqcup=\sqcup^-\sqcup} cc(5)**2/vev**2*(d0stu(s,m(5))_{$\sqcup\&$"}; nl();
printf "______)/36.0"; nl ();
printf "_____!!!__Transversal"; nl ();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup}! !!_{\sqcup} Tensor_{\sqcup} isosinglet"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box} a20(6)_{\Box=\Box} -cc(9)**2/Pi/vev**6*mass(23)**2*mass(24)**2/12_{\Box}*_{\Box}(sostu(s,m(4))_{\Box}\&"; nl(); nl()
printf "_{\square \square \square \square \square}!!!_{\square}Mixed"; nl ();
printf "____!!!_Tensor_isosinglet"; nl ();
printf "_{ \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup } 20(7) =_{ \sqcup } -cc(11)*cc(9)*cc(4) / Pi/vev**4* (mass(23)**2+mass(24)**2) / 12 _{ \sqcup } *_{ \sqcup } (sostu(s,m(4)) _{ \sqcup } (sostu(s
printf "$_{$\cup\cup\cup\cup\cup\cup\cup\cup\cup\cup\cup\cup\cup\cup\cup}*_{$\cup$} (12*s/m(4)**2+12*s**2/m(4)**4+2*kappam$_{$\cup$}$_{$\cup-G}*(s/m(4)**2-s**2/m(4)**4)*s)"$;
printf "____!!!_Fudge-Higgs"; nl ();
printf "____a20_f____fudge_higgs*s/vev**2"; nl ();
printf "____a20_f___a20_f___0*2*(1-ghvva)**2/vev**2*mass(25)**2"; nl ();
printf "_____!!!_Low_energy_theory_alphas"; nl ();
printf "____a20_0_=__16*(2*a4_+_a5)/3*s**2/vev**4"; nl ();
printf "____a20_a_=_a20_0_!!!!_+a20_1/16/Pi**2"; nl ();
printf "____!!!_Unitarize"; nl ();
printf "_{\square \square \square \square \square \square} if_{\square} (fudge_km_{\square}/=_{\square}0)_{\square} then"; nl ();
```

```
printf "____amp_20_=\sum(a20)+a20_f+a20_a"; nl();
printf "_____if__(amp_20_/=_00)_then"; nl();
printf \; \verb"ululululuenduif"; \; nl \; ();
printf "____else"; nl ();
printf ~" \_ \_ \_ \_ \_ = \_ (1-part\_r) \_ * \_ sum(a20) \_ + \_ part\_r \_ * \_ a20(3) "; ~nl~(); \\
printf "____end__if"; nl ();
printf "\( \sum \text{unullamp_20} \) = \( \text{vev**4/s**2} \) \( \text{amp_20"}; \ nl \( ); \)
printf "_uend_function_da20"; nl();
nl();
printf "_{\sqcup\sqcup} %sfunction_{\sqcup} da22_{\sqcup} (cc,_{\sqcup}s,_{\sqcup}m)_{\sqcup} result_{\sqcup} (amp_22) " pure; nl ();
printf " \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ = (kind=default), \_ \_ intent(in) \_ : : \_ \_ s"; nl ();
printf "_____real(kind=default),_dimension(1:12),_intent(in)_::__cc"; nl ();
printf "_____real(kind=default),_dimension(1:5),_intent(in)_::_m"; nl ();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} complex(kind=default), _{\sqcup} dimension(1:7)_{\sqcup}: _{\sqcup}a22"; nl ();
printf "____complex(kind=default)_::_ii,_jj,_amp_22"; nl ();
printf "____real(kind=default)_::__kappal,__kappam,__kappat"; nl ();
printf "____ii_=_cmplx(0.0,1.0/32.0/Pi,default)"; nl ();
printf "____jj_=_s**2/vev**4*ii"; nl ();
printf ~"\_\_\_\_cc(12)*((mass(23)**2+mass(24)**2)/m(4)**2-2*mass(23)**2*mass(24)**2/m(4)**4)
printf "_{UUUUUU}kappam_{U}=_{U}cc(12)*((mass(23)**4+mass(24)**4)/m(4)**2/(mass(23)**2+mass(24)**2)_{U}%"; nl();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} kappat_{\sqcup} = _{\sqcup} cc(12) * mass(23) * * 2 * mass(24) * * 2/m(4) * * 4"; nl();
printf "_{\square \square \square \square \square \square}!!!_{\square}Longitudinal"; nl();
printf "____!!!_Scalar_isosinglet"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box} a22(1)_{\Box} =_{\Box} -_{\Box} 2.0*cc(1)**2/vev**2_{\Box} *_{\Box} s2stu(s,m(1))"; nl();
printf "____!!!_Scalar_isoquintet"; nl ();
printf "____!!!_\Vector_\triplet"; nl ();
printf ~~ " \_ \_ \_ 2.0 * cc(3) * * 2 * (2 * s + m(3) * * 2) * s 2 s tu(s, m(3)) / m(3) * * 4 "; ~ nl~(); \\ (1); ~~ (2); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3); ~~ (3
printf "____!!!._Tensor__isosinglet"; nl ();
printf " = a22(4) =
printf "_____+6.0*s**2/m(4)**4-2*kappal)*s2stu(s,m(4))/3.0)"; nl ();
printf "⊔⊔⊔⊔⊔!!!⊔Tensor⊔isoquintet"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box}a22(5)_{\Box} =_{\Box} -_{\Box}cc(5)**2/vev**2/36._{\Box}*_{\Box}\&"; nl();
printf "______((1.+6.*s/m(5)**2+6.*s**2/m(5)**4_) _&"; nl ();
printf "_____*_s2stu(s,m(5)))"; nl ();
printf "_{ \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} if_{ \sqcup} (cc(5)_{ \sqcup}/=_{ \sqcup} 0)_{ \sqcup} then"; nl();
printf "____a22(5)__=_a22(5)__-_cc(5)**2/vev**2/60_*_&"; nl ();
printf "______s**2/cmplx(s-m(5)**2,m(5)*wkm(5),default)"; nl ();
printf "_\underset end\underset if"; nl ();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup}! !!_{\sqcup} Transversal"; nl ();
printf "_____!!!__Tensor__isosinglet"; nl ();
printf ~"\_\_\_\_cc(9)**2/Pi/vev**6*mass(23)**2*mass(24)**2/12\_*\_(s2stu(s,m(4))\_\&"; nl(); nl
printf "______*_(3*(1+2*s/m(4)**2+2*s**2/m(4)**4)+kappat__))"; nl ();
printf "____!!!_Mixed"; nl ();
printf "____!!!_Tensor_isosinglet"; nl ();
printf "_{ \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup } 22(7) =_{ \sqcup } -cc(11)*cc(9)*cc(4) / Pi/vev**4* (mass(23)**2+mass(24)**2) / 12 _{ \sqcup } *_{ \sqcup } (s2stu(s,m(4)) _{ \sqcup } (s2stu(s,m
printf "______*_(12*s/m(4)**2+12*s**2/m(4)**4+2*kappam__))"; nl ();
printf "_____!!!_Low_energy_theory_alphas"; nl ();
printf "_____a22_1_=_(2*log(lam_reg**2/s)/45_-_247/5400.0_default)*s**2/vev**4"; nl ();
printf "____a22_a_=_a22_0_!!!!_+_a22_1/16/Pi**2"; nl ();
printf " \cup \cup \cup \cup \cup ! !! \cup Unitarize "; nl ();
printf "_{\square \square \square \square \square \square} if_{\square} (fudge_km_{\square}/=_{\square}0)_{\square} then"; nl ();
printf "____amp_22_=\sum(a22)+a22_a"; nl();
printf "_____if__(amp_22__/=_0)_then"; nl();
printf "____end__if"; nl ();
printf "ullullelse"; nl ();
```

```
printf "____end__if"; nl();
printf "____amp_22_=_vev**4/s**2_*amp_22"; nl ();
printf "\sqcup \sqcupend\sqcupfunction\sqcupda22"; nl();
nl();
printf "_{\sqcup \sqcup} %sfunction_{\sqcup} dalzz0_{s_{\sqcup}}(cc,m,k)_{\sqcup} result_{\sqcup} (alzz0_{s}) " pure; nl ();
printf "_\underset \text{type} (momentum), \underset \text{intent(in)} \underset : \underset \text{k"}; nl ();
printf "_\_\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightar
printf "____real(kind=default),_dimension(1:5),_intent(in)_::_m"; nl ();
printf "____complex(kind=default)_::_alzz0_s"; nl();
printf "□□□□□□real(kind=default)□::□s"; nl ();
printf "_{\square \square k*k"; nl ();
printf "____da22(cc,s,m)___da22(cc,s,m))/12)"; nl();
printf "uuendufunctionudalzz0_s"; nl ();
nl();
printf ~"_{ \sqcup \sqcup} \% sfunction_{ \sqcup} dalzz0\_t_{ \sqcup} (\texttt{cc,m,k})_{ \sqcup} result_{ \sqcup} (\texttt{alzz0\_t}) "~pure;~nl~();
printf "____type(momentum),__intent(in)__::__k"; nl ();
printf "_____real(kind=default),__dimension(1:12),__intent(in)__::__cc"; nl ();
printf "____real(kind=default),_dimension(1:5),_intent(in)_::_m"; nl();
printf "_\underset complex(kind=default)\underset ::\underset alzz0_t"; nl ();
printf "_{\square \square k*k"}; nl ();
printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup} alzz0_t = (5.)*g**4/costhw**2*(da02(cc,s,m)_{\sqcup}-_{\sqcup}\&"; nl ();
printf "_____da22(cc,s,m))/4"; nl ();
printf "_{\sqcup \sqcup} end_{\sqcup} function_{\sqcup} dalzz0_t"; nl ();
nl();
printf "_{\sqcup\sqcup} %sfunction_{\sqcup} dalzz1_{-s_{\sqcup}} (cc,m,k)_{\sqcup} result_{\sqcup} (alzz1_{-s}) " pure; nl ();
printf "_\uu_\uu_\uu_\upper (momentum), \upper intent(in)_\upper:: \upper k"; nl ();
printf "uuuuureal(kind=default),udimension(1:12),uintent(in)u::ucc"; nl ();
printf "____real(kind=default),_dimension(1:5),_intent(in)_::_m"; nl ();
printf "____complex(kind=default)_::_alzz1_s"; nl ();
printf "_{\square \square \square \square \square \square} real(kind=default)_{\square} ::_{\square} s"; nl();
printf "_{\square \square k*k"}; nl ();
printf "____alzz1_s_=_g**4/costhw**2*(da20(cc,s,m)/8_&"; nl ();
printf "_____(5.)*da22(cc,s,m)/4)"; nl ();
printf "_uuendufunctionudalzz1_s"; nl ();
nl();
printf "___%sfunction_dalzz1_t_(cc,m,k)_result_(alzz1_t)" pure; nl ();
printf "____type(momentum),_intent(in)_::_k"; nl();
printf "\( \subset \) real(\( kind = default \), \( \subset dimension(1:12) \), \( \subset intent(in) \( \subset : \subset cc''; \ nl \( ); \)
printf "uuuuureal(kind=default),udimension(1:5),uintent(in)u::um"; nl ();
printf "____complex(kind=default)_::_alzz1_t"; nl ();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} real(kind=default)_{\sqcup} :: _{\sqcup} s"; nl();
printf "_{\square \square k*k"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box} alzz1_t = g**4/costhw**2*(-(3.)*da11(cc,s,m)/8(%"; nl (); nl (); )
printf "_____+_3*(5.)*da22(cc,s,m)/8)"; nl ();
printf "_{\sqcup\sqcup} end_{\sqcup} function_{\sqcup} dalzz1_t"; nl ();
nl();
printf "_{\sqcup\sqcup} %sfunction_{\sqcup} dalzz1_{\sqcup\sqcup}(cc,m,k)_{\sqcup} result_{\sqcup}(alzz1_{\sqcup}) " pure; nl ();
printf "____type(momentum),__intent(in)__::__k"; nl ();
printf "uuuuureal(kind=default),udimension(1:12),uintent(in)u::ucc"; nl ();
printf "_{\square \square \square \square \square \square}real(kind=default), _{\square}dimension(1:5), _{\square}intent(in) _{\square}:: _{\square}m"; nl ();
printf "____complex(kind=default)_::_alzz1_u"; nl();
printf "□□□□□□real(kind=default)□::□s"; nl ();
printf "_{\square \square \square \square \square \square \square} s_{\square} = _{\square} k*k"; nl ();
printf ~~ \color= \c
printf "_____+_3*(5.)*da22(cc,s,m)/8)"; nl ();
printf "_{\sqcup\sqcup}end_{\sqcup}function_{\sqcup}dalzz1_{\sqcup}u"; nl ();
nl();
```

```
printf "LLL%sfunction_dalww0_s_(cc,m,k)_result_(alww0_s)" pure; nl ();
printf "____type(momentum),__intent(in)__::__k"; nl ();
printf "_____real(kind=default),_dimension(1:12),_intent(in)_::__cc"; nl ();
printf "_{\square \square \square \square \square \square}real(kind=default), _{\square}dimension(1:5), _{\square}intent(in)_{\square}:: _{\square}m"; nl ();
printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup} complex(kind=default)_{\sqcup}::_{\sqcup}alww0_s"; nl();
printf "____real(kind=default)_::_s"; nl();
printf "____k*k"; nl ();
printf ~" \_ \_ \_ \_ = \_ \_ **4*((2*da00(cc,s,m)_ \_ + \_ da20(cc,s,m))/24 \_ \&"; ~nl~(); \\
printf "______(5.)*(2*da02(cc,s,m)__+__da22(cc,s,m))/12)"; nl ();
printf "_\underline{\text{unction}}\underline{\text{dalww0_s"}}; nl ();
nl();
printf ~\texttt{```} \texttt{``} \texttt{```} \texttt{``} 
printf "____type (momentum),__intent(in)__::__k"; nl ();
printf ~\texttt{"} \verb| uuuuuu real(kind=default), \verb| udimension(1:12), \verb| uintent(in)| u: : \verb| ucc"; nl (); \\
printf "_____real(kind=default),_dimension(1:5),_intent(in)_::_m"; nl ();
printf "____complex(kind=default)_::_alww0_t"; nl ();
printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup}real(kind=default)_{\sqcup}::_{\sqcup}s"; nl();
printf "_{\square \square \square \square \square \square \square} s_{\square} = _{\square} k*k"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box} alww0_t = g**4*(2*(5.)*da02(cc,s,m)_{\Box} = (3.)*da11(cc,s,m)_{\Box} % (3.)*da11(cc,s,m
printf "_____+_(5.)*da22(cc,s,m))/8"; nl ();
printf "_\underset end_function\underset dalww0_t"; nl ();
nl();
printf "⊔⊔%sfunctionudalww0_uu(cc,m,k)uresultu(alww0_u)" pure; nl ();
printf "_\underset \text{type} (momentum), \underset \text{intent(in)} \underset : \underset \text{k"}; nl ();
printf "uuuuureal(kind=default),udimension(1:12),uintent(in)u::ucc"; nl ();
printf "\( \subset \) real (kind=default), \( \subset \) dimension (1:5), \( \subset \) intent(in) \( \subset \): \( \subset \) nl ();
printf "_\underset \text{complex(kind=default)}\underset : \underset \text{alww0_u"}; nl();
printf "_{ \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup } s_{ \sqcup } =_{ \sqcup } k*k"; nl ();
printf "_____+_(5.)*da22(cc,s,m))/8"; nl ();
printf "_u_end_function_dalww0_u"; nl ();
nl();
printf "_{\sqcup \sqcup} %sfunction_{\sqcup} dalww2_{-s_{\sqcup}} (cc,m,k)_{\sqcup} result_{\sqcup} (alww2_{-s}) " pure; nl ();
printf "____real(kind=default),_dimension(1:12),_intent(in)_::_cc"; nl ();
printf "_____real(kind=default),_dimension(1:5),_intent(in)_::_m"; nl ();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} complex(kind=default)_{\sqcup} : _{\sqcup} alww2_s"; nl ();
printf "□□□□□□real(kind=default)□::□s"; nl ();
printf "_{\square \square k*k"; nl ();
printf "____alww2_s_=_g**4*(da20(cc,s,m)__-2*(5.)*da22(cc,s,m))/4__"; nl();
printf "_{\sqcup \sqcup} end_{\sqcup} function_{\sqcup} dalww2_s"; nl ();
nl();
printf "___%sfunction_dalww2_t_(cc,m,k)_result_(alww2_t)" pure; nl ();
printf "uuuuureal(kind=default),udimension(1:12),uintent(in)u::ucc"; nl ();
printf "_{\square \square \square \square \square \square}real(kind=default), _{\square}dimension(1:5), _{\square}intent(in)_{\square}::_{\square}m"; nl ();
printf "____complex(kind=default)_::_alww2_t"; nl ();
printf "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} real(kind=default)_{\sqcup} :: _{\sqcup} s"; nl ();
printf "_{\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup}s_{\sqcup}=_{\sqcup}k*k"; nl ();
printf "_{\Box\Box\Box\Box\Box\Box\Box} alww2_{t_{\Box}=\Box} 3*(5.)*g**4*da22(cc,s,m)/4"; nl ();
printf "_uend_function_dalww2_t"; nl ();
nl();
printf "___%sfunction_dalz4_s_(cc,m,k)_result_(alz4_s)" pure; nl ();
printf "____type(momentum),_intent(in)_::_k"; nl ();
printf "uuuuureal(kind=default),udimension(1:12),uintent(in)u::ucc"; nl ();
printf "\( \subset \) real (kind=default), \( \subset \) dimension (1:5), \( \subset \) intent(in) \( \subset \): \( \subset \) nl ();
printf "□□□□□□□complex(kind=default)□::□alz4_s"; nl ();
printf "____real(kind=default)_::_\s"; nl();
printf "_{ \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup } s_{ \sqcup = \sqcup } k*k"; nl ();
printf "_{\square\square\square\square\square\square\square}alz4_s_=\_g**4/costhw**4*((da00(cc,s,m)\_\&"; nl ();
```

```
printf "_____+_2*da20(cc,s,m))/12_&"; nl ();
     printf "_____(5.)*(da02(cc,s,m)+2*da22(cc,s,m))/6)"; nl ();
     printf "ullendufunctionudalz4_s"; nl ();
     nl();
     printf "___@[<5>";
     printf "_{\sqcup \sqcup} %sfunction_{\sqcup} dalz_{t_{\sqcup}}(cc,m,k)_{\sqcup} result_{\sqcup} (alz_{t}) " pure; nl ();
     printf "_\underset \text{type}(momentum),\underset \text{intent(in)}\underset:\underset \text{k"}; nl ();
     printf "\( \subset \) real(kind=default), \( \subset \) dimension(1:12), \( \subset \) intent(in) \( \subset \): \( \subset \) cc"; \( nl \) ();
     printf "\( \subset \) real(kind=default), \( \subset \) dimension(1:5), \( \subset \) intent(in) \( \subset \): \( \subset \) nl ();
     printf "⊔⊔⊔⊔⊔⊔complex(kind=default)⊔::⊔alz4_t"; nl ();
     printf "_{\square\square\square\square\square\square}s_{\square}=_{\square}k*k"; nl ();
     printf ~\texttt{"uuuuuu} \texttt{alz} 4\_\texttt{tu} = \texttt{ug} **4/\texttt{costhw} **4*(5.)*(\texttt{da}02(\texttt{cc,s,m}) \texttt{u\&"}; ~nl~();
     printf "____+2*da22(cc,s,m))/4"; nl ();
     printf "_{\sqcup \sqcup} end_{\sqcup} function_{\sqcup} dalz4_t"; nl ();
     nl();
end
```

—16— Phase Space

16.1 Interface of Phasespace

```
\begin{array}{ll} \text{module type } T &= \\ \text{sig} \\ \text{type } momentum \\ \\ \text{type } \alpha \ t \\ \text{type } \alpha \ decay \end{array}
```

Sort individual decays and complete phasespaces in a canonical order to determine topological equivalence classes.

```
 \begin{array}{l} \mathsf{val} \; sort \; : \; (\alpha \; \rightarrow \; \alpha \; \rightarrow \; int) \; \rightarrow \; \alpha \; t \; \rightarrow \; \alpha \; t \\ \mathsf{val} \; sort\_decay \; : \; (\alpha \; \rightarrow \; \alpha \; \rightarrow \; int) \; \rightarrow \; \alpha \; decay \; \rightarrow \; \alpha \; decay \\ \end{array}
```

Functionals:

```
\begin{array}{l} \mathsf{val}\ map\ :\ (\alpha\ \to\ \beta)\ \to\ \alpha\ t\ \to\ \beta\ t \\ \mathsf{val}\ map\_decay\ :\ (\alpha\ \to\ \beta)\ \to\ \alpha\ decay\ \to\ \beta\ decay \\ \\ \mathsf{val}\ eval\ :\ (\alpha\ \to\ \beta)\ \to\ (\alpha\ \to\ \beta)\ \to\ (\alpha\ \to\ \beta\ \to\ \beta\ \to\ \beta)\ \to\ \alpha\ t\ \to\ \beta\ t \\ \mathsf{val}\ eval\_decay\ :\ (\alpha\ \to\ \beta)\ \to\ (\alpha\ \to\ \beta\ \to\ \beta\ \to\ \beta)\ \to\ \alpha\ decay\ \to\ \beta\ decay \\ \end{array}
```

of momenta f1 f2 plist constructs the phase space parameterization for a process $f_1f_2 \to X$ with flavor decoration from pairs of outgoing momenta and flavors plist and initial flavors f1 and f2

```
val of momenta: \alpha \to \alpha \to (momentum \times \alpha) \ list \to (momentum \times \alpha) \ t val decay\_of\_momenta: (momentum \times \alpha) \ list \to (momentum \times \alpha) \ decay exception Duplicate of momentum exception Unordered of momentum exception Incomplete of momentum
```

end

module $Make\ (M\ :\ Momentum.T)\ :\ T\ with\ type\ momentum\ =\ M.t$

16.2 Implementation of Phasespace

16.2.1 Tools

These are candidates for *ThoList* and not specific to phase space.

```
\label{eq:local_state} \begin{array}{lll} \text{let rec } \textit{first\_match'} & \textit{mismatch } f & = \text{ function} \\ | & [ ] & \rightarrow \textit{None} \\ | & x :: rest \rightarrow \\ & \text{if } f & \text{x then} \\ & & \textit{Some } (x, \textit{List.rev\_append mismatch rest}) \\ & & \text{else} \\ & & \textit{first\_match'} (x :: \textit{mismatch}) \textit{f rest} \\ \\ & \text{Returns } (x, X \setminus \{x\}) \text{ if } \exists x \in X : f(x). \\ \\ & \text{let } \textit{first\_match } f & l & = \textit{first\_match'} \left[ \right] \textit{f } l \end{array}
```

```
let rec first\_pair' mismatch1 f l1 l2 =
   match l1 with
     [] \rightarrow None
     x1 :: rest1 \rightarrow
         begin match first\_match (f x1) l2 with
            None \rightarrow first\_pair' (x1 :: mismatch1) f rest1 l2
          | Some (x2, rest2) \rightarrow
                Some ((x1, x2), (List.rev\_append\ mismatch1\ rest1,\ rest2))
         end
Returns ((x,y),(X\setminus\{x\},Y\setminus\{y\})) if \exists x\in X:\exists y\in Y:f(x,y).
let first\_pair f l1 l2 = first\_pair' [] f l1 l2
                                        16.2.2 Phase Space Parameterization Trees
module type T =
   sig
      type momentum
      type \alpha t
      type \alpha decay
      \mathsf{val} \ sort \ : \ (\alpha \ \to \ \alpha \ \to \ int) \ \to \ \alpha \ t \ \to \ \alpha \ t
      \mathsf{val} \ \mathit{sort\_decay} \ : \ (\alpha \ \to \ \alpha \ \to \ \mathit{int}) \ \to \ \alpha \ \mathit{decay} \ \to \ \alpha \ \mathit{decay}
      \mathsf{val}\ map\ :\ (\alpha\ \to\ \beta)\ \to\ \alpha\ t\ \to\ \beta\ t
      val\ map\_decay\ :\ (\alpha\ 	o\ \beta)\ 	o\ \alpha\ decay\ 	o\ \beta\ decay
      \mathsf{val}\ eval\ :\ (\alpha\ \to\ \beta)\ \to\ (\alpha\ \to\ \beta)\ \to\ (\alpha\ \to\ \beta\ \to\ \beta)\ \to\ \alpha\ t\ \to\ \beta\ t
      val eval\_decay: (\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \beta \rightarrow \beta \rightarrow \beta) \rightarrow \alpha \ decay \rightarrow \beta \ decay
      val of momenta : \alpha \rightarrow \alpha \rightarrow (momentum \times \alpha) list \rightarrow (momentum \times \alpha) t
      val decay\_of\_momenta : (momentum \times \alpha) list \rightarrow (momentum \times \alpha) decay
      exception Duplicate of momentum
      exception Unordered of momentum
      exception Incomplete of momentum
module\ Make\ (M\ :\ Momentum.T)\ =
   struct
      type momentum = M.t
```



Finally, we came back to binary trees ...

Cascade Decays

```
type \alpha \ decay =
      Leaf of \alpha
    | Branch of \alpha \times \alpha \ decay \times \alpha \ decay
```



Trees of type (momentum $\times \alpha$ option) decay can be build easily and mapped to (momentum $\times \alpha$) decay later, once all the α slots are filled. A more elegant functor operating on β decay directly (with Momentum style functions defined for β) would not allow holes in the β decay during the construction.

```
let \ label = function
    Leaf p \rightarrow p
   | Branch (p, \_, \_) \rightarrow p
let rec sort\_decay \ cmp = function
    Leaf _ as l \rightarrow l
    Branch (p, d1, d2) \rightarrow
        let d1' = sort\_decay \ cmp \ d1
        and d2' = sort\_decay \ cmp \ d2 in
        if cmp (label d1') (label d2') \leq 0 then
```

```
Branch\ (p,\ d1',\ d2') else Branch\ (p,\ d2',\ d1') let rec map\_decay\ f = function |\ Leaf\ p \to Leaf\ (f\ p) |\ Branch\ (p,\ d1,\ d2) \to Branch\ (f\ p,\ map\_decay\ f\ d1,\ map\_decay\ f\ d2) let rec eval\_decay\ fl\ fb = function |\ Leaf\ p \to Leaf\ (fl\ p) |\ Branch\ (p,\ d1,\ d2) \to  let d1' = eval\_decay\ fl\ fb\ d1 and d2' = eval\_decay\ fl\ fb\ d2 in Branch\ (fb\ p\ (label\ d1')\ (label\ d2'),\ d1',\ d2')
```

Assuming that $p > p_D \lor p = p_D \lor p < p_D$, where p_D is the overall momentum of a decay tree D, we can add p to D at the top or somewhere in the middle. Note that '<' is not a total ordering and the operation can fail (raise exceptions) if the set of momenta does not correspond to a tree. Also note that a momentum can already be present without flavor as a complement in a branching entered earlier.

```
exception Duplicate of momentum
exception Unordered of momentum
let rec embed\_in\_decay (p, f as pf) = function
  | Leaf (p', f' \text{ as } pf') \text{ as } d' \rightarrow
       if M.less p' p then
          Branch\ ((p,\ Some\ f),\ d',\ Leaf\ (M.sub\ p\ p',\ None))
       else if M.less p p' then
          Branch (pf', Leaf (p, Some f), Leaf (M.sub p' p, None))
       else if p = p' then
          begin match f' with
            None \rightarrow Leaf (p, Some f)
           Some \ \_ \ \rightarrow \ raise \ (Duplicate \ p)
          end
          raise (Unordered p)
  | Branch\ ((p',\ f'\ \mathsf{as}\ pf'),\ d1,\ d2)\ \mathsf{as}\ d'\ 	o
       let p1, \_ = label d1
       and p2, \_ = label d2 in
       if M.less p' p then
          Branch ((p, Some f), d', Leaf (M.sub p p', None))
       else if M.lesseq p p1 then
          Branch (pf', embed\_in\_decay pf d1, d2)
       else if M.lesseq p p2 then
          Branch (pf', d1, embed\_in\_decay pf d2)
       else if p = p' then
          begin match f' with
           None \rightarrow Branch ((p, Some f), d1, d2)
           Some \ \_ \rightarrow raise (Duplicate \ p)
          end
       else
          raise (Unordered p)
```



Note that both *embed_in_decay* and *embed_in_decays* below do *not* commute, and should process 'bigger' momenta first, because disjoint sub-momenta will create disjoint subtrees in the latter and raise exceptions in the former.

```
exception Incomplete of momentum
```

```
 \begin{array}{ll} \mathsf{let} \; \mathit{finalize1} \; = \; \mathsf{function} \\ \mid \; p, \; \mathit{Some} \; f \; \rightarrow \; (p, \; f) \\ \mid \; p, \; \mathit{None} \; \rightarrow \; \mathit{raise} \; (\mathit{Incomplete} \; p) \\ \\ \mathsf{let} \; \mathit{finalize\_decay} \; t \; = \; \mathit{map\_decay} \; \mathit{finalize1} \; t \\ \end{array}
```

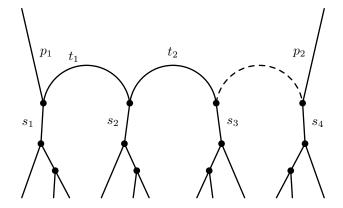


Figure 16.1: Phase space parameterization for $2 \rightarrow n$ scattering by a sequence of cascade decays.

Process the momenta starting in with the highest M.rank:

```
 \begin{array}{l} \text{let } sort\_momenta \ plist = \\ List.sort \ (\text{fun } (p1, \ \_) \ (p2, \ \_) \ \rightarrow \ M.compare \ p1 \ p2) \ plist \\ \\ \text{let } decay\_of\_momenta \ plist = \\ \\ \text{match } sort\_momenta \ plist \ \text{with} \\ \\ | \ (p, \ f) \ :: \ rest \ \rightarrow \\ \\ finalize\_decay \ (List.fold\_right \ embed\_in\_decay \ rest \ (Leaf \ (p, \ Some \ f))) \\ \\ | \ [] \ \rightarrow \ invalid\_arg \ "Phasespace.decay\_of\_momenta: $\sqcup empty" \ \end{tabular}
```

$$2 \rightarrow n$$
 Scattering

A general $2 \to n$ scattering process can be parameterized by a sequence of cascade decays. The most symmetric representation is a little bit redundant and enters each t-channel momentum twice.

```
\mathsf{type} \,\, \alpha \,\, t \,\, = \,\, (\alpha \,\, \times \,\, \alpha \,\, decay \,\, \times \,\, \alpha) \,\, \mathit{list}
```



let $topology = map \ snd$ has type $(momentum \times \alpha) \ t \to \alpha \ t$ and can be used to define topological equivalence classes "up to permutations of momenta," which are useful for calculating Whizard "groves" [11].

```
let sort\ cmp = List.map\ (\operatorname{fun}\ (l,\ d,\ r) \to (l,\ sort\_decay\ cmp\ d,\ r)) let map\ f = List.map\ (\operatorname{fun}\ (l,\ d,\ r) \to (f\ l,\ map\_decay\ f\ d,\ f\ r)) let eval\ ft\ fl\ fb = List.map\ (\operatorname{fun}\ (l,\ d,\ r) \to (ft\ l,\ eval\_decay\ fl\ fb\ d,\ ft\ r))
```

Find a tree with a defined ordering relation with respect to p or create a new one at the end of the list.

```
\begin{array}{lll} \text{let rec } embed\_in\_decays \; (p,\; f \; \text{as} \; pf) \; = \; \text{function} \\ |\;\; [] \; \rightarrow \; [Leaf \; (p,\; Some \; f)] \\ |\;\; d' \; :: \; rest \; \rightarrow \\ &\;\; \text{let} \; p', \; \_ \; = \; label \; d' \; \text{in} \\ &\;\; \text{if} \; M.lesseq \; p' \; p \; \lor \; M.less \; p \; p' \; \text{then} \\ &\;\; embed\_in\_decay \; pf \; d' \; :: \; rest \\ &\;\; \text{else} \\ &\;\; d' \; :: \; embed\_in\_decays \; pf \; rest \\ \end{array}
```

Collecting Ingredients

```
\label{eq:type-aunfinished_decays} \begin{split} &\text{type } \alpha \ unfinished\_decays} \ = \\ & \{ \ n \ : \ int; \\ & t\_channel \ : \ (momentum \ \times \ \alpha \ option) \ list; \\ & decays \ : \ (momentum \ \times \ \alpha \ option) \ decay \ list \ \} \end{split} \text{let } empty \ n \ = \ \{ \ n \ = \ n; \ t\_channel \ = \ [\ ]; \ decays \ = \ [\ ] \ \}
```

¹Not to be confused with gauge invariant classes of Feynman diagrams [12].

```
let insert\_in\_unfinished\_decays (p, f as pf) d =
       if M.Scattering.spacelike p then
          \{ d \text{ with } t\_channel = (p, Some f) :: d.t\_channel \}
       else
          \{ d \text{ with } decays = embed\_in\_decays } f d.decays \}
     let flip\_incoming\ plist\ =
        List.map (fun (p', f') \rightarrow (M.Scattering.flip\_s\_channel\_in p', f')) plist
    let unfinished\_decays\_of\_momenta n f2 p =
        List.fold_right insert_in_unfinished_decays
          (sort\_momenta\ (flip\_incoming\ ((M.of\_ints\ n\ [2],\ f2)\ ::\ p)))\ (empty\ n)
                                                  Assembling Ingredients
    let sort3 compare x y z =
       let a = [|x; y; z|] in
       Array.sort compare a;
       (a.(0), a.(1), a.(2))
Take advantage of the fact that sorting with M.compare sorts with rising values of M.rank:
     let allows\_momentum\_fusion\ (p, \_)\ (p1, \_)\ (p2, \_)\ =
       let p2', p1', p' = sort3 M.compare p p1 p2 in
       match M.try\_fusion p' p1' p2' with
          Some \ \_ \ \to \ \mathsf{true}
          None \rightarrow \mathsf{false}
     let allows\_fusion \ p1 \ p2 \ d = allows\_momentum\_fusion \ (label \ d) \ p1 \ p2
    let rec thread_unfinished_decays' p acc tlist dlist =
       match first_pair (allows_fusion p) tlist dlist with
          None \rightarrow (p, acc, tlist, dlist)
        | Some ((t, \_ as td), (tlist', dlist')) \rightarrow
             thread\_unfinished\_decays' \ t \ (td :: acc) \ tlist' \ dlist'
    let thread\_unfinished\_decays\ p\ c\ =
       match thread\_unfinished\_decays' p [] c.t\_channel c.decays with
        [ -, pairs, [], [] \rightarrow pairs
       \perp \rightarrow failwith "thread_unfinished_decays"
     let rec combine\_decays = function
       | [] \rightarrow []
       | ((t, f \text{ as } tf), d) :: rest \rightarrow
            let p, _{-} = label d in
            begin match M.try\_sub\ t\ p with
             | Some p' \rightarrow (tf, d, (p', f)) :: combine\_decays rest
             None \rightarrow (tf, d, (M.sub (M.neg t) p, f)) :: combine_decays rest
            end
    \mathsf{let} \; \mathit{finalize} \; t \; = \; \mathit{map} \; \mathit{finalize1} \; t
     let of\_momenta f1 f2 = function
       \mid (p, \_) :: \_ \text{ as } l \rightarrow
            let n = M.dim p in
            finalize (combine\_decays)
                            (thread\_unfinished\_decays\ (M.of\_ints\ n\ [1],\ Some\ f1)
                                (unfinished\_decays\_of\_momenta\ n\ f2\ l)))
       | [] \rightarrow []
                                                         Diagnostics
    let p\_to\_string p =
       String.concat "" (List.map string\_of\_int (M.to\_ints (M.abs p)))
```

```
let rec to\_string1 = function
      | Leaf p \rightarrow "(" ^ p\_to\_string p ^ ")"
      | Branch (-, d1, d2) \rightarrow "(" \hat{to\_string1} d1 \hat{to\_string1} d2 \hat{"})"
  let to\_string \ ps =
      String.concat "/"
         (List.map (fun (p1, d, p2) \rightarrow
           p\_to\_string \ p1 \ \hat{} \ to\_string1 \ d \ \hat{} \ p\_to\_string \ p2) \ ps)
                                                               Examples
  {\tt let} \ \mathit{try\_thread\_unfinished\_decays} \ p \ c \ = \\
      thread\_unfinished\_decays' p [] c.t\_channel c.decays
  \mathsf{let}\ try\_of\_momenta\ f\ =\ \mathsf{function}
     \mid (p, \_) :: \_ \text{ as } l \rightarrow
           \mathsf{let}^{'} n \ = \ M.dim \ p \ \mathsf{in}
           try\_thread\_unfinished\_decays
               (M.of\_ints\ n\ [1],\ None)\ (unfinished\_decays\_of\_momenta\ n\ f\ l)
      | \ [ \ ] \ 	o \ invalid\_arg \ "try_of_momenta"
end
```

—17— Whizard

Talk to [11].

17.1 Interface of Whizard

```
\begin{array}{l} \text{module type } T = \\ \text{sig} \\ \text{type } t \\ \text{type } amplitude \\ \text{val } trees : amplitude \rightarrow t \\ \text{val } merge : t \rightarrow t \\ \text{val } write : out\_channel \rightarrow string \rightarrow t \rightarrow unit \\ \text{end} \\ \\ \text{module } Make \ (FM : Fusion.Maker) \ (P : Momentum.T) \\ \text{ } (PW : Momentum.Whizard \ with \ type } t = P.t) \ (M : Model.T) : \\ T \ \text{with \ type } amplitude = FM(P)(M).amplitude \\ \\ \text{val } write\_interface : out\_channel \rightarrow string \ list \rightarrow unit \\ \\ \text{val } write\_makefile : out\_channel \rightarrow \alpha \rightarrow unit \\ \\ \text{val } write\_makefile\_processes : out\_channel \rightarrow string \ list \rightarrow unit \\ \\ \text{val } write\_makefile\_processes : out\_channel \rightarrow string \ list \rightarrow unit \\ \\ \text{val } write\_makefile\_processes : out\_channel \rightarrow string \ list \rightarrow unit \\ \\ \end{array}
```

17.2 Implementation of Whizard

```
open Printf
module type T =
  sig
    type t
    type amplitude
    val trees : amplitude \rightarrow t
    \mathsf{val}\ \mathit{merge}\ :\ t\ \to\ t
    \mathsf{val} \ write \ : \ out\_channel \ \to \ string \to \ t \ \to \ unit
Make (FM : Fusion.Maker) (P : Momentum.T)
     (PW : Momentum. Whizard with type t = P.t) (M : Model. T) =
  struct
     module F = FM(P)(M)
    type tree = (P.t \times F.flavor\ list)\ list
    module \ Poles = Map.Make
          (struct
            \mathsf{type}\ t\ =\ int\ \times\ int
            let compare (s1, t1) (s2, t2) =
              let c = compare \ s2 \ s1 in
              if c \neq 0 then
                 c
```

```
else
                 compare t1 t2
         end)
    let add\_tree maps tree trees =
       Poles.add maps
         (try tree :: (Poles.find maps trees) with Not_found \rightarrow [tree]) trees
    type t =
         \{ in1 : F.flavor; \}
            in2 : F.flavor;
            out : F.flavor\ list;
            trees : tree list Poles.t }
    type amplitude = F.amplitude
                                            17.2.1 Building Trees
A singularity is to be mapped if it is timelike and not the overall s-channel.
    let timelike\_map\ c\ =\ P.Scattering.timelike\ c\ \land\ \neg\ (P.Scattering.s\_channel\ c)
    let count\_maps n clist =
       List.fold\_left (fun (s, t \text{ as } cnt) (c, \_) \rightarrow
         if timelike\_map c then
            (succ\ s,\ t)
         else if P.Scattering.spacelike c then
            (s, succ t)
         else
            cnt) (0, 0) clist
    let poles_to_whizard n trees poles =
       let tree = List.map (fun wf \rightarrow
         (P.Scattering.flip\_s\_channel\_in\ (F.momentum\ wf),\ [F.flavor\ wf]))\ poles\ in
       add_tree (count_maps n tree) tree trees
   I must reinstate the conjugate eventually!
    let trees \ a =
       match F.externals a with
       | in1 :: in2 :: out \rightarrow
           let n = List.length out + 2 in
            \{ in1 = F.flavor in1; \}
              in2 = F.flavor in2;
              out = List.map (fun f \rightarrow (* M.conjugate *) (F.flavor f)) out;
              trees = List.fold\_left
                 (poles\_to\_whizard\ n)\ Poles.empty\ (F.poles\ a)\ \}
       \mid _ \rightarrow invalid\_arg "Whizard().trees"
                                             Merging Homomorphic Trees
    module Pole\_Map =
       Map.Make (struct type t = P.t list let compare = compare end)
    module Flavor\_Set =
       Set.Make (struct type t = F.flavor let compare = compare end)
    let add_flavors flist fset =
       List.fold_right Flavor_Set.add flist fset
    let set\_of\_flavors flist =
       List.fold_right Flavor_Set.add flist Flavor_Set.empty
    let pack\_tree map t =
       let c, f =
```

```
List.split \ (List.sort \ (fun \ (c1, \_) \ (c2, \_) \ \rightarrow
              compare (PW.of\_momentum\ c2)\ (PW.of\_momentum\ c1))\ t) in
        let f' =
           try
              List.map2 add_flavors f (Pole_Map.find c map)
           \mid Not\_found \rightarrow List.map \ set\_of\_flavors f \ in
        Pole\_Map.add\ c\ f'\ map
     let pack_map trees = List.fold_left pack_tree Pole_Map.empty trees
     \mathsf{let}\ \mathit{merge\_sets}\ \mathit{clist}\ \mathit{flist}\ =
        List.map2 (fun c f \rightarrow (c, Flavor\_Set.elements f)) <math>clist flist
     let unpack_map map =
        Pole\_Map.fold (fun c f l \rightarrow (merge\_sets c f) :: l) <math>map []
If a singularity is to be mapped (i.e. if it is timelike and not the overall s-channel), expand merged particles
again:
     let unfold1 (c, f) =
        if timelike\_map \ c then
           List.map (fun f' \rightarrow (c, [f'])) f
        else
           [(c,f)]
     let unfold\_tree \ tree = Product.list \ (fun \ x \rightarrow x) \ (List.map \ unfold1 \ tree)
     let \ unfold \ trees = ThoList.flatmap \ unfold\_tree \ trees
     let merge t =
        \{ t \text{ with } trees = Poles.map \}
              \{\text{fun } t' \rightarrow unfold (unpack\_map (pack\_map t'))) t.trees \}
                                                   17.2.3 Printing Trees
     let flavors\_to\_string f =
        String.concat "/" (List.map M.flavor_to_string f)
     let whizard\_tree t =
        "tree<sub>⊔</sub>"
        (String.concat " \sqcup " (List.rev\_map (fun (c, \_) \rightarrow
           (string\_of\_int\ (PW.of\_momentum\ c)))\ t)) ^
        (String.concat ", " (List.rev_map (fun (, f) \rightarrow flavors_to_string f) t))
     let whizard\_tree\_debug t =
        "tree<sub>⊔</sub>"
        (String.concat " \sqcup " (List.rev\_map (fun (c, \_) \rightarrow
           ("[" ^ (String.concat "+" (List.map string_of_int (P.to_ints c))) ^ "]"))
                                       (List.sort (fun (t1, \_) (t2, \_) \rightarrow
                                          let c =
                                             compare
                                                (List.length (P.to\_ints t2))
                                                (List.length (P.to\_ints t1)) in
                                          if c \neq 0 then
                                             c
                                          else
                                             compare t1 \ t2) \ t))) ^
        (String.concat ", " (List.rev_map (fun (, f) \rightarrow flavors_to_string f) t))
     let format\_maps = function
        (0, 0) \rightarrow "neither_{\sqcup}mapped_{\sqcup}timelike_{\sqcup}nor_{\sqcup}spacelike_{\sqcup}poles"
          (0,\ 1)\ \to\ \verb"noumapped" \verb"timelike" \verb"poles", \verb"one" \verb"spacelike" \verb"pole"
        \mid (0, n) \rightarrow \text{"no} \sqcup \text{mapped} \sqcup \text{timelike} \sqcup \text{poles}, \sqcup \text{"}
```

```
string\_of\_int \ n \ ^ "_spacelike_poles"
   (1, 0) \rightarrow "one mapped timelike pole, no spacelike pole"
   \mid~(1,~1)~
ightarrow~ "one\sqcupmapped\sqcuptimelike\sqcupand\sqcupspacelike\sqcuppole\sqcupeach"
  | (1, n) \rightarrow "one \_mapped \_timelike \_and \_"
         string\_of\_int \ n \ ^ "\spacelike\spoles"
   (n, 0) \rightarrow string\_of\_int n
         \verb|"_{\sqcup} mapped_{\sqcup} timelike_{\sqcup} poles_{\sqcup} and_{\sqcup} no_{\sqcup} spacelike_{\sqcup} pole \verb|"
   (n, 1) \rightarrow string\_of\_int n
         "umappedutimelikeupolesuanduoneuspacelikeupole"
  (n, n') \rightarrow string\_of\_int n ^ "\_mapped\_timelike\_and\_"
         string\_of\_int \ n' \ ^ "\_spacelike\_poles"
let format\_flavor f =
   match flavors\_to\_string \ f with
     "d" \rightarrow "d" \mid "dbar" \rightarrow "D"
     "u" \rightarrow "u" |
                       "ubar" 
ightarrow "U"
                       "sbar" 
ightarrow "S"
     "s" 
ightarrow "s" |
     "c" \rightarrow "c" |
                        "cbar" 
ightarrow "C"
     "b" 
ightarrow "b" ert
                       "bbar" 
ightarrow "B"
     "t" \rightarrow "t" | "tbar" \rightarrow "T"
     "e-" 
ightarrow "e1" | "e+" 
ightarrow "E1"
     "nue" \rightarrow "n1" | "nuebar" \rightarrow "N1"
     "mu-" \rightarrow "e2" \mid "mu+" \rightarrow "E2"
     "numu" 
ightarrow "n2" | "numubar" 
ightarrow "N2"
     "tau-" \rightarrow "e3" \mid "tau+" \rightarrow "E3"
     "nutau" 
ightarrow "n3" | "nutaubar" 
ightarrow "N3"
     \texttt{"g"} \rightarrow \texttt{"G"} \mid \texttt{"A"} \rightarrow \texttt{"A"} \mid \texttt{"Z"} \rightarrow \texttt{"Z"}
     "W+" \rightarrow "W+" \mid "W-" \rightarrow "W-"
     "H" \rightarrow "H"
     s \rightarrow s ^{\circ} "_{\sqcup}(not_{\sqcup}translated)"
module Mappable =
   Set.Make (struct type t = string let compare = compare end)
let mappable =
   List.fold_right Mappable.add
      [ "T"; "Z"; "W+"; "W-"; "H" ] Mappable.empty
let \ analyze\_tree \ ch \ t =
   List.iter (fun (c, f) \rightarrow
     \mathsf{let}\ f'\ =\ \mathit{format\_flavor}\ f
      and c' = PW.of\_momentum c in
      if P.Scattering.timelike\ c then begin
        if P.Scattering.s\_channel\ c then
           else if Mappable.mem\ f'\ mappable then
           fprintf\ ch\ "_{ \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} map_{\sqcup} %d_{\sqcup}s-channel_{\sqcup} %s \ " \ c'\ f'
           fprintf ch
               c' f'
        fprintf\ ch\ "_{\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup}!_{\sqcup}t-channel_{\sqcup}%d_{\sqcup}%s_{\sqcup}not_{\sqcup}mapped_{\square}" c'\ f')\ t
let write \ ch \ pid \ t =
  failwith "Whizard.Make().write:\sqcupincomplete"
  fprintf\ ch\ "process_{\sqcup}%s\n"\ pid;
   Poles.iter (fun maps ds \rightarrow
     fprintf \ ch \ "\n_{\sqcup \sqcup \sqcup \sqcup}!_{\sqcup} d_{\sqcup}times_{\sqcup} s:\n"
         (List.length ds) (format_maps maps);
      List.iter (fun d \rightarrow
        fprintf ch "\n___grove\n";
        fprintf\ ch\ "_{\sqcup\sqcup\sqcup\sqcup}%s\n" (whizard\_tree\ d);
        analyze_tree ch d) ds) t.trees;
```

```
\begin{array}{c} & \textit{fprintf } \textit{ch "\n"} \\ i \times ) & \\ & \text{end} \end{array}
```

17.2.4 Process Dispatcher

```
let arguments = function
   | \quad [] \rightarrow ("", "")
    args \rightarrow
        let arg\_list = String.concat ",\Box" (List.map snd args) in
        (\mathit{arg\_list}, ~", \llcorner" ~^{} ~ \mathit{arg\_list})
let import_prefixed ch pid name =
  fprintf\ ch\ "$\sqcup\sqcup\sqcup\sqcup use_{\sqcup}\%s, \_only:_{\sqcup}\%s_{\_}\%s_{\sqcup}=>_{\sqcup}\%s_{\sqcup}!\ NODEP!\n"
     pid pid name name
let declare\_argument \ ch \ (arg\_type, \ arg) =
  fprintf ch "⊔⊔⊔⊔%s,⊔intent(in)⊔::⊔%s\n" arg_type arg
let call_function ch pid result name args =
  fprintf\ ch\ "_{\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup} case_{\sqcup}(pr_{s}) \ 'pid;
  {\tt let} \ \textit{default\_function} \ \textit{ch} \ \textit{result} \ \textit{default} \ = \\
  fprintf\ ch\ "_{ \cup \cup \cup \cup \cup \cup \cup } case_{ \cup } default \ ";
  \mathit{fprintf}\ \mathit{ch}\ \verb""uuuuuuuuuucalluinvalid_processu(pid)\n"};
  let \ call\_subroutine \ ch \ pid \ name \ args =
  fprintf\ ch\ "_{ \  \  } case_{ \  \  } (pr_%s) \  \  ' pid;
  let default\_subroutine ch =
  fprintf\ ch\ "_{ \  \  \  \  \  \  \  \  \  \  } case_{ \  \  \  \  } default\ '\ '';
  fprintf\ ch\ "
let write_interface_subroutine ch wrapper name args processes =
  let arg\_list, arg\_list' = arguments args in
  fprintf ch "_usubroutine_\%s_(pid\%s)\n" wrapper arg_list';
   List.iter (fun p \rightarrow import\_prefixed \ ch \ p \ name) processes;
   List.iter (declare_argument ch) (("character(len=*)", "pid") :: args);
  fprintf \ ch \ "_{\sqcup\sqcup\sqcup\sqcup}select_{\sqcup}case_{\sqcup}(pid)\n";
   List.iter (fun p \rightarrow call\_subroutine\ ch\ p\ name\ arg\_list) processes;
   default\_subroutine \ ch;
  \mathit{fprintf}\ \mathit{ch}\ "{\sqcup\sqcup\sqcup\sqcup\sqcup} \mathtt{end}_{\sqcup}\mathtt{select}\";
  fprintf\ ch\ "\Box\Box end\Box subroutine \%s\n"\ wrapper
let write_interface_function ch wrapper name
     (result\_type, result, default) args processes =
  let arg\_list, arg\_list' = arguments args in
  List.iter (fun p \rightarrow import\_prefixed \ ch \ p \ name) processes;
   List.iter (declare_argument ch) (("character(len=*)", "pid") :: args);
  fprintf\ ch\ "$\sqcup\sqcup\sqcup\sqcup\sqcup" %s$\" result_type\ result;
  fprintf\ ch\ "_{\sqcup\sqcup\sqcup\sqcup}select_{\sqcup}case_{\sqcup}(pid)\n";
   List.iter (fun p \rightarrow call\_function \ ch \ p \ result \ name \ arg\_list) processes;
   default_function ch result default;
  fprintf\ ch\ "_{\sqcup\sqcup\sqcup\sqcup} = nd_{\sqcup}select\n";
  fprintf\ ch\ "\Box\Box end\Box function \" %s\n" \ wrapper
let write\_other\_interface\_functions \ ch =
  fprintf\ ch\ "_{\sqcup\sqcup}subroutine_{\sqcup}invalid\_process_{\sqcup}(pid)\n";
  fprintf\ ch\ "$\sqcup\sqcup\sqcup\sqcup$\sqcup$ character(len=*),$\sqcup$ intent(in)$_{\sqcup}::$\sqcup$pid$_n";
  fprintf ch "⊔⊔⊔⊔print⊔*, ⊔\"PANIC:";
```

```
fprintf ch "_process_'\"//trim(pid)//\"'_not_available!\"\n";
  fprintf ch "uuendusubroutineuinvalid_process\n";
  fprintf\ ch\ "\Box\Box function\Box n\_tot\Box (pid)\Box result\Box (n)\n";
  fprintf\ ch\ "$\sqcup\sqcup\sqcup\sqcup\sqcup$character(len=*),\sqcupintent(in)_{\sqcup}::\sqcuppid\n";
  fprintf\ ch\ "\Box\Box\Box\Boxinteger\Box::\Boxn\n";
  fprintf\ ch\ "\Box\Box\Box\Box n_{\Box} = \Box n_{\Box} (pid)_{\Box} + \Box n_{\Box} ut(pid) \ ";
  fprintf\ ch\ "\_\_end\_function\_n\_tot\n"
let write\_other\_declarations ch =
  fprintf\ ch\ "\Box public : \Box n_in, \Box n_out, \Box n_tot, \Box pdg\_code \ ";
  fprintf\ ch\ "\Box\Box public_{\Box}::\Box allow\_helicities \";
  fprintf\ ch\ "\Box\Box public\Box::\Box create,\Box destroy \";
  fprintf\ ch\ "_{\sqcup\sqcup}public_{\sqcup}::_{\sqcup}set\_const,_{\sqcup}sqme\n";
  fprintf\ ch\ "_{\sqcup\sqcup}interface_{\sqcup}create\n";
  fprintf ch "____module_procedure_process_create\n";
  fprintf\ ch\ "\Box end interface \";
  fprintf\ ch\ "\Box\Box interface \Box destroy \";
  \textit{fprintf ch "$$$"$$$$"$$ under procedure process_destroy $$n"$;}
  \mathit{fprintf}\ ch\ \verb"`uu" end`uinterface \";
  fprintf\ ch\ "_{\sqcup\sqcup}interface_{\sqcup}set\_const\n";
  fprintf ch "____module_procedure_process_set_const\n";
  fprintf\ ch\ "\Box end interface \";
  fprintf\ ch\ "_{\sqcup\sqcup}interface_{\sqcup}sqme\n";
  \textit{fprintf ch "$$\$"$$$$\sqcup$$\sqcup$$\sqcup$$\sqcup$$\sqcup$$\sqcup$$\sqcup$$\sqcup$$\square$$ module$$\sqcup$$procedure$$\sqcup$$process\_sqme\n"$;}
  fprintf\ ch\ "\Box\Box end\Box interface \"
let write\_interface ch names =
  fprintf ch "module_process_interface\n";
  fprintf ch "⊔⊔use⊔kinds,⊔only:⊔default⊔⊔!NODEP!\n";
  \textit{fprintf ch "$$\sqcup$$} \sqcup \texttt{parameters,$$\sqcup$} \texttt{only:$$\sqcup$} \texttt{parameter\_set} \texttt{""};
  \textit{fprintf } ch \texttt{"} \verb"" implicit \verb"" none \verb"";
  fprintf\ ch\ "{\sqcup\sqcup}private\n"};
   List.iter (fun p \rightarrow
     fprintf ch
        "_\( character(len=*), \( parameter, \( public_\): \( pr_\%s_\=_\\"\%s\"\n" \ p \)
     names:
   write\_other\_declarations ch;
  fprintf ch "contains\n";
   write_interface_function ch "n_in" "n_in" ("integer", "n", "0") [] names;
   write_interface_function ch "n_out" "n_out" ("integer", "n", "0") [] names;
   write_interface_function ch "pdg_code" "pdg_code"
     ("integer", "n", "0") [ "integer", "i" ] names;
   write\_interface\_function\ ch "allow_helicities" "allow_helicities"
     ("logical", "yorn", ".false.") [] names;
   write_interface_subroutine ch "process_create" "create" [] names;
   write_interface_subroutine ch "process_destroy" "destroy" [] names;
   write_interface_subroutine ch "process_set_const" "set_const"
      [ "type(parameter_set)", "par"] names;
   write_interface_function ch "process_sqme" "sqme"
     ("real(kind=default)", "sqme", "0")
       "real(kind=default), dimension(0:,:)", "p";
        "integer, _{\square}dimension(:), _{\square}optional", "h" | names;
   write_other_interface_functions ch;
  fprintf\ ch\ "end\_module\_process\_interface\n"
                                                      17.2.5 Makefile
let write_makefile ch names =
  fprintf ch "KINDS<sub>□</sub>=<sub>□</sub>../@KINDS@\n";
  fprintf\ ch\ "HELAS_{\sqcup}=_{\sqcup}.../@HELAS@\n";
```

 $fprintf\ ch\ "F90 = 0F90@n";$

```
fprintf ch "F90FLAGS_=_@F90FLAGS@\n";
  fprintf\ ch\ "F90INCL_=_-I$(KINDS)_-I$(HELAS)\n";
  fprintf\ ch\ "F90COMMON_{\sqcup} = _{\sqcup}omega\_bundle\_whizard.f90";
  \textit{fprintf ch "$$\_$ile\_utils.f90$$\_$process\_interface.f90$$\\ n";}
  fprintf ch "include_Makefile.processes\n";
  fprintf ch "F90SRC<sub>□</sub>=<sub>□</sub>$(F90COMMON)<sub>□</sub>$(F90PROCESSES)\n";
  fprintf\ ch\ "OBJ_{\perp}=_{\perp}$(F90SRC:.f90=.o)\n";
  fprintf \ ch \ "MOD_{\square}=_{\square}$(F90SRC:.f90=.mod)\n";
  fprintf ch "archive: □processes.a\n";
  fprintf ch "processes.a:_\$(OBJ)\n";
  fprintf\ ch\ "\t\$(AR)_{\sqcup}r_{\sqcup}\$@_{\sqcup}\$(OBJ)\n";
  fprintf ch "\t@RANLIB@_$@\n";
  fprintf ch "clean:\n";
  fprintf\ ch\ "\trm_{\sqcup}-f_{\sqcup}$(OBJ)\n";
  fprintf ch "realclean:\n";
  fprintf\ ch\ "\trm_-f_processes.a\n";
  fprintf ch "parameters.o: _file_utils.o\n";
  \textit{fprintf ch "omega\_bundle\_whizard.o:} \sqcup parameters.o\\ \verb|n";
  fprintf ch "process_interface.o: □parameters.o\n";
  fprintf\ ch\ "%%.o: \_%%.f90\_$(KINDS)/kinds.f90\n";
  fprintf\ ch\ "\t\$(F90)_{\sqcup}\$(F90FLAGS)_{\sqcup}\$(F90INCL)_{\sqcup}-c_{\sqcup}\$<\n"
{\tt let} \ write\_makefile\_processes \ ch \ names \ = \\
  fprintf ch "F90PROCESSES<sub>□</sub>=";
  fprintf \ ch \ "\n";
  List.iter (fun f \rightarrow
     \textit{fprintf ch "\%s.o:} \_\texttt{omega\_bundle\_whizard.o} \_\texttt{parameters.o} \\ \texttt{n"} \ f;
     fprintf\ ch\ "process\_interface.o: \" %s.o\n" f)\ names
```

—18— Applications

18.1 Sample

18.2 Interface of Omega

```
module type T =
  sig
    val main : unit \rightarrow unit
   This used to be only intended for debugging O'Giga, but might live longer ...
    type flavor
    \mathsf{val}\ diagrams\ :\ flavor\ 	o\ flavor\ 	o\ flavor\ list\ 	o
      ((flavor \times Momentum.Default.t) \times
          (flavor \times Momentum. Default.t.)
           flavor \times Momentum.Default.t) Tree.t) list
  end
Wrap the two instances of Fusion. Maker for amplitudes and phase space into a single functor to make sure that
the Dirac and Majorana versions match. Don't export the slightly unsafe module Make (FM: Fusion. Maker) (PM: Fusion.
module\ Binary\ (TM\ :\ Target.Maker)\ (M\ :\ Model.T)\ :\ T\ with\ type\ flavor\ =\ M.flavor
module Binary\_Majorana (TM: Target.Maker) (M: Model.T): T with type flavor = M.flavor
module\ Mixed23\ (TM:\ Target.Maker)\ (M:\ Model.T):\ T\ with\ type\ flavor\ =\ M.flavor
Model. T = Model. T : T with type flavor = M. flavor
module\ Mixed23\_Majorana\_vintage\ (TM\ :\ Target.Maker)\ (M\ :\ Model.T)\ :\ T\ with\ type\ flavor\ =\ M.flavor
module Nary (TM : Target.Maker) (M : Model.T) : T with type flavor = M.flavor
module\ Nary\_Majorana\ (TM\ :\ Target.Maker)\ (M\ :\ Model.T)\ :\ T\ with\ type\ flavor\ =\ M.flavor
```

18.3 Implementation of Omega

```
\begin{array}{l} \text{let } (<<) \ f \ g \ x \ = \ f \ (g \ x) \\ \text{let } (>>) \ f \ g \ x \ = \ g \ (f \ x) \\ \\ \text{module } P = Momentum.Default \\ \\ \text{module } P_-Whizard \ = \ Momentum.DefaultW \\ \\ \text{module type } T \ = \\ \text{sig} \\ \text{val } main \ : \ unit \ \rightarrow \ unit \\ \text{type } flavor \\ \text{val } diagrams \ : \ flavor \ \rightarrow \ flavor \ \rightarrow \ flavor \ list \ \rightarrow \\ ((flavor \ \times \ Momentum.Default.t) \ \times \\ (flavor \ \times \ Momentum.Default.t) \ \times \\ (flavor \ \times \ Momentum.Default.t) \ Tree.t) \ list \\ \text{end} \\ \\ \text{module } Make \ (Fusion\_Maker \ : \ Fusion.Maker) \ (PHS\_Maker \ : \ Fusion.Maker) \end{array}
```

```
(Target\_Maker : Target.Maker) (M : Model.T) =
struct
  module CM = Colorize.It(M)
 type flavor = M.flavor
  module\ Proc = Process.Make(M)
```



We must have initialized the vertices before applying Fusion_Maker, at least if we want to continue using the vertex cache!



NB: this causes the constant initializers in Fusion_Maker more than once. Such side effects must be avoided if the initializers involve expensive computations. Relying on the fact that the functor will be called only once is not a good idea!

```
module F = Fusion\_Maker(P)(M)
module \ CF = Fusion.Multi(Fusion\_Maker)(P)(M)
module T = Target\_Maker(Fusion\_Maker)(P)(M)
\mathsf{module}\ W = \mathit{Whizard}.\mathit{Make}(\mathit{Fusion\_Maker})(P)(P\_\mathit{Whizard})(M)
module C = Cascade.Make(M)(P)
module VSet =
  Set.Make (struct type t = F.constant Coupling.t let compare = compare end)
```

For the phase space, we need asymmetric DAGs.

Since we will not use this to compute amplitudes, there's no need to supply the proper statistics module and we may always use Majorana fermions to be as general as possible. In principle, we could expose in Fusion. T the Fusion.Stat_Maker used by Fusion_Maker to construct it, but that is just not worth the effort.



For the phase space, we should be able to work on the uncolored model.

```
module MT = Modeltools. Topology3(M)
module PHS = PHS\_Maker(P)(MT)
module \ CT = Cascade.Make(MT)(P)
```

Form a α list from a α option array, containing the elements that are not None in order.

```
let opt\_array\_to\_list a =
  let rec opt\_array\_to\_list' acc i a =
    if i < 0 then
       acc
     else
       begin match a.(i) with
         None \rightarrow opt\_array\_to\_list' acc (pred i) a
       | Some x \rightarrow opt\_array\_to\_list'(x :: acc)(pred i) a
  opt\_array\_to\_list' [] (Array.length a - 1) a
```

Return a list of CF. amplitude lists, corresponing to the diagrams for a specific color flow for each flavor combination.

```
let amplitudes_by_flavor amplitudes =
  List.map opt_array_to_list (Array.to_list (CF.process_table amplitudes))
```



if we plan to distiguish different couplings later on, we can no long map all instances of coupling option in the tree to None. In this case, we will need to normalize different fusion orders Coupling.fuse2, Coupling.fuse3 or Coupling.fusen, because they would otherwise lead to inequivalent diagrams. Unfortunately, this stuff packaged deep in Fusion. Tagged_Coupling.

The *Tree.canonicalize* below should be necessary to remove topologically equivalent duplicates.

Take a CF. amplitude list assumed to correspond to the same external states after stripping the color and return a pair of the list of external particles and the corresponding Feynman diagrams without color.

```
let wf1 amplitude =
```

```
match F. externals amplitude with
  | wf :: \_ \rightarrow wf
  [] 
ightarrow \mathit{failwith} "Omega.forest\_sans\_color: \_no\_external\_particles"
let uniq l =
  ThoList.uniq (List.sort compare 1)
let forest\_sans\_color = function
  | \ amplitude \ :: \ \_ \ \text{as} \ amplitudes \ \rightarrow
     let externals = F.externals amplitude in
     let prune\_color \ wf =
       (F.flavor\_sans\_color\ wf,\ F.momentum\_list\ wf) in
     let prune\_color\_and\_couplings\ (wf,\ c) =
       (prune_color wf, None) in
     (List.map prune_color externals,
      uniq
        (List.map
            (fun t \rightarrow
               Tree.\ canonicalize
                 (Tree.map\ prune\_color\_and\_couplings\ prune\_color\ t))
            (ThoList.flatmap (fun a \rightarrow F.forest (wf1 a) a) amplitudes)))
  | [] \rightarrow ([], [])
let dag\_sans\_color = function
  \mid amplitude :: _ as amplitudes 
ightarrow
     let prune a = a in
     List.map prune amplitudes
  | [] \rightarrow []
let p2s p =
  if p~\geq~0~\wedge~p~\leq~9 then
     string\_of\_int p
  else if p \leq 36 then
     String.make\ 1\ (Char.chr\ (Char.code\ 'A' + p - 10))
  else
     11 11
let format_p wf =
  String.concat "" (List.map p2s (F.momentum_list wf))
let variable \ wf =
  M.flavor\_to\_string \ (F.flavor\_sans\_color \ wf) \ ^ "[" \ ^ format\_p \ wf \ ^ "]"
  CM.flavor\_to\_TeX (F.flavor\ wf) ^ "(" ^ format\_p\ wf ^ ")"
let feynmf_style propagator color =
  \{ Tree.style =
       begin match propagator with
         Coupling.Prop\_Feynman
         Coupling.Prop\_Gauge \_ \rightarrow
          begin match color with
           Color.AdjSUN \_ \rightarrow Some ("gluon", "")
              \rightarrow Some ("boson", "")
          end
         Coupling.Prop_Col_Feynman → Some ("gluon", "")
         Coupling. Prop_Unitarity
         Coupling.Prop_Rxi _ → Some ("dbl_wiggly", "")
         Coupling.Prop\_Spinor
         Coupling.Prop\_ConjSpinor \rightarrow Some ("fermion", "")
         \_ \rightarrow None
       end;
     Tree.rev =
       begin match propagator with
       | Coupling.Prop\_Spinor \rightarrow true
```

```
Coupling.Prop\_ConjSpinor \rightarrow false
          _{-} \rightarrow false
       end:
     Tree.label = None;
     Tree.tension = None 
let header incoming outgoing =
   "$<sub>\\</sub>" <sup>'</sup>
  String.concat "\Box"
     (List.map\ (CM.flavor\_to\_TeX\ <<\ F.flavor)\ incoming)\ \hat{\ }
   "_\\to_"
  String.concat "\Box"
     (List.map\ (CM.flavor\_to\_TeX\ <<\ CM.conjugate\ <<\ F.flavor)\ outgoing) ^
   "∟$"
let header_sans_color incoming outgoing =
   "$∟"
  String.concat ~" \llcorner "
     (List.map\ (M.flavor\_to\_TeX\ <<\ fst)\ incoming) ^
  "_{\sqcup} \backslash \backslash \mathsf{to}_{\sqcup}"
  String.concat """
     (List.map\ (M.flavor\_to\_TeX\ <<\ M.conjugate\ <<\ fst)\ outgoing)
   "∟$"
let diagram incoming tree =
  let fmf wf =
     let f = F.flavor wf in
     feynmf\_style\ (CM.propagator\ f)\ (CM.color\ f) in
   Tree.map
     (fun (n, \_) \rightarrow
       let n' = fmf n in
       if List.mem\ n\ incoming\ {\it then}
          \{ n' \text{ with } Tree.rev = \neg n'.Tree.rev \}
       else
          n'
     (fun l \rightarrow
       if List.mem l incoming then
       else
          F.conjugate \ l)
     tree
let diagram\_sans\_color\ incoming\ (tree) =
  let fmf(f, p) =
     feynmf\_style\ (M.propagator\ f)\ (M.color\ f)\ in
   Tree.map
     (\mathsf{fun}\ (n,\ c)\ \to
       let n' = fmf n in
        if List.mem\ n\ incoming\ then
          \{ n' \text{ with } Tree.rev = \neg n'.Tree.rev \}
       else
          n'
     (\mathsf{fun}\ (f,\ p)\ \to
       if List.mem (f, p) incoming then
          (f, p)
       else
          (M.conjugate f, p)
     tree
let feynmf\_set amplitude =
  match F. externals amplitude with
  | wf1 :: wf2 :: wfs \rightarrow
     let incoming = [wf1; wf2] in
     \{ Tree.header = header incoming wfs;
```

```
Tree.incoming = incoming;
       Tree.diagrams =
         List.map\ (diagram\ incoming)\ (F.forest\ wf1\ amplitude)\ \}

ightarrow \ failwith \ "less\_than\_two\_external\_particles"
let feynmf_set_sans_color (externals, trees) =
  match externals with
  | wf1 :: wf2 :: wfs \rightarrow
    let incoming = [wf1; wf2] in
     \{ Tree.header = header\_sans\_color incoming wfs; \}
       Tree.incoming = incoming;
       Tree.diagrams =
         List.map (diagram_sans_color incoming) trees }
      \rightarrow failwith "less_than_two_external_particles"
let feynmf_set_sans_color_empty (externals, trees) =
  match externals with
  | wf1 :: wf2 :: wfs \rightarrow
    let incoming = [wf1; wf2] in
     \{\ Tree.header = header\_sans\_color\ incoming\ wfs;
       Tree.incoming = incoming;
       Tree.diagrams = [] 
  | \_ \rightarrow failwith "less_{\sqcup}than_{\sqcup}two_{\sqcup}external_{\sqcup}particles"
let \ uncolored\_colored \ amplitudes \ =
  \{ Tree.outer = feynmf\_set\_sans\_color (forest\_sans\_color amplitudes); \}
     Tree.inner = List.map\ feynmf\_set\ amplitudes\ \}
let \ uncolored\_only \ amplitudes =
  \{ Tree.outer = feynmf\_set\_sans\_color (forest\_sans\_color amplitudes); \}
     Tree.inner = [] 
let \ colored\_only \ amplitudes \ =
  \{ Tree.outer = feynmf\_set\_sans\_color\_empty (forest\_sans\_color amplitudes); \}
     Tree.inner = List.map\ feynmf\_set\ amplitudes\ 
let momentum\_to\_TeX (_, p) =
  String.concat "" (List.map p2s p)
let wf\_to\_TeX (f, \_as wf) =
  M.flavor\_to\_TeX\ f\ ^"("\ ^momentum\_to\_TeX\ wf\ ^")"
let amplitudes_to_feynmf latex name amplitudes =
     Tree.feynmf_sets_wrapped latex name
       wf_to_TeX momentum_to_TeX variable' format_p
       (List.map uncolored_colored (amplitudes_by_flavor amplitudes))
let amplitudes_to_feynmf_sans_color latex name amplitudes =
     Tree.feynmf_sets_wrapped latex name
       wf_to_TeX momentum_to_TeX variable' format_p
       (List.map uncolored_only (amplitudes_by_flavor amplitudes))
let amplitudes_to_feynmf_color_only latex name amplitudes =
     Tree.feynmf_sets_wrapped latex name
       wf\_to\_TeX momentum\_to\_TeX variable' format\_p
       (List.map\ colored\_only\ (amplitudes\_by\_flavor\ amplitudes))
let debug (str, descr, opt, var) =
  ["-warning:" \hat{} str, Arg. Unit (fun () \rightarrow var := (opt, false) :: !var),
     \verb""uuuuuuuuchecku" ^ descr ^ "uanduprintuwarninguonuerror";
     "-error: " \hat{} str, Arg.Unit (fun () \rightarrow var := (opt, true) :: !var),
     \verb"``uuuuuuuuuchecku" ^ descr ^ "uanduterminateuonuerror" ]
let rec include\_goldstones = function
  | [] \rightarrow \mathsf{false}
    (T.Gauge, \_) :: \_ \rightarrow \mathsf{true}
  | \_ :: rest \rightarrow include\_goldstones\ rest
```

```
let \ read\_lines\_rev \ file \ =
  let ic = open_{-}in file in
  let rev\_lines = ref[] in
  let rec slurp() =
    rev\_lines := input\_line ic :: !rev\_lines;
    slurp() in
  try
     slurp()
  with
  \mid End\_of\_file \rightarrow
       close\_in\ ic;
       !rev\_lines
let read_lines file =
  List.rev (read_lines_rev file)
let unphysical\_polarization = ref None
                                      18.3.1 Main Program
let main() =
  (* Delay evaluation of M.external\_flavors ()! *)
  let usage () =
    "usage:\Box" ^ Sys.argv.(0) ^
    "_[options]_["
       String.concat "|" (List.map M.flavor_to_string
                               (ThoList.flatmap snd
                                   (M.external\_flavors\ ()))) ^ "]"
  and rev\_scatterings = ref[]
  and rev\_decays = ref[]
  and cascades = ref[]
  and checks = ref[]
  and output\_file = ref None
  and print\_forest = ref false
  and template = ref false
  and diagrams\_all = ref None
  and diagrams\_sans\_color = ref None
  and diagrams_color_only = ref None
  and diagrams\_LaTeX = ref false
  and quiet = ref false
  and write = ref true
  and params = ref false
  and poles = ref false
  and dag\_out = ref\ None
  and dag\theta_-out = ref None
  and phase\_space\_out = ref None in
  Options.parse
    (Options.cmdline "-target: " T.options @
      Options.cmdline "-model:" M.options @
      Options.cmdline "-fusion: " CF.options @
      ThoList.flatmap debug
        ["a", "arguments", T.All, checks;
         "n", "#⊔of⊔input⊔arguments", T.Arguments, checks;
         "m", "input_momenta", T.Momenta, checks;
         "g", "internal \sqcup Ward \sqcup identities", T.Gauge, checks @
      [("-o", Arg.String (fun s \rightarrow output\_file := Some s),
        "file____write_to_given_file_instead_of_/dev/stdout");
       ("-scatter",
        Arg.String (fun s \rightarrow rev\_scatterings := s :: !rev\_scatterings),
        "expr_{\square\square\square\square\square\square}in1_\square in2_\square ->_\square out1_\square out2_\square...");
       ("-scatter_file",
```

```
Arg.String (fun s \rightarrow rev\_scatterings := read\_lines\_rev s @ !rev\_scatterings),
          "name, || each, |line: || in1, |in2, |->, || out1, || out2, ||...");
         ("-decay", Arg.String (fun s \rightarrow rev\_decays := s :: !rev\_decays),
           "expr_{\cup\cup\cup\cup\cup\cup\cup\cup}in_{\cup}->_{\cup}out1_{\cup}out2_{\cup}...");
         ("-decay_file",
           Arg.String (fun s \rightarrow rev\_decays := read\_lines\_rev s @ !rev\_decays),
           "name\sqcup \sqcup \sqcup \sqcup \sqcup = ach \sqcup line : \sqcup in \sqcup -> \sqcup out 1 \sqcup out 2 \sqcup ...");
         ("-cascade", Arg.String (fun s \rightarrow cascades := s :: !cascades),
           "expr_{\verb||| \verb||| \verb||| \verb||| \verb||| \verb||| \verb||| select_\verb||| diagrams");
         ("-unphysical", Arg.Int (fun i \rightarrow unphysical\_polarization := Some i),
           "n_{\cup\cup\cup\cup\cup\cup\cup\cup}use_{\cup}unphysical_{\cup}polarization_{\cup}for_{\cup}n-th_{\cup}particle_{\cup}/_{\cup}test_{\cup}WIs");
         ("-template", Arg. Set template,
             ("-forest", Arg.Set print_forest,
           "____Diagrammatic_expansion");
         ("-diagrams", Arg.String (fun s \rightarrow diagrams\_sans\_color := Some s),
          "file____produce_FeynMP_output_for_Feynman_diagrams");
        ("-diagrams:c", Arg.String (fun s \rightarrow diagrams\_color\_only := Some s),
          "file_{\sqcup\sqcup\sqcup\sqcup}produce_{\sqcup}FeynMP_{\sqcup}output_{\sqcup}for_{\sqcup}color_{\sqcup}flow_{\sqcup}diagrams");
        ("-diagrams:C", Arg.String (fun s \rightarrow diagrams\_all := Some s),
           ("-diagrams_LaTeX", Arg.Set diagrams_LaTeX,
           "____enclose_FeynMP_output_in_LaTeX_wrapper");
         ("-quiet", Arg.Set quiet,
           \verb"uuuuuuuuuuudon'tuprintuausummary");
         (\verb"-summary", Arg. Clear write,
           "____print_only_a_summary");
         ("-params", Arg.Set params,
           \verb""" \verb""" the \verb""" model \verb""" parameters");
         ("-poles", Arg. Set poles,
           "\verb| uuuuuuuuuuprintutheuMonteuCarloupoles");
         ("-dag", Arg.String (fun s \rightarrow dag\_out := Some s),
           \verb"uuuuuuuuuuprintuminimaluDAG");
         ("-full_dag", Arg.String (fun s \rightarrow dag0\_out := Some s),
           \verb"uuuuuuuuprintucompleteuDAG");
         ("-phase_space", Arg.String (fun s \rightarrow phase\_space\_out := Some s),
           \verb"``\uu`\uu`\uprint\uminimal\uDAG\uprint\uminimal\uDAG\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uppi\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\uprint\upr
     (fun \_ \rightarrow prerr\_endline (usage ()); exit 1)
     usage;
let \ cmdline =
    String.concat "" (List.map ThoString.quote (Array.to_list Sys.argv)) in
let \ output\_channel, \ close\_output\_channel =
    match !output_file with
       None \rightarrow
          (stdout, fun() \rightarrow ())
    \mid Some name \rightarrow
          let oc = open\_out \ name in
          (oc, fun () \rightarrow close\_out oc) in
let processes =
    try
         ThoList.uniq
             (List.sort compare
                   (match List.rev !rev_scatterings, List.rev !rev_decays with
                      [], [] \rightarrow []
                     scatterings, [] \rightarrow
                            Proc.expand_scatterings (List.map Proc.parse_scattering scatterings)
                   | \ | \ |, \ decays \rightarrow
                            Proc.expand_decays (List.map Proc.parse_decay decays)
                   \mid scatterings, decays \rightarrow
                            invalid\_arg "mixed_scattering_and_decay!"))
```



This is still crude. Eventually, we want to catch *all* exceptions and write an empty (but compilable) amplitude unless one of the special options is selected.

```
begin match processes, !params with
| \  \  \_, \ \mathsf{true} \to
    if !write then
       T.parameters\_to\_channel\ output\_channel;
     exit 0
| [], false \rightarrow
   if !write then
      T.amplitudes_to_channel cmdline output_channel !checks CF.empty;
   exit 0
\mid _, false \rightarrow
  let selectors =
    let fin, fout = List.hd processes in
     C.to_selectors (C.of_string_list (List.length fin + List.length fout)!cascades) in
  \mathsf{let} \ amplitudes \ =
       begin match F.check\_charges () with
         [] \rightarrow ()
       | violators \rightarrow
            let \ violator\_strings =
               String.concat ", "
                 (List.map
                     (fun flist \rightarrow
                        "(" ^{\circ} String.concat "," (List.map M.flavor_to_string flist) ^{\circ} ")")
                     violators) in
            failwith ("charge\_violating\_vertices:\_" ^ violator\_strings)
       end;
       CF.amplitudes\ (include\_goldstones\ !checks)\ !unphysical\_polarization
          CF.no\_exclusions selectors processes
    with
     \mid Fusion.Majorana \rightarrow
        begin
           Printf.eprintf
              "O'Mega: _found_Majorana_fermions, _switching_representation! \n";
           flush stderr;
           close\_output\_channel ();
           Arg.current := 0;
           raise Fusion. Majorana
        end
      exc \rightarrow
          begin
            Printf.eprintf
               "O'Mega: uexception usuninum plitude construction!\n"
               (Printexc.to\_string\ exc);
            flush stderr;
            CF.empty;
          end in
  if !write then
     T.amplitudes_to_channel cmdline output_channel !checks amplitudes;
```

```
if \neg !quiet then begin
  List.iter
     (fun amplitude \rightarrow
       Printf.eprintf \ \verb"SUMMARY:$ $ \bot \& d_{\bot} fusions, $ \bot \& d_{\bot} propagators "
          (F.count\_fusions\ amplitude)\ (F.count\_propagators\ amplitude);
       flush stderr;
       Printf.eprintf \verb|", | % \verb|d| \verb| diagrams" | (F.count\_diagrams | amplitude);
       Printf.eprintf "\n")
     (CF.processes amplitudes);
  let couplings =
     List.fold\_left
       (fun acc p \rightarrow
         let fusions = ThoList.flatmap F.rhs (F.fusions p)
         and brakets = ThoList.flatmap F.ket (F.brakets p) in
         let couplings =
            VSet.of_list (List.map F.coupling (fusions @ brakets)) in
          VSet.union acc couplings)
       VSet.empty (CF.processes amplitudes) in
  let ufo\_couplings =
     VSet.fold
       (fun v acc \rightarrow
         match v with
         | Coupling. Vn (Coupling. UFO (-, v, -, -, -), -, -) \rightarrow
             Sets.String.add v acc
         | \quad acc \rangle
       couplings Sets.String.empty in
  if \neg (Sets.String.is\_empty\ ufo\_couplings) then
     Printf.eprintf
       "SUMMARY: \_ \%d\_UFO\_vertices: \_ \%s \n"
       (Sets.String.cardinal\ ufo\_couplings)
       (String.concat ", " (Sets.String.elements ufo\_couplings))
end;
if !poles then begin
  List.iter
     (fun amplitude \rightarrow
       W.write output_channel "omega" (W.merge (W.trees amplitude)))
     (CF.processes amplitudes)
end;
begin match !dag\theta\_out with
\mid Some name \rightarrow
    let ch = open\_out name in
     List.iter (F.tower_to_dot ch) (CF.processes amplitudes);
     close_out ch
| None \rightarrow ()
end;
begin match !dag\_out with
\mid Some name \rightarrow
    let ch = open\_out name in
     List.iter (F.amplitude_to_dot ch) (CF.processes amplitudes);
     close_out ch
| None \rightarrow ()
end;
begin match !phase_space_out with
\mid Some name \rightarrow
   let selectors =
      let fin, fout = List.hd processes in
      CT.to\_selectors\ (CT.of\_string\_list\ (List.length\ fin\ +\ List.length\ fout)\ !cascades) in
   \mathsf{let}\ ch\ =\ open\_out\ name\ \mathsf{in}
```

```
begin try
               List.iter
                     (fun (fin, fout) \rightarrow
                           Printf.fprintf
                                 ch "%s_{\sqcup}->_{\sqcup}%s_{\sqcup}::\n"
                                 (String.concat `` \ ' \ (List.map M.flavor_to_string fin))
                                 (String.concat `` \ ' \ (List.map M.flavor\_to\_string fout));
                           \mathsf{match}\; \mathit{fin}\; \mathsf{with}\;
                           | \ | \ | \rightarrow
                                    failwith "Omega(): _\phase_\space: _\no_\incoming_\particles"
                                    PHS.phase\_space\_channels
                                          ch
                                          (PHS.amplitude\_sans\_color
                                                   false PHS.no_exclusions selectors fin fout)
                           |[f1; f2]| \rightarrow
                                    PHS.phase\_space\_channels
                                          ch
                                          (PHS.amplitude\_sans\_color
                                                   false PHS.no_exclusions selectors fin fout);
                                    PHS.phase_space_channels_flipped
                                          ch
                                          (PHS.amplitude\_sans\_color
                                                   false PHS.no_exclusions selectors [f2; f1] fout)
                                    failwith "Omega(): _\phase_\space: _\3_\or_\more_\incoming_\particles")
                     processes;
               close_out ch
         with
        | exc \rightarrow
                 begin
                        close_out ch;
                        Printf.eprintf
                              "O'Mega:\_exception\_\%s\_in\_phase\_space\_construction! \\ \label{eq:construction} "O'Mega:\_exception\_space\_construction! \\ \label{eq:construction} "O'Mega:\_exception\_space\_construction" \\ \label{eq:construction} "O'Mega:\_exception\_construction" \\ \label{eq:construction} "O'Mega:\_exception\_construction\_co
                              (Printexc.to\_string\ exc);
                       flush stderr
                  end
        end
| None \rightarrow ()
end;
if !print\_forest then
      List.iter
            (fun amplitude \rightarrow
                  List.iter (fun t \rightarrow Printf.eprintf "%s\n"
                              (Tree.to_string
                                       (Tree.map (fun (wf, \_) \rightarrow variable wf) (fun \_ \rightarrow "") t)))
                        (F.forest\ (List.hd\ (F.externals\ amplitude))\ amplitude))
            (CF.processes amplitudes);
begin match !diagrams\_all with
    Some \ name \rightarrow
      amplitudes_to_feynmf !diagrams_LaTeX name amplitudes
    None \rightarrow ()
end;
begin match !diagrams_sans_color with
\mid Some name \rightarrow
      amplitudes_to_feynmf_sans_color!diagrams_LaTeX name amplitudes
    None \rightarrow ()
end;
begin match !diagrams_color_only with
```

```
\mid Some name \rightarrow
           amplitudes\_to\_feynmf\_color\_only \ ! diagrams\_LaTeX \ name \ amplitudes
        | None \rightarrow ()
         end;
         close_output_channel();
         exit 0
      end
   This was only intended for debugging O'Giga . . .
    let decode \ wf =
      (F.flavor\ wf,\ (F.momentum\ wf\ :\ Momentum.Default.t))
    let diagrams in1 in2 out =
      match F.amplitudes false F.no\_exclusions C.no\_cascades [in1; in2] out with
      \mid \ a \ :: \ \_ \ \rightarrow
           let wf1 = List.hd (F. externals a)
           and wf2 = List.hd (List.tl (F.externals a)) in
           let wf2 = decode \ wf2 in
           List.map (fun t \rightarrow
             (wf2,
              Tree.map (fun (wf, \_) \rightarrow decode wf) decode t))
             (F.forest \ wf1 \ a)
      | [] \rightarrow []
    let diagrams in 1 in 2 out =
      failwith "Omega().diagrams: disabled"
  end
module\ Binary\ (TM\ :\ Target.Maker)\ (M\ :\ Model.T)\ =
  Make(Fusion.Binary)(Fusion.Helac\_Binary)(TM)(M)
module\ Binary\_Majorana\ (TM\ :\ Target.Maker)\ (M\ :\ Model.T)\ =
  Make(Fusion.Binary\_Majorana)(Fusion.Helac\_Binary\_Majorana)(TM)(M)
Mixed 23 (TM : Target.Maker) (M : Model.T) =
  Make(Fusion.Mixed23)(Fusion.Helac\_Mixed23)(TM)(M)
module\ Mixed23\_Majorana\ (TM\ :\ Target.Maker)\ (M\ :\ Model.T)\ =
  Make(Fusion.Mixed23\_Majorana)(Fusion.Helac\_Mixed23\_Majorana)(TM)(M)
module\ Mixed23\_Majorana\_vintage\ (TM\ :\ Target.Maker)\ (M\ :\ Model.T) =
  Make(Fusion\_vintage.Mixed23\_Majorana)(Fusion.Helac\_Mixed23\_Majorana)(TM)(M)
module\ Bound\ (M\ :\ Model.\ T)\ :\ Tuple.Bound\ =
  struct
    (*
   Above max\_degree = 6, the performance drops dramatically!
    let max\_arity() =
      pred\ (M.max\_degree\ ())
Model. T module Nary (TM : Target. Maker) (M : Model. T) =
  Make(Fusion.Nary(Bound(M)))(Fusion.Helac(Bound(M)))(TM)(M)
Model Nary_Majorana (TM : Target.Maker) (M : Model.T) =
  Make(Fusion.Nary\_Majorana(Bound(M)))(Fusion.Helac\_Majorana(Bound(M)))(TM)(M)
                           18.4 Implementation of Omega_QED
Modellib\_SM.QED) module O = Omega.Binary(Targets.Fortran)(Modellib\_SM.QED)
let _{-} = O.main ()
```

18.5 Implementation of Omega_SM

 $\label{eq:module} \begin{array}{lll} \mathsf{module} \ O &= Omega.Mixed23 (\ Targets.Fortran) (Modellib_SM.SM (Modellib_SM.SM_no_anomalous)) \\ \mathsf{let} \ _ &= O.main \ () \end{array}$

18.6 Implementation of Omega_SYM

```
module SYM =
  struct
     open Coupling
     let \ options = Options.empty
     let \ caveats \ () = []
     let <math>nc = 3
     type flavor =
          Q 	ext{ of } int \mid SQ 	ext{ of } int
          G 	ext{ of } int \mid SG 	ext{ of } int
        | Phi
     let generations = ThoList.range 1 1
     let generations\_pairs =
       List.map
          (function [a;b] \rightarrow (a, b)
            | _ → failwith "omega_SYM.generations_pairs")
          (Product.power 2 generations)
     let generations\_triples =
        List.map
          (function [a; b; c] \rightarrow (a, b, c)
            | \_ \rightarrow failwith "omega\_SYM.generations\_triples")
          (Product.power 3 generations)
     let generations_quadruples =
        List.map
          (function [a; b; c; d] \rightarrow (a, b, c, d)
            \perp \rightarrow failwith "omega_SYM.generations_quadruples")
          (Product.power 4 generations)
     let external_flavors () =
       [ "Quarks", List.map (fun i \rightarrow Q i) generations;
          "Anti-Quarks", List.map (fun i \rightarrow Q(-i)) generations;
          "SQuarks", List.map (fun i \rightarrow SQ i) generations;
          "Anti-SQuarks", List.map (fun i \rightarrow SQ (-i)) generations;
          "Gluons", List.map (fun i \rightarrow G i) generations;
          "SGluons", List.map (fun i \rightarrow SG i) generations;
          "Other", [Phi]]
     let flavors() =
        ThoList.flatmap snd (external_flavors ())
     type \ gauge = unit
     type \ constant =
         G-saa of int \times int
          G\_saaa of int \times int \times int
          G3 of int \times int \times int
          I\_G3 of int \times int \times int
        | G4 of int \times int \times int \times int
     type \ orders = unit
     let \ orders = function
       |  \rightarrow  ()
```

```
let \ lorentz = function
  | Q i \rightarrow
       if i > 0 then
          Spinor
        else if i < 0 then
           ConjSpinor
           invalid\_arg "SYM.lorentz_{\sqcup}(Q_{\sqcup}0)"
    SQ - \mid Phi \rightarrow Scalar
    G \_ \to Vector
   SG \_ \to Majorana
\mathsf{let}\ \mathit{color}\ =\ \mathsf{function}
  | Qi | SQi \rightarrow
        Color.SUN (if i > 0 then nc else if i < 0 then -nc else invalid\_arg "SYM.color_{\sqcup}(\mathbb{Q}_{\sqcup}0)")
     G = | SG = \rightarrow Color.AdjSUN \ nc
   \mid Phi \rightarrow Color.Singlet
let nc () = nc
let propagator = function
  | Q i \rightarrow
       if i > 0 then
          Prop_Spinor
        else if i < 0 then
           Prop\_ConjSpinor
        else
           invalid_arg "SYM.lorentz (Q (Q () )"
    SQ - \mid Phi \rightarrow Prop\_Scalar
     G \_ \to Prop\_Feynman
    SG \_ \to Prop\_Majorana
let width _{-} = Timelike
let goldstone _ = None
let conjugate = function
     Q i \rightarrow Q (-i)
     SQ i \rightarrow SQ (-i)
  \mid (G \perp \mid SG \perp \mid Phi) \text{ as } p \rightarrow p
let fermion = function
  | Q i \rightarrow
       if i > 0 then
        else if i < 0 then
          -1
          invalid\_arg "SYM.fermion_{\sqcup}(Q_{\sqcup}0)"
   \mid SQ \mid G \mid Phi \rightarrow 0
   \mid SG \_ \rightarrow 2
module Ch = Charges.Null
let charges _{-} = ()
module F = Modeltools.Fusions (struct)
  \mathsf{type}\ f\ =\ \mathit{flavor}
  type c = constant
  let \ compare = compare
  let \ conjugate = conjugate
end)
let quark\_current =
  List.map
     (\mathsf{fun}\ (i,\ j,\ k)\ \to
        ((Q(-i), Gj, Qk), FBF(-1, Psibar, V, Psi), G3(i, j, k)))
     generations\_triples
```

```
let squark\_current =
   List.map
     (\text{fun }(i, j, k) \rightarrow
        ((G j, SQ i, SQ (-k)), Vector\_Scalar\_Scalar 1, G3 (i, j, k)))
     generations\_triples
let three\_gluon =
   List.map
     (\mathsf{fun}\ (i,\ j,\ k)\ \to\ 
        ((G i, G j, G k), Gauge\_Gauge\_Gauge\_1, I\_G3 (i, j, k)))
     generations\_triples
let gluon2-phi =
   List.map
     (fun (i, j) \rightarrow
        ((Phi, G i, G j), Dim5\_Scalar\_Gauge2 1, G\_saa(i, j)))
     generations\_pairs
let vertices3 =
   quark_current @ squark_current @ three_gluon @ gluon2_phi
let gauge4 = Vector4 [(2, C_13_42); (-1, C_12_34); (-1, C_14_23)]
let squark\_seagull =
   List.map
     (\mathsf{fun}\;(i,\;j,\;k,\;l)\;\to\;
        ((SQ\ i,\ SQ\ (-j),\ G\ k,\ G\ l),\ Scalar2\_Vector2\ 1,\ G4\ (i,\ j,\ k,\ l)))
    generations\_quadruples
let four\_gluon =
   List.map
     (\text{fun }(i, j, k, l) \rightarrow
        ((G \ i, \ G \ j, \ G \ k, \ G \ l), \ gauge 4, \ G4 \ (i, \ j, \ k, \ l)))
    generations\_quadruples
We need at least a Dim6\_Scalar\_Gauge3 vertex to support this.
let gluon3-phi =
   let vertices4 =
   squark\_seagull @ four\_gluon @ gluon3\_phi
let \ vertices () =
   (vertices3, vertices4, [])
let table = F.of\_vertices (vertices ())
let fuse2 = F.fuse2 table
let fuse3 = F.fuse3 table
let fuse = F.fuse \ table
let max\_degree () = 4
let parameters () = { input = []; derived = []; derived_arrays = [] }
let invalid_flavor s =
   invalid_arg ("omega_SYM.flavor_of_string:__" ^ s)
let flavor\_of\_string \ s =
   let l = String.length s in
   if l < 2 then
     invalid\_flavor s
   else if l\ =\ 2 then
     if String.sub \ s \ 0 \ 1 = "q" then
        Q (int\_of\_string (String.sub \ s \ 1 \ 1))
     else if String.sub \ s \ 0 \ 1 \ = \ "Q" then
        Q (-(int\_of\_string (String.sub \ s \ 1 \ 1)))
     else if String.sub \ s \ 0 \ 1 \ = \ "g" then
```

```
G(int\_of\_string(String.sub\ s\ 1\ 1))
                   else
                             invalid\_flavor\ s
          else if l = 3 then
                   if s = "phi" then
                             Phi
                   else if String.sub \ s \ 0 \ 2 \ = \ "sq" then
                              SQ\ (int\_of\_string\ (String.sub\ s\ 2\ 1))
                    else if String.sub \ s \ 0 \ 2 \ = \ "sQ" then
                             SQ (-(int\_of\_string (String.sub \ s \ 2 \ 1)))
                   else if String.sub \ s \ 0 \ 2 \ = \ "sg" then
                              SG (int\_of\_string (String.sub \ s \ 2 \ 1))
                    else
                              invalid\_flavor\ s
          else
                    invalid_flavor s
let flavor\_to\_string = function
          |Qi \rightarrow
                            if i > 0 then
                                       "q" ^ string\_of\_int\ i
                             else if i\ <\ 0 then
                                       "Q" \hat{string\_of\_int} (-i)
                                       invalid\_arg "SYM.flavor_to_string_{\sqcup}(Q_{\sqcup}0)"
          \mid SQ i \rightarrow
                            if i > 0 then
                                       "sq" \hat{string\_of\_int} i
                             \  \, {\rm else} \,\, {\rm if} \,\, i \,\, < \,\, 0 \,\, {\rm then} \,\,
                                       "sQ" \ \hat{\ } string\_of\_int \ (-i)
                            else
                                       invalid\_arg "SYM.flavor_to_string\sqcup(SQ\sqcup0)"
                  G~i~\rightarrow "g" \hat{}~string\_of\_int~i
                  SG~i~\rightarrow "sg" \hat{}~string\_of\_int~i
                 Phi \rightarrow "phi"
let flavor_to_TeX = function
                  Q \ i \ 	o
                            if i > 0 then
                                       "q_{-}\{" \ \hat{\ } string\_of\_int \ i \ \hat{\ } "\}"
                             else \bar{i} \bar{i} < 0 then
                                       "{\color{bar} \downarrow q}_{-}{\color{bar} \downarrow q}_{-}{\color{bar} \downarrow q}_{-} string\_of\_int (-i) ^ "}"
                                       invalid\_arg "SYM.flavor_to_string_{\sqcup}(Q_{\sqcup}0)"
          \mid SQ i \rightarrow
                             if i > 0 then
                                       \{\tilde{i}_{q}\}_{q}^{-1} : string_{q}^{-1} : string
                             else if i < 0 then
                                        "{\hat{-i}^{-1}}_{-1} = \frac{1}{2} \left\{ \text{$i$ ind $-i$ int $(-i)$ $^{-i}$ } \right\} = \frac{1}{2} \left\{ \text{$i$ ind $i$ ind $i$ in $i$
                            else
                                       invalid\_arg "SYM.flavor_to_string_{\sqcup}(SQ_{\sqcup}0)"
           G i \rightarrow "g_{\{}" \hat{string}_{o} f_{i} int i \hat{s}" \}"
                  SG \ i \rightarrow \text{"{}} \text{tilde}_{\sqcup}g}_{-}\{\text{"`} string\_of\_int i`\text{"}}\text{"}
           \mid Phi \rightarrow "phi"
let flavor\_symbol = function
          | Q i \rightarrow
                            \quad \text{if } i \ > \ 0 \ \text{then} \\
                                       "q" \hat{string\_of\_int} i
                             else if i < 0 then
                                        "qbar" \hat{\ } string_of_int (-i)
                             else
                                       invalid\_arg "SYM.flavor_to_string_{\sqcup}(Q_{\sqcup}0)"
```

```
\mid SQ i \rightarrow
             if i > 0 then
                "sq" \hat{\ } string_of_int i
             \  \, {\rm else} \,\, {\rm if} \,\, i \,\, < \,\, 0 \,\, {\rm then} \,\,
                "sqbar" \hat{\ } string_of_int (-i)
                invalid\_arg "SYM.flavor_to_string_{\sqcup}(SQ_{\sqcup}0)"
        |~~G~i~~\rightarrow~~ \text{"g"}~\hat{}~~string\_of\_int~i
          SG~i~\rightarrow "sg" \hat{}~string\_of\_int~i
        \mid Phi \rightarrow "phi"
     let gauge\_symbol() =
        failwith "omega_SYM.gauge_symbol:\sqcupinternal\sqcuperror"
     \texttt{let}\ mass\_symbol\ \_\ =\ \texttt{"O.O\_default"}
     let width_symbol _ = "0.0_default"
     let string\_of\_int\_list int\_list =
        "(" ^ String.concat "," (List.map string_of_int int_list) ^ ")"
     let constant\_symbol = function
          G\_saa\ (i,\ j)\ 	o\ "g\_saa"\ \hat{\ }string\_of\_int\_list\ [i;j]
          G\_saaa~(i,~j,~k)~\rightarrow~\texttt{"g\_saaa"}~\hat{}~string\_of\_int\_list~[i;j;k]
          G3\ (i,\ j,\ k) \rightarrow \text{"g3"}\ \hat{}\ string\_of\_int\_list\ [i;j;k]
          I\_G3 (i, j, k) \rightarrow "ig3" ^ string\_of\_int\_list [i; j; k]
        end
Module O = Omega.Mixed23(Targets.Fortran\_Majorana)(SYM)
let _{-} = O.main ()
```

ACKNOWLEDGEMENTS

We thank Mauro Moretti for fruitful discussions of the ALPHA algorithm [1], that inspired our solution of the double counting problem.

We thank Wolfgang Kilian for providing the WHIZARD environment that turns our numbers into real events with unit weight. Thanks to the ECFA/DESY workshops and their participants for providing a showcase. Thanks to Edward Boos for discussions in Kaluza-Klein gravitons.

This research is supported by Bundesministerium für Bildung und Forschung, Germany, (05 HT9RDA) and Deutsche Forschungsgemeinschaft (MA 676/6-1).

Thanks to the Caml and Objective Caml teams from INRIA for the development and the lean and mean implementation of a programming language that does not insult the programmer's intelligence.

BIBLIOGRAPHY

- [1] F. Caravaglios, M. Moretti, Z. Phys. C74 (1997) 291.
- [2] A. Kanaki, C. Papadopoulos, DEMO-HEP-2000/01, hep-ph/0002082, February 2000.
- [3] Xavier Leroy, The Objective Caml system, documentation and user's guide, Technical Report, INRIA, 1997.
- [4] Chris Okasaki, Purely Functional Data Structures, Cambridge University Press, 1998.
- [5] H. Murayama, I. Watanabe, K. Hagiwara, KEK Report 91-11, January 1992.
- [6] T. Stelzer, W.F. Long, Comput. Phys. Commun. 81 (1994) 357.
- [7] A. Denner, H. Eck, O. Hahn and J. Küblbeck, Phys. Lett. **B291** (1992) 278; Nucl. Phys. **B387** (1992) 467.
- [8] V. Barger, A. L. Stange, R. J. N. Phillips, Phys. Rev. **D45**, (1992) 1751.
- [9] T. Ohl, Lord of the Rings, (Computer algebra library for O'Caml, unpublished).
- [10] T. Ohl, Bocages, (Feynman diagram library for O'Caml, unpublished).
- [11] W. Kilian, WHIZARD, University of Karlsruhe, 2000.
- [12] E. E. Boos, T. Ohl, Phys. Rev. Lett. 83 (1999) 480.
- [13] T. Han, J. D. Lykken and R. Zhang, Phys. Rev. **D59** (1999) 105006 [hep-ph/9811350].
- [14] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, Second Edition, Cambridge University Press, 1992.
- [15] P. Cvitanović, Phys. Rev. **D14** (1976) 1536.
- [16] W. Kilian, T. Ohl, J. Reuter and C. Speckner, JHEP 1210 (2012) 022 [arXiv:1206.3700 [hep-ph]].
- [17] C. Degrande, C. Duhr, B. Fuks, D. Grellscheid, O. Mattelaer and T. Reiter, Comput. Phys. Commun. **183** (2012), 1201-1214 doi:10.1016/j.cpc.2012.01.022 [arXiv:1108.2040 [hep-ph]].

AUTOTOOLS

A.1 Interface of Config

 $\mathsf{val}\ \mathit{version}\ :\ \mathit{string}$ $\mathsf{val}\ date\ :\ string$ $\mathsf{val}\ status\ :\ string$

 ${\tt val} \ default_UFO_dir \ : \ string$

 $\mathsf{val}\ openmp\ :\ bool$



Implementation config.ml unavailable!

—B— TEXTUAL OPTIONS

B.1 Interface of Options

```
type t
 val\ empty : t
 val\ create\ :\ (string \times Arg.spec\ \times\ string)\ list \rightarrow\ t
 val\ extend: t \rightarrow (string \times Arg.spec \times string)\ list \rightarrow t
 \mathsf{val}\ cmdline\ :\ string 	o\ t\ 	o\ (string 	imes Arg.spec\ 	imes\ string)\ list
 This is a clone of Arg.parse with a delayed usage string.
 val parse : (string \times Arg.spec \times string) \ list \rightarrow
    (string \rightarrow unit) \rightarrow (unit \rightarrow string) \rightarrow unit
                                       B.2 Implementation of Options
 module A = Map.Make (struct type t = string let compare = compare end)
 type t =
      \{ actions : Arg.spec A.t; 
         raw : (string \times Arg.spec \times string) \ list \}
 let empty = \{ actions = A.empty; raw = [] \}
 {\sf let} \ extend \ old \ options \ =
    \{ actions = List.fold\_left \}
         (fun a(s, f, \bot) \rightarrow A.add \ s \ f \ a) old.actions options;
       raw = options @ old.raw }
 \mathsf{let}\ create\ =\ extend\ empty
 let cmdline prefix options =
    \textit{List.map} \ (\mathsf{fun} \ (o, \ f, \ d) \ \rightarrow \ (\textit{prefix} \ \widehat{\ } o, \ f, \ d)) \ \textit{options.raw}
Starting with O'Caml version 3.12.1 we can provide a better help* option using Arg.usage\_string. We can finally do this!
     finally do this!
 let parse specs anonymous usage =
   let \ help \ () =
      raise (Arg.Help (usage ())) in
      [("-usage", Arg. Unit help, "Display the external particles");
        ("--usage", \mathit{Arg.Unit\ help}, "Display \sqcup the \sqcup external \sqcup particles")] @ \mathit{specs} in
   try
       Arg.parse_argv Sys.argv specs' anonymous (usage ())
      Arg.Bad\ msg\ 	o\ Printf.eprintf "%s\n" msg;\ exit\ 2;
    |Arg.Help.msg| \rightarrow Printf.printf "%s\n" msg; exit 0
```

—C— PROGRESS REPORTS

C.1 Interface of Progress

```
\begin{array}{l} \text{val } dummy \ : \ t \\ \text{val } channel \ : \ out\_channel \ \rightarrow \ int \ \rightarrow \ t \\ \text{val } file \ : \ string \ \rightarrow \ int \ \rightarrow \ t \\ \text{val } open\_file \ : \ string \ \rightarrow \ int \ \rightarrow \ t \\ \text{val } reset \ : \ t \ \rightarrow \ int \ \rightarrow \ string \ \rightarrow \ unit \\ \text{val } begin\_step \ : \ t \ \rightarrow \ string \ \rightarrow \ unit \\ \text{val } end\_step \ : \ t \ \rightarrow \ string \ \rightarrow \ unit \\ \text{val } summary \ : \ t \ \rightarrow \ string \ \rightarrow \ unit \\ \end{array}
```

C.2 Implementation of Progress

```
type channel =
    Channel\ {\it of}\ out\_channel
    File of string
   Open\_File 	ext{ of } string 	imes out\_channel
\mathsf{type}\ state\ =
    { channel : channel;
       mutable steps: int;
       mutable digits : int;
       mutable step : int;
       created : float;
       mutable last\_reset : float;
       mutable last\_begin : float; }
\mathsf{type}\ t\ =\ \mathit{state}\ \mathit{option}
let digits n =
  if n > 0 then
     succ (truncate (log10 (float n)))
     invalid\_arg "Progress.digits: unon-positive argument"
let mod\_float2 \ a \ b =
  let modulus = mod\_float \ a \ b \ in
  ((a - . modulus) / . b, modulus)
{\tt let} \ time\_to\_string \ seconds \ = \\
  let minutes, seconds = mod\_float2 seconds 60. in
  if minutes > 0.0 then
    let hours, minutes = mod\_float2 minutes 60. in
    if hours > 0.0 then
       let days, hours = mod\_float2 hours 24. in
       if days > 0.0 then
          Printf.sprintf "%.0f:%02.0f days days hours
```

```
else
          Printf.sprintf "%.0f:%02.0f hrs" hours minutes
     else
        Printf.sprintf "%.0f:%02.0f∟mins" minutes seconds
  else
     Printf.sprintf "%.2f⊔secs" seconds
let \ create \ channel \ steps =
  let now = Sys.time () in
  Some \{ channel = channel; 
            steps = steps;
            digits \ = \ digits \ steps;
            step = 0;
            created = now;
            last\_reset = now;
            last\_begin = now }
let dummy =
  None
let channel oc =
  create (Channel oc)
let file name =
  let oc = open\_out name in
  close_out oc;
  create (File name)
let \ open\_file \ name \ =
  let oc = open\_out name in
  create (Open_File (name, oc))
let \ close\_channel \ state =
  match state.channel with
  \mid Channel \ oc \rightarrow
       flush oc
     File \ \_ \ \rightarrow \ ()
    Open\_File (\_, oc) \rightarrow
       flush \ oc;
        close\_out oc
let use\_channel state f =
  match state.channel with
  | Channel \ oc \ | \ Open\_File \ (\_, \ oc) \ \rightarrow
       f \ oc;
       flush oc
  \mid File name \rightarrow
       let oc = open\_out\_gen [Open\_append; Open\_creat] 644_8 name in
       flush \ oc;
        close\_out\ oc
let \ reset \ state \ steps \ msg \ =
  match state with
     None \rightarrow ()
   Some \ state \rightarrow
       let now = Sys.time() in
       state.steps \leftarrow steps;
       state.digits \leftarrow digits steps;
       state.step \leftarrow 0;
       state.last\_reset \leftarrow now;
       state.last\_begin \leftarrow now
let \ begin\_step \ state \ msg =
  \mathsf{match}\ \mathit{state}\ \mathsf{with}
  | None \rightarrow ()
```

```
\mid Some \ state \rightarrow
        let now = Sys.time() in
        state.step \ \leftarrow \ succ \ state.step;
        state.last\_begin \leftarrow now;
        use\_channel\ state\ (fun\ oc\ 
ightarrow
            Printf.fprintf\ oc\ "[\%0*d]_{\%0*d}..."\ state.digits\ state.step\ state.digits\ state.steps\ msg)
\mathsf{let}\ end\_step\ state\ msg\ =
   \mathsf{match}\ \mathit{state}\ \mathsf{with}
     None \rightarrow ()
     Some\ state\ 	o
        let now = Sys.time() in
        let \ last = now - . \ state.last\_begin \ in
        \mathsf{let}\ elapsed\ =\ now\ -\ .\ state.last\_reset\ \mathsf{in}
        let estimated = float state.steps * . elapsed / . float state.step in
        \mathsf{let}\ \mathit{remaining}\ =\ \mathit{estimated}\ -.\ \mathit{elapsed}\ \mathsf{in}
        use\_channel\ state\ (fun\ oc\ 	o
            Printf.fprintf\ oc\ "$\sqcup \%s.$ [time:$\sqcup \%s,$\sqcup total:$\sqcup \%s,$\sqcup remaining:$\sqcup \%s] \n" msg
               (time_to_string last) (time_to_string estimated) (time_to_string remaining))
let summary state msg =
  match state with
     None \rightarrow ()
    Some \ state \rightarrow
        let now = Sys.time () in
         use\_channel\ state\ (fun\ oc\ 
ightarrow
            Printf.fprintf\ oc\ "\%s._{\square}[total_{\square}time:_{\square}\%s]\n"\ msg
               (time\_to\_string\ (now\ -.\ state.created)));
         close\_channel\ state
```

——D— More on Filenames

D.1 Interface of ThoFilename

```
val split: string \rightarrow string \ list
val join: string \ list \rightarrow string
val expand\_home: string \rightarrow string
```

D.2 Implementation of ThoFilename

```
let rec split' acc path =
  match Filename.dirname path, Filename.basename path with
     "/", basename \rightarrow "/" :: basename :: acc
     ".", basename \rightarrow basename :: acc
  | dirname, basename \rightarrow split' (basename :: acc) dirname
let split path =
  split'\ [\ ]\ path
let join = function
  |~[]~\rightarrow~\text{"."}
    [basename] \rightarrow basename
  | dirname :: rest \rightarrow List.fold_left Filename.concat dirname rest
let expand\_home path =
  match split path with
  | ("~" | "$HOME" | "${HOME}}") :: rest \rightarrow
       join~(({\sf try}~Sys.getenv~"HOME"~with~Not\_found~\rightarrow~"/{\sf tmp"})~::~rest)
  - \rightarrow path
```

—E— Cache Files

E.1 Interface of Cache

```
module type T =
   sig
      type key
      type hash = string
      type \ value
      type \alpha \ result =
          | Hit of \alpha
            Miss
            Stale of string
      exception Mismatch of string \times string \times string
      \mathsf{val}\ \mathit{hash}\ :\ \mathit{key}\ \to\ \mathit{hash}
      \mathsf{val}\ exists\ :\ hash\ \to\ string\ \to\ bool
      \mathsf{val}\ \mathit{find}\ :\ \mathit{hash}\ 	o\ \mathit{string}\ 	o\ \mathit{string}\ \mathit{option}
      val write : hash \rightarrow string \rightarrow value \rightarrow unit
      \mathsf{val} \ write\_dir \ : \ hash \ \to \ string \to \ string \to \ value \ \to \ unit
      \mathsf{val}\ \mathit{read}\ :\ \mathit{hash}\ \to\ \mathit{string}\ \to\ \mathit{value}
      val\ maybe\_read : hash \rightarrow string \rightarrow value\ result
   end
module type Key =
   sig
      type t
   end
module type Value =
   sig
      type t
   \quad \text{end} \quad
module Make (Key: Key) (Value: Value):
       T with type key = Key.t and type value = Value.t
```

E.2 Implementation of Cache

```
\begin{array}{ll} \text{let } search\_path &= \\ [ \ Filename.current\_dir\_name \ ] \\ \text{module type } T &= \\ \text{sig} \\ \text{type } key \\ \text{type } hash &= string \\ \text{type } value \\ \\ \text{type } \alpha \ result &= \\ \end{array}
```

```
Hit of \alpha
          Miss
          Stale of string
     exception \mathit{Mismatch} of \mathit{string} \times \mathit{string} \times \mathit{string}
     val hash : key \rightarrow hash
     val\ exists\ :\ hash\ 	o\ string\ 	o\ bool
     \mathsf{val}\ \mathit{find}\ :\ \mathit{hash}\ 	o\ \mathit{string}\ 	o\ \mathit{string}\ \mathit{option}
     val write : hash \rightarrow string \rightarrow value \rightarrow unit
     val\ write\_dir\ :\ hash\ 	o\ string\ 	o\ string\ 	o\ value\ 	o\ unit
     val\ read\ :\ hash\ 	o\ string\ 	o\ value
     val\ maybe\_read : hash \rightarrow string \rightarrow value\ result
module type Key =
  sig
     type t
  end
module type Value =
  sig
     type t
  end
Make (Key : Key) (Value : Value) =
     \mathsf{type}\ key\ =\ Key.t
     type hash = string
     type \ value = Value.t
     type \ tagged =
           \{ tag : hash; 
             value : value; }
     let \ hash \ value =
        Digest.string (Marshal.to_string value [])
     let find_first path name =
        let rec find\_first' = function
           [] \rightarrow raise\ Not\_found
           | dir :: path \rightarrow
                let f = Filename.concat dir name in
                if Sys.file\_exists f then
                   f
                else
                   find_first' path
        find_first' path
     let find hash name =
        try Some (find_first search_path name) with Not_found → None
     let exists hash name =
        match find hash name with
          None \rightarrow \mathsf{false}
         Some \ \_ \ 	o \ \mathsf{true}
     let try_first f path name =
        let rec try\_first' = function
          [] \rightarrow raise\ Not\_found
            dir :: path \rightarrow
                try (f (Filename.concat dir name), dir) with \rightarrow try\_first' path
        in
        try_first' path
```

```
let open_in_bin_first = try_first open_in_bin
  let open_out_bin_last path = try_first open_out_bin (List.rev path)
  {\sf let} \ write \ hash \ name \ value \ =
     let oc, \_ = open\_out\_bin\_last search\_path name in
     Marshal.to\_channel\ oc\ \{\ tag\ =\ hash;\ value\ =\ value\ \}\ [\ ];
     close\_out\ oc
  {\tt let} \ write\_dir \ hash \ dir \ name \ value \ =
     let oc = open\_out\_bin (Filename.concat dir name) in
     Marshal.to\_channel\ oc\ \{\ tag\ =\ hash;\ value\ =\ value\ \}\ [\ ];
      close\_out\ oc
  \mathsf{type} \,\, \alpha \,\, \mathit{result} \,\, = \,\,
        Hit of \alpha
        Miss
       Stale of string
  exception Mismatch of string \times string \times string
  let read hash name =
     \mathsf{let}\ \mathit{ic},\ \mathit{dir}\ =\ \mathit{open\_in\_bin\_first}\ \mathit{search\_path}\ \mathit{name}\ \mathsf{in}
     let { tag = tag; value = value } = Marshal.from\_channel ic in
     close\_in\ ic;
     if tag = hash then
        value
     else
        raise (Mismatch (Filename.concat dir name, hash, tag))
  let maybe\_read hash name =
     try
        Hit (read hash name)
     with
       Not\_found \rightarrow Miss
      | Mismatch (file, \_, \_) \rightarrow Stale file
end
```

——F— More On Lists

F.1 Interface of ThoList

 $splitn \ n \ l = (hdn \ l, \ tln \ l)$, but more efficient.

 $\begin{array}{lll} \mathsf{val} \ hdn \ : \ int \to \ \alpha \ list \to \ \alpha \ list \\ \mathsf{val} \ tln \ : \ int \to \ \alpha \ list \to \ \alpha \ list \end{array}$

 $\mathsf{val} \ \mathit{splitn} \ : \ \mathit{int} \rightarrow \ \alpha \ \mathit{list} \rightarrow \ \alpha \ \mathit{list} \times \alpha \ \mathit{list}$

 $split_last (l @ [a]) = (l, a)$

 $\mathsf{val}\ split_last\ :\ \alpha\ list\ \rightarrow\ \alpha\ list\ \times\ \alpha$

chop $n \mid l$ chops l into pieces of size n (except for the last one, which contains th remainder).

 $\mathsf{val}\ chopn\ :\ int \to\ \alpha\ list \to\ \alpha\ list\ list$

 $cycle_until\ a\ l$ finds a member a in the list l and returns the cyclically permuted list with a as head. Raises Not_found if a is not in l.

 $val\ cycle_until\ :\ \alpha\ o\ \alpha\ list\ o\ \alpha\ list$

cycle n l cyclically permute the list l by $n \geq 0$ positions. Raises Not-found List.length l > n. NB: cycle n l @ hdn n l, but more efficient.

 $\mathsf{val}\ \mathit{cycle}\ :\ \mathit{int}\ \rightarrow\ \alpha\ \mathit{list}\ \rightarrow\ \alpha\ \mathit{list}$

of $_subarray\ n\ m\ a$ is $[a.(n);a.(n+1);\ldots;a.(m)]$. Values of n and m out of bounds are silently shifted towards these bounds.

 $\mathsf{val}\ of_subarray\ :\ int \to\ int \to\ \alpha\ array \to\ \alpha\ list$

range $s \ n \ m \ is \ [n; n+s; n+2s; \ldots; m-((m-n) \ mod \ s)]$

 $\mathsf{val}\ \mathit{range}\ :\ ?\mathit{stride}: \mathit{int} \to \ \mathit{int} \to \ \mathit{int} \to \ \mathit{int} \ \mathit{list}$

enumerate s n [a1; a2; ...] is [(n, a1); (n + s, a2); ...]

val enumerate : ?stride :int \rightarrow int \rightarrow α list \rightarrow (int $\times \alpha$) list

alist_of_list ~predicate ~offset list takes the elements of list that satisfy predicate and forms a list of pairs of an offset into the original list and the element with the offsets starting from offset. NB: the order of the returned alist is not specified!

 $val \ alist_of_list :$

```
?predicate : (\alpha \rightarrow bool) \rightarrow ?offset : int \rightarrow \alpha list \rightarrow (int \times \alpha) list
```

Compress identical elements in a sorted list. Identity is determined using the polymorphic equality function Pervasives.(=).

val $uniq : \alpha \ list \rightarrow \alpha \ list$

Test if all members of a list are structurally identical (actually homogeneous l and List.length (uniq l) ≤ 1 are equivalent, but the former is more efficient if a mismatch comes early).

val $homogeneous: \alpha list \rightarrow bool$

If all elements of the list l appear exactly twice, $pairs\ l$ returns a sorted list with these elements appearing once. Otherwise $Invalid_argument$ is raised.

val $pairs : \alpha \ list \rightarrow \alpha \ list$

compare cmp l1 l2 compare two lists l1 and l2 according to cmp. cmp defaults to the polymorphic Pervasives.compare.

val compare :
$$?cmp : (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha list \rightarrow \alpha list \rightarrow int$$

Collect and count identical elements in a list. Identity is determined using the polymorphic equality function Pervasives.(=). classify does not assume that the list is sorted. However, it is O(n) for sorted lists and $O(n^2)$ in the worst case.

```
val classify: \alpha \ list \rightarrow \ (int \times \alpha) \ list
```

Collect the second factors with a common first factor in lists.

val
$$factorize : (\alpha \times \beta) \ list \rightarrow (\alpha \times \beta \ list) \ list$$

 $flatmap\ f$ is equivalent to $flatten \circ (map\ f)$, but more efficient, because no intermediate lists are built. Unfortunately, it is not tail recursive.

```
\mathsf{val}\ \mathit{flatmap}\ :\ (\alpha\ \to\ \beta\ \mathit{list})\ \to\ \alpha\ \mathit{list}\to\ \beta\ \mathit{list}
```

 $rev_flatmap\ f$ is equivalent to $flatten \circ (rev_map\ (rev \circ f)) = rev \circ (flatmap\ f)$, but more efficient, because no intermediate lists are built. It is tail recursive.

```
\mathsf{val}\ \mathit{rev\_flatmap}\ :\ (\alpha\ \to\ \beta\ \mathit{list})\ \to\ \alpha\ \mathit{list} \to\ \beta\ \mathit{list}
```

clone a n builds a list from n copies of the element a.

```
val clone: \alpha \rightarrow int \rightarrow \alpha \ list
```

multiply n l concatenates n copies of the list l.

```
val multiply : int \rightarrow \alpha \ list \rightarrow \alpha \ list
```

filtermap f l applies f to each element of l and drops the results None.

$$\mathsf{val}\ \mathit{filtermap}\ :\ (\alpha\ \to\ \beta\ \mathit{option})\ \to\ \alpha\ \mathit{list}\ \to\ \beta\ \mathit{list}$$

power a_{-} list computes the list of all sublists of a_{-} list, i.e. the power set. The elements of the sublists are not required to have been sequential in a_list .

```
val power: \alpha \ list \rightarrow \alpha \ list \ list
```



(2) Invent other names to avoid confusions with List.fold_left2 and List.fold_right2.

```
val fold\_right2: (\alpha \rightarrow \beta \rightarrow \beta) \rightarrow \alpha \ list \ list \rightarrow \beta \rightarrow \beta
\mathsf{val}\ \mathit{fold\_left2}\ :\ (\beta\ \rightarrow\ \alpha\ \rightarrow\ \beta)\ \rightarrow\ \beta\ \rightarrow\ \alpha\ \mathit{list}\ \mathit{list}\ \rightarrow\ \beta
```

iteri f n [a; b; c] evaluates f n a, f (n + 1) b and f (n + 2) c.

```
\mathsf{val}\ iteri\ :\ (int\rightarrow\ \alpha\ \rightarrow\ unit)\ \rightarrow\ int\rightarrow\ \alpha\ list\rightarrow\ unit
\mathsf{val}\ \mathit{mapi}\ :\ (\mathit{int}\ \rightarrow\ \alpha\ \rightarrow\ \beta)\ \rightarrow\ \mathit{int}\ \rightarrow\ \alpha\ \mathit{list}\ \rightarrow\ \beta\ \mathit{list}
```

 $iteri2\ f\ n\ m\ [[aa;ab];[ba;bb]]$ evaluates $f\ n\ m\ aa,\ f\ n\ (m+1)\ ab,\ f\ (n+1)\ m\ ba\ and\ f\ (n+1)\ (m+1)\ bb.$ NB: the nested lists need not be rectangular.

```
\textit{val iteri2} \; : \; (int \rightarrow \; int \rightarrow \; \alpha \; \rightarrow \; unit) \; \rightarrow \; int \rightarrow \; int \rightarrow \; \alpha \; list \; list \rightarrow \; unit
```

Just like *List.map3*:

val
$$map3$$
 : $(\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \delta) \rightarrow \alpha \; list \rightarrow \beta \; list \rightarrow \gamma \; list \rightarrow \delta \; list$

Transpose a rectangular list of lists like a matrix.

```
val transpose: \alpha list list <math>\rightarrow \alpha list list
```

interleave f list walks through list and inserts the result of f applied to the reversed list of elements before and the list of elements after. The empty lists at the beginning and end are included!

```
val interleave: (\alpha \ list \rightarrow \ \alpha \ list \rightarrow \ \alpha \ list) \rightarrow \ \alpha \ list \rightarrow \ \alpha \ list
```

interleave_nearest f list is like interleave f list, but f looks only at the nearest neighbors.

val
$$interleave_nearest: (\alpha \rightarrow \alpha \rightarrow \alpha \ list) \rightarrow \alpha \ list \rightarrow \alpha \ list$$

partitioned_sort cmp index_sets list sorts the sublists of list specified by the index_sets and the complement of their union. **NB**: the sorting follows to order in the lists in *index_sets*. **NB**: the indices are 0-based.

val
$$partitioned_sort: (\alpha \rightarrow \alpha \rightarrow int) \rightarrow int \ list \ list \rightarrow \alpha \ list \rightarrow \alpha \ list$$

```
exception Overlapping\_indices exception Out\_of\_bounds
```

ariadne_sort cmp list sorts list according to cmp (default Pervasives.compare) keeping track of the original order by a 0-based list of indices.

```
\label{eq:cont} \begin{array}{l} \mbox{val } ariadne\_sort \ : \ ?cmp : (\alpha \ \rightarrow \ \alpha \ \rightarrow \ int) \ \rightarrow \ \alpha \ list \ \rightarrow \ \alpha \ list \ \times \ int \ list \\ ariadne\_unsort \ (ariadne\_sort \ cmp \ list) \ \mbox{returns } list. \\ \mbox{val } ariadne\_unsort \ : \ \alpha \ list \ \times \ int \ list \ \rightarrow \ \alpha \ list \\ \end{array}
```

lexicographic cmp list1 list2 compares list1 and list2 lexicographically. val lexicographic : ?cmp : $(\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha$ list $\rightarrow \alpha$ list $\rightarrow int$

common l1 l2 returns the elements common to the lists l1 and l2. The lists are not required to be ordered and the result will also not be ordered.

```
val common : \alpha \ list \rightarrow \alpha \ list \rightarrow \alpha \ list
```

complement $l1\ l2$ returns the list l1 with elements of list l2 removed. The lists are not required to be ordered. Raises $Invalid_argument$ "ThoList.complement", if a member of l1 is not in l1.

```
val complement: \alpha \ list \rightarrow \alpha \ list \rightarrow \alpha \ list val to\_string: (\alpha \rightarrow string) \rightarrow \alpha \ list \rightarrow string module Test: sig val \ suite: OUnit.test end
```

F.2 Implementation of ThoList

Avoid referring to *Pervasives.compare*, because *Pervasives* will become *Stdlib.Pervasives* in O'Caml 4.07 and *Stdlib* in O'Caml 4.08.

```
let pcompare = compare
let rec hdn n l =
  if n \leq 0 then
     else
     match l with
     | x :: rest \rightarrow x :: hdn (pred n) rest
     | \ [ \ ] 
ightarrow invalid\_arg "ThoList.hdn"
let rec tln n l =
  if n \leq 0 then
     l
  else
     match l with
     |  :: rest \rightarrow tln (pred n) rest
     [] \rightarrow invalid\_arg "ThoList.tln"
let rec splitn' \ n \ l1 \_rev \ l2 =
  if n \leq 0 then
     (List.rev l1\_rev, l2)
   else
     match l2 with
      | x :: l2' \rightarrow splitn' (pred n) (x :: l1\_rev) l2'
     [] \rightarrow invalid\_arg "ThoList.splitn_{\sqcup}n_{\sqcup}>_{\sqcup}len"
let splitn \ n \ l =
  if n < 0 then
      invalid\_arg "ThoList.splitn_{\sqcup}n_{\sqcup}<_{\sqcup}0"
  else
     splitn' n [] l
let split_last l =
   \mathsf{match}\ \mathit{List.rev}\ \mathit{l}\ \mathsf{with}
   [] \rightarrow invalid\_arg "ThoList.split_last_[]"
```

```
| ln :: l12\_rev \rightarrow (List.rev l12\_rev, ln)
This is splitn' all over again, but without the exception.
let rec chopn'' \ n \ l1 \_rev \ l2 =
  if n \leq 0 then
     (List.rev l1\_rev, l2)
  else
     match l2 with
     | x :: l2' \rightarrow chopn'' (pred n) (x :: l1\_rev) l2'
     [] \rightarrow (List.rev\ l1\_rev,\ [])
let rec chopn' \ n \ ll\_rev = function
  | [] \rightarrow List.rev\ ll\_rev
   l \rightarrow
        begin match chopn'' n [] l with
        [], [] \rightarrow List.rev ll\_rev
        | l1, [] \rightarrow List.rev (l1 :: ll\_rev)
        | l1, l2 \rightarrow chopn' n (l1 :: ll\_rev) l2
        end
let chopn \ n \ l =
  if n < 0 then
     invalid\_arg "ThoList.chopn_{\sqcup}n_{\sqcup}<=_{\sqcup}0"
     chopn' n [] l
Find a member a in the list l and return the cyclically permuted list with a as head.
let cycle\_until a l =
  let rec cycle\_until' acc = function
     | [] \rightarrow raise\ Not\_found
     | a' :: l' \text{ as } al' \rightarrow
         if a' = a then
            al' @ List.rev acc
            cycle\_until' (a' :: acc) l' in
   cycle\_until' [] l
let rec cycle' i acc l =
  if i \leq 0 then
     l @ List.rev acc
  else
     match l with
     [] \rightarrow invalid\_arg "ThoList.cycle"
     | a' :: l' \rightarrow
         cycle' (pred i) (a' :: acc) l'
let cycle \ n \ l =
  if n < 0 then
     invalid_arg "ThoList.cycle"
  else
     cycle' n [] l
let of\_subarray n1 n2 a =
  let rec of \_subarray' n1 n2 =
     if n1 > n2 then
        else
        a.(n1) :: of\_subarray' (succ n1) n2 in
   of_subarray' (max 0 n1) (min n2 (pred (Array.length a)))
\mathsf{let} \ \mathit{range} \ ?(\mathit{stride} = 1) \ \mathit{n1} \ \mathit{n2} \ =
  if stride \leq 0 then
     invalid\_arg "ThoList.range:\sqcupstride\sqcup<=\sqcup0"
  else
```

```
let rec range' n =
        if n > n2 then
           else
           n :: range' (n + stride) in
      range' n1
Tail recursive:
let enumerate ?(stride = 1) n l =
  let _, l_rev =
      List.fold\_left
        (\mathsf{fun}\ (i,\ acc)\ a\ \to\ (i\ +\ stride,\ (i,\ a)\ ::\ acc))
         (n, []) l in
   List.rev l_rev
Take the elements of list that satisfy predicate and form a list of pairs of an offset into the original list and the
element with the offsets starting from offset. NB: the order of the returned alist is not specified!
let alist\_of\_list?(predicate = (fun \_ \rightarrow true))?(offset = 0) list =
  let _{-}, \ alist =
      List.fold\_left
        (fun (n, acc) x \rightarrow
           (succ\ n,\ if\ predicate\ x\ then\ (n,\ x)\ ::\ acc\ else\ acc))
         (offset, []) list in
   alist
This is not tail recursive!
let rec flatmap f = function
    [] \rightarrow []
   | x :: rest \rightarrow f x @ flatmap f rest
This is!
let rev_flatmap f l =
  let rec rev_flatmap' acc f = function
     | [] \rightarrow acc
      x :: rest \rightarrow rev\_flatmap' (List.rev\_append (f x) acc) f rest in
   rev_flatmap' [] f l
let rec power = function
  | [] \rightarrow [[]]
   a :: a\_list \rightarrow
       let power\_a\_list = power a\_list in
       power\_a\_list @ List.map (fun a\_list \rightarrow a :: a\_list) power\_a\_list
let fold\_left2 f acc lists =
   List.fold\_left (List.fold\_left f) acc lists
let fold_right2 f lists acc =
   List.fold\_right\ (List.fold\_right\ f)\ lists\ acc
let iteri f start list =
   ignore\ (List.fold\_left\ (fun\ i\ a\ 	o\ f\ i\ a;\ succ\ i)\ start\ list)
let iteri2 f start\_outer star\_inner lists =
   iteri (fun j \rightarrow iteri (f j) star\_inner) start\_outer lists
let mapi f start list =
  let next, list' =
      List.fold\_left (fun (i, acc) a \rightarrow (succ i, f i a :: acc)) (start, []) list in
   List.rev list'
let rec map3 f l1 l2 l3 =
   match l1, l2, l3 with
    [], [], [] \rightarrow []
   | \hspace{.15cm} a1 \hspace{.15cm} :: \hspace{.15cm} l1 \hspace{.15cm}, \hspace{.15cm} a2 \hspace{.15cm} :: \hspace{.15cm} l2 \hspace{.15cm}, \hspace{.15cm} a3 \hspace{.15cm} :: \hspace{.15cm} l3 \hspace{.15cm} \rightarrow \hspace{.15cm}
      let fa123 = f a1 a2 a3 in
```

```
fa123 :: map3 f l1 l2 l3
  | -, -, - \rightarrow invalid\_arg "ThoList.map3"
Is there a more efficient implementation?
let \ transpose \ lists =
  let rec transpose' rest =
     if List.for\_all\ ((=)\ [\,])\ rest then
     else
         List.map List.hd rest :: transpose' (List.map List.tl rest) in
      transpose' lists
  with
   \mid Failure s \rightarrow
       if s = "tl" then
          invalid\_arg "ThoList.transpose: \_not\_rectangular"
       else
          failwith~("ThoList.transpose:\_unexpected\_Failure("~s~")")
let compare ?(cmp = pcompare) l1 l2 =
  let rec compare' l1' l2' =
      match l1', l2' with
      | [], [] \rightarrow 0
     |[], \rightarrow -1
      | -, [] \rightarrow 1
     \mid n1 :: r1, n2 :: r2 \rightarrow
           let c = cmp \ n1 \ n2 \ in
           if c \neq 0 then
              c
           else
               compare' r1 r2
   compare' l1 l2
\mathsf{let} \ \mathsf{rec} \ \mathit{uniq'} \ x \ = \ \mathsf{function}
   | [] \rightarrow []
   | x' :: rest \rightarrow
        if x' = x then
           uniq' \ x \ rest
         else
           x' :: uniq' x' rest
\mathsf{let}\ uniq\ =\ \mathsf{function}
   | [] \rightarrow []
   | \ x \ :: \ rest \ \rightarrow \ x \ :: \ uniq' \ x \ rest
\mathsf{let} \ \mathsf{rec} \ \mathit{homogeneous} \ = \ \mathsf{function}
  | \ [] \ | \ [\_] \ 	o \ \mathsf{true}
   \mid a1 :: (a2 :: \_ as rest) \rightarrow
         if a1 \neq a2 then
           false
            homogeneous rest
let rec pairs' acc = function
   | [] \rightarrow acc
   [x] \rightarrow invalid\_arg "pairs:\u00cdd\u00cdnumber\u00cdof\u00cdelements"
   | \ x \ :: \ y \ :: \ indices \ \rightarrow
       if x \neq y then
          invalid\_arg "pairs: \( \text{not} \( \text{in} \) \( \text{pairs} \) "
          begin match \mathit{acc} with
          | [] \rightarrow pairs' [x] indices
          | x' :: \_ \rightarrow
```

```
if x = x' then
                invalid\_arg "pairs: \( \text{more} \) than \( \text{twice} \) "
                pairs' (x :: acc) indices
         end
let pairs l =
  pairs' [] (List.sort pcompare l)
If we needed it, we could use a polymorphic version of Set to speed things up from O(n^2) to O(n \ln n). But not
before it matters somewhere ...
let classify l =
  let rec add\_to\_class a = function
     | [] \rightarrow [1, a]
     | (n, a') :: rest \rightarrow
          if a = a' then
             (succ \ n, \ a) :: rest
             (n, a') :: add\_to\_class \ a \ rest
  in
  let rec classify' cl = function
     | [] \rightarrow cl
     |~a~::~rest~\rightarrow~classify'~(add\_to\_class~a~cl)~rest
  in
  classify' [] l
let rec factorize \ l =
  let rec add\_to\_class \ x \ y = function
     [] \rightarrow [(x, [y])]
     | (x', ys) :: rest \rightarrow
          if x = x' then
             (x, y :: ys) :: rest
          else
             (x', ys) :: add\_to\_class x y rest
  in
  let rec factorize' fl = function
     | [] \rightarrow fl
     (x, y) :: rest \rightarrow factorize' (add\_to\_class x y fl) rest
  in
   List.map (fun (x, ys) \rightarrow (x, List.rev ys)) (factorize' [] l)
let rec clone \ x \ n =
  if n < 0 then
     invalid_arg "ThoList.clone"
  else if n = 0 then
     else
     x :: clone \ x \ (pred \ n)
{\tt let} \ interleave \ f \ list =
  let rec interleave' rev_head tail =
     let rev\_head' = List.rev\_append (f rev\_head tail) rev\_head in
     match tail with
       [] \rightarrow List.rev rev\_head'
     | x :: tail' \rightarrow interleave'(x :: rev\_head') tail'
  interleave' [] list
let \ interleave\_nearest \ f \ list =
   interleave
     (fun head tail \rightarrow
        match head, tail with
        | h :: \_, t :: \_ \rightarrow f h t
        | - \rightarrow [])
```

```
list
let rec rev_multiply n rl l =
  if n < 0 then
     invalid_arg "ThoList.multiply"
  else if n = 0 then
  else
     List.rev_append rl (rev_multiply (pred n) rl l)
let multiply \ n \ l = rev\_multiply \ n \ (List.rev \ l) \ l
let filtermap f l =
  let rec rev_filtermap \ acc = function
    | [] \rightarrow List.rev acc
    a :: a\_list \rightarrow
        \mathsf{match}\ f\ a\ \mathsf{with}
         | None \rightarrow rev\_filtermap \ acc \ a\_list
         | Some \ fa \rightarrow rev\_filtermap \ (fa :: acc) \ a\_list
  in
  rev_filtermap [] l
exception Overlapping\_indices
exception Out\_of\_bounds
let iset\_list\_union\ list =
  List.fold_right Sets.Int.union list Sets.Int.empty
let \ complement\_index\_sets \ n \ index\_set\_lists =
  let index\_sets = List.map\ Sets.Int.of\_list\ index\_set\_lists in
  let index\_set = iset\_list\_union index\_sets in
  let size\_index\_sets =
     List.fold\_left (fun acc \ s \rightarrow Sets.Int.cardinal \ s + acc) \ 0 \ index\_sets in
  if size\_index\_sets \neq Sets.Int.cardinal\ index\_set then
     raise Overlapping_indices
  else if Sets.Int.exists (fun i \rightarrow i < 0 \lor i \ge n) index\_set then
     raise Overlapping_indices
  else
     match Sets.Int.elements
               (Sets.Int.diff (Sets.Int.of_list (range 0 (pred n))) index_set) with
     | [] \rightarrow index\_set\_lists
     | complement \rightarrow complement :: index\_set\_lists
let sort_section cmp array index_set =
  List.iter2
     (Array.set array)
     index_set (List.sort cmp (List.map (Array.get array) index_set))
let partitioned_sort cmp index_sets list =
  let array = Array.of\_list\ list\ in
  List.fold\_left
     (fun () \rightarrow sort\_section \ cmp \ array)
     () (complement_index_sets (List.length list) index_sets);
  Array.to_list array
let ariadne\_sort\ ?(cmp = pcompare)\ list =
  let sorted =
     List.sort (fun (n1, a1) (n2, a2) \rightarrow cmp \ a1 \ a2) (enumerate 0 list) in
  (List.map snd sorted, List.map fst sorted)
let ariadne\_unsort (sorted, indices) =
  List.map snd
     (List.sort
         (fun (n1, a1) (n2, a2) \rightarrow pcompare n1 n2)
         (List.map2 (fun n \ a \rightarrow (n, a)) indices sorted))
let lexicographic ?(cmp = pcompare) l1 l2 =
```

```
\mathsf{let} \ \mathsf{rec} \ \mathit{lexicographic'} \ = \ \mathsf{function}
     [],[] \rightarrow 0
     [], \rightarrow -1
       [] \rightarrow 1
     | x1 :: rest1, x2 :: rest2 \rightarrow
         let res = cmp x1 x2 in
         if res \neq 0 then
            res
         else
            lexicographic' (rest1, rest2) in
   lexicographic' (l1, l2)
If there was a polymorphic Set, we could also say Set. elements (Set. union (Set. of _list l1) (Set. of _list l2)).
let common \ l1 \ l2 =
   List.fold\_left
     (fun acc x1 \rightarrow
        if List.mem \ x1 \ l2 then
          x1 :: acc
        else
           acc)
     [] l1
let complement l1 = function
   | [] \rightarrow l1
  | l2 \rightarrow
      if List.for\_all (fun x \rightarrow List.mem \ x \ l1) l2 then
         List.filter (fun x \rightarrow \neg (List.mem x l2)) l1
      else
         invalid\_arg "ThoList.complement"
let to\_string a2s alist =
  "[" \hat{} String.concat "; \Box" (List.map a2s alist) \hat{} "]"
let random_int_list imax n =
  let imax_plus = succ imax in
  Array.to\_list (Array.init n (fun \_ \rightarrow Random.int imax\_plus))
module Test =
  struct
     let int\_list2\_to\_string l2 =
        to_string (to_string string_of_int) l2
Inefficient, must only be used for unit tests.
     let compare\_lists\_by\_size\ l1\ l2\ =
        let lengths = pcompare (List.length l1) (List.length l2) in
        if lengths = 0 then
          pcompare l1 l2
        else
           lengths
     open OUnit
     let suite\_filtermap =
        "filtermap" >:::
          ["filtermap_Some_[]" >::
                (fun () \rightarrow
                   assert_equal ~printer: (to_string string_of_int)
                     [] (filtermap (fun x \rightarrow Some x) []));
             "filtermap_{\sqcup}None_{\sqcup}[]" >::
                (fun () \rightarrow
                   assert_equal ~printer: (to_string string_of_int)
                     [] (filtermap\ (fun\ x\ \rightarrow\ None)\ []));
             "filtermap_{\sqcup}even_{-}neg_{\sqcup}[]" >::
```

```
(\mathsf{fun}\ ()\ \to
              assert_equal ~printer: (to_string string_of_int)
                [0; -2; -4]
                 (filtermap
                     (fun n \rightarrow \text{if } n \mod 2 = 0 \text{ then } Some (-n) \text{ else } None)
                     (range \ 0 \ 5)));
        "filtermap\_odd\_neg_{\sqcup}[] \, ">::
           (fun () \rightarrow
              assert_equal ~printer: (to_string string_of_int)
                 [-1; -3; -5]
                 (filter map
                     (fun n \rightarrow \text{if } n \mod 2 \neq 0 \text{ then } Some (-n) \text{ else } None)
                     (range 0 5)))]
let \ assert\_power \ power\_a\_list \ a\_list \ =
   assert\_equal \ \tilde{\ } printer: int\_list2\_to\_string
     power_a_list
     (List.sort\ compare\_lists\_by\_size\ (power\ a\_list))
let suite\_power =
   "power" >:::
      [ "power_[] " >::
           (fun () \rightarrow
              assert\_power [[]] []);
         "power_[1]" >::
           (fun () \rightarrow
              assert\_power [[]; [1]] [1]);
         "power_[1;2]" >::
           (fun () \rightarrow
              assert_power [[]; [1]; [2]; [1;2]] [1;2]);
         "power_[1;2;3]" >::
           (fun () \rightarrow
              assert\_power
                [[];
                  [1]; [2]; [3];
                  [1;2]; [1;3]; [2;3];
                  [1; 2; 3]]
                [1;2;3]);
         "power_{\perp}[1;2;3;4]" >::
           (fun () \rightarrow
              assert\_power
                [[];
                  [1]; [2]; [3]; [4];
                  [1;2]; [1;3]; [1;4]; [2;3]; [2;4]; [3;4];
                  [1;2;3]; [1;2;4]; [1;3;4]; [2;3;4];
                  [1; 2; 3; 4]
                [1; 2; 3; 4])
let suite\_split =
   "split*" >:::
      [ "split_last_[]" >::
           (fun () \rightarrow
              assert\_raises
                 (Invalid_argument "ThoList.split_last []")
                 (fun () \rightarrow split\_last []));
         "split_last_{\sqcup}[1]">::
           (fun () \rightarrow
              assert\_equal
                 ([], 1)
                 (split\_last [1]);
```

```
"split_last_[2;3;1;4]" >::
            (fun () \rightarrow
               assert\_equal
                  ([2;3;1], 4)
                  (split\_last [2; 3; 1; 4]))
let test\_list = random\_int\_list 1000 100
\mathsf{let} \ \mathit{assert\_equal\_int\_list} \ = \\
   assert_equal ~printer: (to_string string_of_int)
let suite\_cycle =
   "cycle_until" >:::
      ["cycle_{\sqcup}(-1)_{\sqcup}[1;2;3]">::
            (fun () \rightarrow
               assert\_raises
                  (Invalid_argument "ThoList.cycle")
                  (fun () \rightarrow cycle 4 [1;2;3]));
         "cycle_{\perp}4_{\perp}[1;2;3]" >::
            (fun () \rightarrow
               assert\_raises
                  (Invalid_argument "ThoList.cycle")
                  (\mathsf{fun}\ ()\ \rightarrow\ \mathit{cycle}\ 4\ [1;2;3]));
         "cycle_{\sqcup}42_{\sqcup}[\ldots]" >::
            (\mathsf{fun}\ ()\ \to
               \mathsf{let}\ n\ =\ 42\ \mathsf{in}
               assert\_equal\_int\_list
                  (tln \ n \ test\_list @ hdn \ n \ test\_list)
                  (cycle \ n \ test\_list));
         "cycle_until_{\square}1_{\square}[]" >::
            (\mathsf{fun}\ ()\ \to
               assert\_raises
                  (Not\_found)
                  (fun () \rightarrow cycle\_until 1 []));
         "cycle_until_{\square}1_{\square}[2;3;4]" >::
            (\mathsf{fun}\ ()\ \to
               assert\_raises
                  (Not\_found)
                  (fun () \rightarrow cycle\_until 1 [2; 3; 4]));
         "cycle_until_{\square}1_{\square}[1;2;3;4]" >::
            (fun () \rightarrow
               assert\_equal
                  [1; 2; 3; 4]
                  (cycle\_until\ 1\ [1;2;3;4]));
         "cycle_until_{\square}3_{\square}[1;2;3;4]" >::
            (\mathsf{fun}\ ()\ \to
               assert\_equal
                  [3;4;1;2]
                  (cycle\_until\ 3\ [3;4;1;2]));
         "cycle_until_{\sqcup}4_{\sqcup}[1;2;3;4]" >::
            (\mathsf{fun}\ ()\ \to
               assert\_equal
                  [4;1;2;3]
                  (cycle\_until\ 4\ [4;1;2;3]))\ ]
let suite\_alist\_of\_list =
   "alist_of_list" >:::
      [ "simple" >::
            (fun () \rightarrow
               assert\_equal
                  [(46, 4); (44, 2); (42, 0)]
                  (alist\_of\_list
                       [predicate: (fun \ n \rightarrow n \ mod \ 2 = 0) \ [predicate: 42 \ [0; 1; 2; 3; 4; 5]))]
```

```
let suite\_complement =
      "complement" >:::
        [ "simple" >::
              (\mathsf{fun}\ ()\ \to
                 assert\_equal~[2; 4]~(complement~[1; 2; 3; 4]~[1;~3]));
           "empty" >::
              (\mathsf{fun}\ ()\ \to
                 assert_equal [1; 2; 3; 4] (complement [1; 2; 3; 4] []));
           "failure" >::
              (fun () \rightarrow
                 assert\_raises
                    (Invalid_argument ("ThoList.complement"))
                   (\mathsf{fun}\ ()\ \rightarrow\ \mathit{complement}\ (\mathit{complement}\ [1;2;3;4]\ [5])))\ ]
  \mathsf{let} \ \mathit{suite} \ =
     "ThoList" >:::
        [suite_filtermap;
          suite\_power;
          suite\_split;
          suite\_cycle;
          suite\_alist\_of\_list;
          suite\_complement]
end
```

—G— More On Arrays

G.1 Interface of ThoArray

Compressed arrays, i. e. arrays with only unique elements and an embedding that allows to recover the original array. NB: in the current implementation, compressing saves space, if and only if objects of type α require more storage than integers. The main use of α compressed is not for saving space, anyway, but for avoiding the repetition of hard calculations.

```
type \alpha compressed
val uniq: \alpha \ compressed \rightarrow \alpha \ array
val embedding: \alpha \ compressed \rightarrow int \ array
These two are inverses of each other:
val compress: \alpha array \rightarrow \alpha compressed
val uncompress : \alpha \ compressed \rightarrow \alpha \ array
One can play the same game for matrices.
type \alpha compressed2
val uniq2: \alpha \ compressed2 \rightarrow \alpha \ array \ array
\verb|val|| embedding1|: \alpha| compressed2| \rightarrow int| array|
val embedding2: \alpha compressed2 \rightarrow int array
Again, these two are inverses of each other:
val compress2 : \alpha array array \rightarrow \alpha compressed2
val uncompress2: \alpha \ compressed2 \rightarrow \alpha \ array \ array
compare cmp a1 a2 compare two arrays a1 and a2 according to cmp. cmp defaults to the polymorphic
Pervasives.compare.
val compare : ?cmp : (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha \ array \rightarrow \alpha \ array \rightarrow int
Searching arrays
val\ find\_first\ :\ (lpha\ 	o\ bool)\ 	o\ lpha\ array\ 	o\ int
\mathsf{val}\ \mathit{match\_first}\ :\ \alpha\ \to\ \alpha\ \mathit{array}\ \to\ \mathit{int}
val\ find\_all\ :\ (\alpha\ 	o\ bool)\ 	o\ \alpha\ array\ 	o\ int\ list
val match\_all : \alpha \rightarrow \alpha \ array \rightarrow \ int \ list
val num\_rows : \alpha array array \rightarrow int
val num\_columns : \alpha array array \rightarrow int
module Test: sig val suite: OUnit.test end
```

G.2 Implementation of ThoArray

Avoid referring to *Pervasives.compare*, because *Pervasives* will become *Stdlib.Pervasives* in O'Caml 4.07 and *Stdlib* in O'Caml 4.08.

```
\begin{array}{lll} \text{let } pcompare &=& compare \\ \\ \text{type } \alpha \ compressed &=& \\ \{ \ uniq \ : \ \alpha \ array; \\ &=& mbedding : \ int \ array \ \} \end{array}
```

```
let uniq \ a = a.uniq
let embedding \ a = a.embedding
\mathsf{type} \,\, \alpha \,\, compressed 2 \,\, = \,\,
     { uniq2 : \alpha \ array \ array;}
       embedding1 : int array;
       embedding2 : int array }
let uniq2 \ a = a.uniq2
\mathsf{let}\ embedding1\ a\ =\ a.embedding1
let embedding2 a = a.embedding2
module PMap = Pmap.Tree
let compress \ a =
  let last = Array.length \ a - 1 in
  let embedding = Array.make (succ last) (-1) in
  let rec scan num\_uniq uniq elements n =
    if n > last then
       \{ uniq = Array.of\_list (List.rev elements); 
          embedding = embedding }
    else
       match PMap.find_opt compare a.(n) uniq with
       \mid Some \ n' \rightarrow
            embedding.(n) \leftarrow n';
            scan num\_uniq uniq elements (succ n)
       | None \rightarrow
            embedding.(n) \leftarrow num\_uniq;
            scan
              (succ\ num\_uniq)
              (PMap.add\ compare\ a.(n)\ num\_uniq\ uniq)
              (a.(n) :: elements)
              (succ \ n) in
  scan 0 PMap.empty [] 0
let uncompress \ a =
  Array.map (Array.get a.uniq) a.embedding
```

Using transpose simplifies the algorithms, but can be inefficient. If this turns out to be the case, we should add special treatments for symmetric matrices.

```
let transpose \ a =
  let dim1 = Array.length a
  and dim2 = Array.length \ a.(0) in
  let a' = Array.make\_matrix \ dim2 \ dim1 \ a.(0).(0) in
  for i1 = 0 to pred dim1 do
    for i2 = 0 to pred dim2 do
      a'.(i2).(i1) \leftarrow a.(i1).(i2)
    done
  done;
  a'
let compress2 \ a =
  let c2 = compress a in
  let c12\_transposed = compress (transpose c2.uniq) in
  \{ uniq2 = transpose \ c12\_transposed.uniq; \}
    embedding1 = c12\_transposed.embedding;
    embedding2 = c2.embedding }
let uncompress2 \ a = 
  let a2 = uncompress \{ uniq = a.uniq2; embedding = a.embedding2 \} in
  transpose (uncompress \{ uniq = transpose \ a2; \ embedding = a.embedding1 \})
FIXME: not tail recursive!
let compare ?(cmp = pcompare) a1 a2 =
```

```
let l1 = Array.length a1
  and l2 = Array.length \ a2 in
  if l1 < l2 then
     -1
  else if l1 > l2 then
     1
  else
     \mathsf{let} \ \mathsf{rec} \ \mathit{scan} \ i \ = \\
        if i = l1 then
           0
        else
           let c = cmp \ a1.(i) \ a2.(i) in
           if c < 0 then
             -1
           else if c > 0 then
              1
           else
              scan (succ i) in
     scan 0
let find\_first f a =
  let l = Array.length a in
  let rec find_-first' i =
     \text{if } i \ \geq \ l \ \text{then} \\
        raise Not_found
     else if f(a.(i)) then
        i
     else
        find\_first' (succ i)
  in
  find\_first' 0
let match\_first \ x \ a =
  find\_first (fun x' \rightarrow x = x') a
\mathsf{let}\; \mathit{find\_all}\; f\;\; a\;\; =\;\;
  let matches = ref[] in
  \mbox{ for } i \ = \ Array.length \ a \ - \ 1 \ \mbox{downto} \ 0 \ \mbox{do}
     if f(a.(i)) then
        matches := i :: !matches
  done;
  ! matches
\mathsf{let}\ match\_all\ x\ a\ =\ 
  find\_all (fun x' \rightarrow x = x') a
let num\_rows \ a =
   Array.length a
let num\_columns \ a =
  match ThoList.classify (List.map Array.length (Array.to_list a)) with
    [ (-, n) ] \rightarrow n
     \_ \ \to \ \mathit{invalid\_arg} \ \texttt{"ThoArray.num\_columns:} \_ \texttt{inhomogeneous} \_ \texttt{array"}
module Test =
  struct
     open OUnit
     let test\_compare\_empty =
        "empty" >::
           (fun () \rightarrow assert\_equal \ 0 \ (compare [| |] \ [| |]))
     let test\_compare\_shorter =
        "shorter" >::
           (fun () \rightarrow assert\_equal (-1) (compare [|0|] [|0; 1|]))
```

```
let test\_compare\_longer =
   "longer" >::
     (fun () \rightarrow assert\_equal (1) (compare [|0; 1|] [|0|]))
let test\_compare\_less =
   "longer" >::
      (fun () \rightarrow assert\_equal (-1) (compare [|0; 1|] [|0; 2|]))
let test\_compare\_equal =
   "equal" >::
     (fun () \rightarrow assert\_equal (0) (compare [|0; 1|] [|0; 1|]))
let test_compare_more =
   "more" >::
     (fun () \rightarrow assert\_equal (1) (compare [|0; 2|] [|0; 1|]))
let suite\_compare =
   "compare" >:::
     [test\_compare\_empty;
       test\_compare\_shorter;
       test\_compare\_longer;
       test\_compare\_less;
       test\_compare\_equal;
       test\_compare\_more
let test\_find\_first\_not\_found =
   "not_{\square}found" >::
      (fun () \rightarrow
        assert\_raises\ Not\_found
           (\text{fun } () \rightarrow find\_first (\text{fun } n \rightarrow n \text{ mod } 2 = 0) [|1; 3; 5|]))
let test_find_first_first =
   "first" >::
      (fun () \rightarrow
        assert\_equal 0
           (find\_first (fun \ n \rightarrow n \bmod 2 = 0) [|2; 3; 4; 5|]))
let test\_find\_first\_not\_last =
   "last" >::
     (fun () \rightarrow
        assert\_equal 1
           (find\_first (fun \ n \rightarrow n \bmod 2 = 0) [|1; 2; 3; 4|]))
let test\_find\_first\_last =
   "notulast" >::
     (fun () \rightarrow
        assert\_equal 1
           (find\_first (fun \ n \rightarrow n \bmod 2 = 0) [|1;2|]))
let suite\_find\_first =
   "find_first" >:::
     [test\_find\_first\_not\_found;
       test\_find\_first\_first;
       test_find_first_not_last;
       test\_find\_first\_last]
let test\_find\_all\_empty =
   "empty" >::
     (fun () \rightarrow
        assert_equal []
           (find\_all \text{ (fun } n \rightarrow n \text{ mod } 2 = 0) \text{ [}|1;3;5|]))
let test\_find\_all\_first =
   "first" >::
      (fun () \rightarrow
        assert\_equal [0; 2]
           (find\_all \ (fun \ n \rightarrow n \ mod \ 2 = 0) \ [|2; 3; 4; 5|]))
```

```
let test\_find\_all\_not\_last =
     "last" >::
        (fun () \rightarrow
           assert\_equal [1; 3]
             (find\_all \ (fun \ n \rightarrow n \ mod \ 2 = 0) \ [|1; 2; 3; 4; 5|]))
  let test\_find\_all\_last =
     "not_{\sqcup}last">::
        (\mathsf{fun}\ ()\ \to
           assert\_equal [1; 3]
             (find\_all \ (fun \ n \rightarrow n \ mod \ 2 = 0) \ [|1;2;3;4|]))
  let suite\_find\_all =
     "find_all" >:::
        [test\_find\_all\_empty;
         test\_find\_all\_first;
         test\_find\_all\_last;
         test\_find\_all\_not\_last]
  let test_num_columns_ok2 =
     "ok/2" >::
        (\mathsf{fun}\ ()\ \to
           assert\_equal \ 2
             (num\_columns [| [| 11; 12 |];
                                   [| 21; 22 |];
                                   [| 31; 32 |] |]))
  let test\_num\_columns\_ok0 =
     "ok/0" >::
        (fun () \rightarrow
           assert\_equal 0
             (num\_columns [| [| |];
                                   [|\ |]\ |]))
  let test\_num\_columns\_not\_ok =
     "not_ok" >::
        (\mathsf{fun}\ ()\ \to
           assert_raises (Invalid_argument
                                  "ThoArray.num_columns: uinhomogeneous uarray")
             (fun () \rightarrow num\_columns [| [| 11; 12 |];
                                                 [| 21 |];
                                                 [| 31; 32 || |]))
  let suite_num_columns =
     "num_columns" >:::
        [test\_num\_columns\_ok2;
         test\_num\_columns\_ok\theta;
         test\_num\_columns\_not\_ok
  \mathsf{let} \ \mathit{suite} \ =
     "ThoArrays" >:::
        [suite\_compare;
         suite\_find\_first;
         suite\_find\_all;
         suite\_num\_columns]
end
```

—H— More On Strings

H.1 Interface of ThoString

This is a very simple library if stroing manipulation functions missing in O'Caml's standard library. $strip_prefix\ prefix\ string\ returns\ string\ with\ 0\ or\ 1\ occurrences\ of\ a\ leading\ prefix\ removed.$

 $val\ strip_prefix\ :\ string
ightarrow\ string
ightarrow\ string$

strip_prefix_star prefix string returns string with any number of leading occurences of prefix removed.

 $val\ strip_prefix_star\ :\ char
ightarrow\ string
ightarrow\ string$

strip_prefix prefix string returns string with a leading prefix removed, raises Invalid_argument if there's no match.

 $val\ strip_required_prefix : string \rightarrow string \rightarrow string$

 $strip_from_first\ c\ s$ returns s with everything starting from the first c removed. $strip_from_last\ c\ s$ returns s with everything starting from the last c removed.

```
val strip\_from\_first : char \rightarrow string \rightarrow string val strip\_from\_last : char \rightarrow string \rightarrow string
```

 $index_string\ pattern\ string$ returns the index of the first occurence of pattern in string, if any. Raises Not_found , if pattern is not in string.

 $\mathsf{val}\ index_string\ :\ string \to\ string \to\ int$

This silently fails if the argument contains both single and double quotes!

 $val\ quote\ :\ string
ightarrow\ string$

The corresponding functions from *String* have become obsolescent with O'Caml 4.0.3. Quantantine them here.

 $\begin{array}{lll} \mathsf{val} \ uppercase \ : \ string \to \ string \\ \mathsf{val} \ lowercase \ : \ string \to \ string \end{array}$

Ignore the case in comparisons.

 $\verb|val|| compare_caseless : string \rightarrow string \rightarrow int$

Match the regular expression $[A-Za-z][A-Za-z0-9_]*$

 $\mathsf{val}\ valid_fortran_id\ :\ string \to\ bool$

Replace any invalid character by '_' and prepend "N_" iff the string doesn't start with a letter.

 $\mathsf{val}\ sanitize_fortran_id\ :\ string\ \to\ string$

 $module \ Test : sig \ val \ suite : OUnit.test \ end$

H.2 Implementation of ThoString

```
\begin{array}{lll} \text{let } strip\_prefix \ p \ s &= \\ \text{let } lp \ = \ String.length \ p \\ \text{and } ls \ = \ String.length \ s \ \text{in} \\ \text{if } lp \ > \ ls \ \text{then} \\ s \end{array}
```

```
else
     let rec strip\_prefix' i =
        if i \geq lp then
          String.sub \ s \ i \ (ls - i)
        else if p.[i] \neq s.[i] then
        else
          strip\_prefix' (succ \ i)
     in
     strip_prefix' 0
\mathsf{let}\ \mathit{strip\_prefix\_star}\ p\ s\ =\ 
  \mathsf{let}\ \mathit{ls}\ =\ \mathit{String.length}\ \mathit{s}\ \mathsf{in}
  \quad \text{if } \mathit{ls} \; < \; 1 \; \text{then} \\
  else
     let rec strip\_prefix\_star' i =
        if i < ls then begin
          if p \neq s.[i] then
             String.sub \ s \ i \ (ls - i)
          else
             strip\_prefix\_star' (succ i)
        end else
     in
     strip\_prefix\_star' 0
let strip\_required\_prefix p s =
  let lp = String.length p
  and ls = String.length s in
  if lp > ls then
     invalid_arg ("strip_required_prefix:_expected_'" ^ p ^ "'_got_'" ^ s ^ "'")
     let rec strip\_prefix' i =
        if i \geq lp then
           String.sub \ s \ i \ (ls - i)
        else if p.[i] \neq s.[i] then
           else
           strip\_prefix' (succ i)
     in
     strip\_prefix' 0
\mathsf{let}\ \mathit{strip\_from\_first}\ c\ s\ =
     String.sub \ s \ 0 \ (String.index \ s \ c)
  with
  | Not\_found \rightarrow s
\mathsf{let}\ strip\_from\_last\ c\ s\ =
  try
     String.sub \ s \ 0 \ (String.rindex \ s \ c)
  with
  | Not\_found \rightarrow s
let index\_string pat s =
  let lpat = String.length pat
  and ls = String.length s in
  if lpat = 0 then
     0
  else
     let rec index\_string' n =
        \mathsf{let}\ i\ =\ \mathit{String.index\_from}\ s\ n\ \mathit{pat.}[0]\ \mathsf{in}
        if i + lpat > ls then
```

```
raise Not_found
       else
          if String.compare\ pat\ (String.sub\ s\ i\ lpat)\ =\ 0 then
          else
            index\_string' (succ \ i)
    in
     index\_string' 0
let quote \ s =
  if String.contains s ' \lor String.contains s '\n' then begin
    if String.contains\ s '"' then
       else
       "\"" ^ s ^ "\""
  end else
     s
let \ uppercase = String.uppercase\_ascii
{\tt let}\ lowercase\ =\ String.lowercase\_ascii
let compare\_caseless s1 s2 =
  String.compare (lowercase s1) (lowercase s2)
let is\_alpha c =
  ('a' \leq c \land c \leq 'z') \lor ('A' \leq c \land c \leq 'Z')
let is\_numeric\ c\ =
   '0' \leq c \wedge c \leq '9'
let is\_alphanum c =
  is\_alpha\ c\ \lor\ is\_numeric\ c\ \lor\ c\ =\ `\_`
\mathsf{let}\ valid\_fortran\_id\ s\ =
  let rec valid\_fortran\_id' n =
    if n < 0 then
       false
    else if n = 0 then
       is\_alpha \ s.[0]
    else if is\_alphanum \ s.[n] then
       valid\_fortran\_id' (pred n)
    else
       false in
  valid\_fortran\_id' (pred (String.length s))
let \ sanitize\_fortran\_id \ s =
  let sanitize s =
     String.map (fun c \rightarrow if is\_alphanum c then c else '_') s in
  if String.length s \leq 0 then
     invalid\_arg "ThoString.sanitize_fortran_id:\_empty"
  else if is\_alpha \ s.[0] then
     sanitize \ s
  else
     "N_" \hat{\ } sanitize s
module Test =
  struct
    open OUnit
    let fortran\_empty =
       "empty" >::
          (fun () \rightarrow assert\_equal false (valid\_fortran\_id ""))
    let fortran_digit =
       "0" >::
          (fun () \rightarrow assert\_equal false (valid\_fortran\_id "O"))
```

```
let fortran_digit_alpha =
     "0abc" >::
        (fun () \rightarrow assert\_equal false (valid\_fortran\_id "Oabc"))
  let\ fortran\_underscore =
     "<sub>-</sub>" >::
        (fun () \rightarrow assert\_equal false (valid\_fortran\_id "\_"))
  let fortran\_underscore\_alpha =
     "_ABC" >::
       (fun () \rightarrow assert\_equal false (valid\_fortran\_id "\_ABC"))
  let fortran_question mark =
     "A?C" >::
        (fun () \rightarrow assert\_equal false (valid\_fortran\_id "A?C"))
  let fortran_valid =
     "A_xyz_0_" >::
        (fun () \rightarrow assert\_equal true (valid\_fortran\_id "A\_xyz\_0\_"))
  let sanitize\_digit =
     "0" >::
        (fun () \rightarrow assert\_equal "N_0" (sanitize\_fortran\_id "0"))
  let sanitize\_digit\_alpha =
     "0abc" >::
        (fun () \rightarrow assert\_equal "N_Oabc" (sanitize\_fortran\_id "Oabc"))
  let sanitize\_underscore =
     "_" >::
        (fun () \rightarrow assert\_equal "N\_\_" (sanitize\_fortran\_id "\_"))
  let sanitize\_underscore\_alpha =
     "_ABC" >::
        (\mathsf{fun}\ ()\ \to\ \mathit{assert\_equal}\ \verb"N\_\_ABC"\ (\mathit{sanitize\_fortran\_id}\ \verb"\_ABC"))
  let sanitize\_questionmark =
     "A?C" >::
        (fun () \rightarrow assert\_equal "A\_C" (sanitize\_fortran\_id "A?C"))
  let sanitize\_valid =
     "A_xyz_0_" >::
        (fun () \rightarrow assert\_equal "A\_xyz\_0\_" (sanitize\_fortran\_id "A\_xyz\_0\_"))
  let suite\_fortran =
     "valid_fortran_id" >:::
        [fortran\_empty;
         fortran\_digit;
        fortran\_digit\_alpha;
        fortran_underscore;
         fortran\_underscore\_alpha;
         fortran_questionmark;
         fortran\_valid
  let suite\_sanitize =
     "sanitize_fortran_id" >:::
        [sanitize\_digit;
         sanitize\_digit\_alpha;
         sanitize_underscore;
         sanitize\_underscore\_alpha;
         sanitize\_question mark;
         sanitize\_valid
  let suite =
     "ThoString" >:::
       [suite\_fortran;
         suite\_sanitize
end
```

—I—

POLYMORPHIC MAPS

From [9].

I.1 Interface of Pmap

```
Module Pmap: association tables over a polymorphic type<sup>1</sup>.
```

```
module type T =
               sig
                              type ('key, \alpha) t
                                val empty: ('key, \alpha) t
                              val is\_empty: ('key, \alpha) t \rightarrow bool
                              val singleton : 'key \rightarrow \alpha \rightarrow ('key, \alpha) t
                              val add: (key \rightarrow key \rightarrow int) \rightarrow key \rightarrow \alpha \rightarrow (key, \alpha) t \rightarrow (key, \alpha) t
                              val update: (key \rightarrow key \rightarrow int) \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow
                                                 `key \rightarrow \alpha \rightarrow (`key, \alpha) t \rightarrow (`key, \alpha) t
                              val cons: ('key \rightarrow 'key \rightarrow int) \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha \ option) \rightarrow
                                                 `key \rightarrow \alpha \rightarrow (`key, \alpha) t \rightarrow (`key, \alpha) t
                                \mathsf{val}\;\mathit{find}\;:\;(\textit{`key}\;\rightarrow\;\textit{`key}\;\rightarrow\;\mathit{int})\;\rightarrow\;\textit{`key}\;\rightarrow\;(\textit{`key},\;\alpha)\;t\;\rightarrow\;\alpha
                              val\ find\_opt : ('key \rightarrow 'key \rightarrow int) \rightarrow 'key \rightarrow ('key, \alpha) \ t \rightarrow \alpha \ option
                              val choose : ('key, \alpha) t \rightarrow 'key \times \alpha
                              val choose\_opt: ('key, \alpha) t \rightarrow ('key \times \alpha) option
                              val uncons : (key, \alpha) t \rightarrow key \times \alpha \times (key, \alpha) t
                              val uncons\_opt: ('key, \alpha) t \rightarrow ('key \times \alpha \times ('key, \alpha) t) option
                              val elements : ('key, \alpha) t \rightarrow ('key \times \alpha) list
                                val mem: (key \rightarrow key \rightarrow int) \rightarrow key \rightarrow (key, \alpha) t \rightarrow bool
                                \mathsf{val}\ remove\ :\ (\textit{`key}\ \rightarrow\ \textit{`key}\ \rightarrow\ \textit{int})\ \rightarrow\ \textit{`key}\ \rightarrow\ (\textit{`key},\ \alpha)\ t\ \rightarrow\ (\textit{`key},\ \alpha)\ t
                              val union: (key \rightarrow key \rightarrow int) \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow
                                               ('key, \alpha) t \rightarrow ('key, \alpha) t \rightarrow ('key, \alpha) t
                              val compose : ('key \rightarrow 'key \rightarrow int) \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha option) \rightarrow
                                               ('key, \alpha) t \rightarrow ('key, \alpha) t \rightarrow ('key, \alpha) t
                              \mathsf{val}\ iter\ :\ (\mathit{'key}\ \rightarrow\ \alpha\ \rightarrow\ \mathit{unit})\ \rightarrow\ (\mathit{'key},\ \alpha)\ t\ \rightarrow\ \mathit{unit}
                              val map : (\alpha \rightarrow \beta) \rightarrow (key, \alpha) t \rightarrow (key, \beta) t
                                \mathsf{val}\ \mathit{mapi}\ :\ (\mathit{`key}\ \rightarrow\ \alpha\ \rightarrow\ \beta)\ \rightarrow\ (\mathit{`key},\ \alpha)\ t\ \rightarrow\ (\mathit{`key},\ \beta)\ t
                              \mathsf{val}\; \mathit{fold}\; :\; (\,{}^{\backprime}\!\mathit{key}\; \rightarrow\; \alpha\; \rightarrow\; \beta\; \rightarrow\; \beta)\; \rightarrow\; (\,{}^{\backprime}\!\!\mathit{key},\; \alpha)\; t\; \rightarrow\; \beta\; \rightarrow\; \beta
                              \mathsf{val}\ compare\ :\ (\textit{`key}\ \rightarrow\ \textit{`key}\ \rightarrow\ \textit{int})\ \rightarrow\ (\alpha\ \rightarrow\ \alpha\ \rightarrow\ \textit{int})\ \rightarrow\ 
                                               ('key, \alpha) t \rightarrow ('key, \alpha) t \rightarrow int
                              val canonicalize : ('key \rightarrow 'key \rightarrow int) \rightarrow ('key, \alpha) t \rightarrow ('key, \alpha) t
```

Balanced trees: logarithmic access, but representation not unique.

 $\mathsf{module}\ \mathit{Tree}\ :\ \mathit{T}$

Sorted lists: representation unique, but linear access.

 $\mathsf{module}\ \mathit{List}\ :\ \mathit{T}$

 $^{^{1}\}mathrm{Extension}$ of code © 1996 by Xavier Leroy

I.2 Implementation of Pmap

```
module type T =
   sig
       type ('key, \alpha) t
       val empty: ('key, \alpha) t
       val is\_empty : ('key, \alpha) t \rightarrow bool
       val singleton : 'key \rightarrow \alpha \rightarrow ('key, \alpha) t
       \mathsf{val}\ add\ :\ (\textit{`key}\ \rightarrow\ \textit{`key}\ \rightarrow\ \textit{int})\ \rightarrow\ \textit{`key}\ \rightarrow\ \alpha\ \rightarrow\ (\textit{`key},\ \alpha)\ t\ \rightarrow\ (\textit{`key},\ \alpha)\ t
       val update : ('key \rightarrow 'key \rightarrow int) \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow
            'key \rightarrow \alpha \rightarrow ('key, \alpha) t \rightarrow ('key, \alpha) t
       \mathsf{val}\ cons\ :\ (\ 'key\ 	o\ 'key\ 	o\ int)\ 	o\ (lpha\ 	o\ lpha\ option)\ 	o
            'key \rightarrow \alpha \rightarrow ('key, \alpha) t \rightarrow ('key, \alpha) t
       val find: ('key \rightarrow 'key \rightarrow int) \rightarrow 'key \rightarrow ('key, \alpha) t \rightarrow \alpha
       val find\_opt: ('key \rightarrow 'key \rightarrow int) \rightarrow 'key \rightarrow ('key, \alpha) t \rightarrow \alpha option
       val choose : ('key, \alpha) t \rightarrow 'key \times \alpha
       val choose\_opt: ('key, \alpha) t \rightarrow ('key \times \alpha) option
       val uncons: ('key, \alpha) t \rightarrow 'key \times \alpha \times ('key, \alpha) t
       val uncons\_opt: ('key, \alpha) t \rightarrow ('key \times \alpha \times ('key, \alpha) t) option
       val elements: ('key, \alpha) t \rightarrow ('key \times \alpha) list
       val mem : (key \rightarrow key \rightarrow int) \rightarrow key \rightarrow (key, \alpha) t \rightarrow bool
       \mathsf{val}\ \mathit{remove}\ :\ (\mathit{`key}\ \rightarrow\ \mathit{`key}\ \rightarrow\ \mathit{int})\ \rightarrow\ \mathit{`key}\ \rightarrow\ (\mathit{`key},\ \alpha)\ t\ \rightarrow\ (\mathit{`key},\ \alpha)\ t
       val union: ('key \rightarrow 'key \rightarrow int) \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow
           (key, \alpha) t \rightarrow (key, \alpha) t \rightarrow (key, \alpha) t
       \mathsf{val}\ compose\ :\ (\textit{`key}\ \rightarrow\ \textit{`key}\ \rightarrow\ \textit{int})\ \rightarrow\ (\alpha\ \rightarrow\ \alpha\ \rightarrow\ \alpha\ option)\ \rightarrow\ 
           ('key, \alpha) t \rightarrow ('key, \alpha) t \rightarrow ('key, \alpha) t
       val iter: (key \rightarrow \alpha \rightarrow unit) \rightarrow (key, \alpha) t \rightarrow unit
       val map : (\alpha \rightarrow \beta) \rightarrow (key, \alpha) t \rightarrow (key, \beta) t
       val mapi: ('key \rightarrow \alpha \rightarrow \beta) \rightarrow ('key, \alpha) t \rightarrow ('key, \beta) t
       \mathsf{val}\; \mathit{fold}\; :\; (\,{}^{\backprime}\!\mathit{key}\; \rightarrow\; \alpha\; \rightarrow\; \beta\; \rightarrow\; \beta)\; \rightarrow\; (\,{}^{\backprime}\!\mathit{key},\; \alpha)\; t\; \rightarrow\; \beta\; \rightarrow\; \beta
       val compare : (key \rightarrow key \rightarrow int) \rightarrow (\alpha \rightarrow \alpha \rightarrow int) \rightarrow
           (key, \alpha) t \rightarrow (key, \alpha) t \rightarrow int
       val canonicalize : ('key \rightarrow 'key \rightarrow int) \rightarrow ('key, \alpha) t \rightarrow ('key, \alpha) t
    end
module Tree =
   struct
       type ('key, \alpha) t =
              Node of ('key, \alpha) t \times 'key \times \alpha \times ('key, \alpha) t \times int
       let empty = Empty
       let is\_empty = function
            \mid Empty \rightarrow true
           \mid \perp \rightarrow false
       let singleton \ k \ d =
           Node (Empty, k, d, Empty, 1)
       \mathsf{let}\ height\ =\ \mathsf{function}
            Empty \rightarrow 0
            \mid Node (\_,\_,\_,\_,h) \rightarrow h
       let create \ l \ x \ d \ r =
           let hl = height l and hr = height r in
           Node (l, x, d, r, (if hl \ge hr then hl + 1 else hr + 1))
       let bal \ l \ x \ d \ r =
           let hl = \mathsf{match}\ l\ \mathsf{with}\ Empty\ \to\ 0\ |\ Node\ (\_,\_,\_,\_,h)\ \to\ h\ \mathsf{in}
           let hr = \mathsf{match}\ r\ \mathsf{with}\ Empty \to 0 \mid Node\ (\_,\_,\_,h) \to h\ \mathsf{in}
           if hl > hr + 2 then begin
               match l with
               | Empty \rightarrow invalid\_arg  "Map.bal"
```

```
| Node (ll, lv, ld, lr, \_) \rightarrow
                 if height ll > height lr then
                    create \ ll \ lv \ ld \ (create \ lr \ x \ d \ r)
                 else begin
                   match lr with
                      Empty \rightarrow invalid\_arg "Map.bal"
                      Node (lrl, lrv, lrd, lrr, \_) \rightarrow
                         create (create ll lv ld lrl) lrv lrd (create lrr x d r)
                 end
        end else if hr > hl + 2 then begin
           match r with
             Empty → invalid_arg "Map.bal"
           \mid Node (rl, rv, rd, rr, \_) \rightarrow
                if height rr \geq height rl then
                    create (create l x d rl) rv rd rr
                 else begin
                   match rl with
                      Empty \rightarrow invalid\_arg "Map.bal"
                      Node (rll, rlv, rld, rlr, \_) \rightarrow
                         create (create l x d rll) rlv rld (create rlr rv rd rr)
                 end
        end else
           Node (l, x, d, r, (if hl \ge hr then hl + 1 else hr + 1))
     let rec join \ l \ x \ d \ r =
        match bal \ l \ x \ d \ r with
           Empty → invalid_arg "Pmap.join"
          \begin{array}{lll} Node \; (l', \; x', \; d', \; r', \; \_) \; \text{as} \; t' \; \rightarrow \\ & \text{let} \; d \; = \; height \; l' \; - \; height \; r' \; \text{in} \end{array}
              if d < -2 \lor d > 2 then
                join l' x' d' r'
              else
Merge two trees t1 and t2 into one. All elements of t1 must precede the elements of t2. Assumes height t1 - height t2 \leq
     let rec merge t1 t2 =
        match t1, t2 with
           Empty, t \rightarrow t
           t, Empty \rightarrow t
        | Node (l1, v1, d1, r1, h1), Node (l2, v2, d2, r2, h2) \rightarrow
              bal l1 v1 d1 (bal (merge r1 l2) v2 d2 r2)
Same as merge, but does not assume anything about t1 and t2.
     let rec concat \ t1 \ t2 =
        match t1, t2 with
        \mid Empty, t \rightarrow t
           t, Empty \rightarrow t
        | Node (l1, v1, d1, r1, h1), Node (l2, v2, d2, r2, h2) \rightarrow
              join l1 v1 d1 (join (concat r1 l2) v2 d2 r2)
Splitting
     let rec split \ cmp \ x = function
          Empty \rightarrow (Empty, None, Empty)
        | Node (l, v, d, r, \_) \rightarrow
              \mathsf{let}\ c\ =\ cmp\ x\ v\ \mathsf{in}
              \quad \text{if } c \ = \ 0 \ \text{then} \\
                 (l, Some d, r)
              else if c < 0 then
                 \mathsf{let}\ ll,\ vl,\ rl\ =\ split\ cmp\ x\ l\ \mathsf{in}
                 (ll, vl, join rl v d r)
              else (* if c > 0 then *)
```

```
let lr, vr, rr = split cmp x r in
           (join \ l \ v \ d \ lr, \ vr, \ rr)
let rec find \ cmp \ x = function
   \mid Empty \rightarrow raise\ Not\_found
   Node (l, v, d, r, \_) \rightarrow
        let c = cmp x v in
        \quad \text{if } c \ = \ 0 \ \text{then} \\
           d
         else if c\ <\ 0 then
           find cmp \ x \ l
        else (* if c > 0 *)
           find cmp \ x \ r
let rec find\_opt \ cmp \ x = function
     Empty \rightarrow None
     Node (l, v, d, r, \_) \rightarrow
         let c = cmp x v in
         if c = 0 then
            Some d
        else if c\ <\ 0 then
           find\_opt\ cmp\ x\ l
         else (* if c > 0 *)
           find\_opt\ cmp\ x\ r
let rec mem \ cmp \ x = function
     Empty \rightarrow \mathsf{false}
     Node (l, v, d, r, \_) \rightarrow
        let c = cmp x v in
        if c = 0 then
           true
         else if c\ <\ 0 then
           mem\ cmp\ x\ l
        else (* if c > 0 *)
           mem\ cmp\ x\ r
let \ choose = function
     Empty \rightarrow raise\ Not\_found
     Node (l, v, d, r, \_) \rightarrow (v, d)
let \ choose\_opt = function
   \mid Empty \rightarrow None
   | Node (l, v, d, r, \_) \rightarrow Some (v, d) |
\mathsf{let} \; uncons \; = \; \mathsf{function}
     Empty \rightarrow raise\ Not\_found
   | Node (l, v, d, r, h) \rightarrow (v, d, merge l r)
let uncons\_opt = function
     Empty \rightarrow None
     Node (l, v, d, r, h) \rightarrow Some (v, d, merge l r)
\mathsf{let} \ \mathsf{rec} \ \mathit{remove} \ \mathit{cmp} \ x \ = \ \mathsf{function}
   \mid Empty \rightarrow Empty
   | \ \textit{Node} \ (l, \ v, \ d, \ r, \ h) \ \rightarrow
        let c = cmp x v in
        if c = 0 then
           merge\ l\ r
        \  \, {\rm else} \,\, {\rm if} \,\, c \,\, < \,\, 0 \,\, {\rm then} \,\,
            bal\ (remove\ cmp\ x\ l)\ v\ d\ r
         else (* if c > 0 *)
            bal\ l\ v\ d\ (remove\ cmp\ x\ r)
let rec cons \ cmp \ resolve \ x \ data' = function
   | Empty \rightarrow Node (Empty, x, data', Empty, 1)
   | Node (l, v, data, r, h) \rightarrow
```

```
\mathsf{let}\ c\ =\ cmp\ x\ v\ \mathsf{in}
       if c = 0 then
          match resolve data' data with
            Some data" \rightarrow Node (l, x, data'', r, h)
            None \rightarrow merge \ l \ r
        else if c < 0 then
          bal (cons cmp resolve x data' l) v data r
        else (* if c > 0 *)
          bal l v data (cons cmp resolve x data' r)
let rec update \ cmp \ resolve \ x \ data' = function
    Empty \rightarrow Node (Empty, x, data', Empty, 1)
    Node (l, v, data, r, h) \rightarrow
       let c = cmp x v in
        \quad \text{if } c \ = \ 0 \ \text{then} \\
          Node (l, x, resolve data' data, r, h)
        else if c < 0 then
          bal (update cmp resolve x data' l) v data r
        else (* if c > 0 *)
          bal l v data (update cmp resolve x data' r)
let add \ cmp \ x \ data = update \ cmp \ (fun \ n \ o \rightarrow n) \ x \ data
let rec compose \ cmp \ resolve \ s1 \ s2 =
  match s1, s2 with
     Empty, t2 \rightarrow t2
     t1, Empty \rightarrow t1
   | Node (l1, v1, d1, r1, h1), Node (l2, v2, d2, r2, h2) \rightarrow
        if h1 \geq h2 then
          if h2 = 1 then
             cons \ cmp \ (fun \ o \ n \ \rightarrow \ resolve \ n \ o) \ v2 \ d2 \ s1
          else begin
             match split cmp v1 s2 with
             | l2', None, r2' \rightarrow
                  join (compose cmp resolve l1 l2') v1 d1
                     (compose \ cmp \ resolve \ r1 \ r2')
             | l2', Some d, r2' \rightarrow
                  begin match resolve d1 d with
                  | None \rightarrow
                        concat (compose cmp resolve l1 l2')
                          (compose cmp resolve r1 r2')
                  \mid Some \ d \rightarrow
                       join (compose cmp resolve l1 l2') v1 d
                          (compose\ cmp\ resolve\ r1\ r2')
                  end
          end
       else
          if h1 = 1 then
             cons cmp resolve v1 d1 s2
          else begin
             match split \ cmp \ v2 \ s1 with
             | l1', None, r1' \rightarrow
                  join (compose cmp resolve l1' l2) v2 d2
                     (compose cmp resolve r1' r2)
             | l1', Some d, r1' \rightarrow
                  begin match resolve d d2 with
                  | None \rightarrow
                        concat (compose cmp resolve l1' l2)
                          (compose cmp resolve r1' r2)
                  \mid Some \ d \rightarrow
                       join (compose cmp resolve l1' l2) v2 d
                          (compose cmp resolve r1' r2)
                  end
```

end

```
let rec union \ cmp \ resolve \ s1 \ s2 =
  match s1, s2 with
     Empty, t2 \rightarrow t2
     t1, Empty \rightarrow t1
   | Node (l1, v1, d1, r1, h1), Node (l2, v2, d2, r2, h2) \rightarrow
        if h1 \ge h2 then
          if h2 = 1 then
             update \ cmp \ (fun \ o \ n \ 
ightarrow \ resolve \ n \ o) \ v2 \ d2 \ s1
          else begin
             match split\ cmp\ v1\ s2 with
             | l2', None, r2' \rightarrow
                  join (union cmp resolve l1 l2') v1 d1
                     (union cmp resolve r1 \ r2')
             | l2', Some d, r2' \rightarrow
                  join (union cmp resolve l1 l2') v1 (resolve d1 d)
                     (union\ cmp\ resolve\ r1\ r2')
           end
        else
          if h1 = 1 then
             update cmp resolve v1 d1 s2
          else begin
             match split cmp v2 s1 with
             | l1', None, r1' \rightarrow
                  join (union cmp resolve l1' l2) v2 d2
                     (union\ cmp\ resolve\ r1'\ r2)
             | l1', Some d, r1' \rightarrow
                  join (union cmp resolve l1' l2) v2 (resolve d d2)
                     (union cmp resolve r1' r2)
          end
\mathsf{let} \ \mathsf{rec} \ \mathit{iter} \ f \ = \ \mathsf{function}
    Empty \rightarrow ()
   | Node (l, v, d, r, \_) \rightarrow iter f l; f v d; iter f r
let rec map f = function
    Empty \rightarrow Empty
    Node (l, v, d, r, h) \rightarrow Node (map f l, v, f d, map f r, h)
let rec mapi f = function
     Empty \rightarrow Empty
   | Node(l, v, d, r, h) \rightarrow Node(mapi f l, v, f v d, mapi f r, h)|
let \operatorname{rec} fold f \ m \ accu =
  \mathsf{match}\ m\ \mathsf{with}
    Empty \rightarrow accu
   | Node (l, v, d, r, \_) \rightarrow fold f l (f v d (fold f r accu))
let rec compare' cmp\_k cmp\_d l1 l2 =
  match l1, l2 with
    [], [] \rightarrow 0
    [], \rightarrow -1
     [] \rightarrow 1
     Empty :: t1, Empty :: t2 \rightarrow compare' cmp\_k cmp\_d t1 t2
    Node (Empty, v1, d1, r1, \_) :: t1,
        Node (Empty, v2, d2, r2, _{-}) :: t2 \rightarrow
        let cv = cmp_k v1 v2 in
        if cv~\neq~0 then begin
           cv
        end else begin
          let cd = cmp_d d1 d2 in
          if cd \neq 0 then
             cd
```

```
else
                   compare' \ cmp\_k \ cmp\_d \ (r1 :: t1) \ (r2 :: t2)
        | Node (l1, v1, d1, r1, \perp) :: t1, t2 \rightarrow
             compare' \ cmp\_k \ cmp\_d \ (l1 :: Node \ (Empty, \ v1, \ d1, \ r1, \ 0) :: t1) \ t2
        \mid t1, Node (l2, v2, d2, r2, \_) :: t2 \rightarrow
             compare' \ cmp\_k \ cmp\_d \ t1 \ (l2 \ :: \ Node \ (Empty, \ v2, \ d2, \ r2, \ 0) \ :: \ t2)
     let compare \ cmp\_k \ cmp\_d \ m1 \ m2 = compare' \ cmp\_k \ cmp\_d \ [m1] \ [m2]
     let rec elements' accu = function
          Empty \rightarrow accu
        | Node (l, v, d, r, \_) \rightarrow elements' ((v, d) :: elements' accu r) l
     let elements s =
        elements' [] s
     let canonicalize \ cmp \ m =
        fold (add cmp) m empty
  end
module List =
  struct
     type ('key, \alpha) t = ('key \times \alpha) list
     let empty = []
     let is\_empty = function
        \mid \; [\;] \; 
ightarrow \; \mathsf{true}
        \perp \rightarrow false
     let singleton \ k \ d = [(k, \ d)]
     let rec cons \ cmp \ resolve \ k' \ d' = function
        | [] \rightarrow [(k', d')]
        \mid ((k, d) \text{ as } kd :: rest) \text{ as } list \rightarrow
             let c = cmp k' k in
             if c = 0 then
                match resolve d' d with
                  None \rightarrow rest
                Some d'' \rightarrow (k', d'') :: rest
             else if c < 0 then (*k' < k*)
                (k', d') :: list
             else (* if c > 0, i. e. k < k'*)
                kd :: cons\ cmp\ resolve\ k'\ d'\ rest
     let rec update \ cmp \ resolve \ k' \ d' = function
        | [] \rightarrow [(k', d')]
        ((k, d) \text{ as } kd :: rest) \text{ as } list \rightarrow
             let c = cmp k' k in
             if c = 0 then
                (k', resolve d' d) :: rest
             else if c < 0 then (*k' < k*)
                (k', d') :: list
             else (* if c > 0, i. e. k < k' *)
                kd :: update cmp resolve k' d' rest
     let add \ cmp \ k' \ d' \ list =
        update \ cmp \ (fun \ n \ o \rightarrow n) \ k' \ d' \ list
     let rec find \ cmp \ k' = function
        [] \rightarrow raise\ Not\_found
        (k, d) :: rest \rightarrow
             let c = cmp k' k in
             if c = 0 then
                d
             else if c < 0 then (*k' < k*)
```

```
raise\ Not\_found
        else (* if c > 0, i. e. k < k' *)
          find cmp \ k' \ rest
let rec find\_opt \ cmp \ k' = function
  | [] \rightarrow None
  | (k, d) :: rest \rightarrow
        let c = cmp k' k in
        \quad \text{if} \ c \ = \ 0 \ \text{then} \\
           Some d
        else if c < 0 then (* k' < k *)
           None
        else (* if c > 0, i. e. k < k' *)
          find\_opt \ cmp \ k' \ rest
let \ choose = function
  | [] \rightarrow raise\ Not\_found
  \mid kd :: \_ \rightarrow kd
let rec choose\_opt = function
  | [] \rightarrow None
  \mid kd :: \_ \rightarrow Some \ kd
let \ uncons = function
  | \ | \ | \rightarrow raise\ Not\_found
  (k, d) :: rest \rightarrow (k, d, rest)
let uncons\_opt = function
  | [] \rightarrow None
  (k, d) :: rest \rightarrow Some (k, d, rest)
let elements list = list
let rec mem \ cmp \ k' = function
  | [] \rightarrow \mathsf{false}
  (k, d) :: rest \rightarrow
        let c = cmp k' k in
        if c = 0 then
          true
        else if c < 0 then (*k' < k *)
        else (* if c > 0, i. e. k < k' *)
          mem\ cmp\ k'\ rest
let rec remove \ cmp \ k' = function
  | [] \rightarrow []
   ((k, d) \text{ as } kd :: rest) \text{ as } list \rightarrow
        let c = cmp \ k' \ k in
        \quad \text{if } c \ = \ 0 \ \text{then} \\
        else if c < 0 then (*k' < k*)
        else (* if c > 0, i. e. k < k' *)
          kd :: remove cmp k' rest
let rec compare \ cmp\_k \ cmp\_d \ m1 \ m2 =
  match m1, m2 with
    [], [] \rightarrow 0
   [], \rightarrow -1
   [ -, [] \rightarrow 1
  | (k1, d1) :: rest1, (k2, d2) :: rest2 \rightarrow
        let c = cmp_k k1 k2 in
        if c = 0 then begin
          let c' = cmp_{-}d d1 d2 in
          if c' = 0 then
              compare cmp_k cmp_d rest1 rest2
```

```
else
          end else
  let rec iter f = function
     | [] \rightarrow ()
     (k, d) :: rest \rightarrow f k d; iter f rest
  let rec map f = function
     | [] \rightarrow []
     (k, d) :: rest \rightarrow (k, f d) :: map f rest
  let rec mapi f = function
     | [] \rightarrow []
     (k, d) :: rest \rightarrow (k, f k d) :: mapi f rest
  let rec fold f m \ accu =
     match m with
     | \ | \ | \rightarrow accu
     (k, d) :: rest \rightarrow fold \ f \ rest \ (f \ k \ d \ accu)
  let rec compose cmp resolve m1 m2 =
     match m1,\ m2 with
       [], [] \rightarrow []
      [], m \rightarrow m
     | m, [] \rightarrow m
     ((k1, d1) \text{ as } kd1 :: rest1), ((k2, d2) \text{ as } kd2 :: rest2) \rightarrow
          let c = cmp \ k1 \ k2 \ in
          if c = 0 then
            match resolve d1 d2 with
              None \rightarrow compose cmp resolve rest1 rest2
              Some d \rightarrow (k1, d) :: compose cmp resolve rest1 rest2
          else if c < 0 then (*k1 < k2 *)
            kd1 :: compose cmp resolve rest1 m2
          else (* if c > 0, i. e. k2 < k1 *)
            kd2 :: compose cmp resolve m1 rest2
  let rec union cmp resolve m1 m2 =
     match m1,\ m2 with
      [], [] \rightarrow []
     [], m \rightarrow m
     | m, [] \rightarrow m
     (k1, d1) as kd1 :: rest1, (k2, d2) as kd2 :: rest2 \rightarrow
          let c = cmp \ k1 \ k2 \ in
          if c = 0 then
            (k1, resolve \ d1 \ d2) :: union \ cmp \ resolve \ rest1 \ rest2
          else if c < 0 then (*k1 < k2 *)
            kd1 :: union cmp resolve rest1 m2
          else (* if c > 0, i. e. k2 < k1 *)
            kd2 :: union cmp resolve m1 rest2
  let \ canonicalize \ cmp \ x \ = \ x
end
```

I.3 Interface of Partial

Partial maps that are constructed from assoc lists.

```
\begin{array}{ccc} \mathsf{module} \ \mathsf{type} \ T \ = \\ \mathsf{sig} \end{array}
```

The domain of the map. It needs to be compatible with Map.OrderedType.t type domain

The codomain α can be anything we want.

```
type \alpha t
```

A list of argument-value pairs is mapped to a partial map. If an argument appears twice, the later value takes precedence.

```
val of_list : (domain \times \alpha) list \rightarrow \alpha t
```

Two lists of arguments and values (both must have the same length) are mapped to a partial map. Again the later value takes precedence.

```
val of_lists : domain list \rightarrow \alpha list \rightarrow \alpha t
```

If domain and codomain disagree, we must raise an exception or provide a fallback.

```
exception Undefined of domain val apply: \alpha t \rightarrow domain \rightarrow \alpha val apply\_with\_fallback: (<math>domain \rightarrow \alpha) \rightarrow \alpha t \rightarrow domain \rightarrow \alpha
```

Iff domain and codomain of the map agree, we can fall back to the identity map.

I.4 Implementation of Partial

```
module type T =
  sig
     type domain
     type \alpha t
     val of_list : (domain \times \alpha) list \rightarrow \alpha t
     val of_lists : domain list \rightarrow \alpha list \rightarrow \alpha t
     exception Undefined of domain
     \mathsf{val}\ apply\ :\ \alpha\ t\ \to\ domain\ \to\ \alpha
     val apply\_with\_fallback: (domain \rightarrow \alpha) \rightarrow \alpha t \rightarrow domain \rightarrow \alpha
     \mathsf{val}\ auto\ :\ domain\ t\ 	o\ domain\ 	o\ domain
module Make\ (D: Map.OrderedType): T  with type domain = D.t =
  struct
     \mathsf{module}\ M\ =\ \mathit{Map.Make}\ (D)
     type domain = D.t
     type \alpha t = \alpha M.t
     let of_list l =
        List.fold\_left (fun m (d, v) \rightarrow M.add d v m) M.empty l
     let of_lists domain values =
        of\_list
           (try
               List.map2 (fun d v \rightarrow (d, v)) domain values
            with
            | Invalid\_argument\_(* "List.map2" *) \rightarrow
                 invalid_arg "Partial.of_lists: \( \left \) length \( \mismatch \) \( \)
     let auto partial d =
        try
           M.find d partial
        with
        \mid Not\_found \rightarrow d
     exception Undefined of domain
```

```
let apply partial d =
       try
         M.find d partial
       with
       \mid Not\_found \rightarrow raise (Undefined d)
    let apply\_with\_fallback fallback partial d =
       try
         M.find d partial
       with
       | Not\_found \rightarrow fallback d
  end
                                               I.4.1 Unit Tests
module Test : sig val suite : OUnit.test end =
  struct
    open OUnit
    module P = Make (struct type t = int let compare = compare end)
    let apply_ok =
       "apply/ok" >::
         (fun () \rightarrow
           let p = P.of\_list [ (0,"a"); (1,"b"); (2,"c") ]
            and l = [0; 1; 2] in
            assert_equal [ "a"; "b"; "c" ] (List.map (P.apply p) l))
    let apply_ok2 =
       "apply/ok2" >::
         (fun () \rightarrow
           let p = P.of\_lists [0; 1; 2] ["a"; "b"; "c"]
            and l = [0; 1; 2] in
            assert_equal [ "a"; "b"; "c" ] (List.map (P.apply p) l))
    let apply\_shadowed =
       "apply/shadowed" >::
         (fun () \rightarrow
           let p = P.of\_list [ (0,"a"); (1,"b"); (2,"c"); (1,"d") ]
            and l = [0; 1; 2] in
            assert_equal [ "a"; "d"; "c" ] (List.map (P.apply p) l))
    let apply\_shadowed2 =
       "apply/shadowed2" >::
         (\mathsf{fun}\ ()\ \to
           let p = P.of\_lists [0; 1; 2; 1] ["a"; "b"; "c"; "d"]
            and l = [0; 1; 2] in
            assert_equal [ "a"; "d"; "c" ] (List.map (P.apply p) l))
    let apply\_mismatch =
       "apply/mismatch" >::
         (fun () \rightarrow
            assert\_raises
              (Invalid_argument "Partial.of_lists: ulength_mismatch")
              (\text{fun }() \rightarrow P.of\_lists [0; 1; 2] ["a"; "b"; "c"; "d"]))
    let suite\_apply =
       "apply" >:::
         [apply\_ok;
          apply\_ok2;
          apply_shadowed;
          apply\_shadowed2;
          apply\_mismatch]
```

```
let auto\_ok =
     "auto/ok" >::
        (\mathsf{fun}\ ()\ \to
          let p = P.of\_list [ (0, 10); (1, 11) ]
          and l = [0; 1; 2] in
           assert\_equal \ [\ 10;\ 11;\ 2\ ]\ (List.map\ (P.auto\ p)\ l))
  \mathsf{let} \ \mathit{suite\_auto} \ = \\
     "auto" >:::
        [auto\_ok]
  {\tt let} \ \mathit{apply\_with\_fallback\_ok} \ = \\
     "apply_with_fallback/ok" >::
        (fun () \rightarrow
          let p = P.of\_list [(0, 10); (1, 11)]
          and l = [0; 1; 2] in
           assert\_equal
             [10; 11; -2] (List.map (P.apply_with_fallback (fun n \rightarrow -n) p) l))
  {\tt let} \ \mathit{suite\_apply\_with\_fallback} \ = \\
     "apply_with_fallback" >:::
        [apply\_with\_fallback\_ok]
  let suite =
     "Partial" >:::
       [suite\_apply;
         suite\_auto;
         suite\_apply\_with\_fallback]
  let time() =
     ()
end
```

—J— TRIES

From [4], extended for [9].

J.1 Interface of Trie

J.1.1 Monomorphically

```
\begin{array}{l} \text{module type } T = \\ \text{sig} \\ \\ \text{type } key \\ \text{type } (+\alpha) \ t \\ \text{val } empty : \alpha \ t \\ \text{val } is\_empty : \alpha \ t \ \rightarrow \ bool \\ \\ \text{Standard trie interface:} \\ \\ \text{val } add : key \ \rightarrow \ \alpha \ \rightarrow \ \alpha \ t \ \rightarrow \ \alpha \ t \\ \text{val } find : key \ \rightarrow \ \alpha \ t \ \rightarrow \ \alpha \end{array}
```

Functionals:

```
\begin{array}{l} \text{val } remove \ : \ key \ \rightarrow \ \alpha \ t \ \rightarrow \ \alpha \ t \\ \text{val } mem \ : \ key \ \rightarrow \ \alpha \ t \ \rightarrow \ bool \\ \text{val } map \ : \ (\alpha \ \rightarrow \ \beta) \ \rightarrow \ \alpha \ t \ \rightarrow \ \beta \ t \\ \text{val } mapi \ : \ (key \ \rightarrow \ \alpha \ \rightarrow \ \beta) \ \rightarrow \ \alpha \ t \ \rightarrow \ \beta \ t \\ \text{val } iter \ : \ (key \ \rightarrow \ \alpha \ \rightarrow \ unit) \ \rightarrow \ \alpha \ t \ \rightarrow \ unit \\ \text{val } fold \ : \ (key \ \rightarrow \ \alpha \ \rightarrow \ \beta \ \rightarrow \ \beta) \ \rightarrow \ \alpha \ t \ \rightarrow \ \beta \ \rightarrow \ \beta \end{array}
```

Try to match a longest prefix and return the unmatched rest.

```
\mathsf{val}\ longest\ :\ key\ \to\ \alpha\ t\ \to\ \alpha\ option\ \times\ key
```

Try to match a shortest prefix and return the unmatched rest.

```
\mathsf{val}\ shortest\ :\ key\ \to\ \alpha\ t\ \to\ \alpha\ option\ 	imes\ key
```

J.1.2 New in O'Caml 3.08

```
\begin{array}{l} \mathsf{val}\ compare\ :\ (\alpha\ \to\ \alpha\ \to\ int)\ \to\ \alpha\ t\ \to\ \alpha\ t\ \to\ int \\ \mathsf{val}\ equal\ :\ (\alpha\ \to\ \alpha\ \to\ bool)\ \to\ \alpha\ t\ \to\ \alpha\ t\ \to\ bool \end{array}
```

J.1.3 O'Mega customization

export f-open f-close f-descend f-match trie allows us to export the trie trie as source code to another programming language.

end

O'Caml's Map.S prior to Version 3.12:

```
module type Map_-S =
   sig
       type key
       type (+\alpha) t
       val empty: \alpha t
       val\ is\_empty: \alpha\ t \rightarrow bool
       \mathsf{val}\ add:\ key\ \rightarrow\ \alpha\ \rightarrow\ \alpha\ t\ \rightarrow\ \alpha\ t
       \mathsf{val}\; \mathit{find}:\; \mathit{key}\; \rightarrow\; \alpha\; t\; \rightarrow\; \alpha
       val remove: key \rightarrow \alpha t \rightarrow \alpha t
       val\ mem:\ key\ 	o\ lpha\ t\ 	o\ bool
       \mathsf{val}\ iter:\ (key\ \rightarrow\ \alpha\ \rightarrow\ unit)\ \rightarrow\ \alpha\ t\ \rightarrow\ unit
       \mathsf{val}\ map:\ (\alpha\ \to\ \beta)\ \to\ \alpha\ t\ \to\ \beta\ t
       val mapi: (key \rightarrow \alpha \rightarrow \beta) \rightarrow \alpha t \rightarrow \beta t
       \mathsf{val}\ fold:\ (key\ \rightarrow\ \alpha\ \rightarrow\ \beta\ \rightarrow\ \beta)\ \rightarrow\ \alpha\ t\ \rightarrow\ \beta\ \rightarrow\ \beta
       \mathsf{val}\ compare:\ (\alpha\ \rightarrow\ \alpha\ \rightarrow\ int)\ \rightarrow\ \alpha\ t\ \rightarrow\ \alpha\ t\ \rightarrow\ int
       val\ equal: (\alpha \rightarrow \alpha \rightarrow bool) \rightarrow \alpha t \rightarrow \alpha t \rightarrow bool
module Make (M : Map\_S) : T with type key = M.key list
module MakeMap (M : Map\_S) : Map\_S with type key = M.key list
                                                                          J.1.4 Polymorphically
module type Poly =
   sig
        type (\alpha, \beta) t
       val empty : (\alpha, \beta) t
Standard trie interface:
        \mathsf{val}\ add\ :\ (\alpha\ \to\ \alpha\ \to\ int)\ \to\ \alpha\ list\ \to\ \beta\ \to\ (\alpha,\ \beta)\ t\ \to\ (\alpha,\ \beta)\ t
        val find : (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha \ list \rightarrow (\alpha, \beta) \ t \rightarrow \beta
Functionals:
       val remove : (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha list \rightarrow (\alpha, \beta) t \rightarrow (\alpha, \beta) t
       val mem : (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha list \rightarrow (\alpha, \beta) t \rightarrow bool
       \mathsf{val}\ \mathit{map}\ :\ (\beta\ \to\ \gamma)\ \to\ (\alpha,\ \beta)\ t\ \to\ (\alpha,\ \gamma)\ t
       val mapi : (\alpha \ list \rightarrow \beta \rightarrow \gamma) \rightarrow (\alpha, \beta) \ t \rightarrow (\alpha, \gamma) \ t
        \mathsf{val}\ iter\ :\ (\alpha\ list\ \rightarrow\ \beta\ \rightarrow\ unit)\ \rightarrow\ (\alpha,\ \beta)\ t\ \rightarrow\ unit
        \mathsf{val}\ \mathit{fold}\ :\ (\alpha\ \mathit{list}\ \rightarrow\ \beta\ \rightarrow\ \gamma\ \rightarrow\ \gamma)\ \rightarrow\ (\alpha,\ \beta)\ t\ \rightarrow\ \gamma\ \rightarrow\ \gamma
Try to match a longest prefix and return the unmatched rest.
        val longest: (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha \ list \rightarrow (\alpha, \beta) \ t \rightarrow \beta \ option \times \alpha \ list
Try to match a shortest prefix and return the unmatched rest.
        val shortest : (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha list \rightarrow (\alpha, \beta) t \rightarrow \beta option \times \alpha list
                                                                   J.1.5 O'Mega customization
export f_open f_close f_descend f_match trie allows us to export the trie trie as source code to another
programming language.
       val\ export\ :\ (int 
ightarrow\ unit)\ 
ightarrow\ (int 
ightarrow\ unit)\ 
ightarrow
            (int \rightarrow \alpha \ list \rightarrow \ unit) \rightarrow (int \rightarrow \alpha \ list \rightarrow \ \beta \rightarrow \ unit) \rightarrow (\alpha, \ \beta) \ t \rightarrow \ unit)
```

MakePoly (M : Pmap.T) : Poly

J.2 Implementation of Trie

J.2.1 Monomorphically

```
module type T =
   sig
       \mathsf{type}\ \mathit{key}
       type (+\alpha) t
       val\ empty: \alpha\ t
       val\ is\_empty: \alpha\ t \rightarrow bool
       \mathsf{val}\ add\ :\ key\ \rightarrow\ \alpha\ \rightarrow\ \alpha\ t\ \rightarrow\ \alpha\ t
       \mathsf{val} \ \mathit{find} \ : \ \mathit{key} \ \to \ \alpha \ t \ \to \ \alpha
       \mathsf{val}\ remove\ :\ key\ \to\ \alpha\ t\ \to\ \alpha\ t
       \mathsf{val}\ mem\ :\ key\ \to\ \alpha\ t\ \to\ bool
       \mathsf{val}\ map\ :\ (\alpha\ \to\ \beta)\ \to\ \alpha\ t\ \to\ \beta\ t
       val mapi: (key \rightarrow \alpha \rightarrow \beta) \rightarrow \alpha t \rightarrow \beta t
       val\ iter: (key \rightarrow \alpha \rightarrow unit) \rightarrow \alpha t \rightarrow unit
       \mathsf{val}\; fold\; :\; (key\; \rightarrow\; \alpha\; \rightarrow\; \beta\; \rightarrow\; \beta)\; \rightarrow\; \alpha\; t\; \rightarrow\; \beta\; \rightarrow\; \beta
       \mathsf{val}\ longest\ :\ key\ \to\ \alpha\ t\ \to\ \alpha\ option\ \times\ key
       val shortest: key \rightarrow \alpha t \rightarrow \alpha option \times key
       \mathsf{val}\ compare\ :\ (\alpha\ \to\ \alpha\ \to\ int)\ \to\ \alpha\ t\ \to\ \alpha\ t\ \to\ int
       \mathsf{val}\ equal\ :\ (\alpha\ \to\ \alpha\ \to\ bool)\ \to\ \alpha\ t\ \to\ \alpha\ t\ \to\ bool
       val\ export\ :\ (int 
ightarrow\ unit)\ 
ightarrow\ (int 
ightarrow\ unit)\ 
ightarrow
           (int \rightarrow key \rightarrow unit) \rightarrow (int \rightarrow key \rightarrow \alpha \rightarrow unit) \rightarrow \alpha t \rightarrow unit
O'Caml's Map.S prior to Version 3.12:
module type Map\_S =
   sig
       type key
       type (+\alpha) t
       val empty: \alpha t
       \mathsf{val}\ is\_empty:\ \alpha\ t\ \to\ bool
       \mathsf{val}\ add:\ key\ \rightarrow\ \alpha\ \rightarrow\ \alpha\ t\ \rightarrow\ \alpha\ t
       val\ find:\ key\ 	o\ lpha\ t\ 	o\ lpha
       \mathsf{val}\ remove:\ key\ \rightarrow\ \alpha\ t\ \rightarrow\ \alpha\ t
       \mathsf{val}\ mem:\ key\ \rightarrow\ \alpha\ t\ \rightarrow\ bool
       val iter: (key \rightarrow \alpha \rightarrow unit) \rightarrow \alpha t \rightarrow unit
       val\ map: (\alpha \rightarrow \beta) \rightarrow \alpha t \rightarrow \beta t
       val mapi: (key \rightarrow \alpha \rightarrow \beta) \rightarrow \alpha t \rightarrow \beta t
       \mathsf{val}\ fold:\ (key\ \rightarrow\ \alpha\ \rightarrow\ \beta\ \rightarrow\ \beta)\ \rightarrow\ \alpha\ t\ \rightarrow\ \beta\ \rightarrow\ \beta
       val compare: (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha t \rightarrow \alpha t \rightarrow int
       val\ equal: (\alpha \rightarrow \alpha \rightarrow bool) \rightarrow \alpha t \rightarrow \alpha t \rightarrow bool
module Make\ (M\ :\ Map\_S)\ :\ (T\ with\ type\ key\ =\ M.key\ list)\ =
Derived from SML code by Chris Okasaki [4].
       type key = M.key list
       type \alpha t = Trie \text{ of } \alpha \text{ option } \times \alpha t M.t
       let empty = Trie (None, M.empty)
       let is\_empty = function
              Trie\ (None,\ m)\ 	o\ M.is\_empty\ m
           \mid \ \_ \ 	o \ \mathsf{false}
       let rec add key data trie =
           match key, trie with
              [], Trie (\_, children) \rightarrow Trie (Some data, children)
            | k :: rest, Trie (node, children) \rightarrow
```

```
let t = \text{try } M.\text{find } k \text{ children with } Not\_found \rightarrow empty \text{ in}
             Trie (node, M.add k (add rest data t) children)
     let rec find key trie =
       match key, trie with
        [\ ], Trie\ (None,\ \_) \rightarrow raise\ Not\_found
          [], Trie (Some data, \_) \rightarrow data
        | k :: rest, Trie (\_, children) \rightarrow find rest (M.find k children)
The rest is my own fault ...
     let find1 k children =
       try Some (M.find \ k \ children) with Not\_found \rightarrow None
     let \ add\_non\_empty \ k \ t \ children =
       if t = empty then
          M.remove \ k \ children
          M.add \ k \ t \ children
     let rec remove key trie =
       match key, trie with
         [], Trie (\_, children) \rightarrow Trie (None, children)
        k :: rest, (Trie (node, children) as orig) \rightarrow
             match find1 k children with
               None \rightarrow orig
             | Some t \rightarrow Trie (node, add\_non\_empty k (remove rest t) children)
     let rec mem \ key \ trie =
       match key, trie with
        [], Trie (None, \_) \rightarrow false
        [\ ], Trie\ (Some\ data,\ \_) \rightarrow true
       | k :: rest, Trie (\_, children) \rightarrow
             match find1 k children with
               None \rightarrow \mathsf{false}
             | Some t \rightarrow mem \ rest \ t
     let rec map f = function
         Trie\ (Some\ data,\ children)\ 
ightarrow
             Trie (Some (f \ data), M.map \ (map \ f) children)
        | Trie (None, children) \rightarrow Trie (None, M.map (map f) children)
     let rec mapi' key f = function
       | Trie (Some data, children) \rightarrow
             Trie (Some (f key data), descend key f children)
          Trie\ (None,\ children)\ 	o\ Trie\ (None,\ descend\ key\ f\ children)
     and descend \ key \ f = M.mapi \ (fun \ k \rightarrow mapi' \ (key @ [k]) \ f)
     let mapi f = mapi' [] f
     let rec iter' key f = function
          Trie (Some data, children) \rightarrow f key data; descend key f children
          Trie\ (None,\ children)\ 	o\ descend\ key\ f\ children
     and descend \ key \ f = M.iter \ (fun \ k \rightarrow iter' \ (key @ [k]) \ f)
     let iter f = iter' [] f
     let rec fold' key f t acc =
       match t with
          Trie\ (Some\ data,\ children)\ 	o\ descend\ key\ f\ children\ (f\ key\ data\ acc)
          Trie\ (None,\ children)\ 	o\ descend\ key\ f\ children\ acc
     and descend \ key \ f = M.fold \ (fun \ k \rightarrow fold' \ (key @ [k]) \ f)
     let fold f t acc = fold' [] f t acc
     let rec longest' partial partial_rest key trie =
       match key, trie with
        [], Trie (data, \_) \rightarrow (data, [])
       | k :: rest, Trie (data, children) \rightarrow
             match data, find1 k children with
```

```
| None, None \rightarrow (partial, partial\_rest)|
                 | Some \_, None \rightarrow (data, key) |
                |  _, Some t \rightarrow longest' partial partial rest rest <math>t
      let longest key = longest' None key key
      let rec shortest' partial partial_rest key trie =
          \mathsf{match}\ \mathit{key},\ \mathit{trie}\ \mathsf{with}
            [], Trie (data, \_) \rightarrow (data, [])
            k :: rest, Trie (Some \_ as data, children) \rightarrow (data, key)
          k :: rest, Trie (None, children) \rightarrow
                match find1 k children with
                 | None \rightarrow (partial, partial\_rest)|
                | Some t \rightarrow shortest' partial partial rest rest <math>t
      let shortest key = shortest' None key key
                                                       J.2.2 O'Mega customization
      let rec export' n key f_open f_close f_descend f_match = function
          \mid Trie (Some data, children) \rightarrow
                f-match n key data;
                if children \neq M.empty then
                    descend \ n \ key \ f\_open \ f\_close \ f\_descend \ f\_match \ children
          \mid Trie (None, children) \rightarrow
                if children \neq M.empty then begin
                    f\_descend \ n \ key;
                    descend \ n \ key \ f\_open \ f\_close \ f\_descend \ f\_match \ children
      and descend n key f_open f_close f_descend f_match children =
          f\_open n;
          M.iter (fun k \rightarrow
              export' (succ n) (k :: key) f_open f_close f_descend f_match) children;
      let export f\_open f\_close f\_descend f\_match =
          export' \ 0 \ [] f\_open f\_close f\_descend f\_match
      let compare _ _ _ =
          failwith "incomplete"
      let equal \_ \_ \_ =
          failwith "incomplete"
   end
module MakeMap (M : Map\_S) : (Map\_S \text{ with type } key = M.key \ list) = Make(M)
                                                            J.2.3 Polymorphically
module type Poly =
   sig
      type (\alpha, \beta) t
      val empty : (\alpha, \beta) t
      val add: (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha \ list \rightarrow \beta \rightarrow (\alpha, \beta) \ t \rightarrow (\alpha, \beta) \ t
      \mathsf{val} \ \mathit{find} \ : \ (\alpha \ \to \ \alpha \ \to \ \mathit{int}) \ \to \ \alpha \ \mathit{list} \ \to \ (\alpha, \ \beta) \ t \ \to \ \beta
      val remove: (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha \ list \rightarrow (\alpha, \beta) \ t \rightarrow (\alpha, \beta) \ t
      val mem : (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha list \rightarrow (\alpha, \beta) t \rightarrow bool
      val map : (\beta \rightarrow \gamma) \rightarrow (\alpha, \beta) t \rightarrow (\alpha, \gamma) t
      \mathsf{val}\ \mathit{mapi}\ :\ (\alpha\ \mathit{list}\to\ \beta\ \to\ \gamma)\ \to\ (\alpha,\ \beta)\ t\ \to\ (\alpha,\ \gamma)\ t
      \mathsf{val}\ iter\ :\ (\alpha\ list\ \rightarrow\ \beta\ \rightarrow\ unit)\ \rightarrow\ (\alpha,\ \beta)\ t\ \rightarrow\ unit
      val fold: (\alpha \ list \rightarrow \beta \rightarrow \gamma \rightarrow \gamma) \rightarrow (\alpha, \beta) \ t \rightarrow \gamma \rightarrow \gamma
      val longest: (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha \ list \rightarrow (\alpha, \beta) \ t \rightarrow \beta \ option \times \alpha \ list
      val shortest : (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha list \rightarrow (\alpha, \beta) t \rightarrow \beta option \times \alpha list
```

```
val\ export\ :\ (int \rightarrow\ unit)\ \rightarrow\ (int \rightarrow\ unit)\ \rightarrow
        (int \rightarrow \alpha \ list \rightarrow unit) \rightarrow (int \rightarrow \alpha \ list \rightarrow \beta \rightarrow unit) \rightarrow (\alpha, \beta) \ t \rightarrow unit
module\ MakePoly\ (M\ :\ Pmap.T)\ :\ Poly\ =
  struct
Derived from SML code by Chris Okasaki [4].
     type (\alpha, \beta) t = Trie of \beta option \times (\alpha, (\alpha, \beta) t) M.t
     let empty = Trie (None, M.empty)
     let rec add cmp key data trie =
        match key, trie with
          [], Trie (\_, children) \rightarrow Trie (Some data, children)
        \mid k :: rest, Trie (node, children) \rightarrow
             \mathsf{let}\ t\ =\ \mathsf{try}\ M.\mathit{find}\ \mathit{cmp}\ k\ \mathit{children}\ \mathsf{with}\ \mathit{Not\_found}\ \to\ \mathit{empty}\ \mathsf{in}
              Trie (node, M.add cmp k (add cmp rest data t) children)
     let rec find \ cmp \ key \ trie =
        \mathsf{match}\ \mathit{key},\ \mathit{trie}\ \mathsf{with}
        [], Trie (None, \_) \rightarrow raise Not\_found
          [], Trie (Some data, \_) \rightarrow data
        k :: rest, Trie (\_, children) \rightarrow find cmp rest (M.find cmp k children)
The rest is my own fault ...
     let find1 \ cmp \ k \ children =
        try Some (M.find \ cmp \ k \ children) with Not\_found \rightarrow None
     let \ add\_non\_empty \ cmp \ k \ t \ children =
        if t = empty then
           M.remove\ cmp\ k\ children
        else
           M.add\ cmp\ k\ t\ children
     let rec remove cmp key trie =
        match key, trie with
         [], Trie (\_, children) \rightarrow Trie (None, children)
        k :: rest, (Trie (node, children) as orig) \rightarrow
              match find1 cmp k children with
              | None \rightarrow orig
              | Some t \rightarrow Trie (node, add_non_empty cmp k (remove cmp rest t) children)
     let rec mem \ cmp \ key \ trie =
        match key, trie with
          [], Trie (None, \_) \rightarrow false
          [], Trie (Some data, \_) \rightarrow true
        | k :: rest, Trie (\_, children) \rightarrow
             match find1 cmp k children with
              | None \rightarrow false
              \mid Some \ t \rightarrow mem \ cmp \ rest \ t
     let rec map f = function
          Trie\ (Some\ data,\ children)\ 	o
              Trie (Some (f \ data), M.map \ (map \ f) children)
          Trie\ (None,\ children)\ 	o\ Trie\ (None,\ M.map\ (map\ f)\ children)
     let rec mapi' key f = function
          Trie\ (Some\ data,\ children)\ 
ightarrow
              Trie (Some (f key data), descend key f children)
          Trie\ (None,\ children)\ 	o\ Trie\ (None,\ descend\ key\ f\ children)
     and descend key f = M.mapi (fun k \rightarrow mapi' (key @ [k]) f)
     let mapi f = mapi' [] f
     let rec iter' key f = function
        | Trie (Some data, children) \rightarrow f key data; descend key f children
```

```
| Trie\ (None,\ children) \rightarrow descend\ key\ f\ children
  and descend key f = M.iter (fun k \rightarrow iter' (key @ [k]) f)
  let iter f = iter' [] f
  let rec fold' key f t acc =
     match t with
       Trie\ (Some\ data,\ children)\ 	o\ descend\ key\ f\ children\ (f\ key\ data\ acc)
       Trie\ (None,\ children)\ 	o\ descend\ key\ f\ children\ acc
  and descend \ key \ f = M.fold \ (fun \ k \rightarrow fold' \ (key @ [k]) \ f)
  let fold f t acc = fold' [] f t acc
  let rec longest' cmp partial partial_rest key trie =
     match key, trie with
       [], Trie (data, \_) \rightarrow (data, [])
     k :: rest, Trie (data, children) \rightarrow
          match data, find1 cmp k children with
          | None, None \rightarrow (partial, partial\_rest)|
            Some \_, None \rightarrow (data, key)
          |  _, Some t \rightarrow longest' cmp partial partial_rest rest <math>t
  let longest cmp key = longest' cmp None key key
  let rec shortest' cmp partial partial_rest key trie =
     match key, trie with
       [], Trie (data, \_) \rightarrow (data, [])
       k :: rest, Trie (Some \_ as data, children) \rightarrow (data, key)
     | k :: rest, Trie (None, children) \rightarrow
          match find1 cmp k children with
          | None \rightarrow (partial, partial\_rest)|
          | Some t \rightarrow shortest' cmp partial partial rest rest t
  let shortest cmp key = shortest' cmp None key key
                                        J.2.4 O'Mega customization
  let rec export' n key f_open f_close f_descend f_match = function
     | Trie (Some data, children) \rightarrow
          f_{-}match \ n \ key \ data;
          if children \neq M.empty then
             descend \ n \ key \ f\_open \ f\_close \ f\_descend \ f\_match \ children
     | Trie (None, children) \rightarrow
          if children \neq M.empty then begin
            f\_descend \ n \ key;
             descend \ n \ key \ f\_open \ f\_close \ f\_descend \ f\_match \ children
  and descend \ n \ key \ f\_open \ f\_close \ f\_descend \ f\_match \ children =
     f\_open n;
     M.iter (fun k \rightarrow
        export' (succ n) (k :: key) f_open f_close f_descend f_match) children;
  let export f\_open f\_close f\_descend f\_match =
     export' \ 0 \ [] f\_open f\_close f\_descend f\_match
end
```

—K— Tensor Products

From [9].

K.1 Interface of Product

K.1.1 Lists

Since April 2001, we preserve lexicographic ordering.

```
\begin{array}{l} \mathsf{val}\ fold2\ :\ (\alpha \to \beta \to \gamma \to \gamma) \to \alpha\ list \to \beta\ list \to \gamma \to \gamma \\ \mathsf{val}\ fold3\ :\ (\alpha \to \beta \to \gamma \to \delta \to \delta) \to \alpha\ list \to \beta\ list \to \gamma\ list \to \delta \to \delta \\ \mathsf{val}\ fold\ :\ (\alpha\ list \to \beta \to \beta) \to \alpha\ list\ list \to \beta \to \beta \\ \mathsf{val}\ list2\ :\ (\alpha \to \beta \to \gamma) \to \alpha\ list \to \beta\ list \to \gamma\ list \\ \mathsf{val}\ list3\ :\ (\alpha \to \beta \to \gamma \to \delta) \to \alpha\ list \to \beta\ list \to \gamma\ list \to \delta\ list \\ \mathsf{val}\ list\ :\ (\alpha\ list \to \beta) \to \alpha\ list\ list \to \beta\ list \\ \mathsf{Suppress}\ \text{all}\ None\ \text{in the results}. \\ \\ \mathsf{val}\ list2\_opt\ : \end{array}
```

```
\begin{array}{l} \text{Val } list2\_opt \ : \\ (\alpha \to \beta \to \gamma \ option) \to \alpha \ list \to \beta \ list \to \gamma \ list \\ \text{Val } list3\_opt \ : \\ (\alpha \to \beta \to \gamma \to \delta \ option) \to \alpha \ list \to \beta \ list \to \gamma \ list \to \delta \ list \\ \text{Val } list\_opt \ : \\ (\alpha \ list \to \beta \ option) \to \alpha \ list \ list \to \beta \ list \\ \text{Val } power \ : \ int \to \alpha \ list \to \alpha \ list \ list \\ \text{Val } list \to \alpha \ list \ list \to \alpha \ list \ list \\ \end{array}
```

K.1.2 Sets

'a_set is actually α set for a suitable set, but this relation can not be expressed polymorphically (in set) in O'Caml. The two sets can be of different type, but we provide a symmetric version as syntactic sugar.

type α set

```
type (\alpha, \ 'a\_set, \ \beta) \ fold = (\alpha \to \beta \to \beta) \to \ 'a\_set \to \beta \to \beta type (\alpha, \ 'a\_set, \ \beta, \ 'b\_set, \ \gamma) \ fold2 = (\alpha \to \beta \to \gamma \to \gamma) \to \ 'a\_set \to \ 'b\_set \to \gamma \to \gamma val outer: (\alpha, \ 'a\_set, \ \gamma) \ fold \to (\beta, \ 'b\_set, \ \gamma) \ fold \to (\alpha, \ 'a\_set, \ \beta, \ 'b\_set, \ \gamma) \ fold2 val outer\_self: (\alpha, \ 'a\_set, \ \beta) \ fold \to (\alpha, \ 'a\_set, \ \alpha, \ 'a\_set, \ \beta) \ fold2
```

K.2 Implementation of Product

K.2.1 Lists

We use the tail recursive List.fold_left over List.fold_right for efficiency, but revert the argument lists in order to preserve lexicographic ordering. The argument lists are much shorter than the results, so the cost of the List.rev is negligible.

```
let fold2\_rev f l1 l2 acc =
   List.fold\_left (fun acc1 \ x1 \rightarrow
     List.fold\_left (fun acc2 x2 \rightarrow f x1 x2 acc2) acc1 l2) acc l1
let fold2 f l1 l2 acc =
   fold2_rev f (List.rev l1) (List.rev l2) acc
let fold3\_rev f l1 l2 l3 acc =
   List.fold\_left (fun acc1 x1 	o fold2 (f x1) l2 l3 acc1) acc l1
\mathsf{let}\; fold \mathit{3}\; f\; \mathit{l1}\; \mathit{l2}\; \mathit{l3}\; \mathit{acc}\; =\;
  fold3_rev f (List.rev l1) (List.rev l2) (List.rev l3) acc
If all lists have the same type, there's also
let rec fold\_rev f ll acc =
  match ll with
     [] \rightarrow acc
     [l] \rightarrow List.fold\_left (fun acc' x \rightarrow f [x] acc') acc l
        List.fold\_left (fun acc' x \rightarrow fold\_rev (fun xr \rightarrow f(x :: xr)) rest acc') acc l
let fold f ll acc = fold\_rev f (List.map List.rev ll) acc
let list2 op l1 l2 =
  fold2 (fun x1 x2 c \rightarrow op x1 x2 :: c) <math>l1 l2 []
let list3 op l1 l2 l3 =
  fold3 (fun x1 x2 x3 c \rightarrow op x1 x2 x3 ... c) l1 l2 l3 []
let list op ll =
  fold (fun l c \rightarrow op l :: c) ll []
let list2\_opt op l1 l2 =
  fold2
      (fun x1 x2 c \rightarrow
        match op x1 x2 with
          None \rightarrow c
          Some \ op\_x1\_x2 \ \rightarrow \ op\_x1\_x2 \ :: \ c)
     l1 l2 []
let list3\_opt op l1 l2 l3 =
  fold3
      (fun x1 x2 x3 c \rightarrow
        match op x1 x2 x3 with
          None \rightarrow c
         | Some \ op\_x1\_x2\_x3 \rightarrow op\_x1\_x2\_x3 :: c)
     l1 l2 l3 []
let list\_opt op ll =
  fold
      (fun l \ c \rightarrow
        match op l with
           None \rightarrow c
           Some \ op\_l \rightarrow op\_l :: c)
     ll[]
let power n l =
   list (fun x \rightarrow x) (ThoList.clone l n)
Reshuffling lists:
                      [[a_1; \ldots; a_k]; [b_1; \ldots; b_k]; [c_1; \ldots; c_k]; \ldots] \rightarrow [[a_1; b_1; c_1; \ldots]; [a_2; b_2; c_2; \ldots]; \ldots]
                                                                                                                                    (K.1)
    tho: Is this really an optimal implementation?
```

 $\begin{array}{cccc} \mathsf{let} \ thread &=& \mathsf{function} \\ | \ head \ :: \ tail \ \to \end{array}$

K.2.2 Sets

The implementation is amazingly simple:

 $\mathsf{type}\ \alpha\ \mathit{set}$

$$\begin{array}{l} \text{type } (\alpha, \ 'a_set, \ \beta) \ fold = (\alpha \rightarrow \beta \rightarrow \beta) \rightarrow \ 'a_set \rightarrow \beta \rightarrow \beta \\ \text{type } (\alpha, \ 'a_set, \ \beta, \ 'b_set, \ \gamma) \ fold \\ (\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \gamma) \rightarrow \ 'a_set \rightarrow \ 'b_set \rightarrow \gamma \rightarrow \gamma \\ \text{the set } f \ lift \ \ lift$$

let $outer\ fold1\ fold2\ f\ l1\ l2=fold1\ (fun\ x1\to fold2\ (f\ x1)\ l2)\ l1$ let $outer_self\ fold\ f\ l1\ l2=fold\ (fun\ x1\to fold\ (f\ x1)\ l2)\ l1$

—L— (Fiber) Bundles

L.1 Interface of Bundle

See figure L.1 for the geometric intuition behind the bundle structure.



Does the current implementation support faithful projections with a forgetful comparison in the base?

```
\begin{array}{lll} \operatorname{module\ type\ } Elt\_Base\ = \\ \operatorname{sig} \\ \operatorname{type\ } elt \\ \operatorname{type\ } base \\ \operatorname{val\ } compare\_elt\ :\ elt\ \to\ elt\ \to\ int \\ \operatorname{val\ } compare\_base\ :\ base\ \to\ base\ \to\ int \\ \operatorname{end} \\ \operatorname{module\ type\ } Projection\ = \\ \operatorname{sig\ } \\ \operatorname{include\ } Elt\_Base \\ \pi: E \to B \\ \operatorname{val\ } pi\ :\ elt\ \to\ base \\ \operatorname{end} \\ \operatorname{module\ type\ } T\ = \\ \operatorname{sig\ } \\ \end{array}
```

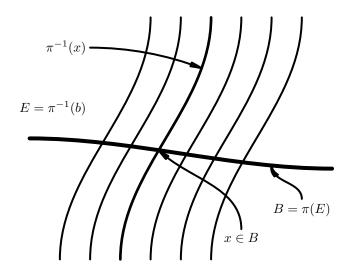


Figure L.1: The bundle structure implemented by Bundle.T

```
type t
      type elt
      type fiber = elt list
      type base
      \mathsf{val}\ add\ :\ elt\ \to\ t\ \to\ t
      val of\_list : elt\ list 	o t
\pi:E\to B
      \mathsf{val}\ pi\ :\ elt\ 	o\ base
\pi^{-1}: B \to E
      val\ inv\_pi : base \rightarrow t \rightarrow fiber
      \mathsf{val}\ base\ :\ t\ \to\ base\ list
\pi^{-1}\circ\pi
      val\ fiber\ :\ elt\ 	o\ t\ 	o\ fiber
      val\ fibers: t \rightarrow (base \times fiber)\ list
module Make\ (P:Projection): T \text{ with type } elt\ =\ P.elt \text{ and type } base\ =\ P.base
The same thing again, but with a projection that is not hardcoded, but passed as an argument at runtime.
module type Dyn =
   sig
      \mathsf{type}\ t
      type elt
      type fiber = elt \ list
      type base
      \mathsf{val}\ add\ :\ (\mathit{elt}\ \rightarrow\ \mathit{base})\ \rightarrow\ \mathit{elt}\ \rightarrow\ t\ \rightarrow\ t
      val \ of\_list : (elt \rightarrow base) \rightarrow elt \ list \rightarrow t
      val\ inv\_pi\ :\ base\ 	o\ t\ 	o\ fiber
      \mathsf{val}\ base\ :\ t\ \to\ base\ list
      \mathsf{val}\; \mathit{fiber}\; :\; (\mathit{elt}\; \rightarrow\; \mathit{base})\; \rightarrow\; \mathit{elt}\; \rightarrow\; t\; \rightarrow\; \mathit{fiber}
      val\ fibers: t \rightarrow (base \times fiber)\ list
module Dyn (P : Elt\_Base) : Dyn \text{ with type } elt = P.elt \text{ and type } base = P.base
```

L.2 Implementation of Bundle

```
module type Elt\_Base =
   sig
      type elt
      type base
      val\ compare\_elt\ :\ elt\ 
ightarrow\ elt\ 
ightarrow\ int
      val\ compare\_base\ :\ base\ 	o\ base\ 	o\ int
   end
module type Dyn =
   sig
      type t
      type elt
      type fiber = elt list
      type base
      val\ add\ :\ (elt\ 	o\ base)\ 	o\ elt\ 	o\ t\ 	o\ t
      \mathsf{val}\ of\_\mathit{list}\ :\ (\mathit{elt}\ \to\ \mathit{base})\ \to\ \mathit{elt}\ \mathit{list}\ \to\ \mathit{t}
      val\ inv\_pi : base \rightarrow t \rightarrow fiber
      \mathsf{val}\ base\ :\ t\ \to\ base\ list
      val\ fiber: (elt \rightarrow base) \rightarrow elt \rightarrow t \rightarrow fiber
```

```
val\ fibers: t \rightarrow (base \times fiber)\ list
module Dyn (P : Elt\_Base) =
  struct
     type elt = P.elt
     type base = P.base
     type fiber = elt list
     module InvPi = Map.Make (struct type t = P.base let compare = P.compare\_base end)
     module Fiber = Set.Make (struct type t = P.elt let compare = P.compare\_elt end)
     \mathsf{type}\ t\ =\ \mathit{Fiber.t}\ \mathit{InvPi.t}
     let add pi element fibers =
       let \ base = pi \ element \ in
       let fiber =
          try InvPi.find\ base\ fibers\ with\ Not\_found\ 	o\ Fiber.empty\ in
       InvPi.add base (Fiber.add element fiber) fibers
     let of\_list pi list =
        List.fold\_right (add pi) list InvPi.empty
     let fibers bundle =
        InvPi.fold
          (fun base fiber acc \rightarrow (base, Fiber.elements fiber) :: acc) bundle []
     let \ base \ bundle =
        InvPi.fold
          (fun base fiber acc \rightarrow base :: acc) bundle []
     let inv_pi base bundle =
       try
          Fiber.elements (InvPi.find base bundle)
       with
       | Not\_found \rightarrow []
     let \ fiber \ pi \ elt \ bundle =
        inv_pi (pi elt) bundle
  end
module type Projection =
     include Elt_Base
     \mathsf{val}\ pi\ :\ elt\ \to\ base
  end
module type T =
  sig
     type t
     type elt
     type fiber = elt \ list
     type base
     \mathsf{val}\ add\ :\ elt\ \to\ t\ \to\ t
     \mathsf{val}\ of\_list\ :\ elt\ list\ \rightarrow\ t
     \mathsf{val}\ pi\ :\ elt\ 	o\ base
     val\ inv\_pi\ :\ base\ 	o\ t\ 	o\ fiber
     \mathsf{val}\ base\ :\ t\ \to\ base\ list
     val\ fiber\ :\ elt\ 	o\ t\ 	o\ fiber
     val\ fibers: t \rightarrow (base \times fiber)\ list
module Make (P : Projection) =
  struct
     \mathsf{module}\ D\ =\ Dyn\ (P)
```

```
type elt = D.elt

type base = D.base

type fiber = D.fiber

type t = D.t

let pi = P.pi

let add = D.add \ pi

let of\_list = D.of\_list \ pi

let base = D.base

let inv\_pi = D.inv\_pi

let fibers = D.fibers

let fiber \ elt \ bundle = inv\_pi \ (pi \ elt) \ bundle
```

end

—M— Power Sets

M.1 Interface of PowSet

Manipulate the power set, i. e. the set of all subsets, of an set $Ordered_Type$. The concrete order is actually irrelevant, we just need it to construct Set.Ss in the implementation. In fact, what we are implementating is the $free\ semilattice\ generated\ from\ the\ set\ of\ subsets\ of\ Ordered_Type$, where the join operation is the set union.

The non trivial operation is *basis*, which takes a set of subsets and returns the smallest set of disjoint subsets from which the argument can be reconstructed by forming unions. It is used in O'Mega for finding coarsest partitions of sets of partiticles.



Eventually, this could be generalized from *power set* or *semi lattice* to *lattice* with a notion of subtraction.

```
module type Ordered\_Type =
  sig
     val\ compare\ :\ t\ 
ightarrow\ t\ 
ightarrow\ int
Debugging ...
     val to\_string : t \rightarrow string
  end
module type T =
  sig
     type elt
     type t
     val\ empty : t
     val\ is\_empty: t \rightarrow bool
Set union (a. k. a. join).
     \mathsf{val}\ union\ :\ t\ list \to\ t
Construct the abstract type from a list of subsets represented as lists and the inverse operation.
     val of\_lists : elt\ list\ list 
ightarrow t
     \mathsf{val}\ to\_\mathit{lists}\ :\ t\ \to\ \mathit{elt}\ \mathit{list}\ \mathit{list}
The smallest set of disjoint subsets that generates the given subset.
     \mathsf{val}\ \mathit{basis}\ :\ t\ \to\ t
Debugging ...
     val\ to\_string : t \rightarrow string
module Make\ (E: Ordered\_Type): T \text{ with type } elt = E.t
```

M.2 Implementation of PowSet

```
\begin{array}{ll} {\sf module\ type\ } {\it Ordered\_Type\ } = \\ {\sf sig\ } \end{array}
```

```
type t
     val compare: t \rightarrow t \rightarrow int
     val to\_string : t \rightarrow string
  end
module type T =
  sig
     \mathsf{type}\ \mathit{elt}
     type t
     val\ empty : t
     val is\_empty : t \rightarrow bool
     \mathsf{val}\ union\ :\ t\ \mathit{list} \to\ t
     val of\_lists : elt\ list\ list 
ightarrow t
     val to\_lists : t \rightarrow elt\ list\ list
     val basis : t \rightarrow t
     val to\_string : t \rightarrow string
module\ Make\ (E\ :\ Ordered\_Type)\ =
  struct
     type elt = E.t
     module ESet = Set.Make (E)
     \mathsf{type}\ set\ =\ ESet.t
     module \ EPowSet = Set.Make \ (ESet)
     type t = EPowSet.t
     let empty = EPowSet.empty
     let is\_empty = EPowSet.is\_empty
     let union \ s\_list =
        List.fold\_right\ EPowSet.union\ s\_list\ EPowSet.empty
     let set_to_string set =
        "{" ^ String.concat "," (List.map E.to_string (ESet.elements set)) ^ "}"
     let to\_string\ powset\ =
        "{" \hat{S}tring.concat "," (List.map set\_to\_string (EPowSet.elements powset)) ^ "}"
     let set\_of\_list = ESet.of\_list
     let of\_lists lists =
        List.fold\_right
           (fun\ list\ acc\ 
ightarrow\ EPowSet.add\ (ESet.of\_list\ list)\ acc)
           lists\ EPowSet.empty
     let to\_lists ps =
        List.map\ ESet.elements\ (EPowSet.elements\ ps)
product\ (s_1, s_2) = s_1 \circ s_2 = \{s_1 \setminus s_2, s_1 \cap s_2, s_2 \setminus s_1\} \setminus \{\emptyset\}
     let product \ s1 \ s2 =
        List.fold\_left
           (fun pset \ set \rightarrow if \ ESet.is\_empty \ set \ then \ pset \ else \ EPowSet.add \ set \ pset)
           EPowSet.empty [ESet.diff s1 s2; ESet.inter s1 s2; ESet.diff s2 s1]
     let disjoint \ s1 \ s2 =
        ESet.is\_empty (ESet.inter\ s1\ s2)
In augment\_basis\_overlapping (s,\{s_i\}_i), we are guaranteed that
                                                          \forall_i : s \cap s_i \neq \emptyset
```

(M.1a)

(M.1b)

Therefore from (M.1b)

$$\forall_{i \neq j} : (s \cap s_i) \cap (s \cap s_j) = s \cap (s_i \cap s_j) = s \cap \emptyset = \emptyset$$
(M.2a)

 $\forall_{i\neq j}: s_i \cap s_j = \emptyset.$

$$\forall_{i \neq j} : (s_i \setminus s) \cap (s_j \setminus s) \subset s_i \cap s_j = \emptyset \tag{M.2b}$$

$$\forall_{i \neq j} : (s \setminus s_i) \cap (s_j \setminus s) \subset s \cap \bar{s} = \emptyset \tag{M.2c}$$

$$\forall_{i \neq j} : (s \cap s_i) \cap (s_j \setminus s) \subset s \cap \bar{s} = \emptyset, \tag{M.2d}$$

but in general

$$\exists_{i \neq j} : (s \setminus s_i) \cap (s \setminus s_j) \neq \emptyset \tag{M.3a}$$

$$\exists_{i \neq j} : (s \setminus s_i) \cap (s \cap s_j) \neq \emptyset, \tag{M.3b}$$

because, e. g., for $s_i = \{i\}$ and $s = \{1, 2, 3\}$

$$(s \setminus s_1) \cap (s \setminus s_2) = \{2, 3\} \cap \{1, 3\} = \{3\}$$
 (M.4a)

$$(s \setminus s_1) \cap (s \cap s_2) = \{2, 3\} \cap \{2\} = \{2\}.$$
 (M.4b)

Summarizing:

Fortunately, we also know from (M.1a) that

$$\forall_i : |s \setminus s_i| < |s| \tag{M.5a}$$

$$\forall_i : |s \cap s_i| < \min(|s|, |s_i|) \tag{M.5b}$$

$$\forall_i : |s_i \setminus s| < |s_i| \tag{M.5c}$$

and can call basis recursively without risking non-termination.

```
let rec basis ps = EPowSet.fold\ augment\_basis\ ps\ EPowSet.empty and augment\_basis\ s\ ps =  if EPowSet.mem\ s\ ps then ps else let no\_overlaps,\ overlaps = EPowSet.partition\ (disjoint\ s)\ ps in if EPowSet.is\_empty\ overlaps then EPowSet.add\ s\ ps
```

else *EPowSet.union no_overlaps (augment_basis_overlapping s overlaps)*

and $augment_basis_overlapping \ s \ ps \ =$

 $basis\ (EPowSet.fold\ (fun\ s'\ o\ EPowSet.union\ (product\ s\ s'))\ ps\ EPowSet.empty)$

end

—N—

COMBINATORICS

N.1 Interface of Combinatorics

This type is defined just for documentation. Below, most functions will construct a (possibly nested) list of partitions or permutations of a α seq.

type $\alpha \ seq \ = \ \alpha \ list$

N.1.1 Simple Combinatorial Functions

The functions

$$factorial: n \to n!$$
 (N.1a)

$$binomial: (n,k) \to \binom{n}{k} = \frac{n!}{k!(n-k)!}$$
(N.1b)

$$multinomial: [n_1; n_2; \dots; n_k] \to \binom{n_1 + n_2 + \dots + n_k}{n_1, n_2, \dots, n_k} = \frac{(n_1 + n_2 + \dots + n_k)!}{n_1! n_2! \cdots n_k!}$$
(N.1c)

have not been optimized. They can quickly run out of the range of native integers.

 $val\ factorial\ :\ int
ightarrow\ int$

 $\begin{array}{lll} \mathsf{val} \ binomial \ : \ int \rightarrow \ int \\ \mathsf{val} \ multinomial \ : \ int \ list \rightarrow \ int \\ \end{array}$

 $symmetry\ l$ returns the size of the symmetric group on l, i.e. the product of the factorials of the numbers of identical elements.

val $symmetry : \alpha \ list \rightarrow \ int$

N.1.2 Partitions

partitions $[n_1; n_2; \ldots; n_k]$ $[x_1; x_2; \ldots; x_n]$, where $n = n_1 + n_2 + \ldots + n_k$, returns all inequivalent partitions of $[x_1; x_2; \ldots; x_n]$ into parts of size n_1, n_2, \ldots, n_k . The order of the n_i is not respected. There are

$$\frac{1}{S(n_1, n_2, \dots, n_k)} \binom{n_1 + n_2 + \dots + n_k}{n_1, n_2, \dots, n_k}$$
(N.2)

such partitions, where the symmetry factor $S(n_1, n_2, ..., n_k)$ is the size of the permutation group of $[n_1; n_2; ...; n_k]$ as determined by the function symmetry.

val partitions: $int\ list
ightarrow \ lpha\ seq\
ightarrow \ lpha\ seq\ list\ list$

 $ordered_partitions$ is identical to partitions, except that the order of the n_i is respected. There are

$$\binom{n_1 + n_2 + \ldots + n_k}{n_1, n_2, \ldots, n_k} \tag{N.3}$$

such partitions.

val $ordered_partitions$: $int list \rightarrow \alpha seq \rightarrow \alpha seq list list$

keystones m l is equivalent to partitions m l, except for the special case when the length of l is even and m contains a part that has exactly half the length of l. In this case only the half of the partitions is created that has the head of l in the longest part.

val $keystones: int list \rightarrow \alpha seq \rightarrow \alpha seq list list$

It can be beneficial to factorize a common part in the partitions and keystones:

```
val factorized\_partitions: int\ list \to \alpha\ seq \to (\alpha\ seq \times \alpha\ seq\ list\ list)\ list val factorized\_keystones: int\ list \to \alpha\ seq \to (\alpha\ seq \times \alpha\ seq\ list\ list)\ list
```

Special Cases

partitions is built from components that can be convenient by themselves, even thepugh they are just special cases of partitions.

 $split\ k\ l$ returns the list of all inequivalent splits of the list l into one part of length k and the rest. There are

$$\frac{1}{S(|l|-k,k)} \binom{|l|}{k} \tag{N.4}$$

such splits. After replacing the pairs by two-element lists, split k l is equivalent to partitions $[k; length \ l-k] \ l$.

val
$$split : int \rightarrow \alpha \ seq \rightarrow (\alpha \ seq \times \alpha \ seq) \ list$$

Create both equipartitions of lists of even length. There are

$$\binom{|l|}{k} \tag{N.5}$$

such splits. After replacing the pairs by two-element lists, the result of $ordered_split\ k\ l$ is equivalent to $ordered_partitions\ [k;\ length\ l\ -\ k]\ l.$

```
val ordered\_split : int \rightarrow \alpha \ seq \rightarrow (\alpha \ seq \times \alpha \ seq) \ list
```

 $multi_split \ n \ k \ l$ returns the list of all inequivalent splits of the list l into n parts of length k and the rest.

```
val multi\_split: int \rightarrow int \rightarrow \alpha \ seq \rightarrow (\alpha \ seq \ list \times \alpha \ seq) \ list val ordered\_multi\_split: int \rightarrow int \rightarrow \alpha \ seq \rightarrow (\alpha \ seq \ list \times \alpha \ seq) \ list
```

N.1.3 Choices

choose $n[x_1; x_2; ...; x_n]$ returns the list of all n-element subsets of $[x_1; x_2; ...; x_n]$. choose n is equivalent to $(map\ fst) \circ (ordered\ split\ n)$.

```
\mathsf{val}\ choose\ :\ int\rightarrow\ \alpha\ seq\ \rightarrow\ \alpha\ seq\ list
```

 $multi_choose \ n \ k$ is equivalent to $(map \ fst) \circ (multi_split \ n \ k)$.

```
\verb|val| multi\_choose|: int \rightarrow int \rightarrow \alpha seq \rightarrow \alpha seq \textit{ list list}|
```

val $ordered_multi_choose: int \rightarrow int \rightarrow \alpha \ seq \rightarrow \alpha \ seq \ list \ list$

N.1.4 Permutations

val $permute : \alpha \ seq \rightarrow \alpha \ seq \ list$

Graded Permutations

```
val permute\_signed: \alpha\ seq \rightarrow (int \times \alpha\ seq)\ list val permute\_even: \alpha\ seq \rightarrow \alpha\ seq\ list val permute\_odd: \alpha\ seq \rightarrow \alpha\ seq\ list val permute\_cyclic: \alpha\ seq \rightarrow \alpha\ seq\ list val permute\_cyclic\_signed: \alpha\ seq \rightarrow (int \times \alpha\ seq)\ list
```

Tensor Products of Permutations

In other words: permutations which respect compartmentalization.

```
val permute\_tensor: \alpha \ seq \ list \rightarrow \alpha \ seq \ list \ list val permute\_tensor\_signed: \alpha \ seq \ list \rightarrow (int \times \alpha \ seq \ list) \ list val permute\_tensor\_even: \alpha \ seq \ list \rightarrow \alpha \ seq \ list \ list val permute\_tensor\_odd: \alpha \ seq \ list \rightarrow \alpha \ seq \ list \ list val sign: ?cmp: (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha \ seq \rightarrow int
```

Sorting

val $sort_signed : ?cmp : (\alpha \rightarrow \alpha \rightarrow int) \rightarrow \alpha seq \rightarrow int \times \alpha seq$

Unit Tests

module Test: sig val suite: OUnit.test end

N.2 Implementation of Combinatorics

Avoid referring to *Pervasives.compare*, because *Pervasives* will become *Stdlib.Pervasives* in O'Caml 4.07 and *Stdlib* in O'Caml 4.08.

let pcompare = compare

type $\alpha \ seq = \alpha \ list$

N.2.1 Simple Combinatorial Functions

```
let rec factorial' fn n =
    if n < 1 then
        fn
    else
        factorial' (n \times fn) (pred \ n)

let factorial n =
    let result = factorial' 1 n in
    if result < 0 then
        invalid\_arg "Combinatorics.factorial\sqcupoverflow"
    else
        result
```

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots1}$$

$$= \frac{n(n-1)\cdots(k+1)}{(n-k)(n-k-1)\cdots1} = \begin{cases} B_{n-k+1}(n,k) & \text{for } k \le \lfloor n/2 \rfloor \\ B_{k+1}(n,n-k) & \text{for } k > \lfloor n/2 \rfloor \end{cases}$$
 (N.6)

where

$$B_{n_{\min}}(n,k) = \begin{cases} nB_{n_{\min}}(n-1,k) & \text{for } n \ge n_{\min} \\ \frac{1}{k}B_{n_{\min}}(n,k-1) & \text{for } k > 1 \\ 1 & \text{otherwise} \end{cases}$$
(N.7)

```
let rec binomial' n\_min n k acc = if n \ge n\_min then binomial' n\_min (pred\ n) k (n \times acc) else if k > 1 then binomial' n\_min n (pred\ k) (acc\ /\ k) else acc let binomial n k = if k > n / 2 then binomial' (k + 1) n (n - k) 1 else binomial' (n - k + 1) n k 1
```

Overflows later, but takes much more time:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \tag{N.8}$$

```
let rec slow\_binomial\ n\ k =
  if n < 0 \lor k < 0 then
    invalid\_arg "Combinatorics.binomial"
  else if k = 0 \lor k = n then
    1
  else slow\_binomial\ (pred\ n)\ k + slow\_binomial\ (pred\ n)\ (pred\ k)

let multinomial\ n\_list =
    List.fold\_left\ (fun\ acc\ n \to acc\ /\ (factorial\ n))
    (factorial\ (List.fold\_left\ (+)\ 0\ n\_list))\ n\_list

let symmetry\ l =
    List.fold\_left\ (fun\ s\ (n,\ \_) \to s\ \times\ factorial\ n)\ 1\ (ThoList.classify\ l)
```

N.2.2 Partitions

The inner steps of the recursion (i. e. n = 1) are expanded as follows

```
split'(1, [p_k; p_{k-1}; \dots; p_1], [x_l; x_{l-1}; \dots; x_1], [x_{l+1}; x_{l+2}; \dots; x_m]) = [([p_1; \dots; p_k; x_{l+1}], [x_1; \dots; x_l; x_{l+2}; \dots; x_m]); ([p_1; \dots; p_k; x_{l+2}], [x_1; \dots; x_l; x_{l+1}; x_{l+3} \dots; x_m]); \dots; ([p_1; \dots; p_k; x_m], [x_1; \dots; x_l; x_{l+1}; \dots; x_{m-1}])] \quad (N.9)
```

while the outer steps (i. e. n > 1) perform the same with one element moved from the last argument to the first argument. At the nth level we have

```
split'(n, [p_k; p_{k-1}; \dots; p_1], [x_l; x_{l-1}; \dots; x_1], [x_{l+1}; x_{l+2}; \dots; x_m]) = [([p_1; \dots; p_k; x_{l+1}; x_{l+2}; \dots; x_{l+n}], [x_1; \dots; x_l; x_{l+n+1}; \dots; x_m]); \dots; ([p_1; \dots; p_k; x_{m-n+1}; x_{m-n+2}; \dots; x_m], [x_1; \dots; x_l; x_{l+1}; \dots; x_{m-n}])] \quad (N.10)
```

where the order of the $[x_1; x_2; ...; x_m]$ is maintained in the partitions. Variations on this multiple recursion idiom are used many times below.

```
\begin{tabular}{ll} \mbox{let rec } split' \ n \ rev\_part \ rev\_head \ = \ \mbox{function} \\ \mbox{let } i \ \mbox{let } rev\_part' \ = \ x \ :: \ rev\_part \ \mbox{and } parts \ = \ split' \ n \ rev\_part \ (x \ :: \ rev\_head) \ tail \ \mbox{in} \\ \mbox{if } n \ < \ 1 \ \mbox{then} \\ \mbox{failwith "Combinatorics.split':$$$$_ican't$$_happen"$ \\ \mbox{else if } n \ = \ 1 \ \mbox{then} \\ \mbox{$(List.rev\ rev\_part',\ List.rev\_append\ rev\_head\ tail)} \ :: \ parts \ \mbox{else} \\ \mbox{$else$} \\ \mbox{$split'$ (pred\ n) \ rev\_part'\ rev\_head\ tail @ parts} \\ \end{tabular}
```

Kick off the recursion for 0 < n < |l| and handle the cases $n \in \{0, |l|\}$ explicitely. Use reflection symmetry for a small optimization.

```
let ordered\_split\_unsafe\ n\ abs\_l\ l=1 let abs\_l=List.length\ l in if n=0 then [[],\ l] else if n=abs\_l then [l,\ []] else if n\leq abs\_l/2 then split'\ n\ []\ []\ l else List.rev\_map\ (\text{fun}\ (a,\ b)\ \rightarrow\ (b,\ a))\ (split'\ (abs\_l-n)\ []\ []\ l)
```

Check the arguments and call the workhorse:

```
let \ ordered\_split \ n \ l =
```

```
let \ abs_l = List.length \ l in
  if n < 0 \lor n > abs\_l then
     invalid_arg "Combinatorics.ordered_split"
  else
     ordered_split_unsafe n abs_l l
Handle equipartitions specially:
let split n l =
  let \ abs\_l = List.length \ l \ in
  if n < 0 \lor n > abs\_l then
     invalid\_arg "Combinatorics.split"
  else begin
     if 2 \times n = abs_l then
        match l with
        [\ ] 
ightarrow \mathit{failwith}  "Combinatorics.split:\Boxcan't\Boxhappen"
          x :: tail \rightarrow
             List.map 	ext{ (fun } (p1, p2) \rightarrow (x :: p1, p2)) 	ext{ (split' (pred n) [] [] tail)}
     else
        ordered\_split\_unsafe\ n\ abs\_l\ l
  end
If we chop off parts repeatedly, we can either keep permutations or suppress them. Generically, attach_to_fst
has type
      (\alpha \times \beta) \ list \rightarrow \alpha \ list \rightarrow (\alpha \ list \times \beta) \ list \rightarrow (\alpha \ list \times \beta) \ list
and semantics
  attach\_to\_fst([(a_1,b_1),(a_2,b_2),\ldots,(a_m,b_m)],[a'_1,a'_2,\ldots]) =
                                                        [([a_1, a'_1, \ldots], b_1), ([a_2, a'_1, \ldots], b_2), \ldots, ([a_m, a'_1, \ldots], b_m)] (N.11)
(where some of the result can be filtered out), assumed to be prepended to the final argument.
let rec multi_split' attach_to_fst n size splits =
  if n < 0 then
     splits
  else
     multi_split' attach_to_fst (pred n) size
        (List.fold\_left (fun acc (parts, tail) \rightarrow
           attach_to_fst (ordered_split size tail) parts acc) [] splits)
let attach\_to\_fst\_unsorted splits parts acc =
   List.fold\_left (fun acc' (p, rest) \rightarrow (p :: parts, rest) :: acc') acc splits
Similarly, if the second argument is a list of lists:
let prepend_to_fst_unsorted splits parts acc =
   List.fold\_left (fun acc' (p, rest) \rightarrow (p @ parts, rest) :: <math>acc') acc \ splits
let attach_to_fst_sorted splits parts acc =
  match parts with
    [] \rightarrow List.fold\_left (fun acc' (p, rest) \rightarrow ([p], rest) :: acc') acc splits
   p :: \_ as parts \rightarrow
        List.fold\_left (fun acc' (p', rest) \rightarrow
          if p' > p then
             (p' :: parts, rest) :: acc'
          else
             acc') acc splits
let multi\_split n size l =
   multi\_split' attach_to_fst_sorted n size [([], l)]
let \ ordered\_multi\_split \ n \ size \ l = 1
   multi\_split' attach_to_fst_unsorted n size [([], l)]
let rec partitions' splits = function
```

```
[] \rightarrow List.map (fun (h, r) \rightarrow (List.rev h, r)) splits
       | (1, size) :: more \rightarrow
                   partitions'
                         (List.fold\_left (fun \ acc \ (parts, \ rest) \rightarrow
                                attach_to_fst_unsorted (split size rest) parts acc)
                                  [] splits) more
       \mid (n, size) :: more \rightarrow
                   partitions'
                         (List.fold\_left (fun acc (parts, rest) \rightarrow
                                prepend_to_fst_unsorted (multi_split n size rest) parts acc)
                                   [] splits) more
location the location of the
      if List.fold\_left (+) 0 multiplicities \neq List.length l then
             invalid\_arg "Combinatorics.partitions"
      else
             List.map\ fst\ (partitions'\ [([\ ],\ l)]
                                                               (ThoList.classify (List.sort compare multiplicities)))
let rec ordered\_partitions' splits = function
      [] \rightarrow List.map (fun (h, r) \rightarrow (List.rev h, r)) splits
       \mid size :: more \rightarrow
                   ordered\_partitions'
                         (List.fold\_left (fun \ acc \ (parts, \ rest) \rightarrow
                                attach_to_fst_unsorted (ordered_split size rest) parts acc)
                                  [] splits) more
let ordered\_partitions multiplicities l =
      if List.fold\_left (+) 0 multiplicities \neq List.length l then
             invalid_arg "Combinatorics.ordered_partitions"
      else
             List.map\ fst\ (ordered\_partitions'\ [([],\ l)]\ multiplicities)
\mathsf{let}\ hdtl\ =\ \mathsf{function}
          [] \rightarrow invalid\_arg "Combinatorics.hdtl"
      | h :: t \rightarrow (h, t)
ThoList.factorize (List.map hdtl (partitions multiplicities l))
```

In order to construct keystones (cf. chapter 3), we must eliminate reflections consistently. For this to work, the lengths of the parts must not be reordered arbitrarily. Ordering with monotonously fallings lengths would be incorrect however, because then some remainders could fake a reflection symmetry and partitions would be dropped erroneously. Therefore we put the longest first and order the remaining with rising lengths:

```
 \begin{array}{l} \text{let } longest\_first \ l = \\ & \text{match } ThoList.classify \ (List.sort \ (\text{fun } n1 \ n2 \ \rightarrow \ compare \ n2 \ n1) \ l) \ \text{with} \\ & \mid [] \ \rightarrow \ [] \\ & \mid longest \ :: \ rest \ \rightarrow \ longest \ :: \ List.rev \ rest \ \\ & \text{let } keystones \ multiplicities \ l = \\ & \text{if } List.fold\_left \ (+) \ 0 \ multiplicities \ \neq \ List.length \ l \ \text{then} \\ & invalid\_arg \ "Combinatorics.keystones" \ \\ & \text{else} \\ & List.map \ fst \ (partitions' \ [([], \ l)] \ (longest\_first \ multiplicities)) \ \\ & \text{let } factorized\_keystones \ multiplicities \ l = \\ & ThoList.factorize \ (List.map \ hdtl \ (keystones \ multiplicities \ l)) \ \\ \end{array}
```

N.2.3 Choices

The implementation is very similar to split', but here we don't have to keep track of the complements of the chosen sets.

```
let rec choose' \ n \ rev\_choice = function
| \ [] \ \rightarrow \ []
```

```
x :: tail \rightarrow
       let rev\_choice' = x :: rev\_choice
       and choices = choose' n rev_choice tail in
       if n < 1 then
          failwith "Combinatorics.choose': _{\sqcup}can't_{\sqcup}happen"
       else if n = 1 then
          List.rev\ rev\_choice'\ ::\ choices
       else
          choose' (pred n) rev_choice' tail @ choices
choose n is equivalent to (List.map fst) \circ (split_ordered n), but more efficient.
let choose \ n \ l =
  let \ abs\_l \ = \ List.length \ l \ in
  if n < 0 then
     invalid_arg "Combinatorics.choose"
  else if n > abs_l then
     else if n = 0 then
     [[]]
  else if n = abs_l then
     [l]
  else
     choose' n [] l
let multi\_choose \ n \ size \ l =
  List.map\ fst\ (multi\_split\ n\ size\ l)
let \ ordered\_multi\_choose \ n \ size \ l = 1
  List.map fst (ordered_multi_split n size l)
                                                 N.2.4 Permutations
let rec insert x = function
  | [] \rightarrow [[x]]
  h :: t \text{ as } l \rightarrow
       (x :: l) :: List.rev\_map (fun l' \rightarrow h :: l') (insert x t)
let permute l =
  List.fold\_left (fun acc \ x \rightarrow ThoList.rev\_flatmap \ (insert \ x) \ acc) [[]] \ l
                                                   Graded Permutations
\mathsf{let} \ \mathsf{rec} \ \mathit{insert\_signed} \ x \ = \ \mathsf{function}
  |(eps, []) \rightarrow [(eps, [x])]
  | (eps, h :: t) \rightarrow (eps, x :: h :: t) ::
       (List.map (fun (eps', l') \rightarrow (-eps', h :: l')) (insert\_signed x (eps, t)))
let rec permute\_signed' = function
  | (eps, []) \rightarrow [(eps, [])]
  (eps, h :: t) \rightarrow ThoList.flatmap (insert\_signed h) (permute\_signed' (eps, t))
let permute\_signed\ l\ =
  permute\_signed' (1, l)
The following are wasting at most a factor of two and there's probably no point in improving on this ...
let filter\_sign \ s \ l =
  List.map \ snd \ (List.filter \ (fun \ (eps, \_) \rightarrow eps = s) \ l)
let permute_even l =
  filter_sign 1 (permute_signed l)
let permute\_odd l =
  filter\_sign (-1) (permute\_signed l)
```



We have a slight inconsistency here: permute[] = [[]], while $permute_cyclic[] = []$. I don't know if it is worth fixing.

```
let permute\_cyclic l =
  let rec permute\_cyclic' acc before = function
     | [] \rightarrow List.rev acc
     \mid x :: rest \text{ as } after \rightarrow
         permute_cyclic' ((after @ List.rev before) :: acc) (x :: before) rest
  in
  permute_cyclic' [] [] l
Algorithm: toggle the signs and at the end map all signs to +1, iff the last sign is positive, i.e. there's an odd
number of elements.
let permute\_cyclic\_signed l =
  let rec permute_cyclic_signed' eps acc before = function
     | [] \rightarrow
         if eps > 0 then
           List.rev\_map (fun (\_, p) \rightarrow (1, p)) acc
           List.rev acc
     \mid x :: rest \text{ as } after \rightarrow
        let eps' = - eps in
         permute\_cyclic\_signed'\ eps'\ ((eps',\ after\ @\ List.rev\ before)\ ::\ acc)\ (x\ ::\ before)\ rest
  in
  permute\_cyclic\_signed'(-1)[][]l
                                            Tensor Products of Permutations
let permute\_tensor ll =
  Product.list (fun l \rightarrow l) (List.map permute ll)
let join\_signs \ l =
  let el, pl = List.split l in
  (List.fold\_left (fun \ acc \ x \rightarrow x \times acc) \ 1 \ el, \ pl)
let\ permute\_tensor\_signed\ ll\ =
  Product.list join_signs (List.map permute_signed ll)
let permute_tensor_even l =
  filter_sign 1 (permute_tensor_signed l)
\mathsf{let}\ permute\_tensor\_odd\ l\ =\ 
  filter\_sign (-1) (permute\_tensor\_signed l)
                                                          Sorting
let insert\_inorder\_signed\ order\ x\ (eps,\ l)\ =
  let rec insert \ eps' \ accu = function
     [] \rightarrow (eps \times eps', List.rev\_append accu [x])
     h :: t \rightarrow
          if order x h = 0 then
             invalid\_arg
               "Combinatorics.insert\_inorder\_signed: \_identical\_elements"
          else if order \ x \ h \ < \ 0 then
             (eps \times eps', List.rev\_append\ accu\ (x :: h :: t))
             insert (-eps') (h :: accu) t
  in
  insert 1 [] l
let sort\_signed\ ?(cmp = pcompare)\ l\ =
  List.fold\_right \ (insert\_inorder\_signed \ cmp) \ l \ (1, \ [])
```

```
let sign\ ?(cmp = pcompare)\ l =
  let eps, \_ = sort\_signed ~cmp l in
let sign2 ? (cmp = pcompare) l =
  let a = Array.of\_list l in
  let eps = ref 1 in
  for j = 0 to Array.length a - 1 do
     \quad \text{for } i \ = \ 0 \ \text{to} \ j \ - \ 1 \ \text{do}
       if cmp \ a.(i) \ a.(j) > 0 then
          eps := -!eps
    done
  done;
  !eps
module Test =
  struct
    open OUnit
    let to\_string =
       ThoList.to\_string\ (ThoList.to\_string\ string\_of\_int)
     let \ assert\_equal\_perms =
       assert_equal ~printer : to_string
    let \ count\_permutations \ n = 1
       let factorial_n = factorial n
       and range = ThoList.range 1 n in
       let sorted = List.sort compare (permute range) in
       (* Verify the count ...*)
       assert_equal factorial_n (List.length sorted);
       (* ... check that they're all different ... *)
       assert_equal factorial_n (List.length (ThoList.uniq sorted));
       (* ... make sure that they a all permutations. *)
       assert\_equal\_perms
          [range] (ThoList.uniq (List.map (List.sort compare) sorted))
     let suite\_permute =
       "permute" >:::
          [ "permute_ [] " >::
              (fun () \rightarrow
                 assert_equal_perms [[]] (permute []));
            "permute_[1]" >::
               (fun () \rightarrow
                 assert_equal_perms [[1]] (permute [1]));
            "permute_[1;2;3]" >::
               (fun () \rightarrow
                 assert\_equal\_perms
                   [2; 3; 1]; [2; 1; 3]; [3; 2; 1];
                      [1; 3; 2]; [1; 2; 3]; [3; 1; 2]
                    (permute [1; 2; 3]);
            "permute_[1;2;3;4]" >::
               (fun () \rightarrow
                 assert\_equal\_perms
                   [ [3; 4; 1; 2]; [3; 1; 2; 4]; [3; 1; 4; 2];
                      [4; 3; 1; 2]; [1; 4; 2; 3]; [1; 2; 3; 4];
                      [1; 2; 4; 3]; [4; 1; 2; 3]; [1; 4; 3; 2];
                      [1; 3; 2; 4]; [1; 3; 4; 2]; [4; 1; 3; 2];
                      [3; 4; 2; 1]; [3; 2; 1; 4]; [3; 2; 4; 1];
                      [4; 3; 2; 1]; [2; 4; 1; 3]; [2; 1; 3; 4];
                      [2; 1; 4; 3]; [4; 2; 1; 3]; [2; 4; 3; 1];
                      [2; 3; 1; 4]; [2; 3; 4; 1]; [4; 2; 3; 1]]
                    (permute [1; 2; 3; 4]));
            "count_{\square}permute_{\square}5">::
```

```
(fun () \rightarrow count\_permutations 5);
        "count_permute_6" >::
           (fun () \rightarrow count\_permutations 6);
        "count_permute_7" >::
           (fun () \rightarrow count\_permutations 7);
        "count_permute_8" > ::
           (fun () \rightarrow count\_permutations 8);
        "cyclic<sub>□</sub>[]" >::
           (fun () \rightarrow
              assert_equal_perms [] (permute_cyclic []));
        "cyclic<sub>□</sub>[1]" >::
           (fun () \rightarrow
              assert\_equal\_perms [[1]] (permute\_cyclic [1]));
        "cyclic<sub>□</sub>[1;2;3]" >::
           (fun () \rightarrow
              assert\_equal\_perms
                [[1;2;3]; [2;3;1]; [3;1;2]]
                 (permute\_cyclic\ [1;2;3]));
        "cyclic<sub>□</sub>[1;2;3;4]" >::
           (fun () \rightarrow
              assert\_equal\_perms
                [[1;2;3;4]; [2;3;4;1]; [3;4;1;2]; [4;1;2;3]]
                 (permute\_cyclic\ [1;2;3;4]));
        "cyclic_{\sqcup}[1;2;3]_{\sqcup}signed" >::
           (fun () \rightarrow
              assert\_equal
                [(1,[1;2;3]); (1,[2;3;1]); (1,[3;1;2])]
                 (permute\_cyclic\_signed [1; 2; 3]));
        "cyclic_{\sqcup}[1;2;3;4]_{\sqcup}signed" >::
           (\mathsf{fun}\ ()\ \to
              assert\_equal
                [(1, [1; 2; 3; 4]); (-1, [2; 3; 4; 1]); (1, [3; 4; 1; 2]); (-1, [4; 1; 2; 3])]
                 (permute\_cyclic\_signed [1; 2; 3; 4]))
let \ sort\_signed\_not\_unique \ =
   "not_unique" >::
      (fun () \rightarrow
        assert\_raises
           (Invalid\_argument
               "Combinatorics.insert_inorder_signed: identical elements")
           (fun () \rightarrow sort\_signed [1; 2; 3; 4; 2]))
let sort\_signed\_even =
   "even" >::
      (fun () \rightarrow
        assert\_equal(1, [1; 2; 3; 4; 5; 6])
           (sort\_signed [1; 2; 4; 3; 6; 5]))
let sort\_signed\_odd =
   "odd" >::
     (fun () \rightarrow
        assert\_equal(-1, [1; 2; 3; 4; 5; 6])
           (sort\_signed [2; 3; 1; 5; 4; 6]))
let sort\_signed\_all =
   "all" >::
   (fun () \rightarrow
     let l = ThoList.range 1 8 in
      assert\_bool "all\_signed\_permutations"
        (List.for\_all
            (\mathsf{fun}\ (eps,\ p)\ \to\ 
               \mathsf{let}\ eps',\ p'\ =\ sort\_signed\ p\ \mathsf{in}
               eps' = eps \land p' = l
```

```
(permute\_signed \ l)))
     let sign\_sign2 =
         "sign/sign2" >::
         (fun () \rightarrow
           let l = ThoList.range 1 8 in
              assert\_bool "all_permutations"
              (List.for\_all
                   (\mathsf{fun}\ p\ \to\ sign\ p\ =\ sign2\ p)
                   (permute \ l)))
     let suite\_sort\_signed =
         "sort_signed" >:::
           [sort\_signed\_not\_unique;
             sort_signed_even;
             sort\_signed\_odd;
             sort\_signed\_all;
             sign\_sign2
     let suite =
         "Combinatorics" >:::
           [suite\_permute;
             suite\_sort\_signed
  end
                                           N.3
                                                     Interface of Permutation
module type T =
  sig
     type t
The argument list [p_1; \ldots; p_n] must contain every integer from 0 to n-1 exactly once.
     val of\_list : int\ list \rightarrow t
     val\ of\_array:\ int\ array 
ightarrow t
list\ (of\_lists\ l\ l')\ l\ =\ l'
     val of_lists : \alpha list \rightarrow \alpha list \rightarrow t
     \mathsf{val}\ inverse\ :\ t\ \to\ t
     \mathsf{val}\ compose\ :\ t\ \to\ t\ \to\ t
compose\_inv \ p \ q = compose \ p \ (inverse \ q), but more efficient.
     \mathsf{val}\ compose\_inv\ :\ t\ \to\ t\ \to\ t
If p is of _list [p_1; \ldots; p_n], then list p [a_1; \ldots; a_n] reorders the list [a_1; \ldots; a_n] in the sequence given by [p_1; \ldots; p_n].
Thus the [p_1; \ldots; p_n] are not used as a map of the indices reshuffling an array. Instead they denote the new
positions of the elements of [a_1; \ldots; a_n]. However list (inverse p) [a_1; \ldots; a_n] is [a_{p_1}; \ldots; a_{p_n}], by duality.
     \mathsf{val}\ \mathit{list}:t\ \to\ \alpha\ \mathit{list}\to\ \alpha\ \mathit{list}
     val array : t \rightarrow \alpha array \rightarrow \alpha array
     \mathsf{val}\ \mathit{all}\ :\ \mathit{int}\ \rightarrow\ t\ \mathit{list}
     val\ even\ :\ int 
ightarrow\ t\ list
     val \ odd : int \rightarrow t \ list
     val\ cyclic\ :\ int 
ightarrow\ t\ list
     val\ signed:\ int \rightarrow\ (int \times\ t)\ list
Assuming fewer than 10 elements!
     val to\_string : t \rightarrow string
  end
module Using\_Lists : T
module Using\_Arrays : T
```

```
\begin{array}{lll} \text{module } Default \ : \ T \\ \\ \text{module } Test \ : \ \text{functor } (P \ : \ T) \ \rightarrow \\ \\ \text{sig val } suite \ : \ OUnit.test \ \text{val } time \ : \ unit \rightarrow \ unit \ \text{end} \end{array}
```

N.4 Implementation of Permutation

```
module type T =
  sig
     type t
     val of\_list : int\ list \rightarrow\ t
     val\ of\_array : int\ array \rightarrow t
     \mathsf{val}\ of\_lists\ :\ \alpha\ list\ \rightarrow\ \alpha\ list\ \rightarrow\ t
     val\ inverse\ :\ t\ 	o\ t
     val compose: t \rightarrow t \rightarrow t
     \mathsf{val}\ compose\_inv\ :\ t\ \to\ t\ \to\ t
     \mathsf{val}\ \mathit{list}: t\ \to\ \alpha\ \mathit{list} \to\ \alpha\ \mathit{list}
     val array : t \rightarrow \alpha array \rightarrow \alpha array
     val \ all : int \rightarrow t \ list
     val\ even\ :\ int 
ightarrow\ t\ list
     \mathsf{val}\ odd\ :\ int\rightarrow\ t\ list
     \mathsf{val}\ cyclic\ :\ int \to\ t\ list
     val\ signed:\ int \rightarrow\ (int \times t)\ list
     val to\_string : t \rightarrow string
  end
let same\_elements\ l1\ l2\ =
   List.sort\ compare\ l1\ =\ List.sort\ compare\ l2
module PM = Pmap.Tree
let offset\_map l =
  \mathsf{let}\ \_,\ \mathit{offsets}\ =
     List.fold\_left
        (fun (i, map) a \rightarrow (succ i, PM.add compare a i map))
         (0, PM.empty) l in
   offsets
TODO: this algorithm fails if the lists contain duplicate elements.
let of\_lists\_list\ l\ l' =
  if same\_elements\ l\ l' then
     let offsets' = offset\_map \ l' in
     let \_, p\_rev =
        List.fold\_left
            (fun (i, acc) a \rightarrow (succ i, PM.find compare a offsets' :: acc))
            (0, []) l in
      List.rev p_rev
      invalid\_arg "Permutation.of_lists:\sqcupincompatible\sqcuplists"
module Using\_Lists : T =
  struct
     type t = int list
     let of\_list p =
        if List.sort compare p \neq (ThoList.range\ 0\ (List.length\ p\ -\ 1)) then
           invalid_arg "Permutation.of_list"
        else
           p
     let of\_array p =
        try
            of\_list (Array.to\_list p)
```

```
with
       | Invalid\_argument s \rightarrow
           if s = "Permutation.of_list" then
             invalid_arg "Permutation.of_array"
           else
             failwith ("Permutation.of_array:\(\_\)unexpected\(\_\)Invalid_argument("\)
                           s ^ ")")
    let of\_lists = of\_lists\_list
    let inverse \ p = snd \ (ThoList.ariadne\_sort \ p)
    let list p l =
       List.map snd
         (List.sort (fun (i, \_) (j, \_) \rightarrow compare i j)
             (try
                 List.rev\_map2 (fun i \ x \rightarrow (i, \ x)) p \ l
              with
              | Invalid\_argument s \rightarrow
                  if s = "List.rev_map2" then
                    invalid\_arg "Permutation.list:_\length\ln mismatch"
                    failwith ("Permutation.list: unexpected Invalid_argument(" ^
                                   s ^ ")")))
    let array p a =
       try
         Array.of\_list\ (list\ p\ (Array.to\_list\ a))
       with
       | Invalid\_argument s \rightarrow
           if s = "Permutation.list: length mismatch" then
             invalid\_arg "Permutation.array:\_length\_mismatch"
             failwith ("Permutation.array: unexpected_Invalid_argument(" ^ s ^ ")")
    \mathsf{let}\ compose\_inv\ p\ q\ =
       list\ q\ p
Probably not optimal (or really inefficient), but correct by associativity.
    let compose p q =
       list (inverse q) p
    let all n =
       List.map of_list (Combinatorics.permute (ThoList.range 0 (pred n)))
    let even n =
       List.map\ of\_list\ (Combinatorics.permute\_even\ (ThoList.range\ 0\ (pred\ n)))
    let odd n =
       List.map\ of\_list\ (Combinatorics.permute\_odd\ (ThoList.range\ 0\ (pred\ n)))
       List.map\ of\_list\ (Combinatorics.permute\_cyclic\ (ThoList.range\ 0\ (pred\ n)))
    let signed n =
       List.map
         (fun (eps, l) \rightarrow (eps, of\_list l))
         (Combinatorics.permute\_signed\ (ThoList.range\ 0\ (pred\ n)))
    let to\_string p =
       String.concat "" (List.map string\_of\_int p)
  end
module Using\_Arrays : T =
  struct
    \mathsf{type}\ t\ =\ int\ array
```

```
let of_list p =
  if List.sort compare p \neq (ThoList.range\ 0\ (List.length\ p\ -\ 1)) then
     invalid_arg "Permutation.of_list"
  else
     Array.of\_list p
let of\_array p =
  try
     of\_list (Array.to\_list p)
  with
  | Invalid\_argument s \rightarrow
      if s = "Permutation.of_list" then
        invalid_arg "Permutation.of_array"
        failwith ("Permutation.of_array:_unexpected_Invalid_argument(" ^
let of_lists \ l \ l' =
  Array.of\_list (of\_lists\_list \ l \ l')
let inverse p =
  let len_p = Array.length p in
  let p' = Array.make len_p p.(0) in
  for i = 0 to pred len_p do
     p'.(p.(i)) \leftarrow i
  done;
  p'
\mathsf{let}\ \mathit{array}\ p\ a\ =
  let len_a = Array.length a
  and len_p = Array.length p in
  if len_{-}a \neq len_{-}p then
     invalid_arg "Permutation.array: □length mismatch";
  let a' = Array.make len_a a.(0) in
  for i = 0 to pred len_a do
     a'.(p.(i)) \leftarrow a.(i)
  done;
  a'
let list p l =
  try
     Array.to\_list (array p (Array.of\_list l))
  with
  | Invalid\_argument s \rightarrow
      if s = "Permutation.array:\_length\_mismatch" then
        invalid\_arg "Permutation.list:\_length\_mismatch"
        failwith ("Permutation.list: unexpected_Invalid_argument(" ^ s ^ ")")
let compose\_inv p q =
  array q p
let compose p q =
  array (inverse q) p
\mathsf{let} \ \mathit{all} \ \mathit{n} \ = \\
  List.map\ of\_list\ (Combinatorics.permute\ (ThoList.range\ 0\ (pred\ n)))
let even n =
  List.map\ of\_list\ (Combinatorics.permute\_even\ (ThoList.range\ 0\ (pred\ n)))
  List.map\ of\_list\ (Combinatorics.permute\_odd\ (ThoList.range\ 0\ (pred\ n)))
let cyclic n =
  List.map\ of\_list\ (Combinatorics.permute\_cyclic\ (ThoList.range\ 0\ (pred\ n)))
```

```
let signed n =
                 List.map
                       (fun (eps, l) \rightarrow (eps, of\_list l))
                       (Combinatorics.permute\_signed\ (ThoList.range\ 0\ (pred\ n)))
           let to\_string p =
                 String.concat "" (List.map string_of_int (Array.to_list p))
     end
module Default = Using\_Arrays
This is the Fisher-Yates shuffle, cf. D. Knuth, Seminumerical algorithms. The Art of Computer Programming.
2. Reading, MA: Addison-Wesley. pp. 139-140.
location l
     let a = Array.of\_list l in
     for n = Array.length \ a - 1 downto 1 do
           let k = Random.int (succ n) in
           if k \neq n then
                 \mathsf{let}\ tmp\ =\ Array.get\ a\ n\ \mathsf{in}
                 Array.set \ a \ n \ (Array.get \ a \ k);
                 Array.set \ a \ k \ tmp
     done;
     Array.to\_list a
let time f x =
     let start = Sys.time() in
     let <math>f_-x = f x in
     \mathsf{let}\ stop\ =\ Sys.time\ ()\ \mathsf{in}
     (f_x, stop - start)
\mathsf{let}\ print\_time\ msg\ f\ x\ =
     let f_-x, seconds = time f x in
      Printf.printf "%s_took_\%10.2f_ms\n" msg (seconds * . 1000.);
     f_{-}x
let random_int_list imax n =
     let imax\_plus = succ imax in
      Array.to\_list (Array.init n (fun \_ \rightarrow Random.int imax\_plus))
module Test\ (P\ :\ T)\ : sig val suite\ :\ OUnit.test\ val\ time\ :\ unit\ 	o\ unit\ end\ =
     struct
           open OUnit
           open P
           let of list_overlap =
                 "overlap" >::
                       (fun () \rightarrow
                             assert_raises (Invalid_argument "Permutation.of_list")
                                  (fun () \rightarrow
                                        of_list [0; 1; 2; 2])
           let of\_list\_gap =
                 "gap" >::
                       (fun () \rightarrow
                            assert_raises (Invalid_argument "Permutation.of_list")
                                  (fun () \rightarrow
                                        of_list [0; 1; 2; 4; 5])
           let of_list_ok =
                 "ok" >::
                      (\mathsf{fun}\ ()\ \to
                            \mathsf{let}\ l\ =\ ThoList.range\ 0\ 10\ \mathsf{in}
                             assert\_equal\ (of\_list\ l)\ (of\_list\ l))
           let suite\_of\_list =
```

```
"of_list" >:::
     [of\_list\_overlap;
      of\_list\_gap;
      of\_list\_ok
let suite\_of\_lists =
   "of_lists" >:::
     [ "ok" >::
          (fun () \rightarrow
             for i = 1 to 10 do
               \mathsf{let}\ l\ =\ random\_int\_list\ 1000000\ 100\ \mathsf{in}
               let l' = shuffle l in
               assert\_equal
                   \tilde{printer}: (ThoList.to\_string\_string\_of\_int)
                  l' (list (of_lists l l') l)
             done)
let apply_invalid_lengths =
   "invalid/lengths" >::
     (\mathsf{fun}\ ()\ \to
        assert\_raises
          (Invalid_argument "Permutation.list:|length|mismatch")
          (fun () \rightarrow
             list\ (of\_list\ [0;1;2;3;4])\ [0;1;2;3]))
let apply_ok =
   "ok" >::
     (fun () \rightarrow
        assert\_equal [2; 0; 1; 3; 5; 4]
          (list\ (of\_list\ [1;2;0;3;5;4])\ [0;1;2;3;4;5]))
let suite\_apply =
   "apply" >:::
     [apply_invalid_lengths;
      apply\_ok
let inverse\_ok =
   "ok" >::
     (fun () \rightarrow
       let l = shuffle ( ThoList.range 0 1000) in
       let p = of\_list (shuffle l) in
       assert\_equal\ l\ (list\ (inverse\ p)\ (list\ p\ l)))
let suite\_inverse =
   "inverse" >:::
     [inverse\_ok]
let compose\_ok =
   "ok" >::
     (fun () \rightarrow
       let id = ThoList.range 0 1000 in
       let p = of\_list (shuffle id)
       and q = of\_list (shuffle id)
       and l = id in
       assert\_equal\ (list\ p\ (list\ q\ l))\ (list\ (compose\ p\ q)\ l))
let compose_inverse_ok =
  "inverse/ok" >::
     (fun () \rightarrow
       let id = ThoList.range \ 0 \ 1000 in
       let p = of\_list (shuffle id)
       and q = of\_list (shuffle id) in
        assert\_equal
          (compose\ (inverse\ p)\ (inverse\ q))
          (inverse\ (compose\ q\ p)))
```

```
let suite\_compose =
     "compose" >:::
        [compose\_ok;
         compose\_inverse\_ok]
  \mathsf{let} \ \mathit{suite} \ =
      "Permutations" >:::
        [suite\_of\_list;
         suite\_of\_lists;
         suite\_apply;
         suite\_inverse;
         suite\_compose]
  {\sf let} \ repeat \ repetitions \ size \ = \\
     \mathsf{let}\ id\ =\ ThoList.range\ 0\ size\ \mathsf{in}
     \mathsf{let}\ p\ =\ of\_list\ (\mathit{shuffle}\ id)
     and l = shuffle (List.map string\_of\_int id) in
     print\_time\ (Printf.sprintf\ "reps=%d, \_len=%d"\ repetitions\ size)
        (fun () \rightarrow
          for i = 1 to repetitions do
             ignore (P.list p l)
        ()
  let time() =
     repeat 100000 10;
     repeat 10000 100;
     repeat\ 1000\ 1000;
     repeat\ 100\ 10000;
     repeat 10 100000;
     ()
end
```

—O—

PARTITIONS

O.1 Interface of Partition

pairs n n1 n2 returns all (unordered) pairs of integers with the sum n in the range from n1 to n2.

```
\begin{array}{lll} \mathsf{val} \ pairs \ : \ int \to \ int \to \ int \to \ (int \times int) \ list \\ \mathsf{val} \ triples \ : \ int \to \ int \to \ int \to \ (int \times int \times int) \ list \end{array}
```

tuples d n n-min n-max returns all $[n_1; n_2; \ldots; n_d]$ with $n_{\min} \leq n_1 \leq n_2 \leq \ldots \leq n_d \leq n_{\max}$ and

$$\sum_{i=1}^{d} n_i = n \tag{O.1}$$

 $\mathsf{val}\ tuples\ :\ int\rightarrow\ int\rightarrow\ int\rightarrow\ int\ \rightarrow\ int\ list\ list$

O.2 Implementation of Partition

All unordered pairs of integers with the same sum n in a given range $\{n_1, \ldots, n_2\}$:

$$pairs: (n, n_1, n_2) \to \{(i, j) \mid i + j = n \land n_1 \le i \le j \le n_2\}$$
(O.2)

```
let rec pairs' acc n1 n2 =
  if n1 > n2 then
     List.rev acc
  else
     pairs' ((n1, n2) :: acc) (succ n1) (pred n2)
let pairs sum min_n1 max_n2 =
  let n1 = max min_n1 (sum - max_n2) in
  let n2 = sum - n1 in
  if n2 \leq max_n2 then
     pairs' [] n1 n2
  else
let rec tuples \ d \ sum \ n\_min \ n\_max =
  if d \leq 0 then
     invalid\_arg "tuples"
   \  \, {\rm else} \,\, {\rm if} \,\, d \,\, > \,\, 1 \,\, {\rm then} \,\,
     tuples' \ d \ sum \ n\_min \ n\_max \ n\_min
  else if sum \geq n\_min \wedge sum \leq n\_max then
     [[sum]]
  else
and tuples' \ d \ sum \ n\_min \ n\_max \ n =
  if n > n_{-}max then
  else
     List.fold\_right (fun \ l \ ll \rightarrow (n :: l) :: ll)
```

```
(tuples (pred d) (sum - n) (max n_min n) n_max)
(tuples' \ d \ sum \ n\_min \ n\_max \ (succ \ n))
```



When I find a little spare time, I can provide a dedicated implementation, but we *know* that *Impossible* is never raised and the present approach is just as good (except for a real little in the control of the cont never raised and the present approach is just as good (except for a possible tiny inefficiency).

```
exception Impossible of string
let impossible name = raise (Impossible name)
let triples \ sum \ n\_min \ n\_max =
  List.map (function [n1; n2; n3] \rightarrow (n1, n2, n3) \mid \_ \rightarrow impossible "triples")
     (tuples \ 3 \ sum \ n\_min \ n\_max)
```

—P—

Young Diagrams and Tableaux

P.1 Interface of Young

Caveat: the following are not optimized for large Young diagrams and tableaux. They are straightforward implementations of the definitions, since we are unlikely to meet large diagrams.

To make matters worse, native integer arithmetic will overflow already for diagrams with more than 20 cells. Since the Num library has been removed from the O'Caml distribution with version 4.06, we can not use it as a shortcut. Requiring Whizard/O'Mega users to install Num or its successor Zarith is probably not worth the effort.

P.1.1 Young Diagrams

Young diagrams can be represented by a non-increasing list of positive integers, corresponding to the number of boxes in each row:

$$\iff [5;4;4;2] \tag{P.1}$$

 $\mathsf{type}\ \mathit{diagram}\ =\ \mathit{int}\ \mathit{list}$

Check that the diagram is valid, i. e. the number of boxes is non-increasing from top to bottom.

 $\verb|val| valid_diagram : diagram \rightarrow bool|$

Count the number of cells.

 $val num_cells_diagram : diagram \rightarrow int$

Conjugate a diagram:



 $\verb|val|| conjugate_diagram| : diagram| \to diagram|$

The product of all the "hook lengths" in the diagram, e.g.

$$\mapsto \frac{\begin{vmatrix} 8 & 7 & 5 & 4 & 1 \\ 6 & 5 & 3 & 2 \\ \hline 5 & 4 & 2 & 1 \end{vmatrix}}{\begin{vmatrix} 5 & 4 & 2 & 1 \\ 2 & 1 \end{vmatrix}} \mapsto 8 \cdot 7 \cdot 6 \cdot 5^3 \cdot 4^2 \cdot 3 \cdot 2^3 = 16128000$$
 (P.3)

where the intermediate step is only for illustration and does not represent a Young tableau!

 $val\ hook_lengths_product: diagram \rightarrow int$

Number of standard tableaux corresponding to the diagram. Also, the dimension of the representation of S_n described by this diagram

$$d = \frac{n!}{\prod_{i=1}^{n} h_i} \tag{P.4}$$

with n the number of cells and h_i the hook length of the ith cell.

 $val\ num_standard_tableaux : diagram \rightarrow int$

Normalization of the projector on the representation of GL(N) described by the diagram

$$\alpha = \frac{\prod_{R} |R|! \prod_{C} |C|!}{\prod_{i=1}^{n} h_i} \tag{P.5}$$

with |R| and |C| the lengths of the row R and column C, respectively. Returned as a pair of numerator and denominator, because it is not guaranteed to be integer.

 $\mathsf{val}\ normalization\ :\ diagram\ \to\ int\ \times\ int$

P.1.2 Young Tableaux

There is an obvious representation as a list of lists:

$$\begin{array}{c|c}
\hline
0 & 2 & 3 \\
\hline
1 & 4
\end{array}
\iff [[0; 2; 3]; [1; 4]]$$
(P.6)

type α tableau = α list list

Ignoring the contents of the cells of a Young tableau produces a unique corresponding Young diagram.

$$\begin{array}{c|c}
0 & 2 & 3 \\
\hline
1 & 4 & 1
\end{array}$$
(P.7)

 $val\ diagram_of_tableau\ :\ \alpha\ tableau\ o\ diagram$

The number of columns must be non-increasing. Obviously, $valid_tableau$ is the composition of $diagram_of_tableau$ and $valid_diagram$.

 $val\ valid_tableau\ :\ \alpha\ tableau\ o\ bool$

A tableau is called *semistandard*, iff the entries don't increase along rows and strictly increase along columns. Therefore, the conjugate of a semistandard tableau is *not* necessarily semistandard.

 $\verb|val|| semistandard_tableau| : \alpha |tableau| \rightarrow |bool|$

A tableau is called *standard*, iff it is semistandard and the entries are an uninterrupted sequence of natural numbers. If the optional *offset* is specified, it must match the smallest of these numbers. Some authors expect offset = 1, but we want to be able to start from 0 as well. The conjugate of a standard tableau is again a standard tableau.

 $\verb|val|| standard_tableau : ?offset : int \rightarrow int \ tableau \rightarrow bool$

The contents of the cells and their number.

val $cells_tableau: \alpha \ tableau \rightarrow \alpha \ list$ val $num_cells_tableau: \alpha \ tableau \rightarrow int$

Conjugate a Young tableau

$$\begin{array}{c|c}
\hline
0 & 2 & 3 \\
\hline
1 & 4 & 3
\end{array}
\mapsto
\begin{array}{c|c}
\hline
0 & 1 \\
\hline
2 & 4 \\
\hline
3
\end{array}$$
(P.8)

 $val\ conjugate_tableau\ :\ \alpha\ tableau\ o\ \alpha\ tableau$

P.1.3 Unit Tests

 $\begin{array}{ll} \text{module type } Test &= \\ \text{sig} \\ \text{val } suite \ : \ OUnit.test \\ \text{val } suite_long \ : \ OUnit.test \\ \text{end} \end{array}$

module Test: Test

P.2 Implementation of Young

Avoid referring to *Pervasives.compare*, because *Pervasives* will become *Stdlib.Pervasives* in O'Caml 4.07 and *Stdlib* in O'Caml 4.08.

```
let pcompare = compare
type diagram = int \ list
type \alpha tableau = \alpha list list
Not exposed. Just for documentation.
\mathsf{type} \,\, \alpha \,\, table \,\, = \,\, \alpha \,\, option \,\, array \,\, array
The following three are candidates for ThoList.
let rec sum = function
  | [] \rightarrow 0
  \mid n :: rest \rightarrow n + sum \ rest
let rec product = function
   | [] \rightarrow 1
  n :: rest \rightarrow n \times product \ rest
Test a predicate for each pair of consecutive elements of a list. Trivially true for empty and one-element lists.
let rec for\_all\_pairs\ predicate = function
  | [] | [\_] \rightarrow \mathsf{true}
  | a1 :: (a2 :: \_ as a\_list) \rightarrow
      if \neg (predicate \ a1 \ a2) then
         false
      else
         for_all_pairs predicate a_list
let decreasing l = for\_all\_pairs (fun a1 a2 \rightarrow pcompare a1 a2 > 0) l
let increasing \ l = for\_all\_pairs (fun a1 a2 \rightarrow pcompare a1 a2 < 0) l
let non\_increasing \ l = for\_all\_pairs (fun a1 a2 \rightarrow pcompare a1 a2 \geq 0) l
let non\_decreasing \ l = for\_all\_pairs (fun a1 a2 \rightarrow pcompare a1 a2 \leq 0) l
let \ valid\_diagram = non\_increasing
let diagram\_rows d =
   List.length d
let diagram\_columns = function
    [] \rightarrow 0
   | nc :: \_ \rightarrow nc
let take\_column d =
  let rec take\_column' len acc = function
     [] \rightarrow (len, List.rev acc)
     | cols :: rest \rightarrow
         if cols < 1 then
            take_column' (succ len) acc rest
            take_column' (succ len) (pred cols :: acc) rest in
   take\_column' \ 0 \ [] \ d
let conjugate\_diagram\_new d =
  let rec conjugate\_diagram' rows =
     match take_column rows with
     \mid n, \mid \rightarrow \mid n \mid
     \mid n, rest \rightarrow n :: conjugate\_diagram' rest in
   conjugate_diagram' d
let tableau\_rows t =
   List.length t
\mathsf{let}\ tableau\_columns\ =\ \mathsf{function}
  | [] \rightarrow 0
```

```
| row :: \_ \rightarrow List.length row
let num\_cells\_diagram d =
  sum d
let cells\_tableau t =
  List.flatten t
let num\_cells\_tableau t =
  List.fold\_left (fun acc\ row\ 	o \ acc\ +\ List.length\ row) 0 t
let diagram\_of\_tableau t =
  List.map List.length t
let tableau\_of\_diagram cell d =
  List.map (ThoList.clone cell) d
Note that the first index counts the rows and the second the columns!
let array_of_tableau t =
  let nr = tableau\_rows t
  and nc = tableau\_columns t in
  let a = Array.make\_matrix nr nc None in
  List.iteri
     (\text{fun } ir \rightarrow List.iteri (\text{fun } ic \ cell \rightarrow a.(ir).(ic) \leftarrow Some \ cell))
     t;
  a
let transpose\_array a =
  let nr = Array.length a in
  if nr \leq 0 then
     invalid_arg "Young.transpose_array"
    let nc = Array.length \ a.(0) in
    let a' = Array.make\_matrix nc nr None in
    for ic = 0 to pred nc do
       for ir = 0 to pred nr do
         a'.(ic).(ir) \ \leftarrow \ a.(ir).(ic)
       done
    done;
     a'
let list\_of\_array\_row a =
  let n = Array.length a in
  let rec list\_of\_array\_row' ic =
    if ic \geq n then
       else
       match a.(ic) with
         None \rightarrow []
        | Some cell \rightarrow cell :: list\_of\_array\_row' (succ ic) in
  list\_of\_array\_row' 0
let tableau\_of\_array \ a =
  Array.fold\_right (fun row acc \rightarrow list\_of\_array\_row row :: acc) a []
let conjugate\_tableau t =
  array\_of\_tableau\ t\ | > transpose\_array\ | > tableau\_of\_array
let conjugate\_diagram d =
  tableau\_of\_diagram () d \mid > conjugate\_tableau \mid > diagram\_of\_tableau
let \ valid\_tableau \ t =
  valid\_diagram\ (diagram\_of\_tableau\ t)
let semistandard\_tableau t =
  let rows = t
  and columns = conjugate\_tableau \ t in
```

```
valid\_tableau t
   \land List.for\_all\ non\_decreasing\ rows
   \land List.for_all increasing columns
let standard\_tableau ?offset t =
   match List.sort pcompare (cells_tableau t) with
     [] \rightarrow \mathsf{true}
   \mid cell :: \_ as cell\_list 
ightarrow
       (\mathsf{match}\ \mathit{offset}\ \mathsf{with}\ \mathit{None}\ \rightarrow\ \mathsf{true}\ |\ \mathit{Some}\ o\ \rightarrow\ \mathit{cell}\ =\ o)
       \land for\_all\_pairs (fun c1 c2 \rightarrow c2 = c1 + 1) cell\_list
       \land semistandard_tableau t
let\ hook\_lengths\_table\ d\ =
  let nr = diagram\_rows d
   and nc = diagram\_columns d in
  if min \ nr \ nc \ \le \ 0 then
      invalid_arg "Young.hook_lengths_table"
  else
     let a = array\_of\_tableau (tableau\_of\_diagram 0 d) in
     \mathsf{let}\ cols\ =\ Array.of\_list\ d
     and rows = transpose\_array \ a \mid > tableau\_of\_array
                      -> diagram_of_tableau | > Array.of_list in
     for ir = 0 to pred nr do
        \  \, \hbox{for}\,\, ic\,\,=\,\,0\,\,\hbox{to}\,\,pred\,\,cols.(ir)\,\,\hbox{do}
           a.(ir).(ic) \leftarrow Some (rows.(ic) - ir + cols.(ir) - ic - 1)
     done;
```



The following products and factorials can easily overflow, even if the final ratio is a smallish number. We can avoid this by representing them as lists of factors (or maps from factors to powers). The ratio can be computed by first cancelling all common factors and multiplying the remaining factors at the very end.

```
let hook\_lengths\_product d =
  let nr = diagram\_rows d
  and nc = diagram\_columns d in
  if min \ nr \ nc \ \leq \ 0 then
    0
  else
    let cols = Array.of\_list d
    and rows = Array.of\_list (conjugate\_diagram d) in
    let n = ref 1 in
    for ir = 0 to pred nr do
      for ic = 0 to pred\ cols.(ir) do
         n := !n \times (rows.(ic) - ir + cols.(ir) - ic - 1)
      done
    done;
    !n
let num\_standard\_tableaux d =
  let num = Combinatorics.factorial (num_cells_diagram d)
  and den = hook\_lengths\_product d in
  if num \mod den \neq 0 then
    failwith "Young.num_standard_tableaux"
  else
    num / den
```

Note that $hook_lengths_product$ calls $conjugate_diagram$ and this calls it again. This is wasteful, but probably no big deal for our applications.

```
\begin{array}{lll} \text{let } normalization \ d &= \\ \text{let } num &= \\ product \ (List.map \ Combinatorics.factorial \ (d @ conjugate\_diagram \ d)) \end{array}
```

```
and den = hook\_lengths\_product d in
  (num, den)
module type Test =
  sig
    val\ suite\ :\ OUnit.test
    val\ suite\_long\ :\ OUnit.test
  end
module \ Test =
  struct
    open OUnit
    let random_int ratio =
       truncate (Random.float \ ratio + .0.5)
    let random\_diagram\ ?(ratio = 1.0)\ rows =
       let \ rec \ random\_diagram' \ acc \ row \ cols \ =
         if row \geq rows then
            acc
         else
            let cols' = cols + random\_int ratio in
            random_diagram' (cols' :: acc) (succ row) cols' in
       random_diagram' [] 0 (1 + random_int ratio)
    let \ suite\_hook\_lengths\_product \ =
       "hook_lengths_product" >:::
         ["[4;3;2]">::
              (fun () \rightarrow assert\_equal \ 2160 \ (hook\_lengths\_product \ [4; \ 3; \ 2])) ]
    let suite\_num\_standard\_tableaux =
       "num_standard_tableaux" >:::
         ["[4;3;2]">::
              (fun () \rightarrow assert\_equal 168 (num\_standard\_tableaux [4; 3; 2]))]
    let suite\_normalization =
       "normalization" >:::
         ["[2;1]">::
              (fun () \rightarrow assert\_equal (4, 3) (normalization [2; 1])) ]
    let suite =
       "Young" >:::
         [suite\_hook\_lengths\_product;
          suite\_num\_standard\_tableaux;
          suite\_normalization]
    let suite\_long =
       "Young_long" >:::
         end
```

—Q—

From [10]: Trees with one root admit a straightforward recursive definition

$$T(N,L) = L \cup N \times T(N,L) \times T(N,L) \tag{Q.1}$$

that is very well adapted to mathematical reasoning. Such recursive definitions are useful because they allow us to prove properties of elements by induction

$$\forall l \in L : p(l) \land (\forall n \in N : \forall t_1, t_2 \in T(N, L) : p(t_1) \land p(t_2) \Rightarrow p(n \times t_1 \times t_2))$$

$$\Longrightarrow \forall t \in T(N, L) : p(t) \quad (Q.2)$$

i.e. establishing a property for all leaves and showing that a node automatically satisfies the property if it is true for all children proves the property for *all* trees. This induction is of course modelled after standard mathematical induction

$$p(1) \land (\forall n \in \mathbf{N} : p(n) \Rightarrow p(n+1)) \Longrightarrow \forall n \in \mathbf{N} : p(n)$$
 (Q.3)

The recursive definition (Q.1) is mirrored by the two tree construction functions¹

$$leaf: \nu \times \lambda \to (\nu, \lambda)T$$
 (Q.4a)

$$node: \nu \times (\nu, \lambda)T \times (\nu, \lambda)T \to (\nu, \lambda)T$$
 (Q.4b)

Renaming leaves and nodes leaves the structure of the tree invariant. Therefore, morphisms $L \to L'$ and $N \to N'$ of the sets of leaves and nodes induce natural homomorphisms $T(N,L) \to T(N',L')$ of trees

$$map: (\nu \to \nu') \times (\lambda \to \lambda') \times (\nu, \lambda)T \to (\nu', \lambda')T \tag{Q.5}$$

The homomorphisms constructed by map are trivial, but ubiquitous. More interesting are the morphisms

$$fold: (\nu \times \lambda \to \alpha) \times (\nu \times \alpha \times \alpha \to \alpha) \times (\nu, \lambda) T \to \alpha$$

$$(f_1, f_2, l \in L) \mapsto f_1(l)$$

$$(f_1, f_2, (n, t_1, t_2)) \mapsto f_2(n, fold(f_1, f_2, t_1), fold(f_1, f_2, t_2))$$
(Q.6)

and

$$fan: (\nu \times \lambda \to \{\alpha\}) \times (\nu \times \alpha \times \alpha \to \{\alpha\}) \times (\nu, \lambda)T \to \{\alpha\}$$

$$(f_1, f_2, l \in L) \mapsto f_1(l)$$

$$(f_1, f_2, (n, t_1, t_2)) \mapsto f_2(n, fold(f_1, f_2, t_1) \otimes fold(f_1, f_2, t_2))$$

$$(Q.7)$$

where the tensor product notation means that f_2 is applied to all combinations of list members in the argument:

$$\phi(\{x\} \otimes \{y\}) = \{\phi(x,y) | x \in \{x\} \land y \in \{y\}\}$$
 (Q.8)

But note that due to the recursive nature of trees, fan is not a morphism from T(N, L) to $T(N \otimes N, L)$.

If we identify singleton sets with their members, fold could be viewed as a special case of fan, but that is probably more confusing than helpful. Also, using the special case $\alpha = (\nu', \lambda')T$, the homomorphism map can be expressed in terms of fold and the constructors

$$map: (\nu \to \nu') \times (\lambda \to \lambda') \times (\nu, \lambda)T \to (\nu', \lambda')T$$

$$(f, g, t) \mapsto fold(leaf \circ (f \times g), node \circ (f \times id \times id), t)$$
(Q.9)

¹To make the introduction more accessible to non-experts, I avoid the 'curried' notation for functions with multiple arguments and use tuples instead. The actual implementation takes advantage of curried functions, however. Experts can read $\alpha \to \beta \to \gamma$ for $\alpha \times \beta \to \gamma$.

fold is much more versatile than map, because it can be used with constructors for other tree representations to translate among different representations. The target type can also be a mathematical expression. This is used extensively below for evaluating Feynman diagrams.

Using fan with $\alpha = (\nu', \lambda')T$ can be used to construct a multitude of homomorphic trees. In fact, below it will be used extensively to construct all Feynman diagrams $\{(\nu, \{p_1, \dots, p_n\})T\}$ of a given topology $t \in (\emptyset, \{1, \dots, n\})T$.



The physicist in me guesses that there is another morphism of trees that is related to fan like a Lie-algebra is related to the it's Lie-group. I have not been able to pin it down, but I guess that it is a generalization of grow below.

Q.1 Interface of Tree

This module provides utilities for generic decorated trees, such as FeynMF output.

Q.1.1 Abstract Data Type

type $(\nu, \lambda) t$

 $leaf\ n\ l$ returns a tree consisting of a single leaf node of type n with a label l.

 $\mathsf{val}\ \mathit{leaf}\ :\ \nu\ \to\ \lambda\ \to\ (\nu,\ \lambda)\ \mathit{t}$

 $cons \ n \ ch$ returns a tree node.

val $cons : \nu \rightarrow (\nu, \lambda) \ t \ list \rightarrow (\nu, \lambda) \ t$

Note that cons node [] constructs a terminal node, but not a leaf, since the latter must have a label!



This approach was probably tailored to Feynman diagrams, where we have external propagators as nodes with additional labels (cf. the function to_feynmf on page 700 below). I'm not so sure anymore that this was a good choice.

 $node \ t$ returns the top node of the tree t.

val $node: (\nu, \lambda) t \rightarrow \nu$

leafs t returns a list of all leaf labels in order.

val $leafs: (\nu, \lambda) t \rightarrow \lambda list$

nodes t returns a list of all nodes that are not leafs in post-order. This guarantees that the root node can be stripped from the result by List.tl.

val $nodes: (\nu, \lambda) t \rightarrow \nu list$

fuse conjg root contains_root trees joins the trees, using the leaf root in one of the trees as root of the new tree. contains_root guides the search for the subtree containing root as a leaf. fun $t \to List.mem$ root (leafs t) is acceptable, but more efficient solutions could be available in special circumstances.

$$\mathsf{val}\ \mathit{fuse}\ :\ (\nu\ \to\ \nu)\ \to\ \lambda\ \to\ ((\nu,\ \lambda)\ t\ \to\ \mathit{bool})\ \to\ (\nu,\ \lambda)\ t\ \mathit{list}\ \to\ (\nu,\ \lambda)\ t$$

 $sort\ lesseq\ t$ return a sorted copy of the tree t: node labels are ignored and nodes are according to the supremum of the leaf labels in the corresponding subtree.

val $sort: (\lambda \to \lambda \to bool) \to (\nu, \lambda) \ t \to (\nu, \lambda) \ t$ val $canonicalize: (\nu, \lambda) \ t \to (\nu, \lambda) \ t$

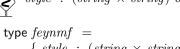
Q.1.2 Homomorphisms

 $\begin{array}{l} \text{val } map \ : \ ("n1 \ \rightarrow \ "n2) \ \rightarrow \ ("l1 \ \rightarrow \ "l2) \ \rightarrow \ ("n1, \ "l1) \ t \ \rightarrow \ ("n2, \ "l2) \ t \\ \text{val } fold \ : \ (\nu \ \rightarrow \ \lambda \ \rightarrow \ \alpha) \ \rightarrow \ (\nu \ \rightarrow \ \alpha \ list \ \rightarrow \ \alpha) \ \rightarrow \ (\nu, \ \lambda) \ t \ \rightarrow \ \alpha \\ \text{val } fan \ : \ (\nu \ \rightarrow \ \lambda \ \rightarrow \ \alpha \ list) \ \rightarrow \ (\nu \ \rightarrow \ \alpha \ list \ \rightarrow \ \alpha \ list) \ \rightarrow \\ (\nu, \ \lambda) \ t \ \rightarrow \ \alpha \ list \end{array}$

Q.1.3 Output

```
val\ to\_string : (string, string)\ t \rightarrow string
```

Feynmf



type $\lambda feynmf_set =$

style: (string × string) option should be replaced by style: string option; tex_label: string option

```
\{ style : (string \times string) \ option; \}
        rev : bool;
        label: string option;
        tension : float option }
val vanilla : feynmf
val\ sty\ :\ (string \times string)\ \times\ bool\ \times\ string\ 	o\ feynmf
```

to_feynmf file to_string incoming t write the trees in the list t to the file named file. The leaves incoming are used as incoming particles and to_string is use to convert leaf labels to LATEX-strings.

```
{ header : string;
      incoming : \lambda list;
      diagrams : (feynmf, \lambda) \ t \ list \}
type (\lambda, \mu) feynmf_sets =
   { outer : \lambda feynmf\_set;
      inner : \mu feynmf\_set list 
\verb|val| feynmf_sets_plain : bool \rightarrow int \rightarrow string \rightarrow |
   (\lambda \rightarrow string) \rightarrow (\lambda \rightarrow string) \rightarrow
   (\mu \rightarrow string) \rightarrow (\mu \rightarrow string) \rightarrow (\lambda, \mu) feynmf\_sets \ list \rightarrow unit
val\ feynmf\_sets\_wrapped : bool \rightarrow string \rightarrow
   (\lambda \rightarrow string) \rightarrow (\lambda \rightarrow string) \rightarrow
   (\mu \rightarrow string) \rightarrow (\mu \rightarrow string) \rightarrow (\lambda, \mu) feynmf\_sets \ list \rightarrow unit
If the diagrams at all levels are of the same type, we can recurse to arbitrary depth.
type \lambda feynmf_levels =
   { this : \lambda feynmf_set;
      lower: \lambda feynmf\_levels list 
to_feynmf_levels_plain sections level file wf_to_TeX p_to_TeX levels ...
val\ feynmf\_levels\_plain : bool \rightarrow int \rightarrow string \rightarrow
   (\lambda \rightarrow string) \rightarrow (\lambda \rightarrow string) \rightarrow \lambda feynmf\_levels \ list \rightarrow unit
to_feynmf_levels_wrapped file wf_to_TeX p_to_TeX levels ...
val feynmf\_levels\_wrapped: string \rightarrow
```

Least Squares Layout

A general graph with edges of type ε , internal nodes of type ν , and external nodes of type 'ext.

```
type (\varepsilon, \nu, 'ext) graph
val graph\_of\_tree : (\nu \rightarrow \nu \rightarrow \varepsilon) \rightarrow (\nu \rightarrow \nu) \rightarrow
    \nu \rightarrow (\nu, \nu) t \rightarrow (\varepsilon, \nu, \nu) graph
```

A general graph with the layout of the external nodes fixed.

```
type (\varepsilon, \nu, 'ext) \ ext\_layout
val left\_to\_right: int \rightarrow (\varepsilon, \nu, 'ext) \ graph \rightarrow (\varepsilon, \nu, 'ext) \ ext\_layout
```

 $(\lambda \rightarrow string) \rightarrow (\lambda \rightarrow string) \rightarrow \lambda feynmf_levels \ list \rightarrow unit$

A general graph with the layout of all nodes fixed.

```
\begin{array}{l} {\rm type} \ (\varepsilon, \ \nu, \ 'ext) \ layout \\ {\rm val} \ layout \ : \ (\varepsilon, \ \nu, \ 'ext) \ ext\_layout \ \rightarrow \ (\varepsilon, \ \nu, \ 'ext) \ layout \\ {\rm val} \ dump \ : \ (\varepsilon, \ \nu, \ 'ext) \ layout \ \rightarrow \ unit \\ {\rm val} \ iter\_edges \ : \ (\varepsilon \ \rightarrow \ float \times \ float \ \rightarrow \ float \times \ float \ \rightarrow \ unit) \ \rightarrow \\ (\varepsilon, \ \nu, \ 'ext) \ layout \ \rightarrow \ unit \\ {\rm val} \ iter\_incerning \ : \ ('ext \times \ float \times \ float \ \rightarrow \ unit) \ \rightarrow \\ (\varepsilon, \ \nu, \ 'ext) \ layout \ \rightarrow \ unit \\ {\rm val} \ iter\_outgoing \ : \ ('ext \times \ float \times \ float \ \rightarrow \ unit) \ \rightarrow \\ (\varepsilon, \ \nu, \ 'ext) \ layout \ \rightarrow \ unit \\ {\rm val} \ iter\_outgoing \ : \ ('ext \times \ float \times \ float \ \rightarrow \ unit) \ \rightarrow \\ (\varepsilon, \ \nu, \ 'ext) \ layout \ \rightarrow \ unit \\ \end{array}
```

Q.2 Implementation of Tree

Q.2.1 Abstract Data Type

```
\begin{array}{lll} \text{type } (\nu,\ \lambda)\ t &= \\ &|\ \textit{Leaf of } \nu\ \times\ \lambda \\ &|\ \textit{Node of } \nu\ \times\ (\nu,\ \lambda)\ t\ \textit{list} \end{array} \text{let } \textit{leaf } n\ l &=\ \textit{Leaf } (n,\ l) \text{let } \textit{cons } n\ \textit{children} &=\ \textit{Node } (n,\ \textit{children}) \end{array}
```

Presenting the leafs in order comes naturally, but will be useful below.

```
\begin{array}{lll} \text{let rec } leafs &=& \text{function} \\ | & Leaf \ (\_, \ l) \ \rightarrow \ [l] \\ | & Node \ (\_, \ ch) \ \rightarrow \ ThoList.flatmap \ leafs \ ch \\ \\ \text{let } node &=& \text{function} \\ | & Leaf \ (n, \ \_) \ \rightarrow \ n \\ | & Node \ (n, \ \_) \ \rightarrow \ n \end{array}
```

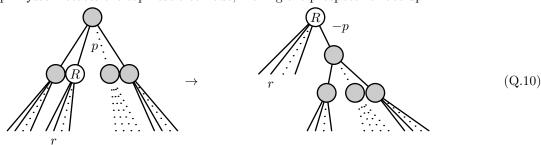
This guarantees that the root node can be stripped from the result by List.tl.

```
 \begin{array}{lll} \text{let rec } nodes &= \text{ function} \\ \mid \textit{Leaf } \bot &\to & [ \ ] \\ \mid \textit{Node } (n, \textit{ ch}) &\to & n \ :: \ \textit{ThoList.flatmap nodes ch} \end{array}
```

 $first_match\ p\ list\ returns\ (x, list'),$ where x is the first element of list for which $p\ x\ =\ true\ and\ list'$ is $list\ sans\ x$

```
\begin{array}{l} \text{let } \mathit{first\_match} \ p \ \mathit{list} = \\ \\ \text{let } \mathit{rec} \ \mathit{first\_match'} \ \mathit{no\_match} \ = \ \mathsf{function} \\ \\ \mid \ [] \ \rightarrow \ \mathit{invalid\_arg} \ "\mathsf{Tree.fuse}: \ _\mathsf{prospective\_root\_not\_found"} \\ \mid \ t \ :: \ \mathit{rest} \ \mathsf{when} \ p \ t \ \rightarrow \ (t, \ \mathit{List.rev\_append} \ \mathit{no\_match} \ \mathit{rest}) \\ \mid \ t \ :: \ \mathit{rest} \ \rightarrow \ \mathit{first\_match'} \ (t \ :: \ \mathit{no\_match}) \ \mathit{rest} \ \mathsf{in} \\ \\ \mathit{first\_match'} \ [] \ \mathit{list} \end{array}
```

One recursion step in fuse' rotates the topmost tree node, moving the prospective root up:



```
let fuse conjg root contains_root trees =
  let rec fuse' subtrees =
  match first_match contains_root subtrees with
```

If the prospective root is contained in a leaf, we have either found the root—in which case we're done—or have failed catastrophically:

```
 | \ Leaf \ (n, \ l), \ children \rightarrow \\  | \ if \ l = root \ then \\  | \ Node \ (conjg \ n, \ children) \\  | \ else \\  | \ invalid\_arg \ "Tree.fuse: \_root\_predicate\_inconsistent"
```

Otherwise, we perform a rotation as in (Q.10) and connect all nodes that do not contain the root to a new node. For efficiency, we append the new node at the end and prevent $first_match$ from searching for the root in it in vain again. Since $root_children$ is probably rather short, this should be a good strategy.

```
|\ Node\ (n,\ root\_children),\ other\_children\ \rightarrow fuse'\ (root\_children\ @\ [Node\ (conjg\ n,\ other\_children)])\ in\ fuse'\ trees
```

Sorting is also straightforward, we only have to keep track of the suprema of the subtrees:

```
type (\alpha, \beta) with_supremum = \{ sup : \alpha; data : \beta \}
```

Since the lists are rather short, List.sort could be replaced by an optimized version, but we're not (yet) dealing with the most important speed bottleneck here:

Q.2.2 Homomorphisms

Isomophisms are simple:

let rec map fn fl = function

```
 \mid Leaf\ (n,\ l) \rightarrow Leaf\ (fn\ n,\ fl\ l)  \mid Node\ (n,\ ch) \rightarrow Node\ (fn\ n,\ List.map\ (map\ fn\ fl)\ ch)  homomorphisms are not more complicated:  |\text{let rec } fold\ leaf\ node\ =\ \text{function}  \mid Leaf\ (n,\ l) \rightarrow leaf\ n\ l  \mid Node\ (n,\ ch) \rightarrow node\ n\ (List.map\ (fold\ leaf\ node)\ ch)  and tensor products are fun:  |\text{let rec } fan\ leaf\ node\ =\ \text{function}  \mid Leaf\ (n,\ l) \rightarrow leaf\ n\ l  \mid Node\ (n,\ ch) \rightarrow Product.fold    (\text{fun\ } ch'\ t \rightarrow node\ n\ ch'\ @\ t)\ (List.map\ (fan\ leaf\ node)\ ch)\ []
```

Q.2.3 Output

```
else if l = "" then
  else
     n ^ "(" ^ l ^ ")"
let node\_to\_string \ n \ ch =
  "(" \hat{} (if n = \hat{} "" then "" else n \hat{} ":") \hat{} (String.concat "," ch) \hat{} ")"
let to\_string t =
  fold leaf_to_string node_to_string t
                                                            Feynmf
Add a value that is greater than all suprema
type \alpha supremum_or_infinity = Infinity | Sup of \alpha
type (\alpha, \beta) with_supremum_or_infinity =
     \{ sup : \alpha supremum\_or\_infinity; data : \beta \}
let with\_infinity\ cmp\ x\ y\ =
  match x.sup, y.sup with
     Infinity, \_ \rightarrow 1
     \_, Infinity \rightarrow -1
  | Sup x', Sup y' \rightarrow cmp x' y'
```

let rec $sort_2i'$ lesseq i2 = function

Using this, we can sort the tree in another way that guarantees that a particular leaf (i2) is moved as far to the end as possible. We can then flip this leaf from outgoing to incoming without introducing a crossing:

```
\mid Leaf (-, l) as e \rightarrow
       \{ sup = if \ l = i2 \ then \ Infinity \ else \ Sup \ l; \ data = e \ \}
  | Node (n, ch) \rightarrow
       let \ \mathit{ch'} \ = \ \mathit{List.sort} \ (\mathit{with\_infinity} \ \mathit{compare})
            (List.map\ (sort\_2i'\ lesseq\ i2)\ ch)\ in
       \{ sup = (List.hd (List.rev ch')).sup; \}
         data = Node (n, List.map (fun x \rightarrow x.data) ch') 
again, throw away the overall supremum:
let sort_2i lesseq i2 t = (sort_2i' lesseq i2 t).data
type feynmf =
     \{ style : (string \times string) \ option; \}
       rev : bool;
       label: string option;
       tension : float option }
open Printf
let style prop =
  match prop.style with
    None \rightarrow ("plain","")
   Some \ s \rightarrow s
let species prop = fst (style prop)
let tex\_lbl prop = snd (style prop)
let leaf\_label tex io leaf lab = function
    \mid Some \ s \rightarrow
       fprintf\ tex\ "lullu \ \fmflabel{$s^{(s)}}$$\
let leaf_label tex io leaf lab label =
  ()
```

We try to draw diagrams more symmetrically by reducing the tension on the outgoing external lines.



This is insufficient for asymmetrical cascade decays.

```
let rec leaf_node tex to_label i2 n prop leaf =
    let io, tension, rev =
        if leaf = i2 then
            ("i", "", ¬ prop.rev)
        else
            ("o", ",tension=0.5", prop.rev) in
    leaf_label tex io (to_label leaf) (tex_lbl prop) prop.label;
    if rev then
        fprintf\ tex\ "$\sqcup \sqcup \sqcup \sqcup \sqcup \ \{\%s\%s\}\{\%s\%s,v\%d\}\n$
             (species\ prop)\ tension\ io\ (to\_label\ leaf)\ n
    else
        (species prop) tension n io (to_label leaf)
and int\_node\ tex\ to\_label\ i2\ n\ n'\ prop\ t\ =
    if prop.rev then
        fprintf tex
             (x,y)
             (species prop) (tex\_lbl prop) n' n
    else
        fprintf tex
             "_{\sqcup\sqcup\sqcup\sqcup} \left( s, label = \ scriptsize \right) \ (scriptsize) \ (scripts
             (species prop) (tex\_lbl prop) n n';
    edges_feynmf' tex to_label i2 n' t
and leaf\_or\_int\_node tex to_label i2 n n' = function
       Leaf (prop, l) \rightarrow leaf\_node tex to\_label i2 n prop l
     \mid Node (prop, \_) as t 	o int_node tex to_label i2 n n' prop t
and edges\_feynmf' tex to\_label i2 n = function
        Leaf\ (prop,\ l) \rightarrow leaf\_node\ tex\ to\_label\ i2\ n\ prop\ l
        Node (\_, ch) \rightarrow
             ignore (List.fold_right)
                                  (fun t' n' \rightarrow
                                      leaf_or_int_node tex to_label i2 n n' t';
                                      succ\ n')\ ch\ (4\times n))
let edges\_feynmf tex to_label i1 i2 t =
    \mathsf{let}\ n\ =\ 1\ \mathsf{in}
    begin match t with
        Leaf \ \_ \ \rightarrow \ ()
     Node\ (prop, \_) \rightarrow
            leaf_label tex "i" "1" (tex_lbl prop) prop.label;
            if prop.rev then
                 fprintf tex "LLLL\\fmf{%s}{v\d,i\%s}\n" (species prop) n (to_label i1)
                 fprintf\ tex\ "lullu \ \ fmf{%s}{i%s,v%d}\n"\ (species\ prop)\ (to\_label\ i1)\ n
    end;
    edges_feynmf' tex to_label i2 n t
let to_feynmf_channel tex to_TeX to_label incoming t =
    match incoming with
    | i1 :: i2 :: _ →
            let t' = sort_2i (\leq) i2 t in
            let out = List.filter (fun a \rightarrow i2 \neq a) (leafs t') in
            fprintf tex "\fmfframe(8,7)(8,6){\%\n"};
            fprintf tex "uu\\begin{fmfgraph*}(35,30)\n";
            fprintf\ tex\ "_{\sqcup\sqcup\sqcup} \fmfpen{thin} \n";
            fprintf\ tex\ "$\lund$$ fmfleft{i%s,i%s}\n"\ (to\_label\ i1)\ (to\_label\ i2);
            fprintf tex "uuuu\\fmfright{o%s}\n"
```

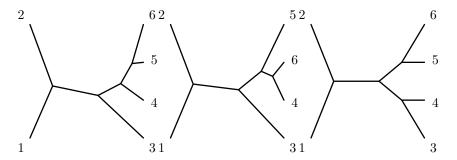


Figure Q.1: Note that this is subtly different ...

```
(String.concat ",o" (List.map to_label out));
         List.iter
           (fun s \rightarrow
              fprintf\ tex\ "lull" \ \fmflabel{$\{\%s\}}{i\%s}\n
                (to\_TeX\ s)\ (to\_label\ s))
           [i1; i2];
        List.iter
           (fun s \rightarrow
              fprintf\ tex\ "lull \fmflabel{${\%s}}{o\%s}\n"
                 (to\_TeX \ s) \ (to\_label \ s))
           out:
         edges_feynmf tex to_label i1 i2 t';
        fprintf\ tex\ "$\sqcup$\bot\\end{fmfgraph*}}\\\hfil\\allowbreak\n"
   | - \rightarrow ()
let vanilla = \{ style = None; rev = false; label = None; tension = None \}
let sty(s, r, l) = \{ vanilla \text{ with } style = Some s; rev = r; label = Some l \}
\mathsf{type}\,\,\lambda\,\, \mathit{feynmf\_set}\,\,=\,\,
   \{ header : string; 
      incoming : \lambda \ list;
      diagrams: (feynmf, \lambda) t list }
type (\lambda, \mu) feynmf_sets =
   \{ outer : \lambda feynmf\_set; \}
     inner: \mu feynmf\_set list }
type \lambda \ feynmf\_levels =
   \{ this : \lambda feynmf\_set; \}
     lower: \lambda feynmf\_levels list 
let \ latex\_section \ = \ function
     level \ \ when \ level \ < \ 0 \ 	o \ \ "part"
     0 \ \to \ \texttt{"chapter"}
     1 \rightarrow "section"
     2 \rightarrow "subsection"
     3\,\rightarrow\, "subsubsection"
     4 \rightarrow \text{"paragraph"}

ightarrow "subparagraph"
let\ rec\ \mathit{feynmf\_set}\ \mathit{tex}\ \mathit{sections}\ \mathit{level}\ \mathit{to\_TeX}\ \mathit{to\_label}\ \mathit{set}\ =
  fprintf tex "%s\\%s{\%s}\n"
     (if sections then "" else "%%%\square")
     (latex_section level)
     set.header;
   List.iter
     (to_feynmf_channel tex to_TeX to_label set.incoming)
     set.diagrams
```

```
let feynmf_sets tex sections level
     to\_TeX\_outer\ to\_label\_outer\ to\_TeX\_inner\ to\_label\_inner\ set\ =
  feynmf_set tex sections level to_TeX_outer to_label_outer set.outer;
  List.iter
     (feynmf_set tex sections (succ level) to_TeX_inner to_label_inner)
     set.inner
let feynmf_sets_plain sections level file
     to\_TeX\_outer\ to\_label\_outer\ to\_TeX\_inner\ to\_label\_inner\ sets\ =
  let tex = open\_out (file ^ ".tex") in
  List.iter
     (feynmf_sets tex sections level
        to_TeX_outer to_label_outer to_TeX_inner to_label_inner)
  close_out tex
let feynmf_header tex file =
  fprintf tex "\\documentclass[10pt]{article}\n";
  fprintf tex "\\usepackage{ifpdf}\n";
  fprintf tex "\\usepackage[colorlinks]{hyperref}\n";
  fprintf tex "\\usepackage[a4paper,margin=1cm]{geometry}\n";
  fprintf tex "\\usepackage{feynmp}\n";
  fprintf tex "\\ifpdf\n";
  fprintf\ tex\ "

\\DeclareGraphicsRule{*}{mps}{*}\\n";
  fprintf tex "\\else\n";
  fprintf\ tex\ "_{\sqcup\sqcup\sqcup} \\\ensuremath{\mbox{\sc lareGraphicsRule}} \{*\} \{eps\} \{*\} \{\} \n";
  fprintf tex "\\fi\n";
  fprintf tex "\\setlength{\\unitlength}{1mm}\n";
  fprintf tex "\\setlength{\\parindent}{Opt}\n";
  fprintf tex
     "\renewcommand{\\mathstrut}{\\protect\\vphantom\{\hat\{0123456789\}\}\n";
  fprintf tex "\\begin{document}\n";
  fprintf tex "\\tableofcontents\n";
  fprintf tex "\\begin{fmffile}{%s-fmf}\n\n" file
let feynmf\_footer tex =
  fprintf tex "\n";
  fprintf tex "\\end{fmffile}..\n";
  fprintf tex "\\end{document}_\\n"
let feynmf_sets_wrapped latex file
     to\_TeX\_outer\ to\_label\_outer\ to\_TeX\_inner\ to\_label\_inner\ sets\ =
  let tex = open\_out (file ` ".tex") in
  if latex then feynmf_header tex file;
  List.iter
     (feynmf_sets tex latex 1
        to_TeX_outer to_label_outer to_TeX_inner to_label_inner)
     sets:
  if latex then feynmf_footer tex;
  close_out tex
{\tt let\ rec}\ \mathit{feynmf\_levels}\ \mathit{tex}\ \mathit{sections}\ \mathit{level}\ \mathit{to\_TeX}\ \mathit{to\_label}\ \mathit{set}\ =
  fprintf tex "%s\\%s{%s}\n"
     (if sections then "" else "%%%∟")
     (latex_section level)
     set.this.header;
  List.iter
     (to_feynmf_channel tex to_TeX to_label set.this.incoming)
     set.this.diagrams;
  List.iter (feynmf_levels tex sections (succ level) to_TeX to_label) set.lower
let feynmf_levels_plain sections level file to_TeX to_label sets =
  let tex = open\_out (file ^ ".tex") in
  List.iter (feynmf_levels tex sections level to_TeX to_label) sets;
```

close_out tex

Q.2.4 Least Squares Layout

$$L = \frac{1}{2} \sum_{i \neq i'} T_{ii'} (x_i - x_{i'})^2 + \frac{1}{2} \sum_{i,j} T'_{ij} (x_i - e_j)^2$$
 (Q.11)

and thus

$$0 = \frac{\partial L}{\partial x_i} = \sum_{i' \neq i} T_{ii'} (x_i - x_{i'}) + \sum_j T'_{ij} (x_i - e_j)$$
 (Q.12)

or

$$\left(\sum_{i'\neq i} T_{ii'} + \sum_{j} T'_{ij}\right) x_i - \sum_{i'\neq i} T_{ii'} x_{i'} = \sum_{j} T'_{ij} e_j \tag{Q.13}$$

where we can assume that

$$T_{ii'} = T_{i'i} \tag{Q.14a}$$

$$T_{ii} = 0 (Q.14b)$$

 $\mathsf{type} \,\, \alpha \,\, node_with_tension \,\, = \,\, \{ \,\, node \,\, : \,\, \alpha; \,\, tension \,\, : \,\, float \,\, \}$

 $\mathsf{let}\ unit_tension\ t\ =\$

$$map \ (\text{fun } n \rightarrow \{ \ node = n; \ tension = 1.0 \}) \ (\text{fun } l \rightarrow l) \ t$$

let $leafs_and_nodes$ i2 t =

let $t' = sort_{-}2i (\leq) i2 t$ in

 $\mathsf{match}\ \mathit{nodes}\ t'\ \mathsf{with}$

```
| [] \rightarrow failwith "Tree.nodes_and_leafs:uimpossible" | i1 :: _as n \rightarrow (i1, i2, List.filter (fun <math>l \rightarrow l \neq i2) (leafs \ t'), n)
```

Not tail recursive, but they're unlikely to meet any deep trees:

let rec $internal_edges_from n = function$

The root node of the tree represents a vertex (node) and an external line (leaf) of the Feynman diagram simultaneously. Thus it requires special treatment:

```
let internal\_edges = function
```

```
| Leaf \_ \rightarrow []
| Node (n, ch) \rightarrow ThoList.flatmap (internal\_edges\_from n) ch
```

let rec $external_edges_from n = function$

```
| Leaf(n', \_) \rightarrow [(n', n)]
```

 $| Node (n', ch) \rightarrow ThoList.flatmap (external_edges_from n') ch$

 $\mathsf{let}\ external_edges\ =\ \mathsf{function}$

```
| Leaf (n, \_) \rightarrow [(n, n)] |
```

Node $(n, ch) \rightarrow (n, n)$:: ThoList.flatmap (external_edges_from n) ch

```
type ('edge, 'node, 'ext) graph =
    { int_nodes : 'node array;
```

ext_nodes : 'ext array;
int_edges : ('edge \times int \times int)

 $int_edges: ('edge \times int \times int) \ list; \\ ext_edges: ('edge \times int \times int) \ list \}$

module M = Pmap.Tree

Invert an array, viewed as a map from non-negative integers into a set. The result is a map from the set to the integers: val invert_array: α array \rightarrow (α, int) M.t

```
\mathsf{let}\ invert\_array\_unsafe\ a\ =
  fst (Array.fold\_left (fun (m, i) a\_i \rightarrow
     (M.add\ compare\ a\_i\ i\ m,\ succ\ i))\ (M.empty,\ 0)\ a)
exception Not_invertible
let \ add\_unique \ key \ data \ map =
  if M.mem compare key map then
     raise Not_invertible
  else
     M.add compare key data map
let invert\_array a =
  fst (Array.fold\_left (fun (m, i) a\_i \rightarrow
     (add\_unique\ a\_i\ i\ m,\ succ\ i))\ (M.empty,\ 0)\ a)
let graph_of_tree nodes2edge conjugate i2 t =
  let i1, i2, out, vertices = leafs\_and\_nodes i2 t in
  let int\_nodes = Array.of\_list vertices
  and ext\_nodes = Array.of\_list (conjugate i1 :: i2 :: out) in
  let int\_nodes\_index\_table = invert\_array int\_nodes
  and ext\_nodes\_index\_table = invert\_array \ ext\_nodes \ in
  {\tt let} \ int\_nodes\_index \ n \ = \ M.find \ compare \ n \ int\_nodes\_index\_table
  and ext\_nodes\_index n = M.find compare n ext\_nodes\_index\_table in
  \{ int\_nodes = int\_nodes; \}
     ext\_nodes = ext\_nodes;
     int\_edges = List.map
       (fun (n1, n2) \rightarrow
          (nodes2edge n1 n2, int_nodes_index n1, int_nodes_index n2))
       (internal\_edges\ t);
     ext\_edges = List.map
       (\mathsf{fun}\ (e,\ n)\ \to
          let e' =
            if e = i1 then
               conjugate e
            else
               e in
          (nodes2edge\ e'\ n,\ ext\_nodes\_index\ e',\ int\_nodes\_index\ n))
       (external\_edges\ t) }
let int\_incidence\ f\ null\ g\ =
  let n = Array.length g.int\_nodes in
  let incidence = Array.make\_matrix n n null in
  List.iter (fun (edge, n1, n2) \rightarrow
     if n1 \neq n2 then begin
       let edge' = f \ edge \ g.int\_nodes.(n1) \ g.int\_nodes.(n2) in
       incidence.(n1).(n2) \leftarrow edge';
       incidence.(n2).(n1) \leftarrow edge'
     end)
     g.int\_edges;
  incidence
let ext\_incidence\ f\ null\ g\ =
  let n\_int = Array.length \ g.int\_nodes
  and n\_ext = Array.length \ g.ext\_nodes in
  let incidence = Array.make\_matrix n\_int n\_ext null in
  List.iter (fun (edge, e, n) \rightarrow
     incidence.(n).(e) \leftarrow f \ edge \ g.ext\_nodes.(e) \ g.int\_nodes.(n))
     q.ext\_edges;
  incidence
let division n =
```

```
\quad \text{if } n \ < \ 0 \ \text{then} \\
     \mathsf{else} \; \mathsf{if} \; n \; = \; 1 \; \mathsf{then}
     [0.5]
   else
     let n' = pred n in
     let d = 1.0 /. (float n') in
     \mathsf{let} \ \mathsf{rec} \ \mathit{division'} \ i \ \mathit{acc} \ = \\
        if i < 0 then
           acc
        else
            division' (pred i) (float i * . d :: acc) in
      division' n'
type (\varepsilon, \nu, 'ext) \ ext\_layout = (\varepsilon, \nu, 'ext \times float \times float) \ graph
type (\varepsilon, \nu, 'ext) layout = (\varepsilon, \nu \times float \times float, 'ext) ext_layout
let left\_to\_right num\_in g =
  if num_in < 1 then
     invalid_arg "left_to_right"
  else
     let num\_out = Array.length g.ext\_nodes - num\_in in
     if num\_out < 1 then
        invalid_arg "left_to_right"
     else
        let incoming =
           List.map2 (fun e y \rightarrow (e, 0.0, y))
              (Array.to\_list\ (Array.sub\ g.ext\_nodes\ 0\ num\_in))
              (division num_in)
        and outgoing =
           List.map2 (fun e y \rightarrow (e, 1.0, y))
              (Array.to\_list\ (Array.sub\ g.ext\_nodes\ num\_in\ num\_out))
              (division \ num\_out) in
        \{g \text{ with } ext\_nodes = Array.of\_list (incoming @ outgoing) \}
```

Reformulating (Q.13)

$$Ax = b_x (Q.15a)$$

$$Ay = b_y (Q.15b)$$

with

$$A_{ii'} = \left(\sum_{i''\neq i} T_{ii''} + \sum_{j} T'_{ij}\right) \delta_{ii'} - T_{ii'}$$
 (Q.16a)

$$(b_{x/y})_i = \sum_j T'_{ij}(e_{x/y})_j \tag{Q.16b}$$

```
let sum\ a = Array.fold\_left\ (+.)\ 0.0\ a

let tension\_to\_equation\ t\ t'\ e =
let xe,\ ye = List.split\ e in
let bx = Linalg.matmulv\ t'\ (Array.of\_list\ xe)
and by = Linalg.matmulv\ t'\ (Array.of\_list\ ye)
and a = Array.init\ (Array.length\ t)
(fun i \to
let a\_i = Array.map\ (~-.)\ t.(i) in
a\_i.(i) \leftarrow a\_i.(i) + .sum\ t.(i) + .sum\ t'.(i);
a\_i) in
(a,\ bx,\ by)

let layout\ g =
let ext\_nodes =
List.map\ (fun\ (\_,\ x,\ y) \to (x,\ y))\ (Array.to\_list\ g.ext\_nodes) in
```

```
let a, bx, by =
     tension\_to\_equation
        (int\_incidence (fun \_ \_ \_ \rightarrow 1.0) 0.0 g)
        (ext\_incidence (fun \_ \_ \_ \rightarrow 1.0) \ 0.0 \ g) \ ext\_nodes in
  match Linalg.solve\_many a [bx; by] with
  [x; y] \rightarrow \{g \text{ with } int\_nodes = Array.mapi\}
                         (fun i n \rightarrow (n, x.(i), y.(i))) g.int\_nodes)
  \mid _ 
ightarrow failwith "impossible"
let iter\_edges\ f\ g\ =
  List.iter (fun (edge, n1, n2) \rightarrow
     \mathsf{let}\ \_,\ x1,\ y1\ =\ g.int\_nodes.(n1)
     and _, x2, y2 = g.int\_nodes.(n2) in
     f \ edge \ (x1, \ y1) \ (x2, \ y2)) \ g.int\_edges;
  List.iter~(\mathsf{fun}~(edge,~e,~n)~\rightarrow
     \mathsf{let} \ \_, \ x1, \ y1 \ = \ g.ext\_nodes.(e)
     and _{-}, x2, y2 = g.int\_nodes.(n) in
     f \ edge \ (x1, \ y1) \ (x2, \ y2)) \ g.ext\_edges
\mathsf{let}\ iter\_internal\ f\ g\ =
   Array.iter (fun (node, x, y) \rightarrow f(x, y)) g.int_nodes
\mathsf{let}\ iter\_incoming\ f\ g\ =
  f \ g.ext\_nodes.(0);
  f \ g.ext\_nodes.(1)
let iter\_outgoing f g =
  for i = 2 to pred (Array.length g.ext\_nodes) do
     f \ g.ext\_nodes.(i)
  done
let dump g =
  Array.iter (fun (\_, x, y) \rightarrow Printf.eprintf "(%g,%g)_{\sqcup}" x y) g.ext\_nodes;
   Printf.eprintf "\n_=>_\";
  Array.iter (fun (\_, x, y) \rightarrow Printf.eprintf "(%g,%g)_{\sqcup}" x y) g.int\_nodes;
  Printf.eprintf "\n"
```

—R— Dependency Trees

R.1 Interface of Tree2

Dependency trees for wavefunctions.

```
\begin{array}{l} {\rm type}\ (\nu,\ \varepsilon)\ t \\ {\rm val}\ cons\ :\ (\varepsilon\ \times\ \nu\ \times\ (\nu,\ \varepsilon)\ t\ list)\ list\ \to\ (\nu,\ \varepsilon)\ t \\ {\rm val}\ leaf\ :\ \nu\ \to\ (\nu,\ \varepsilon)\ t\ \to\ bool \\ {\rm val}\ to\_string\ :\ (\nu\ \to\ string)\ \to\ (\varepsilon\ \to\ string)\ \to\ (\nu,\ \varepsilon)\ t\ \to\ string \\ {\rm val}\ to\_channel\ :\ out\_channel\ \to\ (\nu\ \to\ string)\ \to\ (\varepsilon\ \to\ string)\ \to\ (\nu,\ \varepsilon)\ t\ \to\ unit \\ \end{array}
```

R.2 Implementation of Tree2

Dependency trees for wavefunctions.

```
type (\nu, \varepsilon) t =
  | Node of (\varepsilon \times \nu \times (\nu, \varepsilon) t \text{ list}) \text{ list}
    Leaf of \nu
let leaf node = Leaf node
let sort\_children (edge, node, children) =
  (edge, node, List.sort compare children)
let cons fusions = Node (List.sort compare (List.map sort_children fusions))
let is\_singleton = function
  \mid Leaf \_ \rightarrow true
  \mid \ \_ \rightarrow \mathsf{false}
let rec to\_string \ n2s \ e2s = function
  | Leaf n \rightarrow n2s n
  | Node [children] \rightarrow
      children_to_string n2s e2s children
  \mid Node \ children2 \rightarrow
         and children\_to\_string\ n2s\ e2s\ (e,\ n,\ children)\ =
  "(" ^ (match e2s e with "" \rightarrow "" | s \rightarrow s ^ ">") ^ n2s n ^ ":" ^
     (String.concat "," (List.map (to_string n2s e2s) children)) ^ ")"
let rec to\_channel\ ch\ n2s\ e2s\ =\ {\sf function}
    Leaf n \rightarrow Printf.fprintf ch "%s" (n2s n)
     Node [] \rightarrow Printf.fprintf ch "{\sqcup}";
    Node \ [children] \rightarrow children\_to\_channel \ ch \ n2s \ e2s \ children
   | Node (children :: children 2) \rightarrow
      Printf.fprintf ch "{\_";}
      children_to_channel ch n2s e2s children;
```

```
List.iter
          (fun children \rightarrow
             \textit{Printf.fprintf ch "$\sqcup$\backslash n$\sqcup L$\sqcup"};
             children_to_channel ch n2s e2s children)
          children 2;
       Printf.fprintf\ ch\ "{}_{\sqcup}\}"
and children\_to\_channel\ ch\ n2s\ e2s\ (e,\ n,\ children)\ =
   Printf.fprintf ch "(";
   begin match e2s e with
     "" \rightarrow ()
   \mid s \rightarrow Printf.fprintf ch "%s>" s
   end;
   Printf.fprintf ch "%s:" (n2s n);
   begin match {\it children} with
   | [] \rightarrow ()
   | [child] \rightarrow to\_channel \ ch \ n2s \ e2s \ child
   \mid child :: children \rightarrow
       to\_channel\ ch\ n2s\ e2s\ child;
       List.iter
          (fun child \rightarrow
             Printf.fprintf\ ch\ ",";
             to_channel ch n2s e2s child)
          children
   end;
   Printf.fprintf ch ")"
```

—S— CONSISTENCY CHECKS



Application count.ml unavailable!

—T— Complex Numbers



Interface complex.mli unavailable!

Implementation complex.ml unavailable!

—U— Algebra

U.1 Interface of Algebra

```
\begin{array}{ll} \operatorname{module\ type\ } Test\ = \\ \operatorname{sig} \\ \operatorname{val\ } suite\ :\ OUnit.test \\ \operatorname{end} \end{array}
```

U.1.1 Coefficients

For our algebra, we need coefficient rings.

```
\begin{array}{l} \text{module type } CRing = \\ \text{sig} \\ \text{type } t \\ \text{val } null : t \\ \text{val } unit : t \\ \text{val } mul : t \rightarrow t \rightarrow t \\ \text{val } add : t \rightarrow t \rightarrow t \\ \text{val } sub : t \rightarrow t \rightarrow t \\ \text{val } neg : t \rightarrow t \\ \text{val } to\_string : t \rightarrow string \\ \text{end} \end{array}
```

And rational numbers provide a particularly important example:

```
module type Rational =
   sig
      include CRing
      \mathsf{val}\ is\_null\ :\ t\ \to\ bool
      \mathsf{val}\ is\_unit\ :\ t\ \to\ bool
      val is\_positive : t \rightarrow bool
      \mathsf{val}\ is\_negative\ :\ t\ \to\ bool
      \mathsf{val}\ is\_integer\ :\ t\ \to\ bool
      \mathsf{val}\ make\ :\ int \to\ int \to\ t
      \mathsf{val}\ abs\ :\ t\ \to\ t
      \mathsf{val}\ inv\ :\ t\ \to\ t
      \mathsf{val}\ div\ :\ t\ \to\ t\ \to\ t
      \mathsf{val}\ pow\ :\ t\ \to\ int\ \to\ t
      val sum : t list \rightarrow t
      \mathsf{val}\ to\_ratio\ :\ t\ \to\ int\ \times\ int
      \mathsf{val}\ to\_float\ :\ t\ \to\ float
      val to\_integer: t \rightarrow int
      module \ Test : Test
   end
```

U.1.2 Naive Rational Arithmetic



This is dangerous and will overflow even for simple applications. The production code will have to be linked to a library for large integer arithmetic.

```
\begin{array}{ll} \operatorname{module} \ Small\_Rational \ : \ Rational \\ \operatorname{module} \ Q \ : \ Rational \end{array}
```

U.1.3 Rational Complex Numbers

```
module type QComplex =
   sig
      type q
      type t
      val make: q \rightarrow q \rightarrow t
      val null : t
      val unit: t
      val real : t \rightarrow q
      val\ imag\ :\ t\ 
ightarrow\ q
      \mathsf{val}\ \mathit{conj}\ :\ t\ \to\ t
      \mathsf{val}\ neg\ :\ t\ \to\ t
      \mathsf{val}\ add\ :\ t\ \to\ t\ \to\ t
      \mathsf{val}\ sub\ :\ t\ \to\ t\ \to\ t
      \mathsf{val}\ mul\ :\ t\ \to\ t\ \to\ t
      \mathsf{val}\ inv\ :\ t\ \to\ t
      \mathsf{val}\ div\ :\ t\ \to\ t\ \to\ t
      \mathsf{val}\ pow\ :\ t\ 	o\ int\ 	o\ t
      \mathsf{val}\ sum\ :\ t\ list \to\ t
      val is\_null : t \rightarrow bool
      val is\_unit : t \rightarrow bool
      val is\_positive : t \rightarrow bool
      val is\_negative : t \rightarrow bool
      val is\_integer : t \rightarrow bool
      val is\_real : t \rightarrow bool
      val to\_string : t \rightarrow string
      module Test: Test
   end
module QComplex: functor (Q': Rational) \rightarrow QComplex with type q = Q'.t
\mathsf{module}\ QC\ :\ QComplex\ \mathsf{with}\ \mathsf{type}\ q\ =\ Q.t
```

U.1.4 Laurent Polynomials

Polynomials, including negative powers, in one variable. In our applications, the variable x will often be N_C , the number of colors

$$\sum_{n} c_n N_C^n \tag{U.1}$$

```
module type Laurent =  sig
```

The type of coefficients. In the implementation below, it is QComplex.t: complex numbers with rational real and imaginary parts.

```
\begin{array}{c} {\rm type} \ c \\ {\rm type} \ t \end{array}
```

Elementary constructors

```
\mathsf{val}\ null\ :\ t
```

```
val\ is\_null\ :\ t\ 	o\ bool val\ unit\ :\ t
```

atom c n constructs a term cx^n , where x denotes the variable.

Elementary arithmetic

```
\begin{array}{l} \text{val } scale \ : \ c \ \rightarrow \ t \ \rightarrow \ t \\ \text{val } neg \ : \ t \ \rightarrow \ t \\ \text{val } add \ : \ t \ \rightarrow \ t \ \rightarrow \ t \\ \text{val } diff \ : \ t \ \rightarrow \ t \ \rightarrow \ t \\ \text{val } sum \ : \ t \ list \ \rightarrow \ t \\ \text{val } mul \ : \ t \ \rightarrow \ t \ \rightarrow \ t \\ \text{val } product \ : \ t \ list \ \rightarrow \ t \\ \text{val } pow \ : \ int \ \rightarrow \ t \ \rightarrow \ t \end{array}
```

 $eval\ c\ p$ evaluates the polynomial p by substituting the constant c for the variable.

```
\mathsf{val}\ eval\ :\ c\ \to\ t\ \to\ c
```

A total ordering. Does not correspond to any mathematical order.

```
val\ compare\ :\ t\ 
ightarrow\ t\ 
ightarrow\ int
```

Logging, debugging and toplevel integration.

```
val to\_string: string 
ightarrow t 
ightarrow string val pp: Format.formatter 
ightarrow t 
ightarrow unit module Test: Test end
```

\$

Could (should?) be functorialized over *QComplex*. We had to wait until we upgraded our O'Caml requirements to 4.02, but that has been done.

module Laurent: Laurent with type c = QC.t

U.1.5 Expressions: Terms, Rings and Linear Combinations

The tensor algebra will be spanned by an abelian monoid:

```
\begin{array}{l} \text{module type } \textit{Term} = \\ \text{sig} \\ \text{type } \alpha \ t \\ \text{val } \textit{unit} : \textit{unit} \rightarrow \alpha \ t \\ \text{val } \textit{is\_unit} : \alpha \ t \rightarrow \textit{bool} \\ \text{val } \textit{atom} : \alpha \rightarrow \alpha \ t \\ \text{val } \textit{power} : \textit{int} \rightarrow \alpha \ t \rightarrow \alpha \ t \\ \text{val } \textit{mul} : \alpha \ t \rightarrow \alpha \ t \rightarrow \alpha \ t \\ \text{val } \textit{mul} : \alpha \ t \rightarrow \alpha \ t \rightarrow \alpha \ t \\ \text{val } \textit{map} : (\alpha \rightarrow \beta) \rightarrow \alpha \ t \rightarrow \beta \ t \\ \text{val } \textit{to\_string} : (\alpha \rightarrow \textit{string}) \rightarrow \alpha \ t \rightarrow \textit{string} \end{array}
```

The derivative of a term is *not* a term, but a sum of terms instead:

$$D(f_1^{p_1} f_2^{p_2} \cdots f_n^{p_n}) = \sum_i (Df_i) p_i f_1^{p_1} f_2^{p_2} \cdots f_i^{p_i-1} \cdots f_n^{p_n}$$
 (U.2)

The function returns the sum as a list of triples $(Df_i, p_i, f_1^{p_1} f_2^{p_2} \cdots f_i^{p_i-1} \cdots f_n^{p_n})$. Summing the terms is left to the calling module and the Df_i are not guaranteed to be different. NB: The function implementating the inner derivative, is supposed to return $Some\ Df_i$ and None, iff Df_i vanishes.

```
val derive: (\alpha \rightarrow \beta \ option) \rightarrow \alpha \ t \rightarrow (\beta \times int \times \alpha \ t) \ list
```

convenience function

```
val product: \alpha t list <math>\rightarrow \alpha t
       val atoms : \alpha t \rightarrow \alpha list
   end
module type Ring =
   sig
       \mathsf{module}\ C\ :\ Rational
       type \alpha t
       \mathsf{val}\ null\ :\ unit\ \rightarrow\ \alpha\ t
       val\ unit: unit \rightarrow \ \alpha\ t
       val is\_null : \alpha t \rightarrow bool
       val is\_unit : \alpha t \rightarrow bool
       val atom : \alpha \rightarrow \alpha t
       val scale : C.t \rightarrow \alpha t \rightarrow \alpha t
       \mathsf{val}\ add\ :\ \alpha\ t\ \to\ \alpha\ t\ \to\ \alpha\ t
       \mathsf{val}\ sub\ :\ \alpha\ t\ \to\ \alpha\ t\ \to\ \alpha\ t
       \mathsf{val}\ mul\ :\ \alpha\ t\ \to\ \alpha\ t\ \to\ \alpha\ t
       \mathsf{val}\ neg\ :\ \alpha\ t\ \to\ \alpha\ t
```

Again

$$D(f_1^{p_1} f_2^{p_2} \cdots f_n^{p_n}) = \sum_i (Df_i) p_i f_1^{p_1} f_2^{p_2} \cdots f_i^{p_i-1} \cdots f_n^{p_n}$$
 (U.3)

but, iff Df_i can be identified with a f', we know how to perform the sum.

```
val derive\_inner: (\alpha \rightarrow \alpha \ t) \rightarrow \alpha \ t \rightarrow \alpha \ t \ (* this? *) val derive\_inner': (\alpha \rightarrow \alpha \ t \ option) \rightarrow \alpha \ t \rightarrow \alpha \ t \ (* or \ that? *)
```

Below, we will need partial derivatives that lead out of the ring: $derive_outer\ derive_atom\ term$ returns a list of partial derivatives β with non-zero coefficients α t:

```
val derive\_outer: (\alpha \rightarrow \beta \ option) \rightarrow \alpha \ t \rightarrow (\beta \times \alpha \ t) \ list
```

convenience functions

```
\begin{array}{lll} \mathsf{val} \ sum \ : \ \alpha \ t \ list \rightarrow \ \alpha \ t \\ \mathsf{val} \ product \ : \ \alpha \ t \ list \rightarrow \ \alpha \ t \end{array}
```

The list of all generators appearing in an expression:

```
\begin{array}{l} \text{val } atoms \ : \ \alpha \ t \ \rightarrow \ \alpha \ list \\ \\ \text{val } to\_string \ : \ (\alpha \ \rightarrow \ string) \ \rightarrow \ \alpha \ t \ \rightarrow \ string \\ \\ \text{end} \\ \\ \text{module type } Linear \ = \\ \\ \text{sig} \\ \\ \text{module } C \ : \ Ring \end{array}
```

module C: Ringtype (α, γ) tval $null: unit \rightarrow (\alpha, \gamma)$ tval $atom: \alpha \rightarrow (\alpha, \gamma)$ tval $singleton: \gamma C.t \rightarrow \alpha \rightarrow (\alpha, \gamma)$ tval $scale: \gamma C.t \rightarrow (\alpha, \gamma)$ $t \rightarrow (\alpha, \gamma)$ tval $add: (\alpha, \gamma)$ $t \rightarrow (\alpha, \gamma)$ $t \rightarrow (\alpha, \gamma)$ tval $sub: (\alpha, \gamma)$ $t \rightarrow (\alpha, \gamma)$ $t \rightarrow (\alpha, \gamma)$ t

A partial derivative w.r.t. a vector maps from a coefficient ring to the dual vector space.

```
\mathsf{val}\ partial\ :\ (\gamma\ \to\ (\alpha,\ \gamma)\ t)\ \to\ \gamma\ C.t\ \to\ (\alpha,\ \gamma)\ t
```

A linear combination of vectors

$$linear[(v_1, c_1); (v_2, c_2); \dots; (v_n, c_n)] = \sum_{i=1}^{n} c_i \cdot v_i$$
 (U.4)

val $linear : ((\alpha, \gamma) \ t \times \gamma \ C.t) \ list \rightarrow (\alpha, \gamma) \ t$

Some convenience functions

```
\begin{array}{lll} \mathrm{val} \ map \ : \ (\alpha \ \rightarrow \ \gamma \ C.t \ \rightarrow \ (\beta, \ \delta) \ t) \ \rightarrow \ (\alpha, \ \gamma) \ t \ \rightarrow \ (\beta, \ \delta) \ t \\ \mathrm{val} \ sum \ : \ (\alpha, \ \gamma) \ t \ list \ \rightarrow \ (\alpha, \ \gamma) \ t \end{array}
```

The list of all generators and the list of all generators of coefficients appearing in an expression:

```
val atoms: (\alpha, \gamma) \ t \rightarrow \alpha \ list \times \gamma \ list val to\_string: (\alpha \rightarrow string) \rightarrow (\gamma \rightarrow string) \rightarrow (\alpha, \gamma) \ t \rightarrow string end module Term: Term module Make\_Ring\ (C: Rational)\ (T: Term): Ring module Make\_Linear\ (C: Ring): Linear\ with\ module\ C = C
```

U.2 Implementation of Algebra

Avoid referring to *Pervasives.compare*, because *Pervasives* will become *Stdlib.Pervasives* in O'Caml 4.07 and *Stdlib* in O'Caml 4.08.

```
\begin{array}{lll} \text{let } pcompare &=& compare \\ \\ \text{module type } Test &=& \\ \\ \text{sig} & \text{val } suite \ : \ OUnit.test \\ \\ \text{end} & \end{array}
```

The terms will be small and there's no need to be fancy and/or efficient. It's more important to have a unique representation.

module PM = Pmap.List

U.2.1 Coefficients

For our algebra, we need coefficient rings.

```
\begin{array}{l} \text{module type } CRing = \\ \text{sig} \\ \text{type } t \\ \text{val } null : t \\ \text{val } unit : t \\ \text{val } mul : t \rightarrow t \rightarrow t \\ \text{val } add : t \rightarrow t \rightarrow t \\ \text{val } sub : t \rightarrow t \rightarrow t \\ \text{val } neg : t \rightarrow t \\ \text{val } to\_string : t \rightarrow string \\ \text{end} \end{array}
```

And rational numbers provide a particularly important example:

 $\begin{array}{l} \text{module type } Rational = \\ \text{sig} \\ \text{include } CRing \\ \text{val } is_null : t \rightarrow bool \\ \text{val } is_nuit : t \rightarrow bool \\ \text{val } is_positive : t \rightarrow bool \\ \text{val } is_negative : t \rightarrow bool \\ \text{val } is_integer : t \rightarrow bool \\ \text{val } is_integer : t \rightarrow bool \\ \text{val } make : int \rightarrow int \rightarrow t \\ \text{val } abs : t \rightarrow t \\ \text{val } inv : t \rightarrow t \\ \text{val } div : t \rightarrow t \rightarrow t \\ \text{val } pow : t \rightarrow int \rightarrow t \\ \text{val } sum : t \ list \rightarrow t \\ \text{val } to_ratio : t \rightarrow int \times int \\ \end{array}$

 $val to_float : t \rightarrow float$

U.2.2 Naive Rational Arithmetic

\$

This is dangerous and will overflow even for simple applications. The production code will have to be linked to a library for large integer arithmetic.

Anyway, here's Euclid's algorithm:

```
let rec gcd i1 i2 =
  if i2 = 0 then
     abs i1
  else
     gcd i2 (i1 mod i2)
let lcm i1 i2 = (i1 / gcd i1 i2) \times i2
module Small\_Rational : Rational =
  struct
     \mathsf{type}\ t\ =\ int\ \times\ int
     let is_null\ (n, \ \_) = (n = 0)
     let is\_unit(n, d) = (n \neq 0) \land (n = d)
     let is_{-}positive\ (n,\ d)\ =\ n\ \times\ d\ >\ 0
     let is\_negative\ (n,\ d)\ =\ n\ \times\ d\ <\ 0
     let is\_integer (n, d) = (gcd n d = d)
     let null = (0, 1)
     let unit = (1, 1)
     \mathsf{let}\ \mathit{make}\ \mathit{n}\ \mathit{d}\ =
       \mathsf{let}\ c\ =\ gcd\ n\ d\ \mathsf{in}
       (n / c, d / c)
     let abs (n, d) = (abs n, abs d)
     let inv (n, d) = (d, n)
     let mul\ (n1,\ d1)\ (n2,\ d2)\ =\ make\ (n1\ \times\ n2)\ (d1\ \times\ d2)
     let div q1 q2 = mul q1 (inv q2)
     let add~(n1,~d1)~(n2,~d2)~=~make~(n1~\times~d2~+~n2~\times~d1)~(d1~\times~d2)
     let sub\ (n1,\ d1)\ (n2,\ d2)\ =\ make\ (n1\ \times\ d2\ -\ n2\ \times\ d1)\ (d1\ \times\ d2)
     let neg (n, d) = (-n, d)
     let rec pow q p =
       if p = 0 then
          unit
       else if p < 0 then
          pow (inv q) (-p)
       else
          mul \ q \ (pow \ q \ (pred \ p))
     let sum qs =
       List.fold_right add qs null
     let to_ratio(n, d) =
       \quad \text{if} \ d \ < \ 0 \ \text{then} \\
          (-n, -d)
       else
          (n, d)
     let to_float (n, d) = float n / float d
     let to\_string(n, d) =
       if d = 1 then
          Printf.sprintf "%d" n
          let n, d = to\_ratio(n, d) in
          Printf.sprintf "(%d/%d)" n \ d
     let to\_integer(n, d) =
       if is\_integer (n, d) then
```

```
else
            invalid_arg "Algebra.Small_Rational.to_integer"
      module \ Test =
        struct
           open OUnit
           \mathsf{let}\ equal\ z1\ z2\ =
              is\_null (sub z1 z2)
           let assert\_equal\_rational\ z1\ z2\ =
               assert\_equal\ \~printer: to\_string\ \~cmp: equal\ z1\ z2
           let suite\_mul =
               "mul" >:::
                 [ "1*1=1" >::
                       (fun () \rightarrow
                          assert_equal_rational (mul unit unit) unit) ]
           let suite =
               "Algebra.Small_Rational" >:::
                 [suite\_mul]
         end
  end
\mathsf{module}\ Q\ =\ Small\_Rational
                                             U.2.3 Rational Complex Numbers
module type QComplex =
  sig
     type q
     type t
     \mathsf{val}\ make\ :\ q\ \to\ q\ \to\ t
     \mathsf{val}\ null\ :\ t
     val \ unit : t
     \mathsf{val}\ \mathit{real}\ :\ t\ \to\ q
     \mathsf{val}\ imag\ :\ t\ \to\ q
     val\ conj\ :\ t\ 	o\ t
     \mathsf{val}\ neg\ :\ t\ \to\ t
     \mathsf{val}\ add\ :\ t\ \to\ t\ \to\ t
     \mathsf{val}\ sub\ :\ t\ \to\ t\ \to\ t
     \mathsf{val}\ mul\ :\ t\ \to\ t\ \to\ t
     \mathsf{val}\ inv\ :\ t\ \to\ t
     \mathsf{val}\ div\ :\ t\ \to\ t\ \to\ t
     \mathsf{val}\ pow\ :\ t\ \to\ int\ \to\ t
     val sum : t list \rightarrow t
     \mathsf{val}\ is\_null\ :\ t\ \to\ bool
     \mathsf{val}\ is\_unit\ :\ t\ \to\ bool
     val is\_positive : t \rightarrow bool
     val is\_negative : t \rightarrow bool
     \mathsf{val}\ is\_integer\ :\ t\ \to\ bool
     val is\_real : t \rightarrow bool
     val to\_string : t \rightarrow string
     module Test: Test
  end
```

n

```
module QComplex (Q : Rational) : QComplex with type <math>q = Q.t =
  struct
     \mathsf{type}\ q\ =\ Q.t
    \mathsf{type}\ t\ =\ \{\ re\ :\ q;\ im\ :\ q\ \}
     let make re im = \{ re; im \}
     let null = \{ re = Q.null; im = Q.null \}
    let unit = \{ re = Q.unit; im = Q.null \}
    let real z = z.re
    let imag z = z.im
    let conj z = \{ re = z.re; im = Q.neg z.im \}
     let neg z = \{ re = Q.neg z.re; im = Q.neg z.im \}
     let \ add \ z1 \ z2 \ = \ \left\{ \ re \ = \ Q.add \ z1.re \ z2.re; \ im \ = \ Q.add \ z1.im \ z2.im \ \right\}
    let sub z1 z2 = \{ re = Q.sub z1.re z2.re; im = Q.sub z1.im z2.im \}
    let sum qs =
       List.fold_right add qs null
Save one multiplication with respect to the standard formula
                            (x+iy)(u+iv) = [xu - yv] + i[(x+u)(y+v) - xu - yv]
                                                                                                                 (U.5)
at the expense of one addition and two subtractions.
     let mul \ z1 \ z2 =
       \mathsf{let}\ re12\ =\ Q.mul\ z1.re\ z2.re
       and im12 = Q.mul \ z1.im \ z2.im in
       \{ re = Q.sub \ re12 \ im12; 
          im = Q.sub
                  (Q.sub\ (Q.mul\ (Q.add\ z1.re\ z1.im)\ (Q.add\ z2.re\ z2.im))\ re12)
    let inv z =
       let modulus = Q.add (Q.mul z.re z.re) (Q.mul z.im z.im) in
       \{ re = Q.div z.re modulus; \}
          im = Q.div (Q.neg z.im) modulus 
    let div \ n \ d =
       mul\ (inv\ d)\ n
     \mathsf{let} \ \mathsf{rec} \ pow \ q \ p \ =
       if p = 0 then
         unit
       else if p < 0 then
         pow (inv q) (-p)
          mul \ q \ (pow \ q \ (pred \ p))
     let is\_real \ q =
       Q.is\_null\ q.im
    let test\_real test q =
       is\_real \ q \land test \ q.re
    let is\_null = test\_real Q.is\_null
    let is\_unit = test\_real Q.is\_unit
    let is\_positive = test\_real Q.is\_positive
    let is\_negative = test\_real Q.is\_negative
    let is\_integer = test\_real Q.is\_integer
```

(if $Q.is_negative\ q$ then "-" else " $_{\sqcup}$ ") $\hat{}$ $Q.to_string\ (Q.abs\ q)$

let $q_to_string q =$

let $to_string z =$

if $Q.is_null\ z.im$ then $q_to_string\ z.re$

```
else if Q.is\_null \ z.re then
           if Q.is\_unit\ z.im then
              "LI"
           else if Q.is\_unit (Q.neg z.im) then
              "-I"
           else
              q\_to\_string\ z.im\ ^ "*I"
        else
           Printf.sprintf "(%s%s*I)" (Q.to\_string\ z.re) (q\_to\_string\ z.im)
     module Test =
        struct
           open OUnit
           let equal \ z1 \ z2 =
              is\_null (sub z1 z2)
           let assert\_equal\_complex \ z1 \ z2 =
              assert_equal ~printer: to_string ~cmp: equal z1 z2
           let suite\_mul =
              "mul" >:::
                 [ "1*1=1" >::
                       (fun () \rightarrow
                          assert_equal_complex (mul unit unit) unit) ]
           let suite =
              "Algebra.QComplex" >:::
                 [suite\_mul]
        end
  end
module QC = QComplex(Q)
                                                 U.2.4 Laurent Polynomials
module type Laurent =
  sig
     type c
     type t
     val \ null : t
     \mathsf{val}\ is\_null\ :\ t\ \to\ bool
     val unit: t
     \mathsf{val}\ atom\ :\ c\ \to\ int\ \to\ t
     \mathsf{val}\ const\ :\ c\ \to\ t
     \mathsf{val}\ scale\ :\ c\ \to\ t\ \to\ t
     val\ neg: t \rightarrow t
     \mathsf{val}\ add\ :\ t\ \to\ t\ \to\ t
     \mathsf{val}\ \mathit{diff}\ :\ t\ \to\ t\ \to\ t
     \mathsf{val}\ \mathit{sum}\ :\ t\ \mathit{list} \to\ t
     \mathsf{val}\ mul\ :\ t\ \to\ t\ \to\ t
     val\ product : t\ list \rightarrow t
     \mathsf{val}\ pow\ :\ int\rightarrow\ t\ \rightarrow\ t
     \mathsf{val}\ eval\ :\ c\ \to\ t\ \to\ c
     val compare: t \rightarrow t \rightarrow int
     val to\_string : string \rightarrow t \rightarrow string
     \mathsf{val}\ pp\ :\ Format.formatter\ 	o\ t\ 	o\ unit
     module Test: Test
  end
```

module Laurent: Laurent with type c = QC.t =

struct

```
module IMap =
        Map.Make
           (struct
             \mathsf{type}\ t\ =\ int
             let compare i1 i2 =
                pcompare i2 i1
           end)
     \mathsf{type}\ c\ =\ QC.t
     let qc\_minus\_one =
        QC.neg\ QC.unit
     \mathsf{type}\ t\ =\ c\ \mathit{IMap.t}
     let null = IMap.empty
     let is\_null l = IMap.for\_all (fun \_ \rightarrow QC.is\_null) l
     let atom \ qc \ n =
        \text{if } qc \ = \ QC.null \ \text{then} \\
           null
           IMap.singleton\ n\ qc
     \mathsf{let}\ const\ z\ =\ atom\ z\ 0
     let unit = const QC.unit
     let add1 \ n \ qc \ l =
        try
           let qc' = QC.add \ qc \ (IMap.find \ n \ l) in
           if qc' = QC.null then
              IMap.remove n l
           else
              IMap.add \ n \ qc' \ l
        with
        \mid Not\_found \rightarrow IMap.add \ n \ qc \ l
     let add l1 l2 =
        IMap.fold add1 l1 l2
     let sum = function
        | [] \rightarrow null
        | [l] \rightarrow l
        l :: l\_list \rightarrow
            List.fold_left add l l_list
     \mathsf{let} \ \mathit{scale} \ \mathit{qc} \ \mathit{l} \ = \\
        IMap.map\ (QC.mul\ qc)\ l
     let neg l =
        IMap.map\ QC.neg\ l
     let diff l1 l2 =
        add l1 (scale qc_minus_one l2)
cf. Product.fold2\_rev
     let fold2 f l1 l2 acc =
        IMap.fold
           (fun n1 qc1 acc1 \rightarrow
                (\mathsf{fun}\ n2\ qc2\ acc2\ \rightarrow\ f\ n1\ qc1\ n2\ qc2\ acc2)
                 l2 acc1)
           l1 \ acc
     let mul \ l1 \ l2 =
        fold2
           (fun n1 qc1 n2 qc2 acc \rightarrow
              add1 (n1 + n2) (QC.mul qc1 qc2) acc)
```

```
l1 l2 null
let product = function
  | [] \rightarrow unit
  | [l] \rightarrow l
  l :: l\_list \rightarrow
       List.fold_left mul l l_list
let poly_pow multiply one inverse n x =
  let rec pow' i x' acc =
     \text{if } i \ < \ 1 \ \text{then} \\
        acc
     else
        pow' (pred i) x' (multiply x' acc) in
  if n < 0 then
     \mathsf{let}\ x'\ =\ \mathit{inverse}\ x\ \mathsf{in}
     pow' \ (pred \ (-n)) \ x' \ x'
   \  \, {\rm else} \,\, {\rm if} \,\, n \,\, = \,\, 0 \,\, {\rm then} \,\,
     one
  else
     pow' (pred n) x x
let qc\_pow \ n \ z =
  poly\_pow\ QC.mul\ QC.unit\ QC.inv\ n\ z
let pow \ n \ l =
  poly\_pow\ mul\ unit\ (\mathsf{fun}\ \_\ \to\ invalid\_arg\ \texttt{"Algebra.Laurent.pow"})\ n\ l
let q_to_string q =
  (if Q.is\_positive\ q then "+" else "-") ^ Q.to\_string\ (Q.abs\ q)
let qc\_to\_string z =
  let r = QC.real z
  and i = QC.imag z in
  if Q.is\_null\ i then
     q\_to\_string \ r
  else if Q.is\_null \ r then
     if Q.is\_unit i then
        "+I"
     else if Q.is\_unit (Q.neg i) then
        "-I"
     else
        q\_to\_string\ i\ `"*I"
  else
     Printf.sprintf "(%s%s*I)" (Q.to\_string\ r) (q\_to\_string\ i)
let to\_string1 name (n, qc) =
  \quad \text{if } n \ = \ 0 \ \text{then} \\
     qc\_to\_string \ qc
  else if n = 1 then
     if QC.is\_unit\ qc then
        name
     else if qc = qc\_minus\_one then
        \verb"-" ^ name
     else
        Printf.sprintf "%s*%s" (qc_to_string qc) name
  else if n = -1 then
     Printf.sprintf "%s/%s" (qc\_to\_string \ qc) name
  else if n > 1 then
     if QC.is\_unit\ qc then
        Printf.sprintf "%s^%d" name n
     else if qc = qc\_minus\_one then
        Printf.sprintf "-%s^%d" name \ n
     else
        Printf.sprintf "%s*%s^%d" (qc\_to\_string \ qc) name \ n
```

```
else
       Printf.sprintf "%s/%s^%d" (qc\_to\_string \ qc) name \ (-n)
  let to\_string name l =
    match IMap.bindings \ l with
     | [] → "0"
    l \rightarrow String.concat "" (List.map (to_string1 name) l)
  let pp fmt l =
    Format.fprintf fmt "%s" (to_string "N" l)
  let eval \ v \ l =
    IMap.fold
       (\text{fun } n \ qc \ acc \rightarrow QC.add \ (QC.mul \ qc \ (qc\_pow \ n \ v)) \ acc)
  let compare l1 l2 =
    pcompare
       (List.sort pcompare (IMap.bindings l1))
       (List.sort\ pcompare\ (IMap.bindings\ l2))
  let compare l1 l2 =
     IMap.compare prompare l1 l2
  module Test =
    struct
       open OUnit
       let equal \ l1 \ l2 =
         compare l1 l2 = 0
       let \ assert\_equal\_laurent \ l1 \ l2 =
         assert_equal ~printer: (to_string "N") ~cmp: equal l1 l2
       let suite\_mul =
         "mul" >:::
            ["(1+N)(1-N)=1-N^2">::
                (\mathsf{fun}\ ()\ \to
                   assert\_equal\_laurent
                     (sum [unit; atom (QC.neg QC.unit) 2])
                     (product [sum [unit; atom QC.unit 1];
                                 sum [unit; atom (QC.neg QC.unit) 1]]));
              "(1+N)(1-1/N)=N-1/N">::
                (fun () \rightarrow
                   assert\_equal\_laurent
                     (sum [atom QC.unit 1; atom (QC.neg QC.unit) (-1)])
                     (product [sum [unit; atom QC.unit 1];
                                 sum [unit; atom (QC.neg QC.unit) (-1)])); ]
       let suite =
         "Algebra.Laurent" >:::
            [suite\_mul]
    end
end
                  U.2.5
                          Expressions: Terms, Rings and Linear Combinations
```

The tensor algebra will be spanned by an abelian monoid:

```
module type Term =
   sig
       type \alpha t
       val\ unit: unit \rightarrow \alpha \ t
       \mathsf{val}\ is\_unit\ :\ \alpha\ t\ \to\ bool
       \mathsf{val}\ atom\ :\ \alpha\ \to\ \alpha\ t
```

```
\mathsf{val}\ power\ :\ int\rightarrow\ \alpha\ t\ \rightarrow\ \alpha\ t
       \mathsf{val}\ \mathit{mul}\ :\ \alpha\ t\ \to\ \alpha\ t\ \to\ \alpha\ t
       \mathsf{val}\ map\ :\ (\alpha\ \to\ \beta)\ \to\ \alpha\ t\ \to\ \beta\ t
       val to_string : (\alpha \rightarrow string) \rightarrow \alpha t \rightarrow string
       \mathsf{val}\ \mathit{derive}\ :\ (\alpha\ \rightarrow\ \beta\ \mathit{option})\ \rightarrow\ \alpha\ t\ \rightarrow\ (\beta\ \times\ \mathit{int}\ \times\ \alpha\ t)\ \mathit{list}
        val product : \alpha \ t \ list \rightarrow \alpha \ t
        val atoms: \alpha t \rightarrow \alpha list
    end
module type Ring =
   sig
       \mathsf{module}\ C\ :\ Rational
       type \alpha t
       \mathsf{val}\ null\ :\ unit\ \rightarrow\ \alpha\ t
        \mathsf{val}\ unit: unit \to \ \alpha\ t
       val is\_null : \alpha t \rightarrow bool
       val is\_unit : \alpha t \rightarrow bool
       \mathsf{val}\ atom\ :\ \alpha\ \to\ \alpha\ t
       val scale : C.t \rightarrow \alpha t \rightarrow \alpha t
       \mathsf{val}\ add\ :\ \alpha\ t\ \to\ \alpha\ t\ \to\ \alpha\ t
       \mathsf{val}\ sub\ :\ \alpha\ t\ \to\ \alpha\ t\ \to\ \alpha\ t
       \mathsf{val}\ mul\ :\ \alpha\ t\ \to\ \alpha\ t\ \to\ \alpha\ t
       \mathsf{val}\ neg\ :\ \alpha\ t\ \to\ \alpha\ t
       val derive\_inner: (\alpha \rightarrow \alpha t) \rightarrow \alpha t \rightarrow \alpha t (* this? *)
       val derive\_inner': (\alpha \rightarrow \alpha \ t \ option) \rightarrow \alpha \ t \rightarrow \alpha \ t \ (* \ or \ that? *)
       val derive\_outer: (\alpha \rightarrow \beta \ option) \rightarrow \alpha \ t \rightarrow (\beta \times \alpha \ t) \ list
       \mathsf{val}\ \mathit{sum}\ :\ \alpha\ t\ \mathit{list}\ \to\ \alpha\ t
       \mathsf{val}\ \mathit{product}\ :\ \alpha\ t\ \mathit{list}\ \rightarrow\ \alpha\ t
       val atoms : \alpha t \rightarrow \alpha list
        val to\_string : (\alpha \rightarrow string) \rightarrow \alpha t \rightarrow string
module type Linear =
   sig
        module C: Ring
       type (\alpha, \gamma) t
       val null: unit \rightarrow (\alpha, \gamma) t
       val atom : \alpha \rightarrow (\alpha, \gamma) t
       val singleton : \gamma C.t \rightarrow \alpha \rightarrow (\alpha, \gamma) t
       \mathsf{val}\ scale\ :\ \gamma\ C.t\ \to\ (\alpha,\ \gamma)\ t\ \to\ (\alpha,\ \gamma)\ t
       val add: (\alpha, \gamma) t \rightarrow (\alpha, \gamma) t \rightarrow (\alpha, \gamma) t
       val sub : (\alpha, \gamma) t \rightarrow (\alpha, \gamma) t \rightarrow (\alpha, \gamma) t
       val partial : (\gamma \rightarrow (\alpha, \gamma) t) \rightarrow \gamma C.t \rightarrow (\alpha, \gamma) t
       val linear : ((\alpha, \gamma) \ t \times \gamma \ C.t) \ list \rightarrow (\alpha, \gamma) \ t
       \mathsf{val}\ \mathit{map}\ :\ (\alpha\ \rightarrow\ \gamma\ \mathit{C.t}\ \rightarrow\ (\beta,\ \delta)\ t)\ \rightarrow\ (\alpha,\ \gamma)\ t\ \rightarrow\ (\beta,\ \delta)\ t
       val sum : (\alpha, \gamma) \ t \ list \rightarrow (\alpha, \gamma) \ t
       val atoms: (\alpha, \gamma) t \rightarrow \alpha list \times \gamma list
        val to_string : (\alpha \rightarrow string) \rightarrow (\gamma \rightarrow string) \rightarrow (\alpha, \gamma) t \rightarrow string
   end
module Term : Term =
   struct
        \mathsf{module}\ M\ =\ PM
       type \alpha t = (\alpha, int) M.t
        let unit() = M.empty
        let is\_unit = M.is\_empty
        let atom f = M.singleton f 1
       let power p x = M.map ((\times) p) x
       let insert1 binop f p term =
            let p' = binop (try M.find compare f term with Not_found \rightarrow 0) p in
```

```
if p' = 0 then
           M.remove\ compare\ f\ term
        else
           M.add compare f p' term
     let mul1 f p term = insert1 (+) f p term
     let mul x y = M.fold mul1 x y
     \mathsf{let} \ \mathit{map} \ f \ \mathit{term} \ = \ \mathit{M.fold} \ (\mathsf{fun} \ t \ \to \ \mathit{mul1} \ (f \ t)) \ \mathit{term} \ \mathit{M.empty}
     let to\_string\ fmt\ term\ =
        String.concat "*"
           (M.fold (fun f p acc \rightarrow
             (if p = 0 then
                "1"
             \mathsf{else} \; \mathsf{if} \; p \; = \; 1 \; \mathsf{then}
                fmt f
             else
                "[" ^{\hat{}} fmt f ^{\hat{}} "]^{\hat{}}" ^{\hat{}} string_of_int p) :: acc) term [])
     let derive \ derive1 \ x =
        M.fold (fun f p dx \rightarrow
          if p \neq 0 then
             match derive1 f with
               Some df \rightarrow (df, p, mul1 \ f \ (pred \ p) \ (M.remove \ compare \ f \ x)) :: dx
               None \rightarrow dx
           else
              dx) x []
     let product factors =
        List.fold\_left\ mul\ (unit\ ())\ factors
     let atoms t =
        List.map\ fst\ (PM.elements\ t)
  end
Make\_Ring (C : Rational) (T : Term) : Ring =
  struct
     \mathsf{module}\ C\ =\ C
     let one = C.unit
     \mathsf{module}\ M\ =\ PM
     type \alpha t = (\alpha T.t, C.t) M.t
     let null () = M.empty
     \mathsf{let}\ is\_null\ =\ M.is\_empty
     let power t p = M.singleton t p
     let \ unit \ () \ = \ power \ (T.unit \ ()) \ one
     let is\_unit t = unit () = t
    The following should be correct too, but produces to many false positives instead! What's going on?
     let broken\_is\_unit t =
        match M.elements t with
        | [(t, p)] \rightarrow T.is\_unit t \lor C.is\_null p
        \mid \ \_ \ 	o \ \mathsf{false}
     let atom t = power (T.atom t) one
     let scale \ c \ x = M.map \ (C.mul \ c) \ x
     let insert1 binop t c sum =
        let c' = binop (try M.find compare t sum with Not\_found \rightarrow C.null) c in
        if C.is\_null\ c' then
```

```
M.remove\ compare\ t\ sum
       else
          M.add compare t c' sum
    let add x y = M.fold (insert1 C.add) x y
     let sub x y = M.fold (insert1 C.sub) y x
One might be tempted to use Product.outer\_self\ M.fold instead, but this would require us to combine tx and cx
to (tx, cx).
    let fold2 f x y =
       M.fold (fun tx \ cx \rightarrow M.fold (f tx \ cx) y) x
       fold2 (fun tx \ cx \ ty \ cy \rightarrow insert1 \ C.add (T.mul \ tx \ ty) (C.mul \ cx \ cy))
         x \ y \ (null \ ())
    let neg x =
       sub\ (null\ ())\ x
    let neg x =
       scale (C.neg C.unit) x
Multiply the derivatives by c and add the result to dx.
    let add_derivatives derivatives c dx =
       List.fold\_left (fun acc (df, dt\_c, dt\_t) \rightarrow
          add (mul df (power dt_t (C.mul c (C.make dt_c 1)))) acc) dx derivatives
    let derive\_inner derive1 x =
       M.fold (fun t \rightarrow
          add\_derivatives\ (T.derive\ (fun\ f\ \to\ Some\ (derive1\ f))\ t))\ x\ (null\ ())
    let derive\_inner' derive1 x =
       M.fold (fun t \rightarrow add\_derivatives (T.derive\ derive1\ t)) x\ (null\ ())
    let collect\_derivatives\ derivatives\ c\ dx\ =
       List.fold\_left (fun acc (df, dt\_c, dt\_t) \rightarrow
          (df, power dt_t (C.mul c (C.make dt_c 1))) :: acc) dx derivatives
    let derive\_outer \ derive1 \ x =
       M.fold (fun t \rightarrow collect\_derivatives (T.derive\ derive1\ t)) x []
    let sum terms =
       List.fold_left add (null ()) terms
    let product factors =
       List.fold_left mul (unit ()) factors
    let atoms t =
       ThoList.uniq (List.sort compare
                           (ThoList.flatmap\ (fun\ (t, \_) \rightarrow T.atoms\ t)\ (PM.elements\ t)))
    let to\_string fmt sum =
       "(" \hat{String.concat} "_{\sqcup}+_{\sqcup}"
                 (M.fold (fun \ t \ c \ acc \rightarrow
                   if C.is\_null\ c then
                      acc
                   else if C.is\_unit\ c then
                      T.to\_string\ fmt\ t\ ::\ acc
                   else if C.is\_unit (C.neg c) then
                      ("(-"^T.to\_string fmt t^")") :: acc
                      (C.to\_string\ c\ ^"*["\ ^T.to\_string\ fmt\ t\ ^"]")\ ::\ acc)\ sum\ [])\ ^")"
  end
module Make\_Linear (C:Ring): Linear with module C=C=
  struct
```

```
\mathsf{module}\ C\ =\ C
  module M = PM
  type (\alpha, \gamma) t = (\alpha, \gamma C.t) M.t
  let null() = M.empty
  let is\_null = M.is\_empty
  let atom \ a = M.singleton \ a \ (C.unit \ ())
  let singleton \ c \ a = M.singleton \ a \ c
  let scale \ c \ x = M.map \ (C.mul \ c) \ x
  let insert1 binop t c sum =
     let c' = binop \text{ (try } M.find \ compare \ t \ sum \ with \ Not\_found \rightarrow C.null \text{ ())} \ c \ in
     if C.is\_null\ c' then
        M.remove\ compare\ t\ sum
     else
        M.add compare t c' sum
  let add x y = M.fold (insert1 C.add) x y
  let \ sub \ x \ y \ = \ M.fold \ (insert1 \ C.sub) \ y \ x
  let map f t =
     M.fold (fun a \ c \rightarrow add \ (f \ a \ c)) t \ M.empty
  let sum terms =
     List.fold_left add (null ()) terms
  let \ linear \ terms =
     List.fold\_left (fun acc (a, c) \rightarrow add (scale c a) acc) (null ()) terms
  let partial derive t =
     let d t' =
       let dt' = derive t' in
       if is\_null \ dt' then
          None
        else
          Some dt' in
     linear (C.derive\_outer \ d \ t)
     let a, c = List.split (PM.elements t) in
     (a, ThoList.uniq (List.sort compare (ThoList.flatmap C.atoms c)))
  let to\_string fmt \ cfmt \ sum =
     "(" \hat{String.concat} "_{\sqcup}+_{\sqcup}"
               (M.fold (fun \ t \ c \ acc \rightarrow
                  if C.is\_null\ c then
                  else if C.is\_unit c then
                    fmt\ t\ ::\ acc
                  else if C.is\_unit (C.neg\ c) then
                     ("(-"^fmt t^")") :: acc
                     (C.to\_string\ cfmt\ c\ ^ "*"\ ^ fmt\ t)\ ::\ acc)
                  sum []) ^ ")"
end
```

—V— SIMPLE LINEAR ALGEBRA

V.1 Interface of Linalg

```
exception Singular exception Not\_Square val copy\_matrix: float\ array\ array \rightarrow float\ array\ array \rightarrow float\ array\ array val matmulv: float\ array\ array \rightarrow float\ array \rightarrow float\ array val lu\_decompose: float\ array\ array \rightarrow float\ array\ array \rightarrow float\ array val solve: float\ array\ array \rightarrow float\ array float\ array val solve\_many: float\ array\ array \rightarrow float\ array\ list \rightarrow float\ array\ list
```

V.2 Implementation of Linalg

This is not a functional implementations, but uses imperative array in Fotran style for maximimum speed.

```
exception Singular
exception Not\_Square
let copy\_matrix a =
  Array.init (Array.length a)
    (fun i \rightarrow Array.copy a.(i))
let matmul \ a \ b =
  let ni = Array.length a
  and nj = Array.length b.(0)
  and n = Array.length b in
  let ab = Array.make\_matrix \ ni \ nj \ 0.0 \ in
  for i = 0 to pred ni do
    for j = 0 to pred nj do
       for k = 0 to pred n do
         ab.(i).(j) \leftarrow ab.(i).(j) + .a.(i).(k) * .b.(k).(j)
       done
    done
  done;
  ab
let matmulv \ a \ v =
  let na = Array.length a in
  let nv = Array.length v in
  let v' = Array.make na 0.0 in
  for i = 0 to pred na do
    for j = 0 to pred nv do
       v'.(i) \leftarrow v'.(i) + . a.(i).(j) * . v.(j)
    done
  done:
  v'
\mathsf{let}\ maxabsval\ a\ :\ float =
```

```
\begin{array}{lll} \text{let } x &=& ref \ (abs\_float \ a.(0)) \ \text{in} \\ \text{for } i &=& 1 \ \text{to} \ Array.length \ a \ - \ 1 \ \text{do} \\ x &:=& max \ !x \ (abs\_float \ a.(i)) \\ \text{done;} \\ !x \end{array}
```

$$A = LU (V.1a)$$

In more detail

$$\begin{pmatrix} a_{00} & a_{01} & \dots & a_{0(n-1)} \\ a_{10} & a_{11} & \dots & a_{1(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{(n-1)0} & a_{(n-1)1} & \dots & a_{(n-1)(n-1)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ l_{10} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ l_{(n-1)0} & l_{(n-1)1} & \dots & 1 \end{pmatrix} \begin{pmatrix} u_{00} & u_{01} & \dots & u_{0(n-1)} \\ 0 & u_{11} & \dots & u_{1(n-1)} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & u_{(n-1)(n-1)} \end{pmatrix} \quad (V.1b)$$

Rewriting (V.1) in block matrix notation

$$\begin{pmatrix} a_{00} & a_{0.} \\ a_{.0} & A \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ l_{.0} & L \end{pmatrix} \begin{pmatrix} u_{00} & u_{0.} \\ 0 & U \end{pmatrix} = \begin{pmatrix} u_{00} & u_{0.} \\ l_{.0}u_{00} & l_{.0} \otimes u_{0.} + LU \end{pmatrix}$$
(V.2)

we can solve it easily

$$u_{00} = a_{00}$$
 (V.3a)

$$u_{0.} = a_{0.}$$
 (V.3b)

$$l._0 = \frac{a._0}{a_{00}} \tag{V.3c}$$

$$LU = A - \frac{a_{.0} \otimes a_{0.}}{a_{00}} \tag{V.3d}$$

and (V.3c) and (V.3d) define a simple iterative algorithm if we work from the outside in. It just remains to add pivoting.

```
let swap a i j =
  let a_-i = a.(i) in
  a.(i) \leftarrow a.(j);
  a.(j) \leftarrow a_{-}i
let pivot\_column \ v \ a \ n =
  \mathsf{let}\ n'\ =\ \mathit{ref}\ n
  and max\_va = ref(v.(n) *. (abs\_float a.(n).(n))) in
  for i = succ \ n to Array.length \ v - 1 do
    let va_i = v(i) * (abs_float a(i)(n)) in
    if va_i > !max_va then begin
       n' := i;
       max\_va := va\_i
    end
  done;
  !n'
let lu\_decompose\_in\_place \ a =
  let n = Array.length a in
  let eps = ref 1
  and pivots = Array.make n 0
  and v \; = \;
    try
       Array.init n (fun i \rightarrow
```

```
let a_i = a(i) in
          if Array.length \ a_{-}i \neq n then
            raise Not_Square;
          1.0 /. (maxabsval \ a_{-}i))
     with
     | Division\_by\_zero \rightarrow raise Singular in
  for i = 0 to pred n do
     \mathsf{let}\ pivot\ =\ pivot\_column\ v\ a\ i\ \mathsf{in}
     if pivot \neq i then begin
       swap a pivot i;
        eps := -!eps;
       v.(pivot) \leftarrow v.(i)
     end;
     pivots.(i) \leftarrow pivot;
     let inv_a_i =
       try 1.0 /. a.(i).(i) with Division\_by\_zero \rightarrow raise Singular in
     for j = succ i to pred n do
       a.(j).(i) \leftarrow inv_a_i i *. a.(j).(i)
     done;
     for j = succ i to pred n do
       for k = succ i to pred n do
          a.(j).(k) \ \leftarrow \ a.(j).(k) \ - . \ a.(j).(i) \ * . \ a.(i).(k)
       done
     done
  done;
  (pivots, !eps)
let lu\_decompose\_split \ a \ pivots =
  let n = Array.length pivots in
  let l = Array.make\_matrix \ n \ n \ 0.0 in
  let u = Array.make\_matrix \ n \ n \ 0.0 in
  for i = 0 to pred n do
     l.(i).(i) \leftarrow 1.0;
     for j = succ i to pred n do
       l.(j).(i) \leftarrow a.(j).(i)
     done
  done;
  for i = pred n downto 0 do
     swap \ l \ i \ pivots.(i)
  done;
  for i = 0 to pred n do
     for j = 0 to i do
       u.(j).(i) \leftarrow a.(j).(i)
     done
  done;
  (l, u)
let lu\_decompose \ a =
  let a = copy\_matrix a in
  let pivots, \_ = lu\_decompose\_in\_place a in
  lu\_decompose\_split\ a\ pivots
let lu\_backsubstitute a pivots b =
  let n = Array.length a in
  let nonzero = ref (-1) in
  let b = Array.copy b in
  for i = 0 to pred n do
     let ll = pivots.(i) in
     let b_i = ref(b(ll)) in
     b.(ll) \leftarrow b.(i);
     if !nonzero \ge 0 then
       for j = !nonzero to pred i do
          b_{-i} := !b_{-i} - . a.(i).(j) * . b.(j)
```

```
done
      else if !b_{-}i \neq 0.0 then
         nonzero := i;
      b.(i) \leftarrow !b_i
   done;
   \quad \text{for } i \ = \ pred \ n \ \mathsf{downto} \ 0 \ \mathsf{do}
      let b_i = ref(b(i)) in
      \text{ for } j \ = \ succ \ i \ \text{to} \ pred \ n \ \text{do}
         b_{-}i := !b_{-}i - . a.(i).(j) * . b.(j)
      b.(i) \leftarrow !b_i /. a.(i).(i)
   done;
   b
\mathsf{let}\ solve\_destructive\ a\ b\ =
   \mathsf{let}\ \mathit{pivot},\ \_\ =\ \mathit{lu\_decompose\_in\_place}\ \mathit{a}\ \mathsf{in}
   lu\_backsubstitute a pivot b
let solve\_many\_destructive \ a \ bs =
   \mathsf{let}\ pivot,\ \_\ =\ lu\_decompose\_in\_place\ a\ \mathsf{in}
   List.map (lu_backsubstitute a pivot) bs
let solve \ a \ b =
   solve\_destructive\ (copy\_matrix\ a)\ b
let solve\_many a bs =
   solve\_many\_destructive\ (copy\_matrix\ a)\ bs
```

—W— Partial Maps

W.1 Interface of Partial

Partial maps that are constructed from assoc lists.

```
\begin{array}{ccc} \mathsf{module} \ \mathsf{type} \ T \ = \\ \mathsf{sig} \end{array}
```

The domain of the map. It needs to be compatible with Map. Ordered Type.t

type domain

The codomain α can be anything we want.

```
type \alpha t
```

A list of argument-value pairs is mapped to a partial map. If an argument appears twice, the later value takes precedence.

```
val of_list : (domain \times \alpha) list \rightarrow \alpha t
```

Two lists of arguments and values (both must have the same length) are mapped to a partial map. Again the later value takes precedence.

```
val of_lists : domain list \rightarrow \alpha list \rightarrow \alpha t
```

If domain and codomain disagree, we must raise an exception or provide a fallback.

```
exception Undefined of domain val apply: \alpha t \rightarrow domain \rightarrow \alpha val apply\_with\_fallback: <math>(domain \rightarrow \alpha) \rightarrow \alpha t \rightarrow domain \rightarrow \alpha
```

Iff domain and codomain of the map agree, we can fall back to the identity map.

```
\mbox{ val } auto \ : \ domain \ t \ \to \ domain \ \to \ domain \mbox{ end} \mbox{module } Make \ : \ \mbox{functor } (D \ : \ Map.OrderedType) \ \to \ T \ \mbox{with type } domain \ = \ D.t \mbox{module } Test \ : \ \mbox{sig val } suite \ : \ OUnit.test \ \mbox{end}
```

W.2 Implementation of Partial

```
module type T= sig  \text{type } domain \\ \text{type } \alpha \ t \\ \text{val } of\_list: (domain \times \alpha) \ list \rightarrow \alpha \ t \\ \text{val } of\_lists: domain \ list \rightarrow \alpha \ list \rightarrow \alpha \ t \\ \text{exception } Undefined \ \text{of } domain \\ \text{val } apply: \alpha \ t \rightarrow domain \rightarrow \alpha \\ \text{val } apply\_with\_fallback: (domain \rightarrow \alpha) \rightarrow \alpha \ t \rightarrow domain \rightarrow \alpha \\ \text{val } auto: domain \ t \rightarrow domain \rightarrow domain \\ \text{end} \\ \\ \text{module } Make \ (D: Map.OrderedType): T \ \text{with type } domain = D.t = 0
```

```
struct
     \mathsf{module}\ M\ =\ Map.Make\ (D)
    type domain = D.t
     type \alpha t = \alpha M.t
    let of_list l =
       List.fold\_left (fun m (d, v) \rightarrow M.add d v m) M.empty <math>l
    let of_lists domain values =
       of\_list
          (try
              List.map2 (fun d \ v \rightarrow (d, \ v)) domain \ values
           | Invalid\_argument\_(* "List.map2" *) \rightarrow
               invalid_arg "Partial.of_lists:_length_mismatch")
    let auto partial d =
       try
          M.find d partial
       with
       | Not\_found \rightarrow d
    exception Undefined of domain
    \mathsf{let} \ \mathit{apply} \ \mathit{partial} \ d \ = \\
       try
          M.find d partial
       with
       | Not\_found \rightarrow raise (Undefined d)
    let apply\_with\_fallback fallback partial d =
       try
          M.find d partial
       with
       \mid Not\_found \rightarrow fallback d
  end
                                                 W.2.1
                                                          Unit Tests
module Test : sig val suite : OUnit.test end =
  struct
     open OUnit
     module P = Make (struct type t = int let compare = compare end)
    let apply_ok =
       "apply/ok" >::
          (fun () \rightarrow
            let p = P.of\_list [ (0,"a"); (1,"b"); (2,"c") ]
            and l = [0; 1; 2] in
            assert_equal [ "a"; "b"; "c" ] (List.map (P.apply p) l))
    let apply_ok2 =
       "apply/ok2" >::
          (fun () \rightarrow
            let p = P.of\_lists [0; 1; 2] ["a"; "b"; "c"]
            and l = [0; 1; 2] in
            assert\_equal [ "a"; "b"; "c" ] (List.map (P.apply p) l))
    let apply\_shadowed =
       "apply/shadowed" >::
          (fun () \rightarrow
            let p = P.of\_list [ (0,"a"); (1,"b"); (2,"c"); (1,"d") ]
```

```
and l = [0; 1; 2] in
          assert_equal [ "a"; "d"; "c" ] (List.map (P.apply p) l))
  let apply\_shadowed2 =
     "apply/shadowed2" >::
       (fun () \rightarrow
         \mathsf{let}\ p\ =\ P.of\_lists\ [0;\ 1;\ 2;\ 1]\ ["a";\ "b";\ "c";\ "d"]
          and l = [0; 1; 2] in
          assert\_equal [ "a"; "d"; "c" ] (List.map (P.apply p) l))
  let apply\_mismatch =
     "apply/mismatch" >::
       (\mathsf{fun}\ ()\ \to
          assert\_raises
            (Invalid_argument "Partial.of_lists: ulength_mismatch")
            (fun () \rightarrow P.of\_lists [0; 1; 2] ["a"; "b"; "c"; "d"]))
  let suite\_apply =
     "apply" >:::
       [apply\_ok;
        apply\_ok2;
        apply\_shadowed;
        apply\_shadowed2;
        apply\_mismatch]
  let auto\_ok =
     "auto/ok" >::
       (fun () \rightarrow
         let p = P.of\_list [(0, 10); (1, 11)]
         and l = [0; 1; 2] in
          assert\_equal [10; 11; 2] (List.map (P.auto p) l))
  \mathsf{let} \ \mathit{suite\_auto} \ = \\
     "auto" >:::
       [auto\_ok]
  let apply_with_fallback_ok =
     "apply_with_fallback/ok" >::
       (fun () \rightarrow
         let p = P.of\_list [(0, 10); (1, 11)]
         and l = [\ 0;\ 1;\ 2\ ] in
          assert\_equal
            [10; 11; -2] (List.map (P.apply_with_fallback (fun n \rightarrow -n) p) l))
  let \ suite\_apply\_with\_fallback \ =
     "apply_with_fallback" >:::
       [apply\_with\_fallback\_ok]
  let suite =
     "Partial" >:::
       [suite\_apply;
        suite\_auto;
        suite\_apply\_with\_fallback]
  let time() =
     ()
end
```

—X— TALK TO THE WHIZARD ...

Talk to [11].



Temporarily disabled, until, we implement some conditional weaving...

—Y— FORTRAN LIBRARIES

Y.1 Trivia

```
\langle omega\_spinors.f90 \rangle \equiv
   \langle Copyleft \rangle
  module omega_spinors
     use kinds
     use constants
     implicit none
     private
     public :: operator (*), operator (+), operator (-)
     public :: abs, set_zero
     \langle \mathtt{intrinsic} :: \mathtt{abs} \rangle
     type, public :: conjspinor
         ! private (omegalib needs access, but DON'T TOUCH IT!)
         complex(kind=default), dimension(4) :: a
     end type conjspinor
     type, public :: spinor
         ! private (omegalib needs access, but DON'T TOUCH IT!)
         complex(kind=default), dimension(4) :: a
     end type spinor
     ⟨Declaration of operations for spinors⟩
     integer, parameter, public :: omega_spinors_2010_01_A = 0
   contains
     ⟨Implementation of operations for spinors⟩
  end module omega_spinors
\langle intrinsic :: abs (if working) \rangle \equiv
  intrinsic :: abs
\langle intrinsic :: conjg (if working) \rangle \equiv
  intrinsic :: conjg
well, the Intel Fortran Compiler chokes on these with an internal error:
\langle \mathtt{intrinsic} :: \mathtt{abs} \rangle \equiv
\langle \mathtt{intrinsic} :: \mathtt{conjg} \rangle \equiv
To reenable the pure functions that have been removed for OpenMP, one should set this chunk to pure &
\langle pure \ unless \ OpenMP \rangle \equiv
                                                  Y.1.1 Inner Product
\langle Declaration \ of \ operations \ for \ spinors \rangle \equiv
  interface operator (*)
      module procedure conjspinor_spinor
  end interface
  private :: conjspinor_spinor
                                                              \bar{\psi}\psi'
                                                                                                                           (Y.1)
```

NB: dot_product conjugates its first argument, we can either cancel this or inline dot_product: $\langle Implementation\ of\ operations\ for\ spinors\rangle \equiv$

```
pure function conjspinor_spinor (psibar, psi) result (psibarpsi)
    complex(kind=default) :: psibarpsi
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    psibarpsi = psibar%a(1)*psi%a(1) + psibar%a(2)*psi%a(2) &
               + psibar%a(3)*psi%a(3) + psibar%a(4)*psi%a(4)
  end function conjspinor_spinor
                                        Y.1.2 Spinor Vector Space
\langle Declaration \ of \ operations \ for \ spinors \rangle + \equiv
  interface set_zero
    module procedure set_zero_spinor, set_zero_conjspinor
  end interface
 private :: set_zero_spinor, set_zero_conjspinor
\langle Implementation \ of \ operations \ for \ spinors \rangle + \equiv
  elemental subroutine set_zero_spinor (x)
    type(spinor), intent(out) :: x
    x\%a = 0
  end subroutine set_zero_spinor
\langle Implementation \ of \ operations \ for \ spinors \rangle + \equiv
  elemental subroutine set_zero_conjspinor (x)
    type(conjspinor), intent(out) :: x
    x\%a = 0
  end subroutine set_zero_conjspinor
                                              Scalar Multiplication
\langle Declaration \ of \ operations \ for \ spinors \rangle + \equiv
  interface operator (*)
     module procedure integer_spinor, spinor_integer, &
          real_spinor, double_spinor, &
           complex_spinor, dcomplex_spinor, &
           spinor_real, spinor_double, &
           spinor_complex, spinor_dcomplex
  end interface
 private :: integer_spinor, spinor_integer, real_spinor, &
       double_spinor, complex_spinor, dcomplex_spinor, &
       spinor_real, spinor_double, spinor_complex, spinor_dcomplex
\langle Implementation \ of \ operations \ for \ spinors \rangle + \equiv
 pure function integer_spinor (x, y) result (xy)
    integer, intent(in) :: x
    type(spinor), intent(in) :: y
    type(spinor) :: xy
    xy\%a = x * y\%a
  end function integer_spinor
\langle Implementation of operations for spinors \rangle + \equiv
 pure function real_spinor (x, y) result (xy)
    real(kind=single), intent(in) :: x
    type(spinor), intent(in) :: y
    type(spinor) :: xy
    xy\%a = x * y\%a
  end function real_spinor
  pure function double_spinor (x, y) result (xy)
    real(kind=default), intent(in) :: x
    type(spinor), intent(in) :: y
    type(spinor) :: xy
    xy\%a = x * y\%a
  end function double_spinor
  pure function complex_spinor (x, y) result (xy)
    complex(kind=single), intent(in) :: x
    type(spinor), intent(in) :: y
    type(spinor) :: xy
```

```
xy\%a = x * y\%a
 end function complex_spinor
 pure function dcomplex_spinor (x, y) result (xy)
   complex(kind=default), intent(in) :: x
   type(spinor), intent(in) :: y
   type(spinor) :: xy
   xy\%a = x * y\%a
 end function dcomplex_spinor
 pure function spinor_integer (y, x) result (xy)
   integer, intent(in) :: x
    type(spinor), intent(in) :: y
   type(spinor) :: xy
   xy\%a = x * y\%a
 end function spinor_integer
 pure function spinor_real (y, x) result (xy)
   real(kind=single), intent(in) :: x
   type(spinor), intent(in) :: y
   type(spinor) :: xy
   xy\%a = x * y\%a
 end function spinor_real
 pure function spinor_double (y, x) result (xy)
   real(kind=default), intent(in) :: x
   type(spinor), intent(in) :: y
   type(spinor) :: xy
   xy\%a = x * y\%a
  end function spinor_double
 pure function spinor_complex (y, x) result (xy)
   complex(kind=single), intent(in) :: x
   type(spinor), intent(in) :: y
   type(spinor) :: xy
   xy\%a = x * y\%a
  end function spinor_complex
 pure function spinor_dcomplex (y, x) result (xy)
   complex(kind=default), intent(in) :: x
   type(spinor), intent(in) :: y
   type(spinor) :: xy
   xy\%a = x * y\%a
  end function spinor_dcomplex
\langle Declaration \ of \ operations \ for \ spinors \rangle + \equiv
 interface operator (*)
    module procedure integer_conjspinor, conjspinor_integer, &
          real_conjspinor, double_conjspinor, &
          complex_conjspinor, dcomplex_conjspinor, &
          conjspinor_real, conjspinor_double, &
          conjspinor_complex, conjspinor_dcomplex
 end interface
 private :: integer_conjspinor, conjspinor_integer, real_conjspinor, &
       double_conjspinor, complex_conjspinor, dcomplex_conjspinor, &
       conjspinor_real, conjspinor_double, conjspinor_complex, &
       conjspinor_dcomplex
\langle Implementation of operations for spinors \rangle + \equiv
 pure function integer_conjspinor (x, y) result (xy)
   integer, intent(in) :: x
    type(conjspinor), intent(in) :: y
   type(conjspinor) :: xy
   xy\%a = x * y\%a
  end function integer_conjspinor
 pure function real_conjspinor (x, y) result (xy)
   real(kind=single), intent(in) :: x
   type(conjspinor), intent(in) :: y
   type(conjspinor) :: xy
   xy\%a = x * y\%a
 end function real_conjspinor
 pure function double_conjspinor (x, y) result (xy)
   real(kind=default), intent(in) :: x
```

```
type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy\%a = x * y\%a
  end function double_conjspinor
 pure function complex_conjspinor (x, y) result (xy)
    complex(kind=single), intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy\%a = x * y\%a
  end function complex_conjspinor
  pure function dcomplex_conjspinor (x, y) result (xy)
    complex(kind=default), intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy\%a = x * y\%a
  end function dcomplex_conjspinor
 pure function conjspinor_integer (y, x) result (xy)
    integer, intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy\%a = x * y\%a
  end function conjspinor_integer
  pure function conjspinor_real (y, x) result (xy)
    real(kind=single), intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy\%a = x * y\%a
  end function conjspinor_real
 pure function conjspinor_double (y, x) result (xy)
    real(kind=default), intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy\%a = x * y\%a
  end function conjspinor_double
 pure function conjspinor_complex (y, x) result (xy)
    complex(kind=single), intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy\%a = x * y\%a
  end function conjspinor_complex
 pure function conjspinor_dcomplex (y, x) result (xy)
    complex(kind=default), intent(in) :: x
    type(conjspinor), intent(in) :: y
    type(conjspinor) :: xy
    xy\%a = x * y\%a
  end function conjspinor_dcomplex
                                          Unary Plus and Minus
\langle Declaration \ of \ operations \ for \ spinors \rangle + \equiv
  interface operator (+)
     module procedure plus_spinor, plus_conjspinor
  end interface
 private :: plus_spinor, plus_conjspinor
  interface operator (-)
     module procedure neg_spinor, neg_conjspinor
  end interface
 private :: neg_spinor, neg_conjspinor
\langle Implementation \ of \ operations \ for \ spinors \rangle + \equiv
 pure function plus_spinor (x) result (plus_x)
    type(spinor), intent(in) :: x
    type(spinor) :: plus_x
    plus_x%a = x%a
  end function plus_spinor
 pure function neg_spinor (x) result (neg_x)
```

```
type(spinor), intent(in) :: x
    type(spinor) :: neg_x
    neg_x%a = - x%a
  end function neg_spinor
\langle Implementation \ of \ operations \ for \ spinors \rangle + \equiv
 pure function plus_conjspinor (x) result (plus_x)
    type(conjspinor), intent(in) :: x
    type(conjspinor) :: plus_x
    plus_x%a = x%a
  end function plus_conjspinor
  pure function neg_conjspinor (x) result (neg_x)
    type(conjspinor), intent(in) :: x
    type(conjspinor) :: neg_x
    neg_x%a = - x%a
  end function neg_conjspinor
                                            Addition and Subtraction
\langle Declaration \ of \ operations \ for \ spinors \rangle + \equiv
  interface operator (+)
     module procedure add_spinor, add_conjspinor
  end interface
  private :: add_spinor, add_conjspinor
  interface operator (-)
     module procedure sub_spinor, sub_conjspinor
  end interface
 private :: sub_spinor, sub_conjspinor
\langle Implementation \ of \ operations \ for \ spinors \rangle + \equiv
 pure function add_spinor (x, y) result (xy)
    type(spinor), intent(in) :: x, y
    type(spinor) :: xy
    xy\%a = x\%a + y\%a
  end function add_spinor
 pure function sub_spinor (x, y) result (xy)
    type(spinor), intent(in) :: x, y
    type(spinor) :: xy
    xy\%a = x\%a - y\%a
  end function sub_spinor
\langle Implementation \ of \ operations \ for \ spinors \rangle + \equiv
 pure function add_conjspinor (x, y) result (xy)
    type(conjspinor), intent(in) :: x, y
    type(conjspinor) :: xy
    xy\%a = x\%a + y\%a
  end function add_conjspinor
  pure function sub_conjspinor (x, y) result (xy)
    type(conjspinor), intent(in) :: x, y
    type(conjspinor) :: xy
    xy\%a = x\%a - y\%a
  end function sub_conjspinor
                                                  Y.1.3 Norm
\langle Declaration \ of \ operations \ for \ spinors \rangle + \equiv
  interface abs
     module procedure abs_spinor, abs_conjspinor
  end interface
 private :: abs_spinor, abs_conjspinor
\langle Implementation \ of \ operations \ for \ spinors \rangle + \equiv
 pure function abs_spinor (psi) result (x)
    type(spinor), intent(in) :: psi
    real(kind=default) :: x
    x = sqrt (real (dot_product (psi%a, psi%a)))
  end function abs_spinor
```

```
\langle Implementation \ of \ operations \ for \ spinors \rangle + \equiv
  pure function abs_conjspinor (psibar) result (x)
    real(kind=default) :: x
    type(conjspinor), intent(in) :: psibar
    x = sqrt (real (dot_product (psibar%a, psibar%a)))
  end function abs_conjspinor
                                          Y.2 Spinors Revisited
\langle \mathtt{omega\_bispinors.f90} \rangle \equiv
  \langle Copyleft \rangle
  module omega_bispinors
    use kinds
    use constants
    implicit none
    private
    public :: operator (*), operator (+), operator (-)
    public :: abs, set_zero
    type, public :: bispinor
        ! private (omegalib needs access, but DON'T TOUCH IT!)
        {\tt complex(kind=default),\ dimension(4)\ ::\ a}
    end type bispinor
    ⟨Declaration of operations for bispinors⟩
    integer, parameter, public :: omega_bispinors_2010_01_A = 0
  contains
     (Implementation of operations for bispinors)
  end module omega_bispinors
\langle Declaration \ of \ operations \ for \ bispinors \rangle \equiv
  interface operator (*)
    module procedure spinor_product
  end interface
  private :: spinor_product
                                                         \bar{\psi}\psi'
                                                                                                                 (Y.2)
NB: dot_product conjugates its first argument, we have to cancel this.
\langle Implementation \ of \ operations \ for \ bispinors \rangle \equiv
  pure function spinor_product (psil, psir) result (psilpsir)
    complex(kind=default) :: psilpsir
     type(bispinor), intent(in) :: psil, psir
    type(bispinor) :: psidum
    psidum%a(1) = psir%a(2)
    psidum%a(2) = - psir%a(1)
    psidum%a(3) = - psir%a(4)
    psidum%a(4) = psir%a(3)
    psilpsir = dot_product (conjg (psil%a), psidum%a)
  end function spinor_product
                                          Y.2.1 Spinor Vector Space
\langle Declaration \ of \ operations \ for \ bispinors \rangle + \equiv
  interface set_zero
    module procedure set_zero_bispinor
  end interface
  private :: set_zero_bispinor
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  elemental subroutine set_zero_bispinor (x)
    type(bispinor), intent(out) :: x
    x\%a = 0
  end subroutine set_zero_bispinor
```

Scalar Multiplication

```
\langle Declaration \ of \ operations \ for \ bispinors \rangle + \equiv
  interface operator (*)
     module procedure integer_bispinor, bispinor_integer, &
             real_bispinor, double_bispinor, &
             complex_bispinor, dcomplex_bispinor, &
             bispinor_real, bispinor_double, &
             bispinor_complex, bispinor_dcomplex
  end interface
  private :: integer_bispinor, bispinor_integer, real_bispinor, &
       double_bispinor, complex_bispinor, dcomplex_bispinor, &
       bispinor_real, bispinor_double, bispinor_complex, bispinor_dcomplex
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  pure function integer_bispinor (x, y) result (xy)
    type(bispinor) :: xy
    integer, intent(in) :: x
    type(bispinor), intent(in) :: y
    xy\%a = x * y\%a
  end function integer_bispinor
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  pure function real_bispinor (x, y) result (xy)
    type(bispinor) :: xy
    real(kind=single), intent(in) :: x
    type(bispinor), intent(in) :: y
    xy\%a = x * y\%a
  end function real_bispinor
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  pure function double_bispinor (x, y) result (xy)
    type(bispinor) :: xy
    real(kind=default), intent(in) :: x
    type(bispinor), intent(in) :: y
    xy\%a = x * y\%a
  end function double_bispinor
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  pure function complex_bispinor (x, y) result (xy)
    type(bispinor) :: xy
    complex(kind=single), intent(in) :: x
    type(bispinor), intent(in) :: y
    xy\%a = x * y\%a
  end function complex_bispinor
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  pure function dcomplex_bispinor (x, y) result (xy)
    type(bispinor) :: xy
    complex(kind=default), intent(in) :: x
    type(bispinor), intent(in) :: y
    xy\%a = x * y\%a
  end function dcomplex_bispinor
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  pure function bispinor_integer (y, x) result (xy)
    type(bispinor) :: xy
    integer, intent(in) :: x
    type(bispinor), intent(in) :: y
    xy\%a = x * y\%a
  end function bispinor_integer
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  pure function bispinor_real (y, x) result (xy)
    type(bispinor) :: xy
    real(kind=single), intent(in) :: x
    type(bispinor), intent(in) :: y
    xy\%a = x * y\%a
  end function bispinor_real
```

```
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  pure function bispinor_double (y, x) result (xy)
    type(bispinor) :: xy
    real(kind=default), intent(in) :: x
    type(bispinor), intent(in) :: y
    xy\%a = x * y\%a
  end function bispinor_double
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  pure function bispinor_complex (y, x) result (xy)
    type(bispinor) :: xy
    complex(kind=single), intent(in) :: x
    type(bispinor), intent(in) :: y
    xy\%a = x * y\%a
  end function bispinor_complex
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  pure function bispinor_dcomplex (y, x) result (xy)
    type(bispinor) :: xy
    complex(kind=default), intent(in) :: x
    type(bispinor), intent(in) :: y
    xy\%a = x * y\%a
  end function bispinor_dcomplex
                                               Unary Plus and Minus
\langle Declaration\ of\ operations\ for\ bispinors \rangle + \equiv
  interface operator (+)
     module procedure plus_bispinor
  end interface
  private :: plus_bispinor
  interface operator (-)
     module procedure neg_bispinor
  end interface
  private :: neg_bispinor
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  pure function plus_bispinor (x) result (plus_x)
    type(bispinor) :: plus_x
    type(bispinor), intent(in) :: x
    plus_x\%a = x\%a
  end function plus_bispinor
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  pure function neg_bispinor (x) result (neg_x)
    type(bispinor) :: neg_x
    type(bispinor), intent(in) :: x
    neg_x%a = - x%a
  end function neg_bispinor
                                              Addition and Subtraction
\langle Declaration \ of \ operations \ for \ bispinors \rangle + \equiv
  interface operator (+)
     module procedure add_bispinor
  end interface
  private :: add_bispinor
  interface operator (-)
     module procedure sub_bispinor
  end interface
  private :: sub_bispinor
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  pure function add_bispinor (x, y) result (xy)
    type(bispinor) :: xy
    type(bispinor), intent(in) :: x, y
    xy\%a = x\%a + y\%a
  end function add_bispinor
```

```
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  pure function sub_bispinor (x, y) result (xy)
    type(bispinor) :: xy
    type(bispinor), intent(in) :: x, y
    xy\%a = x\%a - y\%a
  end function sub_bispinor
                                                    Y.2.2 Norm
\langle Declaration \ of \ operations \ for \ bispinors \rangle + \equiv
  interface abs
     module procedure abs_bispinor
  end interface
  private :: abs_bispinor
\langle Implementation \ of \ operations \ for \ bispinors \rangle + \equiv
  pure function abs_bispinor (psi) result (x)
    real(kind=default) :: x
    type(bispinor), intent(in) :: psi
    x = sqrt (real (dot_product (psi%a, psi%a)))
  end function abs_bispinor
                                             Y.3 Vectorspinors
\langle {\tt omega\_vectorspinors.f90} \rangle \equiv
  \langle Copyleft \rangle
  module omega_vectorspinors
    use kinds
    use constants
    use omega_bispinors
    use omega_vectors
    implicit none
    private
    public :: operator (*), operator (+), operator (-)
    public :: abs, set_zero
    type, public :: vectorspinor
        ! private (omegalib needs access, but DON'T TOUCH IT!)
        type(bispinor), dimension(4) :: psi
    end type vectorspinor
    \langle Declaration \ of \ operations \ for \ vectorspinors \rangle
    integer, parameter, public :: omega_vectorspinors_2010_01_A = 0
  contains
    ⟨Implementation of operations for vectorspinors⟩
  end module omega_vectorspinors
\langle Declaration \ of \ operations \ for \ vectorspinors \rangle \equiv
  interface operator (*)
    module procedure vspinor_product
  end interface
  private :: vspinor_product
                                                         \bar{\psi}^{\mu}\psi'_{\mu}
                                                                                                                   (Y.3)
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle \equiv
  pure function vspinor_product (psil, psir) result (psilpsir)
    complex(kind=default) :: psilpsir
    type(vectorspinor), intent(in) :: psil, psir
    psilpsir = psil%psi(1) * psir%psi(1) &
               - psil%psi(2) * psir%psi(2) &
               - psil%psi(3) * psir%psi(3) &
               - psil%psi(4) * psir%psi(4)
  end function vspinor_product
```

Y.3.1 Vectorspinor Vector Space

```
\langle Declaration \ of \ operations \ for \ vectorspinors \rangle + \equiv
  interface set_zero
    module procedure set_zero_vectorspinor
  end interface
  private :: set_zero_vectorspinor
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
  elemental subroutine set_zero_vectorspinor (x)
    type(vectorspinor), intent(out) :: x
    call set_zero (x%psi)
  end subroutine set_zero_vectorspinor
                                               Scalar Multiplication
\langle Declaration \ of \ operations \ for \ vectorspinors \rangle + \equiv
  interface operator (*)
     module procedure integer_vectorspinor, vectorspinor_integer, &
             real_vectorspinor, double_vectorspinor, &
             complex_vectorspinor, dcomplex_vectorspinor, &
             vectorspinor_real, vectorspinor_double, &
             vectorspinor_complex, vectorspinor_dcomplex, &
             momentum_vectorspinor, vectorspinor_momentum
  end interface
  private :: integer_vectorspinor, vectorspinor_integer, real_vectorspinor, &
       double_vectorspinor, complex_vectorspinor, dcomplex_vectorspinor, &
        vectorspinor_real, vectorspinor_double, vectorspinor_complex, &
       vectorspinor_dcomplex
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
  pure function integer_vectorspinor (x, y) result (xy)
    type(vectorspinor) :: xy
    integer, intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
      xy\%psi(k) = x * y\%psi(k)
    end do
  end function integer_vectorspinor
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
  pure function real_vectorspinor (x, y) result (xy)
    type(vectorspinor) :: xy
    real(kind=single), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy\%psi(k) = x * y\%psi(k)
    end do
  end function real_vectorspinor
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
  pure function double_vectorspinor (x, y) result (xy)
    type(vectorspinor) :: xy
    real(kind=default), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy\%psi(k) = x * y\%psi(k)
    end do
  end function double_vectorspinor
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
  pure function complex_vectorspinor (x, y) result (xy)
    type(vectorspinor) :: xy
    complex(kind=single), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
```

```
do k = 1,4
    xy\%psi(k) = x * y\%psi(k)
    end do
  end function complex_vectorspinor
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
 pure function dcomplex_vectorspinor (x, y) result (xy)
    type(vectorspinor) :: xy
    complex(kind=default), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy\%psi(k) = x * y\%psi(k)
    end do
  end function dcomplex_vectorspinor
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
 pure function vectorspinor_integer (y, x) result (xy)
    type(vectorspinor) :: xy
    integer, intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy\%psi(k) = y\%psi(k) * x
    end do
  end function vectorspinor_integer
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
 pure function vectorspinor_real (y, x) result (xy)
    type(vectorspinor) :: xy
    real(kind=single), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy\%psi(k) = y\%psi(k) * x
    end do
  end function vectorspinor_real
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
 pure function vectorspinor_double (y, x) result (xy)
    type(vectorspinor) :: xy
    real(kind=default), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy\%psi(k) = y\%psi(k) * x
    end do
  end function vectorspinor_double
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
 pure function vectorspinor_complex (y, x) result (xy)
    type(vectorspinor) :: xy
    complex(kind=single), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy\%psi(k) = y\%psi(k) * x
    end do
  end function vectorspinor_complex
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
 pure function vectorspinor_dcomplex (y, x) result (xy)
    type(vectorspinor) :: xy
    complex(kind=default), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy\%psi(k) = y\%psi(k) * x
  end function vectorspinor_dcomplex
```

```
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
  pure function momentum_vectorspinor (y, x) result (xy)
    type(bispinor) :: xy
    type(momentum), intent(in) :: y
    type(vectorspinor), intent(in) :: x
    integer :: k
    do k = 1,4
                        * x\%psi(1)\%a(k) - y\%x(1) * x\%psi(2)\%a(k) - &
    xy\%a(k) = y\%t
             y\%x(2) * x\%psi(3)\%a(k) - y\%x(3) * x\%psi(4)\%a(k)
    end do
  end function momentum_vectorspinor
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
  pure function vectorspinor_momentum (y, x) result (xy)
    type(bispinor) :: xy
    type(momentum), intent(in) :: x
    type(vectorspinor), intent(in) :: y
    integer :: k
    do k = 1,4
    xy\%a(k) = x\%t
                        * y\%psi(1)\%a(k) - x\%x(1) * y\%psi(2)\%a(k) - &
             x\%x(2) * y\%psi(3)\%a(k) - x\%x(3) * y\%psi(4)\%a(k)
  end function vectorspinor_momentum
                                              Unary Plus and Minus
\langle Declaration \ of \ operations \ for \ vectorspinors \rangle + \equiv
  interface operator (+)
     module procedure plus_vectorspinor
  end interface
  private :: plus_vectorspinor
  interface operator (-)
     module procedure neg_vectorspinor
  end interface
  private :: neg_vectorspinor
\langle Implementation\ of\ operations\ for\ vectorspinors \rangle + \equiv
  pure function plus_vectorspinor (x) result (plus_x)
    type(vectorspinor) :: plus_x
    \verb|type(vectorspinor), intent(in) :: x|\\
    integer :: k
    do k = 1,4
    plus_x\%psi(k) = + x\%psi(k)
    end do
  end function plus_vectorspinor
\langle Implementation of operations for vectorspinors \rangle + \equiv
  pure function neg_vectorspinor (x) result (neg_x)
    type(vectorspinor) :: neg_x
    type(vectorspinor), intent(in) :: x
    integer :: k
    do k = 1,4
    neg_x%psi(k) = - x%psi(k)
    end do
  end function neg_vectorspinor
                                             Addition and Subtraction
\langle Declaration \ of \ operations \ for \ vectorspinors \rangle + \equiv
  interface operator (+)
     module procedure add_vectorspinor
  end interface
  private :: add_vectorspinor
  interface operator (-)
     module procedure sub_vectorspinor
  end interface
  private :: sub_vectorspinor
```

```
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
  pure function add_vectorspinor (x, y) result (xy)
    type(vectorspinor) :: xy
    type(vectorspinor), intent(in) :: x, y
    integer :: k
    do k = 1,4
    xy\%psi(k) = x\%psi(k) + y\%psi(k)
    end do
  end function add_vectorspinor
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
  pure function sub_vectorspinor (x, y) result (xy)
    type(vectorspinor) :: xy
    type(vectorspinor), intent(in) :: x, y
    integer :: k
    do k = 1,4
    xy\%psi(k) = x\%psi(k) - y\%psi(k)
    end do
  end function sub_vectorspinor
                                                  Y.3.2 Norm
\langle Declaration \ of \ operations \ for \ vectorspinors \rangle + \equiv
  interface abs
     module procedure abs_vectorspinor
  end interface
  private :: abs_vectorspinor
\langle Implementation \ of \ operations \ for \ vectorspinors \rangle + \equiv
  pure function abs_vectorspinor (psi) result (x)
    real(kind=default) :: x
    type(vectorspinor), intent(in) :: psi
    x = sqrt (real (dot_product (psi%psi(1)%a, psi%psi(1)%a) &
             - dot_product (psi%psi(2)%a, psi%psi(2)%a)
             - dot_product (psi%psi(3)%a, psi%psi(3)%a)
             - dot_product (psi%psi(4)%a, psi%psi(4)%a)))
  end function abs_vectorspinor
```

Y.4 Vectors and Tensors

Condensed representation of antisymmetric rank-2 tensors:

$$\begin{pmatrix} T^{00} & T^{01} & T^{02} & T^{03} \\ T^{10} & T^{11} & T^{12} & T^{13} \\ T^{20} & T^{21} & T^{22} & T^{23} \\ T^{30} & T^{31} & T^{32} & T^{33} \end{pmatrix} = \begin{pmatrix} 0 & T_e^1 & T_e^2 & T_e^3 \\ -T_e^1 & 0 & T_b^3 & -T_b^2 \\ -T_e^2 & -T_b^3 & 0 & T_b^1 \\ -T_e^3 & T_b^2 & -T_b^1 & 0 \end{pmatrix}$$
 (Y.4)

```
\langle \mathtt{omega\_vectors.f90} \rangle \equiv
  \langle Copyleft \rangle
 module omega_vectors
    use kinds
    use constants
    implicit none
    private
    public :: assignment (=), operator(==)
    public :: operator (*), operator (+), operator (-), operator (.wedge.)
    public :: abs, conjg, set_zero
    public :: random_momentum
    ⟨intrinsic :: abs⟩
    (intrinsic :: conjg)
    type, public :: momentum
       ! private (omegalib needs access, but DON'T TOUCH IT!)
       real(kind=default) :: t
       real(kind=default), dimension(3) :: x
    end type momentum
```

```
type, public :: vector
       ! private (omegalib needs access, but DON'T TOUCH IT!)
       complex(kind=default) :: t
       complex(kind=default), dimension(3) :: x
    end type vector
    {\tt type,\ public :: tensor2odd}
       ! private (omegalib needs access, but DON'T TOUCH IT!)
       complex(kind=default), dimension(3) :: e
       complex(kind=default), dimension(3) :: b
    end type tensor2odd
    \langle Declaration \ of \ operations \ for \ vectors \rangle
    integer, parameter, public :: omega_vectors_2010_01_A = 0
    (Implementation of operations for vectors)
  end module omega_vectors
                                            Y.4.1 Constructors
\langle Declaration \ of \ operations \ for \ vectors \rangle \equiv
  interface assignment (=)
     module procedure momentum_of_array, vector_of_momentum, &
          vector_of_array, vector_of_double_array, &
          array_of_momentum, array_of_vector
  end interface
  private :: momentum_of_array, vector_of_momentum, vector_of_array, &
       vector_of_double_array, array_of_momentum, array_of_vector
\langle Implementation \ of \ operations \ for \ vectors \rangle \equiv
 pure subroutine momentum_of_array (m, p)
    type(momentum), intent(out) :: m
    real(kind=default), dimension(0:), intent(in) :: p
    m\%t = p(0)
    m%x = p(1:3)
  end subroutine momentum_of_array
  pure subroutine array_of_momentum (p, v)
    real(kind=default), dimension(0:), intent(out) :: p
    type(momentum), intent(in) :: v
    p(0) = v\%t
    p(1:3) = v\%x
  end subroutine array_of_momentum
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure subroutine vector_of_array (v, p)
    type(vector), intent(out) :: v
    complex(kind=default), dimension(0:), intent(in) :: p
    v\%t = p(0)
    v%x = p(1:3)
  end subroutine vector_of_array
 pure subroutine vector_of_double_array (v, p)
    type(vector), intent(out) :: v
    real(kind=default), dimension(0:), intent(in) :: p
    v\%t = p(0)
    v%x = p(1:3)
  end subroutine vector_of_double_array
 pure subroutine array_of_vector (p, v)
    complex(kind=default), dimension(0:), intent(out) :: p
    type(vector), intent(in) :: v
    p(0) = v\%t
    p(1:3) = v%x
  end subroutine array_of_vector
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure subroutine vector_of_momentum (v, p)
    {\tt type(vector),\ intent(out)\ ::\ v}
    type(momentum), intent(in) :: p
    v\%t = p\%t
    v\%x = p\%x
  end subroutine vector_of_momentum
```

```
\langle Declaration \ of \ operations \ for \ vectors \rangle + \equiv
  interface operator(==)
     module procedure momentum_eq
  end interface
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
  elemental function momentum_eq (lhs, rhs) result (yorn)
    logical :: yorn
    type(momentum), intent(in) :: lhs
    type(momentum), intent(in) :: rhs
    yorn = all (abs(lhs%x - rhs%x) < eps0) .and. abs(lhs%t - rhs%t) < eps0
  end function momentum_eq
                                             Y.4.2 Inner Products
\langle Declaration \ of \ operations \ for \ vectors \rangle + \equiv
  interface operator (*)
     module procedure momentum_momentum, vector_vector, &
           vector_momentum, momentum_vector, tensor2odd_tensor2odd
  end interface
  private :: momentum_momentum, vector_vector, vector_momentum, &
        momentum_vector, tensor2odd_tensor2odd
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
  pure function momentum_momentum (x, y) result (xy)
    type(momentum), intent(in) :: x
    type(momentum), intent(in) :: y
    real(kind=default) :: xy
    xy = x\%t*y\%t - x\%x(1)*y\%x(1) - x\%x(2)*y\%x(2) - x\%x(3)*y\%x(3)
  end function momentum_momentum
  pure function momentum_vector (x, y) result (xy)
    type(momentum), intent(in) :: x
    type(vector), intent(in) :: y
    complex(kind=default) :: xy
    xy = x\%t*y\%t - x\%x(1)*y\%x(1) - x\%x(2)*y\%x(2) - x\%x(3)*y\%x(3)
  end function momentum_vector
  pure function vector_momentum (x, y) result (xy)
    type(vector), intent(in) :: x
    type(momentum), intent(in) :: y
    complex(kind=default) :: xy
    xy = x\%t*y\%t - x\%x(1)*y\%x(1) - x\%x(2)*y\%x(2) - x\%x(3)*y\%x(3)
  end function vector_momentum
  pure function vector_vector (x, y) result (xy)
    type(vector), intent(in) :: x
    type(vector), intent(in) :: y
    complex(kind=default) :: xy
    xy = x/t * y/t - x/x(1) * y/x(1) - x/x(2) * y/x(2) - x/x(3) * y/x(3)
  end function vector_vector
Just like classical electrodynamics:
                          \frac{1}{2}T_{\mu\nu}U^{\mu\nu} = \frac{1}{2}\left(-T^{0i}U^{0i} - T^{i0}U^{i0} + T^{ij}U^{ij}\right) = T_b^k U_b^k - T_e^k U_e^k
                                                                                                                 (Y.5)
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
  pure function tensor2odd_tensor2odd (x, y) result (xy)
    {\tt type(tensor2odd),\ intent(in)\ ::\ x}
    type(tensor2odd), intent(in) :: y
    complex(kind=default) :: xy
    xy = x\%b(1)*y\%b(1) + x\%b(2)*y\%b(2) + x\%b(3)*y\%b(3) &
        - x\%e(1)*y\%e(1) - x\%e(2)*y\%e(2) - x\%e(3)*y\%e(3)
  end function tensor2odd_tensor2odd
                                     Y.4.3 Not Entirely Inner Products
\langle Declaration \ of \ operations \ for \ vectors \rangle + \equiv
```

interface operator (*)

$$y^{\nu} = x_{\mu} T^{\mu\nu} : y^{0} = -x^{i} T^{i0} = x^{i} T^{0i}$$
(Y.6a)

$$y^{1} = x^{0}T^{01} - x^{2}T^{21} - x^{3}T^{31}$$
 (Y.6b)

$$y^2 = x^0 T^{02} - x^1 T^{12} - x^3 T^{32} (Y.6c)$$

$$y^3 = x^0 T^{03} - x^1 T^{13} - x^2 T^{23} (Y.6d)$$

 $\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv$ pure function vector_tensor2odd (x, t2) result (xt2) type(vector), intent(in) :: x type(tensor2odd), intent(in) :: t2 type(vector) :: xt2 xt2%t = x%x(1)*t2%e(1) + x%x(2)*t2%e(2) + x%x(3)*t2%e(3)xt2%x(1) = x%t*t2%e(1) + x%x(2)*t2%b(3) - x%x(3)*t2%b(2)xt2%x(2) = x%t*t2%e(2) + x%x(3)*t2%b(1) - x%x(1)*t2%b(3)xt2%x(3) = x%t*t2%e(3) + x%x(1)*t2%b(2) - x%x(2)*t2%b(1)end function vector_tensor2odd pure function momentum_tensor2odd (x, t2) result (xt2) type(momentum), intent(in) :: x type(tensor2odd), intent(in) :: t2 type(vector) :: xt2 xt2%t = x%x(1)*t2%e(1) + x%x(2)*t2%e(2) + x%x(3)*t2%e(3)xt2%x(1) = x%t*t2%e(1) + x%x(2)*t2%b(3) - x%x(3)*t2%b(2)xt2%x(2) = x%t*t2%e(2) + x%x(3)*t2%b(1) - x%x(1)*t2%b(3)xt2%x(3) = x%t*t2%e(3) + x%x(1)*t2%b(2) - x%x(2)*t2%b(1)end function momentum_tensor2odd

$$y^{\mu} = T^{\mu\nu} x_{\nu} : y^{0} = -T^{0i} x^{i} \tag{Y.7a}$$

$$y^{1} = T^{10}x^{0} - T^{12}x^{2} - T^{13}x^{3}$$
 (Y.7b)

$$y^2 = T^{20}x^0 - T^{21}x^1 - T^{23}x^3 (Y.7c)$$

$$y^3 = T^{30}x^0 - T^{31}x^1 - T^{32}x^2 (Y.7d)$$

 $\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv$ pure function tensor2odd_vector (t2, x) result (t2x) type(tensor2odd), intent(in) :: t2 type(vector), intent(in) :: x type(vector) :: t2x t2x%t = -t2%e(1)*x%x(1) - t2%e(2)*x%x(2) - t2%e(3)*x%x(3)t2x%x(1) = -t2%e(1)*x%t + t2%b(2)*x%x(3) - t2%b(3)*x%x(2)t2x%x(2) = -t2%e(2)*x%t + t2%b(3)*x%x(1) - t2%b(1)*x%x(3)t2x%x(3) = -t2%e(3)*x%t + t2%b(1)*x%x(2) - t2%b(2)*x%x(1)end function tensor2odd_vector pure function tensor2odd_momentum (t2, x) result (t2x) type(tensor2odd), intent(in) :: t2 type(momentum), intent(in) :: x type(vector) :: t2x t2x%t = -t2%e(1)*x%x(1) - t2%e(2)*x%x(2) - t2%e(3)*x%x(3)t2x%x(1) = -t2%e(1)*x%t + t2%b(2)*x%x(3) - t2%b(3)*x%x(2)t2x%x(2) = -t2%e(2)*x%t + t2%b(3)*x%x(1) - t2%b(1)*x%x(3)t2x%x(3) = -t2%e(3)*x%t + t2%b(1)*x%x(2) - t2%b(2)*x%x(1)end function tensor2odd_momentum

Y.4.4 Outer Products

```
\langle Declaration\ of\ operations\ for\ vectors 
angle + \equiv interface operator (.wedge.) module procedure momentum_wedge_momentum, &
```

```
momentum_wedge_vector, vector_wedge_momentum, vector_wedge_vector
  end interface
 private :: momentum_wedge_momentum, momentum_wedge_vector, &
       vector_wedge_momentum, vector_wedge_vector
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure function momentum_wedge_momentum (x, y) result (t2)
    type(momentum), intent(in) :: x
    type(momentum), intent(in) :: y
    type(tensor2odd) :: t2
    t2\%e = x\%t * y\%x - x\%x * y\%t
    t2\%b(1) = x\%x(2) * y\%x(3) - x\%x(3) * y\%x(2)
    t2\%b(2) = x\%x(3) * y\%x(1) - x\%x(1) * y\%x(3)
    t2\%b(3) = x\%x(1) * y\%x(2) - x\%x(2) * y\%x(1)
  end function momentum_wedge_momentum
  pure function momentum_wedge_vector (x, y) result (t2)
    type(momentum), intent(in) :: x
    type(vector), intent(in) :: y
    type(tensor2odd) :: t2
    t2\%e = x\%t * y\%x - x\%x * y\%t
    t2\%b(1) = x\%x(2) * y\%x(3) - x\%x(3) * y\%x(2)
    t2\%b(2) = x\%x(3) * y\%x(1) - x\%x(1) * y\%x(3)
    t2\%b(3) = x\%x(1) * y\%x(2) - x\%x(2) * y\%x(1)
  end function momentum_wedge_vector
  pure function vector_wedge_momentum (x, y) result (t2)
    type(vector), intent(in) :: x
    type(momentum), intent(in) :: y
    type(tensor2odd) :: t2
    t2\%e = x\%t * y\%x - x\%x * y\%t
    t2\%b(1) = x\%x(2) * y\%x(3) - x\%x(3) * y\%x(2)
    t2\%b(2) = x\%x(3) * y\%x(1) - x\%x(1) * y\%x(3)
    t2\%b(3) = x\%x(1) * y\%x(2) - x\%x(2) * y\%x(1)
  end function vector_wedge_momentum
 pure function vector_wedge_vector (x, y) result (t2)
    type(vector), intent(in) :: x
    type(vector), intent(in) :: y
    type(tensor2odd) :: t2
    t2\%e = x\%t * y\%x - x\%x * y\%t
    t2\%b(1) = x\%x(2) * y\%x(3) - x\%x(3) * y\%x(2)
    t2\%b(2) = x\%x(3) * y\%x(1) - x\%x(1) * y\%x(3)
    t2\%b(3) = x\%x(1) * y\%x(2) - x\%x(2) * y\%x(1)
  end function vector_wedge_vector
                                             Y.4.5 Vector Space
\langle Declaration \ of \ operations \ for \ vectors \rangle + \equiv
  interface set_zero
    module procedure set_zero_vector, set_zero_momentum, &
      set_zero_tensor2odd, set_zero_real, set_zero_complex
  end interface
 private :: set_zero_vector, set_zero_momentum, set_zero_tensor2odd
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
  elemental subroutine set_zero_vector (x)
    type(vector), intent(out) :: x
    x\%t = 0
    x\%x = 0
  end subroutine set_zero_vector
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
  elemental subroutine set_zero_momentum (x)
    type(momentum), intent(out) :: x
    x\%t = 0
    x\%x = 0
  end subroutine set_zero_momentum
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
  elemental subroutine set_zero_tensor2odd (x)
```

```
type(tensor2odd), intent(out) :: x
    x\%e = 0
    x\%b = 0
  end subroutine set_zero_tensor2odd
Doesn't really belong here, but there is no better place \dots
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
  elemental subroutine set_zero_real (x)
    real(kind=default), intent(out) :: x
    x = 0
  end subroutine set_zero_real
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
  elemental subroutine set_zero_complex (x)
    complex(kind=default), intent(out) :: x
    x = 0
  end subroutine set_zero_complex
                                             Scalar Multiplication
\langle Declaration \ of \ operations \ for \ vectors \rangle + \equiv
  interface operator (*)
     module procedure integer_momentum, real_momentum, double_momentum, &
           complex_momentum, dcomplex_momentum, &
           integer_vector, real_vector, double_vector, &
           complex_vector, dcomplex_vector, &
           integer_tensor2odd, real_tensor2odd, double_tensor2odd, &
           complex_tensor2odd, dcomplex_tensor2odd, &
           momentum_integer, momentum_real, momentum_double, &
          momentum_complex, momentum_dcomplex, &
          vector_integer, vector_real, vector_double, &
          vector_complex, vector_dcomplex, &
          tensor2odd_integer, tensor2odd_real, tensor2odd_double, &
          {\tt tensor2odd\_complex},\ {\tt tensor2odd\_dcomplex}
  end interface
  private :: integer_momentum, real_momentum, double_momentum, &
        complex_momentum, dcomplex_momentum, integer_vector, real_vector, &
       double_vector, complex_vector, dcomplex_vector, &
       integer_tensor2odd, real_tensor2odd, double_tensor2odd, &
       complex_tensor2odd, dcomplex_tensor2odd, momentum_integer, &
       momentum_real, momentum_double, momentum_complex, &
       momentum_dcomplex, vector_integer, vector_real, vector_double, &
       vector_complex, vector_dcomplex, tensor2odd_integer, &
       tensor2odd_real, tensor2odd_double, tensor2odd_complex, &
       tensor2odd_dcomplex
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
  pure function integer_momentum (x, y) result (xy)
    integer, intent(in) :: x
    type(momentum), intent(in) :: y
    type(momentum) :: xy
    xy\%t = x * y\%t
    xy\%x = x * y\%x
  end function integer_momentum
  pure function real_momentum (x, y) result (xy)
    real(kind=single), intent(in) :: x
    type(momentum), intent(in) :: y
    type(momentum) :: xy
    xy\%t = x * y\%t
    xy\%x = x * y\%x
  end function real_momentum
  pure function double_momentum (x, y) result (xy)
    real(kind=default), intent(in) :: x
    type(momentum), intent(in) :: y
    type(momentum) :: xy
    xy\%t = x * y\%t
    xy\%x = x * y\%x
```

```
end function double_momentum
  pure function complex_momentum (x, y) result (xy)
    complex(kind=single), intent(in) :: x
    type(momentum), intent(in) :: y
    type(vector) :: xy
    xy\%t = x * y\%t
    xy\%x = x * y\%x
  end function complex_momentum
  pure function dcomplex_momentum (x, y) result (xy)
    complex(kind=default), intent(in) :: x
    type(momentum), intent(in) :: y
    type(vector) :: xy
    xy\%t = x * y\%t
    xy\%x = x * y\%x
  end function dcomplex_momentum
\langle Implementation\ of\ operations\ for\ vectors \rangle + \equiv
 pure function integer_vector (x, y) result (xy)
    integer, intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy\%t = x * y\%t
    xy\%x = x * y\%x
  end function integer_vector
  pure function real_vector (x, y) result (xy)
    real(kind=single), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy\%t = x * y\%t
    xy\%x = x * y\%x
  end function real_vector
  pure function double_vector (x, y) result (xy)
    real(kind=default), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy\%t = x * y\%t
    xy\%x = x * y\%x
  end function double_vector
 pure function complex_vector (x, y) result (xy)
    complex(kind=single), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy\%t = x * y\%t
    xy\%x = x * y\%x
  end function complex_vector
  pure function dcomplex_vector (x, y) result (xy)
    complex(kind=default), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy\%t = x * y\%t
    xy\%x = x * y\%x
  end function dcomplex_vector
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure function integer_tensor2odd (x, t2) result (xt2)
    integer, intent(in) :: x
    type(tensor2odd), intent(in) :: t2
    type(tensor2odd) :: xt2
    xt2\%e = x * t2\%e
    xt2\%b = x * t2\%b
  end function integer_tensor2odd
 pure function real_tensor2odd (x, t2) result (xt2)
    real(kind=single), intent(in) :: x
    type(tensor2odd), intent(in) :: t2
    type(tensor2odd) :: xt2
    xt2\%e = x * t2\%e
    xt2\%b = x * t2\%b
```

```
end function real_tensor2odd
 pure function double_tensor2odd (x, t2) result (xt2)
   real(kind=default), intent(in) :: x
   type(tensor2odd), intent(in) :: t2
   type(tensor2odd) :: xt2
   xt2\%e = x * t2\%e
   xt2\%b = x * t2\%b
 end function double_tensor2odd
 pure function complex_tensor2odd (x, t2) result (xt2)
   complex(kind=single), intent(in) :: x
    type(tensor2odd), intent(in) :: t2
   type(tensor2odd) :: xt2
   xt2\%e = x * t2\%e
   xt2\%b = x * t2\%b
  end function complex_tensor2odd
 pure function dcomplex_tensor2odd (x, t2) result (xt2)
   complex(kind=default), intent(in) :: x
   type(tensor2odd), intent(in) :: t2
   type(tensor2odd) :: xt2
   xt2\%e = x * t2\%e
   xt2\%b = x * t2\%b
 end function dcomplex_tensor2odd
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure function momentum_integer (y, x) result (xy)
   integer, intent(in) :: x
   type(momentum), intent(in) :: y
   type(momentum) :: xy
   xy\%t = x * y\%t
   xy\%x = x * y\%x
 end function momentum_integer
 pure function momentum_real (y, x) result (xy)
   real(kind=single), intent(in) :: x
   type(momentum), intent(in) :: y
   type(momentum) :: xy
   xy\%t = x * y\%t
   xy\%x = x * y\%x
 end function momentum_real
 pure function momentum_double (y, x) result (xy)
   real(kind=default), intent(in) :: x
   type(momentum), intent(in) :: y
   type(momentum) :: xy
   xy\%t = x * y\%t
   xy\%x = x * y\%x
  end function momentum_double
 pure function momentum_complex (y, x) result (xy)
   complex(kind=single), intent(in) :: x
   type(momentum), intent(in) :: y
   type(vector) :: xy
   xy\%t = x * y\%t
   xy\%x = x * y\%x
 end function momentum_complex
 pure function momentum_dcomplex (y, x) result (xy)
   complex(kind=default), intent(in) :: x
   type(momentum), intent(in) :: y
    type(vector) :: xy
   xy\%t = x * y\%t
   xy\%x = x * y\%x
 end function momentum_dcomplex
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure function vector_integer (y, x) result (xy)
   integer, intent(in) :: x
   type(vector), intent(in) :: y
   type(vector) :: xy
   xy\%t = x * y\%t
   xy\%x = x * y\%x
```

```
end function vector_integer
 pure function vector_real (y, x) result (xy)
    real(kind=single), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy\%t = x * y\%t
    xy\%x = x * y\%x
  end function vector_real
  pure function vector_double (y, x) result (xy)
    real(kind=default), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy\%t = x * y\%t
    xy\%x = x * y\%x
  end function vector_double
 pure function vector_complex (y, x) result (xy)
    complex(kind=single), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy\%t = x * y\%t
    xy\%x = x * y\%x
  end function vector_complex
  pure function vector_dcomplex (y, x) result (xy)
    complex(kind=default), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy\%t = x * y\%t
    xy\%x = x * y\%x
  end function vector_dcomplex
\langle Implementation\ of\ operations\ for\ vectors \rangle + \equiv
 pure function tensor2odd_integer (t2, x) result (t2x)
    type(tensor2odd), intent(in) :: t2
    integer, intent(in) :: x
    type(tensor2odd) :: t2x
    t2x\%e = x * t2\%e
    t2x\%b = x * t2\%b
  end function tensor2odd_integer
 pure function tensor2odd_real (t2, x) result (t2x)
    type(tensor2odd), intent(in) :: t2
    real(kind=single), intent(in) :: x
    type(tensor2odd) :: t2x
    t2x\%e = x * t2\%e
    t2x\%b = x * t2\%b
  end function tensor2odd_real
  pure function tensor2odd_double (t2, x) result (t2x)
    type(tensor2odd), intent(in) :: t2
    real(kind=default), intent(in) :: x
    type(tensor2odd) :: t2x
    t2x\%e = x * t2\%e
    t2x\%b = x * t2\%b
  end function tensor2odd_double
  pure function tensor2odd_complex (t2, x) result (t2x)
    type(tensor2odd), intent(in) :: t2
    complex(kind=single), intent(in) :: x
    type(tensor2odd) :: t2x
    t2x\%e = x * t2\%e
    t2x\%b = x * t2\%b
  \verb"end function tensor2odd_complex"
 pure function tensor2odd_dcomplex (t2, x) result (t2x)
    type(tensor2odd), intent(in) :: t2
    complex(kind=default), intent(in) :: x
    type(tensor2odd) :: t2x
    t2x\%e = x * t2\%e
    t2x\%b = x * t2\%b
  end function tensor2odd_dcomplex
```

Unary Plus and Minus

```
\langle Declaration \ of \ operations \ for \ vectors \rangle + \equiv
  interface operator (+)
     module procedure plus_momentum, plus_vector, plus_tensor2odd
  end interface
 private :: plus_momentum, plus_vector, plus_tensor2odd
  interface operator (-)
     module procedure neg_momentum, neg_vector, neg_tensor2odd
  end interface
 private :: neg_momentum, neg_vector, neg_tensor2odd
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure function plus_momentum (x) result (plus_x)
    type(momentum), intent(in) :: x
    type(momentum) :: plus_x
    plus_x = x
  end function plus_momentum
 pure function neg_momentum (x) result (neg_x)
    \verb"type(momentum"), intent(in) :: x
    type(momentum) :: neg_x
    neg_x\%t = - x\%t
    neg_x%x = - x%x
  end function neg_momentum
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure function plus_vector (x) result (plus_x)
    type(vector), intent(in) :: x
    type(vector) :: plus_x
    plus_x = x
  end function plus_vector
 pure function neg_vector (x) result (neg_x)
    type(vector), intent(in) :: x
    type(vector) :: neg_x
    neg_x\%t = - x\%t
    neg_x%x = - x%x
  end function neg_vector
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure function plus_tensor2odd (x) result (plus_x)
    type(tensor2odd), intent(in) :: x
    type(tensor2odd) :: plus_x
    plus_x = x
  end function plus_tensor2odd
 pure function neg_tensor2odd (x) result (neg_x)
    type(tensor2odd), intent(in) :: x
    type(tensor2odd) :: neg_x
    neg_x\%e = - x\%e
    neg_x\%b = - x\%b
  end function neg_tensor2odd
                                            Addition and Subtraction
\langle Declaration \ of \ operations \ for \ vectors \rangle + \equiv
  interface operator (+)
     module procedure add_momentum, add_vector, &
          add_vector_momentum, add_momentum_vector, add_tensor2odd
  end interface
  private :: add_momentum, add_vector, add_vector_momentum, &
       add_momentum_vector, add_tensor2odd
  interface operator (-)
     module procedure sub_momentum, sub_vector, &
           sub_vector_momentum, sub_momentum_vector, sub_tensor2odd
 end interface
 private :: sub_momentum, sub_vector, sub_vector_momentum, &
       sub_momentum_vector, sub_tensor2odd
```

```
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure function add_momentum (x, y) result (xy)
    type(momentum), intent(in) :: x, y
    type(momentum) :: xy
    xy\%t = x\%t + y\%t
    xy\%x = x\%x + y\%x
  end function add_momentum
  pure function add_vector (x, y) result (xy)
    type(vector), intent(in) :: x, y
    type(vector) :: xy
    xy\%t = x\%t + y\%t
    xy\%x = x\%x + y\%x
  end function add_vector
  pure function add_momentum_vector (x, y) result (xy)
    type(momentum), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy\%t = x\%t + y\%t
    xy\%x = x\%x + y\%x
  end function add_momentum_vector
  pure function add_vector_momentum (x, y) result (xy)
    type(vector), intent(in) :: x
    type(momentum), intent(in) :: y
    type(vector) :: xy
    xy\%t = x\%t + y\%t
    xy\%x = x\%x + y\%x
  end function add_vector_momentum
 pure function add_tensor2odd (x, y) result (xy)
    type(tensor2odd), intent(in) :: x, y
    type(tensor2odd) :: xy
    xy\%e = x\%e + y\%e
    xy\%b = x\%b + y\%b
  end function add_tensor2odd
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure function sub_momentum (x, y) result (xy)
    type(momentum), intent(in) :: x, y
    type(momentum) :: xy
    xy\%t = x\%t - y\%t
    xy\%x = x\%x - y\%x
  end function sub_momentum
  pure function sub_vector (x, y) result (xy)
    type(vector), intent(in) :: x, y
    type(vector) :: xy
    xy\%t = x\%t - y\%t
    xy\%x = x\%x - y\%x
  end function sub_vector
  pure function sub_momentum_vector (x, y) result (xy)
    type(momentum), intent(in) :: x
    type(vector), intent(in) :: y
    type(vector) :: xy
    xy\%t = x\%t - y\%t
    xy\%x = x\%x - y\%x
  end function sub_momentum_vector
  pure function sub_vector_momentum (x, y) result (xy)
    type(vector), intent(in) :: x
    type(momentum), intent(in) :: y
    type(vector) :: xy
    xy\%t = x\%t - y\%t
    xy\%x = x\%x - y\%x
  end function sub_vector_momentum
 pure function sub_tensor2odd (x, y) result (xy)
    type(tensor2odd), intent(in) :: x, y
    type(tensor2odd) :: xy
    xy\%e = x\%e - y\%e
    xy\%b = x\%b - y\%b
  end function sub_tensor2odd
```

Y.4.6 Norm

```
Not the covariant length!
\langle Declaration \ of \ operations \ for \ vectors \rangle + \equiv
  interface abs
     module procedure abs_momentum, abs_vector, abs_tensor2odd
  end interface
  private :: abs_momentum, abs_vector, abs_tensor2odd
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
  pure function abs_momentum (x) result (absx)
    type(momentum), intent(in) :: x
    real(kind=default) :: absx
    absx = sqrt (real (x\%t*x\%t + dot_product (x\%x, x\%x)))
  \verb"end function abs_momentum"
  pure function abs_vector (x) result (absx)
    type(vector), intent(in) :: x
    real(kind=default) :: absx
    absx = sqrt (real (conjg(x%t)*x%t + dot_product (x%x, x%x)))
  end function abs_vector
  pure function abs_tensor2odd (x) result (absx)
    type(tensor2odd), intent(in) :: x
    real(kind=default) :: absx
    absx = sqrt (real (dot_product (x%e, x%e) + dot_product (x%b, x%b)))
  end function abs_tensor2odd
                                               Y.4.7 Conjugation
\langle Declaration \ of \ operations \ for \ vectors \rangle + \equiv
  interface conjg
     module procedure conjg_momentum, conjg_vector, conjg_tensor2odd
  end interface
  private :: conjg_momentum, conjg_vector, conjg_tensor2odd
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
  pure function conjg_momentum (x) result (conjg_x)
    type(momentum), intent(in) :: x
    type(momentum) :: conjg_x
    conjg_x = x
  end function conjg_momentum
  pure function conjg_vector (x) result (conjg_x)
    type(vector), intent(in) :: x
    type(vector) :: conjg_x
    conjg_x\%t = conjg(x\%t)
    conjg_x%x = conjg(x%x)
  end function conjg_vector
  pure function conjg_tensor2odd (t2) result (conjg_t2)
    type(tensor2odd), intent(in) :: t2
    type(tensor2odd) :: conjg_t2
    conjg_t2%e = conjg (t2%e)
    conjg_t2%b = conjg (t2%b)
  end function conjg_tensor2odd
                                                 Y.4.8 \epsilon-Tensors
                                                 \epsilon_{0123} = 1 = -\epsilon^{0123}
                                                                                                                  (Y.8)
in particular
                             \epsilon(p_1,p_2,p_3,p_4) = \epsilon_{\mu_1\mu_2\mu_3\mu_4} p_1^{\mu_1} p_2^{\mu_2} p_3^{\mu_3} p_4^{\mu_4} = p_1^0 p_2^1 p_3^2 p_4^3 \pm \dots
                                                                                                                  (Y.9)
\langle Declaration \ of \ operations \ for \ vectors \rangle + \equiv
  interface pseudo_scalar
     module procedure pseudo_scalar_momentum, pseudo_scalar_vector, &
           pseudo_scalar_vec_mom
  end interface
  public :: pseudo_scalar
  private :: pseudo_scalar_momentum, pseudo_scalar_vector
```

```
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
  pure function pseudo_scalar_momentum (p1, p2, p3, p4) result (eps1234)
    type(momentum), intent(in) :: p1, p2, p3, p4
    real(kind=default) :: eps1234
    eps1234 = &
         p1%t
                  * p2\%x(1) * (p3\%x(2) * p4\%x(3) - p3\%x(3) * p4\%x(2)) &
       + p1%t
                  * p2\%x(2) * (p3\%x(3) * p4\%x(1) - p3\%x(1) * p4\%x(3)) &
                  * p2\%x(3) * (p3\%x(1) * p4\%x(2) - p3\%x(2) * p4\%x(1)) &
       + p1%t
       - p1\%x(1) * p2\%x(2) * (p3\%x(3) * p4\%t
                                                   - p3%t
                                                                * p4%x(3)) &
       - p1\%x(1) * p2\%x(3) * (p3\%t)
                                        * p4\%x(2) - p3\%x(2) * p4\%t
       - p1\%x(1) * p2\%t
                            * (p3\%x(2) * p4\%x(3) - p3\%x(3) * p4\%x(2)) &
       + p1\%x(2) * p2\%x(3) * (p3\%t * p4\%x(1) - p3\%x(1) * p4\%t ) &
                           * (p3\%x(1) * p4\%x(3) - p3\%x(3) * p4\%x(1)) & 1) * <math>(p3\%x(3) * p4\%t - p3\%t * p4\%x(3)) & 2
       + p1%x(2) * p2%t
       + p1\%x(2) * p2\%x(1) * (p3\%x(3) * p4\%t
                           * (p3\%x(1) * p4\%x(2) - p3\%x(2) * p4\%x(1)) &
       - p1%x(3) * p2%t
       - p1\%x(3) * p2\%x(1) * (p3\%x(2) * p4\%t - p3\%t
                                                              * p4%x(2)) &
       - p1\%x(3) * p2\%x(2) * (p3\%t)
                                          * p4\%x(1) - p3\%x(1) * p4\%t
  end function pseudo_scalar_momentum
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure function pseudo_scalar_vector (p1, p2, p3, p4) result (eps1234)
    type(vector), intent(in) :: p1, p2, p3, p4
    complex(kind=default) :: eps1234
    eps1234 = &
         p1%t
                  * p2\%x(1) * (p3\%x(2) * p4\%x(3) - p3\%x(3) * p4\%x(2)) &
       + p1%t
                  * p2\%x(2) * (p3\%x(3) * p4\%x(1) - p3\%x(1) * p4\%x(3)) &
                  * p2\%x(3) * (p3\%x(1) * p4\%x(2) - p3\%x(2) * p4\%x(1)) &
       + p1%t
       - p1\%x(1) * p2\%x(2) * (p3\%x(3) * p4\%t
                                                                * p4%x(3)) &
                                                   - p3%t
       - p1\%x(1) * p2\%x(3) * (p3\%t * p4\%x(2) - p3\%x(2) * p4\%t
                                                                         ) &
       - p1\%x(1) * p2\%t
                           * (p3\%x(2) * p4\%x(3) - p3\%x(3) * p4\%x(2)) &
       + p1\%x(2) * p2\%x(3) * (p3\%t * p4\%x(1) - p3\%x(1) * p4\%t
                           * (p3\%x(1) * p4\%x(3) - p3\%x(3) * p4\%x(1)) &
       + p1%x(2) * p2%t
       + p1\%x(2) * p2\%x(1) * (p3\%x(3) * p4\%t
                                                   - p3%t
                                                                * p4%x(3)) &
       - p1\%x(3) * p2\%t
                           * (p3\%x(1) * p4\%x(2) - p3\%x(2) * p4\%x(1)) &
       - p1\%x(3) * p2\%x(1) * (p3\%x(2) * p4\%t - p3\%t
                                                              * p4%x(2)) &
        - p1\%x(3) * p2\%x(2) * (p3\%t)
                                         * p4\%x(1) - p3\%x(1) * p4\%t
  end function pseudo_scalar_vector
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure function pseudo_scalar_vec_mom (p1, v1, p2, v2) result (eps1234)
    type(momentum), intent(in) :: p1, p2
    type(vector), intent(in) :: v1, v2
    complex(kind=default) :: eps1234
    eps1234 = &
                  * v1\%x(1) * (p2\%x(2) * v2\%x(3) - p2\%x(3) * v2\%x(2)) &
         p1%t
       + p1%t
                  * v1\%x(2) * (p2\%x(3) * v2\%x(1) - p2\%x(1) * v2\%x(3)) &
                  * v1\%x(3) * (p2\%x(1) * v2\%x(2) - p2\%x(2) * v2\%x(1)) &
       + p1%t
       - p1\%x(1) * v1\%x(2) * (p2\%x(3) * v2\%t
                                                  - p2%t
                                                              * v2%x(3)) &
                                       * v2\%x(2) - p2\%x(2) * v2\%t
       - p1\%x(1) * v1\%x(3) * (p2\%t)
                                                                         ) &
                           * (p2\%x(2) * v2\%x(3) - p2\%x(3) * v2\%x(2)) &
       - p1\%x(1) * v1\%t
       + p1\%x(2) * v1\%x(3) * (p2\%t * v2\%x(1) - p2\%x(1) * v2\%t ) &
       + p1\%x(2) * v1\%t * (p2\%x(1) * v2\%x(3) - p2\%x(3) * v2\%x(1)) &
       + p1\%x(2) * v1\%x(1) * (p2\%x(3) * v2\%t - p2\%t
                                                               * v2%x(3)) &
       - p1\%x(3) * v1\%t
                           * (p2\%x(1) * v2\%x(2) - p2\%x(2) * v2\%x(1)) &
       - p1\%x(3) * v1\%x(1) * (p2\%x(2) * v2\%t - p2\%t
                                                              * v2%x(2)) &
       - p1\%x(3) * v1\%x(2) * (p2\%t)
                                         * v2\%x(1) - p2\%x(1) * v2\%t
  end function pseudo_scalar_vec_mom
                                       \epsilon_{\mu}(p_1, p_2, p_3) = \epsilon_{\mu\mu_1\mu_2\mu_3} p_1^{\mu_1} p_2^{\mu_2} p_3^{\mu_3}
```

$$\epsilon_{\mu}(p_1, p_2, p_3) = \epsilon_{\mu\mu_1\mu_2\mu_3} p_1^{\mu_1} p_2^{\mu_2} p_3^{\mu_3}$$
 (Y.10)

i.e.

$$\epsilon_0(p_1, p_2, p_3) = p_1^1 p_2^2 p_3^3 \pm \dots$$
 (Y.11a)

$$\epsilon_1(p_1, p_2, p_3) = p_1^2 p_2^3 p_3^0 \pm \dots$$
 (Y.11b)

$$\epsilon_2(p_1, p_2, p_3) = -p_1^3 p_2^0 p_3^1 \pm \dots$$
 (Y.11c)

$$\epsilon_3(p_1, p_2, p_3) = p_1^0 p_2^1 p_3^2 \pm \dots$$
 (Y.11d)

```
\langle Declaration \ of \ operations \ for \ vectors \rangle + \equiv
  interface pseudo_vector
     module procedure pseudo_vector_momentum, pseudo_vector_vector, &
          pseudo_vector_vec_mom
  end interface
 public :: pseudo_vector
 private :: pseudo_vector_momentum, pseudo_vector_vector
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure function pseudo_vector_momentum (p1, p2, p3) result (eps123)
    type(momentum), intent(in) :: p1, p2, p3
    type(momentum) :: eps123
    eps123\%t = &
      + p1\%x(1) * (p2\%x(2) * p3\%x(3) - p2\%x(3) * p3\%x(2)) &
      + p1\%x(2) * (p2\%x(3) * p3\%x(1) - p2\%x(1) * p3\%x(3)) &
      + p1\%x(3) * (p2\%x(1) * p3\%x(2) - p2\%x(2) * p3\%x(1))
    eps123%x(1) = &
      + p1\%x(2) * (p2\%x(3) * p3\%t
                                       - p2%t
                                                  * p3%x(3)) &
      + p1\%x(3) * (p2\%t)
                           * p3\%x(2) - p2\%x(2) * p3\%t ) &
      + p1%t
               * (p2\%x(2) * p3\%x(3) - p2\%x(3) * p3\%x(2))
    eps123%x(2) = &
                             * p3\%x(1) - p2\%x(1) * p3\%t ) &
      - p1%x(3) * (p2%t)
               * (p2\%x(1) * p3\%x(3) - p2\%x(3) * p3\%x(1)) &
      - p1\%x(1) * (p2\%x(3) * p3\%t
                                      - p2%t
                                                  * p3%x(3))
    eps123%x(3) = &
      + p1%t
               * (p2\%x(1) * p3\%x(2) - p2\%x(2) * p3\%x(1)) &
      + p1\%x(1) * (p2\%x(2) * p3\%t - p2\%t * p3\%x(2)) &
                             * p3%x(1) - p2%x(1) * p3%t )
      + p1\%x(2) * (p2\%t)
  end function pseudo_vector_momentum
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure function pseudo_vector_vector (p1, p2, p3) result (eps123)
    type(vector), intent(in) :: p1, p2, p3
    type(vector) :: eps123
    eps123%t = &
      + p1\%x(1) * (p2\%x(2) * p3\%x(3) - p2\%x(3) * p3\%x(2)) &
      + p1\%x(2) * (p2\%x(3) * p3\%x(1) - p2\%x(1) * p3\%x(3) & + p1\%x(3) * (p2\%x(1) * p3\%x(2) - p2\%x(2) * p3\%x(1))
    eps123%x(1) = &
      + p1\%x(2) * (p2\%x(3) * p3\%t
                                        - p2%t
                                                    * p3%x(3)) &
                           * p3%x(2) - p2%x(2) * p3%t ) &
      + p1\%x(3) * (p2\%t)
               * (p2\%x(2) * p3\%x(3) - p2\%x(3) * p3\%x(2))
      + p1%t
    eps123\%x(2) = &
                             * p3%x(1) - p2%x(1) * p3%t
      - p1%x(3) * (p2%t)
                * (p2\%x(1) * p3\%x(3) - p2\%x(3) * p3\%x(1)) &
      - p1\%x(1) * (p2\%x(3) * p3\%t
                                       - p2%t
                                                  * p3%x(3))
    eps123%x(3) = &
      + p1%t
               * (p2\%x(1) * p3\%x(2) - p2\%x(2) * p3\%x(1)) &
      + p1%x(1) * (p2%x(2) * p3%t
                                      - p2%t
                                                  * p3%x(2)) &
      + p1\%x(2) * (p2\%t)
                             * p3\%x(1) - p2\%x(1) * p3\%t
  end function pseudo_vector_vector
\langle Implementation \ of \ operations \ for \ vectors \rangle + \equiv
 pure function pseudo_vector_vec_mom (p1, p2, v) result (eps123)
    type(momentum), intent(in) :: p1, p2
    type(vector), intent(in)
    type(vector) :: eps123
    eps123\%t = &
      + p1\%x(1) * (p2\%x(2) * v\%x(3) - p2\%x(3) * v\%x(2)) &
      + p1\%x(2) * (p2\%x(3) * v\%x(1) - p2\%x(1) * v\%x(3)) &
      + p1\%x(3) * (p2\%x(1) * v\%x(2) - p2\%x(2) * v\%x(1))
    eps123%x(1) = &
      + p1%x(2) * (p2%x(3) * v%t
                                       - p2%t
                                                  * v%x(3)) &
      + p1\%x(3) * (p2\%t * v\%x(2) - p2\%x(2) * v\%t ) &
               * (p2\%x(2) * v\%x(3) - p2\%x(3) * v\%x(2))
      + p1%t
    eps123%x(2) = &
      - p1\%x(3) * (p2\%t)
                             * v\%x(1) - p2\%x(1) * v\%t ) &
               * (p2\%x(1) * v\%x(3) - p2\%x(3) * v\%x(1)) &
```

Y.4.9 Utilities

 $\langle Declaration \ of \ operations \ for \ vectors \rangle + \equiv$

```
\langle Implementation of operations for vectors\rangle +=
subroutine random_momentum (p, pabs, m)
type(momentum), intent(out) :: p
real(kind=default), intent(in) :: pabs, m
real(kind=default), dimension(2) :: r
real(kind=default) :: phi, cos_th
call random_number (r)
phi = 2*PI * r(1)
cos_th = 2 * r(2) - 1
p%t = sqrt (pabs**2 + m**2)
p%x = pabs * (/ cos_th * cos(phi), cos_th * sin(phi), sqrt (1 - cos_th**2) /)
end subroutine random_momentum
```

Y.5 Polarization vectors

```
\langle (comega_polarizations.f90)\leftauring
\langle (Copyleft)
module omega_polarizations
use kinds
use constants
use omega_vectors
implicit none
private
\langle Declaration of polarization vectors\rangle
integer, parameter, public :: omega_polarizations_2010_01_A = 0
contains
\langle Implementation of polarization vectors\rangle
end module omega_polarizations
```

Here we use a phase convention for the polarization vectors compatible with the angular momentum coupling to spin 3/2 and spin 2.

$$\epsilon_1^{\mu}(k) = \frac{1}{|\vec{k}|\sqrt{k_x^2 + k_y^2}} \left(0; k_z k_x, k_y k_z, -k_x^2 - k_y^2\right) \tag{Y.12a}$$

$$\epsilon_2^{\mu}(k) = \frac{1}{\sqrt{k_x^2 + k_y^2}} (0; -k_y, k_x, 0)$$
(Y.12b)

$$\epsilon_3^{\mu}(k) = \frac{k_0}{m|\vec{k}|} \left(\vec{k}^2/k_0; k_x, k_y, k_z\right)$$
 (Y.12c)

and

$$\epsilon_{\pm}^{\mu}(k) = \frac{1}{\sqrt{2}} (\epsilon_1^{\mu}(k) \pm i\epsilon_2^{\mu}(k)) \tag{Y.13a}$$

$$\epsilon_0^{\mu}(k) = \epsilon_3^{\mu}(k) \tag{Y.13b}$$

i.e.

$$\epsilon_{+}^{\mu}(k) = \frac{1}{\sqrt{2}\sqrt{k_x^2 + k_y^2}} \left(0; \frac{k_z k_x}{|\vec{k}|} - ik_y, \frac{k_y k_z}{|\vec{k}|} + ik_x, -\frac{k_x^2 + k_y^2}{|\vec{k}|} \right)$$
 (Y.14a)

$$\epsilon_{-}^{\mu}(k) = \frac{1}{\sqrt{2}\sqrt{k_x^2 + k_y^2}} \left(0; \frac{k_z k_x}{|\vec{k}|} + ik_y, \frac{k_y k_z}{|\vec{k}|} - ik_x, -\frac{k_x^2 + k_y^2}{|\vec{k}|} \right)$$
 (Y.14b)

$$\epsilon_0^{\mu}(k) = \frac{k_0}{m|\vec{k}|} \left(\vec{k}^2/k_0; k_x, k_y, k_z\right)$$
 (Y.14c)

Determining the mass from the momenta is a numerically haphazardous for light particles. Therefore, we accept some redundancy and pass the mass explicitely.

```
\langle Declaration\ of\ polarization\ vectors \rangle {\equiv}
  public :: eps
\langle Implementation\ of\ polarization\ vectors \rangle \equiv
  pure function eps (m, k, s) result (e)
    type(vector) :: e
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: k
    integer, intent(in) :: s
    real(kind=default) :: kt, kabs, kabs2, sqrt2
    sqrt2 = sqrt (2.0_default)
    kabs2 = dot_product (k%x, k%x)
    e%t = 0
    e%x = 0
    if (kabs2 > 0) then
       kabs = sqrt (kabs2)
       select case (s)
       case (1)
          kt = sqrt (k%x(1)**2 + k%x(2)**2)
          if (abs(kt) <= epsilon(kt) * kabs) then
             if (k\%x(3) > 0) then
                 e%x(1) = cmplx (
                                   1,
                                         0, kind=default) / sqrt2
                 e%x(2) = cmplx (
                                   0,
                                         1, kind=default) / sqrt2
             else
                                         0, kind=default) / sqrt2
                e%x(1) = cmplx ( - 1,
                e%x(2) = cmplx ( 0,
                                         1, kind=default) / sqrt2
             end if
          else
             e%x(1) = cmplx ( k%x(3)*k%x(1)/kabs, &
                  - k%x(2), kind=default) / kt / sqrt2
             e\%x(2) = cmplx ( k\%x(2)*k\%x(3)/kabs, &
                  k%x(1), kind=default) / kt / sqrt2
             e%x(3) = - kt / kabs / sqrt2
          end if
       case (-1)
          kt = sqrt (k%x(1)**2 + k%x(2)**2)
          if (abs(kt) <= epsilon(kt) * kabs) then
             if (k\%x(3) > 0) then
                                   1, 0, kind=default) / sqrt2
                e%x(1) = cmplx (
                                   0, - 1, kind=default) / sqrt2
                e%x(2) = cmplx (
                e%x(1) = cmplx ( -1,  0, kind=default) / sqrt2
                 e%x(2) = cmplx (
                                   0, - 1, kind=default) / sqrt2
             end if
          else
             e%x(1) = cmplx ( k%x(3)*k%x(1)/kabs, &
                  k%x(2), kind=default) / kt / sqrt2
             e%x(2) = cmplx ( k%x(2)*k%x(3)/kabs, &
                   - k%x(1), kind=default) / kt / sqrt2
             e%x(3) = - kt / kabs / sqrt2
          end if
       case (0)
          if (m > 0) then
             e\%t = kabs / m
             e%x = k%t / (m*kabs) * k%x
          end if
       case (3)
          e = (0,1) * k
       case (4)
          if (m > 0) then
             e = (1 / m) * k
          else
```

```
e = (1 / k\%t) * k
        end if
     end select
  else !!! for particles in their rest frame defined to be
          !!! polarized along the 3-direction
     select case (s)
     case (1)
        e%x(1) = cmplx ( 1, 0, kind=default) / sqrt2
        e%x(2) = cmplx ( 0, 1, kind=default) / sqrt2
     case (-1)
         e\%x(1) = cmplx ( 1, 0, kind=default) / sqrt2 \\ e\%x(2) = cmplx ( 0, -1, kind=default) / sqrt2 
     case (0)
        if (m > 0) then
            e%x(3) = 1
        end if
     case (4)
        if (m > 0) then
           e = (1 / m) * k
            e = (1 / k\%t) * k
         end if
     end select
  end if
end function eps
```

Y.6 Polarization vectors revisited

```
\langle (omega_polarizations_madgraph.f90)\\
\langle \langle (Copyleft)
\text{module omega_polarizations_madgraph}
\text{ use kinds}
\text{ use constants}
\text{ use omega_vectors}
\text{ implicit none}
\text{ private}
\langle (Declaration of polarization vectors for madgraph)}
\text{ integer, parameter, public :: omega_pols_madgraph_2010_01_A = 0}
\text{ contains}
\langle (Implementation of polarization vectors for madgraph)}
\text{ end module omega_polarizations_madgraph}
\end{arrange}
\]
```

This set of polarization vectors is compatible with HELAS [5]:

$$\epsilon_1^{\mu}(k) = \frac{1}{|\vec{k}|\sqrt{k_x^2 + k_y^2}} \left(0; k_z k_x, k_y k_z, -k_x^2 - k_y^2\right) \tag{Y.15a}$$

$$\epsilon_2^{\mu}(k) = \frac{1}{\sqrt{k_x^2 + k_y^2}} (0; -k_y, k_x, 0)$$
(Y.15b)

$$\epsilon_3^{\mu}(k) = \frac{k_0}{m|\vec{k}|} \left(\vec{k}^2/k_0; k_x, k_y, k_z\right)$$
 (Y.15c)

and

$$\epsilon_{\pm}^{\mu}(k) = \frac{1}{\sqrt{2}} (\mp \epsilon_1^{\mu}(k) - i\epsilon_2^{\mu}(k))$$
 (Y.16a)

$$\epsilon_0^{\mu}(k) = \epsilon_3^{\mu}(k) \tag{Y.16b}$$

i.e.

$$\epsilon_{+}^{\mu}(k) = \frac{1}{\sqrt{2}\sqrt{k_x^2 + k_y^2}} \left(0; -\frac{k_z k_x}{|\vec{k}|} + ik_y, -\frac{k_y k_z}{|\vec{k}|} - ik_x, \frac{k_x^2 + k_y^2}{|\vec{k}|} \right)$$
 (Y.17a)

$$\epsilon_{-}^{\mu}(k) = \frac{1}{\sqrt{2}\sqrt{k_x^2 + k_y^2}} \left(0; \frac{k_z k_x}{|\vec{k}|} + ik_y, \frac{k_y k_z}{|\vec{k}|} - ik_x, -\frac{k_x^2 + k_y^2}{|\vec{k}|} \right)$$
 (Y.17b)

$$\epsilon_0^{\mu}(k) = \frac{k_0}{m|\vec{k}|} \left(\vec{k}^2/k_0; k_x, k_y, k_z\right)$$
 (Y.17c)

Fortunately, for comparing with squared matrix generated by Madgraph we can also use the modified version, since the difference is only a phase and does *not* mix helicity states. Determining the mass from the momenta is a numerically haphazardous for light particles. Therefore, we accept some redundancy and pass the mass explicitly.

```
\langle Declaration \ of \ polarization \ vectors \ for \ madgraph \rangle \equiv
  public :: eps
\langle Implementation of polarization vectors for madgraph \rangle \equiv
 pure function eps (m, k, s) result (e)
    type(vector) :: e
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: k
    integer, intent(in) :: s
    real(kind=default) :: kt, kabs, kabs2, sqrt2
    sqrt2 = sqrt (2.0_default)
    kabs2 = dot_product (k%x, k%x)
    e%t = 0
    e%x = 0
    if (kabs2 > 0) then
       kabs = sqrt (kabs2)
       select case (s)
       case (1)
          kt = sqrt (k%x(1)**2 + k%x(2)**2)
          if (abs(kt) <= epsilon(kt) * kabs) then
             if (k\%x(3) > 0) then
                e%x(1) = cmplx (-1, 0, kind=default) / sqrt2
                e%x(2) = cmplx ( 0, -1, kind=default) / sqrt2
                e%x(1) = cmplx (
                                   1, 0, kind=default) / sqrt2
                e%x(2) = cmplx (
                                   0, - 1, kind=default) / sqrt2
             end if
             e%x(1) = cmplx ( - k%x(3)*k%x(1)/kabs, &
                  k%x(2), kind=default) / kt / sqrt2
             e%x(2) = cmplx ( - k%x(2)*k%x(3)/kabs, &
                  - k%x(1), kind=default) / kt / sqrt2
             e%x(3) = kt / kabs / sqrt2
          end if
       case (-1)
          kt = sqrt (k\%x(1)**2 + k\%x(2)**2)
          if (abs(kt) <= epsilon(kt) * kabs) then
             if (k\%x(3) > 0) then
                e%x(1) = cmplx (
                                   1, 0, kind=default) / sqrt2
                e%x(2) = cmplx ( 0, -1, kind=default) / sqrt2
             else
                e%x(1) = cmplx ( -1, 0, kind=default) / sqrt2
                e%x(2) = cmplx ( 0, -1, kind=default) / sqrt2
             end if
          else
             e%x(1) = cmplx ( k%x(3)*k%x(1)/kabs, &
                  k%x(2), kind=default) / kt / sqrt2
             e%x(2) = cmplx ( k%x(2)*k%x(3)/kabs, &
                   - k%x(1), kind=default) / kt / sqrt2
             e%x(3) = - kt / kabs / sqrt2
          end if
       case (0)
          if (m > 0) then
             e\%t = kabs / m
             e%x = k%t / (m*kabs) * k%x
          end if
       case (3)
          e = (0,1) * k
       case (4)
          if (m > 0) then
```

```
e = (1 / m) * k
       else
          e = (1 / k\%t) * k
       end if
    end select
        !!! for particles in their rest frame defined to be
         !!! polarized along the 3-direction
    select case (s)
    case (1)
       e%x(1) = cmplx (-1, 0, kind=default) / sqrt2
       e%x(2) = cmplx ( 0, -1, kind=default) / sqrt2
    case (-1)
       e%x(1) = cmplx ( 1, 0, kind=default) / sqrt2
       e%x(2) = cmplx ( 0, -1, kind=default) / sqrt2
    case (0)
       if (m > 0) then
          e%x(3) = 1
       end if
    case (4)
       if (m > 0) then
          e = (1 / m) * k
          e = (1 / k\%t) * k
       end if
    end select
 end if
end function eps
```

Y.7 Symmetric Tensors

Spin-2 polarization tensors are symmetric, transversal and traceless

$$\epsilon_m^{\mu\nu}(k) = \epsilon_m^{\nu\mu}(k)$$
 (Y.18a)

$$k_{\mu}\epsilon_{m}^{\mu\nu}(k) = k_{\nu}\epsilon_{m}^{\mu\nu}(k) = 0 \tag{Y.18b}$$

$$\epsilon_{m,\mu}^{\mu}(k) = 0 \tag{Y.18c}$$

with m = 1, 2, 3, 4, 5. Our current representation is redundant and does not enforce symmetry or tracelessness.

```
\langle Copyleft \rangle
module omega_tensors
  use kinds
  use constants
  use omega_vectors
  implicit none
  private
  public :: operator (*), operator (+), operator (-), &
        operator (.tprod.)
  public :: abs, conjg, set_zero
  ⟨intrinsic :: abs⟩
  ⟨intrinsic :: conjg⟩
  type, public :: tensor
      ! private (omegalib needs access, but DON'T TOUCH IT!)
     complex(kind=default), dimension(0:3,0:3) :: t
  end type tensor
  \langle Declaration \ of \ operations \ for \ tensors \rangle
  integer, parameter, public :: omega_tensors_2010_01_A = 0
contains
  \langle Implementation \ of \ operations \ for \ tensors \rangle
end module omega_tensors
```

Y.7.1 Vector Space

```
⟨Declaration of operations for tensors⟩≡ interface set_zero
```

 $\langle omega_tensors.f90 \rangle \equiv$

```
module procedure set_zero_tensor
  end interface
 private :: set_zero_tensor
\langle Implementation \ of \ operations \ for \ tensors \rangle \equiv
 elemental subroutine set_zero_tensor (x)
    type(tensor), intent(out) :: x
    x\%t = 0
  end subroutine set_zero_tensor
                                              Scalar Multiplication
\langle Declaration \ of \ operations \ for \ tensors \rangle + \equiv
  interface operator (*)
     module procedure integer_tensor, real_tensor, double_tensor, &
           complex_tensor, dcomplex_tensor
 private :: integer_tensor, real_tensor, double_tensor
 private :: complex_tensor, dcomplex_tensor
\langle Implementation of operations for tensors \rangle + \equiv
 pure function integer_tensor (x, y) result (xy)
    \verb|integer, intent(in)| :: x
    type(tensor), intent(in) :: y
    type(tensor) :: xy
    xy\%t = x * y\%t
  end function integer_tensor
 pure function real_tensor (x, y) result (xy)
    real(kind=single), intent(in) :: x
    type(tensor), intent(in) :: y
    type(tensor) :: xy
    xy\%t = x * y\%t
  end function real_tensor
 pure function double_tensor (x, y) result (xy)
    real(kind=default), intent(in) :: x
    type(tensor), intent(in) :: y
    type(tensor) :: xy
    xy\%t = x * y\%t
  end function double_tensor
  pure function complex_tensor (x, y) result (xy)
    complex(kind=single), intent(in) :: x
    type(tensor), intent(in) :: y
    type(tensor) :: xy
    xy\%t = x * y\%t
  end function complex_tensor
 pure function dcomplex_tensor (x, y) result (xy)
    complex(kind=default), intent(in) :: x
    type(tensor), intent(in) :: y
    type(tensor) :: xy
    xy\%t = x * y\%t
  end function dcomplex_tensor
                                            Addition and Subtraction
\langle Declaration \ of \ operations \ for \ tensors \rangle + \equiv
  interface operator (+)
     module procedure plus_tensor
  end interface
 private :: plus_tensor
  interface operator (-)
    module procedure neg_tensor
  end interface
 private :: neg_tensor
\langle Implementation \ of \ operations \ for \ tensors \rangle + \equiv
 pure function plus_tensor (t1) result (t2)
    type(tensor), intent(in) :: t1
```

```
type(tensor) :: t2
    t2 = t1
  end function plus_tensor
  pure function neg_tensor (t1) result (t2)
    type(tensor), intent(in) :: t1
    type(tensor) :: t2
    t2\%t = - t1\%t
  end function neg_tensor
\langle Declaration \ of \ operations \ for \ tensors \rangle + \equiv
  interface operator (+)
     module procedure add_tensor
  end interface
  private :: add_tensor
  interface operator (-)
     module procedure sub_tensor
  end interface
  private :: sub_tensor
\langle Implementation \ of \ operations \ for \ tensors \rangle + \equiv
  pure function add_tensor (x, y) result (xy)
    type(tensor), intent(in) :: x, y
    type(tensor) :: xy
    xy\%t = x\%t + y\%t
  end function add_tensor
  pure function sub_tensor (x, y) result (xy)
    type(tensor), intent(in) :: x, y
    type(tensor) :: xy
    xy\%t = x\%t - y\%t
  end function sub_tensor
\langle Declaration \ of \ operations \ for \ tensors \rangle + \equiv
  interface operator (.tprod.)
     module procedure out_prod_vv, out_prod_vm, &
           out_prod_mv, out_prod_mm
  end interface
  private :: out_prod_vv, out_prod_vm, &
        out_prod_mv, out_prod_mm
\langle Implementation of operations for tensors \rangle + \equiv
  pure function out_prod_vv (v, w) result (t)
    type(tensor) :: t
    type(vector), intent(in) :: v, w
    integer :: i, j
    t\%t(0,0) = v\%t * w\%t
    t\%t(0,1:3) = v\%t * w\%x
    t\%t(1:3,0) = v\%x * w\%t
    do i = 1, 3
       do j = 1, 3
           t\%t(i,j) = v\%x(i) * w\%x(j)
        end do
    end do
  end function out_prod_vv
\langle Implementation \ of \ operations \ for \ tensors \rangle + \equiv
  pure function out_prod_vm (v, m) result (t)
    type(tensor) :: t
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: m
    integer :: i, j
    t\%t(0,0) = v\%t * m\%t
    t\%t(0,1:3) = v\%t * m\%x
    t\%t(1:3,0) = v\%x * m\%t
    do i = 1, 3
       do j = 1, 3
           t\%t(i,j) = v\%x(i) * m\%x(j)
       end do
    end do
  end function out_prod_vm
```

```
\langle Implementation \ of \ operations \ for \ tensors \rangle + \equiv
  pure function out_prod_mv (m, v) result (t)
    type(tensor) :: t
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: m
    integer :: i, j
    t\%t(0,0) = m\%t * v\%t
    t\%t(0,1:3) = m\%t * v\%x
    t\%t(1:3,0) = m\%x * v\%t
    do i = 1, 3
        do j = 1, 3
           t\%t(i,j) = m\%x(i) * v\%x(j)
        end do
    end do
  end function out_prod_mv
\langle Implementation\ of\ operations\ for\ tensors \rangle + \equiv
  pure function out_prod_mm (m, n) result (t)
    type(tensor) :: t
    type(momentum), intent(in) :: m, n
    integer :: i, j
    t\%t(0,0) = m\%t * n\%t
    t\%t(0,1:3) = m\%t * n\%x
    t\%t(1:3,0) = m\%x * n\%t
    do i = 1, 3
        do j = 1, 3
           t\%t(i,j) = m\%x(i) * n\%x(j)
        end do
    end do
  end function out_prod_mm
\langle Declaration \ of \ operations \ for \ tensors \rangle + \equiv
  interface abs
     module procedure abs_tensor
  end interface
  private :: abs_tensor
\langle Implementation\ of\ operations\ for\ tensors \rangle + \equiv
  pure function abs_tensor (t) result (abs_t)
    type(tensor), intent(in) :: t
    real(kind=default) :: abs_t
    abs_t = sqrt (sum ((abs (t%t))**2))
  end function abs_tensor
\langle Declaration \ of \ operations \ for \ tensors \rangle + \equiv
  interface conjg
     module procedure conjg_tensor
  end interface
  private :: conjg_tensor
\langle Implementation\ of\ operations\ for\ tensors \rangle + \equiv
  pure function conjg_tensor (t) result (conjg_t)
    type(tensor), intent(in) :: t
    type(tensor) :: conjg_t
    conjg_t%t = conjg (t%t)
  end function conjg_tensor
\langle Declaration \ of \ operations \ for \ tensors \rangle + \equiv
  interface operator (*)
     module procedure tensor_tensor, vector_tensor, tensor_vector, &
           momentum_tensor, tensor_momentum
  end interface
  private :: tensor_tensor, vector_tensor, tensor_vector, &
        momentum_tensor, tensor_momentum
\langle Implementation \ of \ operations \ for \ tensors \rangle + \equiv
  pure function tensor_tensor (t1, t2) result (t1t2)
    type(tensor), intent(in) :: t1
    type(tensor), intent(in) :: t2
    complex(kind=default) :: t1t2
```

```
integer :: i1, i2
    t1t2 = t1\%t(0,0)*t2\%t(0,0) &
          - dot_product (conjg (t1%t(0,1:)), t2%t(0,1:)) &
          - dot_product (conjg (t1%t(1:,0)), t2%t(1:,0))
    do i1 = 1, 3
       do i2 = 1, 3
          t1t2 = t1t2 + t1\%t(i1,i2)*t2\%t(i1,i2)
       end do
    end do
  end function tensor_tensor
\langle Implementation \ of \ operations \ for \ tensors \rangle + \equiv
 pure function tensor_vector (t, v) result (tv)
    type(tensor), intent(in) :: t
    type(vector), intent(in) :: v
    type(vector) :: tv
               t\%t(0,0) * v\%t - dot_product (conjg (t\%t(0,1:)), v\%x)
    tv%x(1) = t%t(0,1) * v%t - dot_product (conjg (t%t(1,1:)), v%x)
    tv%x(2) = t%t(0,2) * v%t - dot_product (conjg (t%t(2,1:)), v%x)
    tv%x(3) = t%t(0,3) * v%t - dot_product (conjg (t%t(3,1:)), v%x)
  end function tensor_vector
\langle Implementation \ of \ operations \ for \ tensors \rangle + \equiv
 pure function vector_tensor (v, t) result (vt)
    type(vector), intent(in) :: v
    type(tensor), intent(in) :: t
    type(vector) :: vt
             v%t * t%t(0,0) - dot_product (conjg (v%x), t%t(1:,0))
    vt%x(1) = v%t * t%t(0,1) - dot_product (conjg (v%x), t%t(1:,1))
    vt%x(2) = v%t * t%t(0,2) - dot_product (conjg (v%x), t%t(1:,2))
    vt%x(3) = v%t * t%t(0,3) - dot_product (conjg (v%x), t%t(1:,3))
  end function vector_tensor
\langle Implementation \ of \ operations \ for \ tensors \rangle + \equiv
 pure function tensor_momentum (t, p) result (tp)
    type(tensor), intent(in) :: t
    type(momentum), intent(in) :: p
    type(vector) :: tp
               t%t(0,0) * p%t - dot_product (conjg (t%t(0,1:)), p%x)
    tp%x(1) = t%t(0,1) * p%t - dot_product (conjg (t%t(1,1:)), p%x)
    tp%x(2) = t%t(0,2) * p%t - dot_product (conjg (t%t(2,1:)), p%x)
    tp%x(3) = t%t(0,3) * p%t - dot_product (conjg (t%t(3,1:)), p%x)
  end function tensor_momentum
\langle Implementation \ of \ operations \ for \ tensors \rangle + \equiv
 pure function momentum_tensor (p, t) result (pt)
    type(momentum), intent(in) :: p
    type(tensor), intent(in) :: t
    type(vector) :: pt
    pt%t =
               p%t * t%t(0,0) - dot_product (p%x, t%t(1:,0))
    pt%x(1) = p%t * t%t(0,1) - dot_product (p%x, t%t(1:,1))
    pt\%x(2) = p\%t * t\%t(0,2) - dot_product (p\%x, t\%t(1:,2))

pt\%x(3) = p\%t * t\%t(0,3) - dot_product (p\%x, t\%t(1:,3))
  end function momentum_tensor
```

Y.8 Symmetric Polarization Tensors

$$\epsilon_{+2}^{\mu\nu}(k) = \epsilon_{+}^{\mu}(k)\epsilon_{+}^{\nu}(k) \tag{Y.19a}$$

$$\epsilon_{+1}^{\mu\nu}(k) = \frac{1}{\sqrt{2}} \left(\epsilon_{+}^{\mu}(k) \epsilon_{0}^{\nu}(k) + \epsilon_{0}^{\mu}(k) \epsilon_{+}^{\nu}(k) \right)$$
 (Y.19b)

$$\epsilon_0^{\mu\nu}(k) = \frac{1}{\sqrt{6}} \left(\epsilon_+^{\mu}(k) \epsilon_-^{\nu}(k) + \epsilon_-^{\mu}(k) \epsilon_+^{\nu}(k) - 2\epsilon_0^{\mu}(k) \epsilon_0^{\nu}(k) \right) \tag{Y.19c}$$

$$\epsilon_{-1}^{\mu\nu}(k) = \frac{1}{\sqrt{2}} \left(\epsilon_{-}^{\mu}(k) \epsilon_{0}^{\nu}(k) + \epsilon_{0}^{\mu}(k) \epsilon_{-}^{\nu}(k) \right) \tag{Y.19d}$$

$$\epsilon_{-2}^{\mu\nu}(k) = \epsilon_{-}^{\mu}(k)\epsilon_{-}^{\nu}(k) \tag{Y.19e}$$

```
Note that \epsilon_{\pm 2,\mu}^{\mu}(k) = \epsilon_{\pm}^{\mu}(k)\epsilon_{\pm,\mu}(k) \propto \epsilon_{\pm}^{\mu}(k)\epsilon_{\mp,\mu}^{*}(k) = 0 and that the sign in \epsilon_{0}^{\mu\nu}(k) insures its tracelessness<sup>1</sup>.
\langle omega\_tensor\_polarizations.f90 \rangle \equiv
   \langle Copyleft \rangle
  module omega_tensor_polarizations
     use kinds
     use constants
     use omega_vectors
     use omega_tensors
     use omega_polarizations
     implicit none
     private
     \langle Declaration \ of \ polarization \ tensors \rangle
     integer, parameter, public :: omega_tensor_pols_2010_01_A = 0
  contains
     \langle Implementation \ of \ polarization \ tensors \rangle
  end module omega_tensor_polarizations
\langle Declaration \ of \ polarization \ tensors \rangle \equiv
  public :: eps2
\langle Implementation \ of \ polarization \ tensors \rangle \equiv
  pure function eps2 (m, k, s) result (t)
     type(tensor) :: t
     real(kind=default), intent(in) :: m
     type(momentum), intent(in) :: k
     integer, intent(in) :: s
     type(vector) :: ep, em, e0
     t%t = 0
     select case (s)
     case (2)
         ep = eps (m, k, 1)
        t = ep.tprod.ep
     case (1)
         ep = eps (m, k, 1)
         e0 = eps (m, k, 0)
         t = (1 / sqrt (2.0_default)) &
               * ((ep.tprod.e0) + (e0.tprod.ep))
     case (0)
         ep = eps (m, k, 1)
         e0 = eps (m, k, 0)
         em = eps (m, k, -1)
         t = (1 / sqrt (6.0_default)) &
                * ((ep.tprod.em) + (em.tprod.ep) - 2*(e0.tprod.e0))
     case (-1)
         e0 = eps (m, k, 0)
         em = eps (m, k, -1)
         t = (1 / sqrt (2.0_default)) &
                * ((em.tprod.e0) + (e0.tprod.em))
     case (-2)
         em = eps (m, k, -1)
         t = em.tprod.em
     end select
   end function eps2
                                                    Y.9 Couplings
\langle omega\_couplings.f90 \rangle \equiv
   \langle Copyleft \rangle
  module omega_couplings
```

```
use kinds
use constants
```

$$L_{-}^{2}|++\rangle = 2e^{2i\phi}|00\rangle + e^{i(\phi+\chi)}(|+-\rangle + |-+\rangle)$$

¹On the other hand, with the shift operator $L_- |+\rangle = e^{i\phi} |0\rangle$ and $L_- |0\rangle = e^{i\chi} |-\rangle$, we find

i. e. $\chi - \phi = \pi$, if we want to identify $\epsilon^{\mu}_{-,0,+}$ with $|-,0,+\rangle$.

```
use omega_vectors
    use omega_tensors
    implicit none
    private
     \langle Declaration \ of \ couplings \rangle
     \langle Declaration \ of \ propagators \rangle
    integer, parameter, public :: omega_couplings_2010_01_A = 0
  contains
     \langle Implementation \ of \ couplings \rangle
     (Implementation of propagators)
  end module omega_couplings
\langle Declaration \ of \ propagators \rangle \equiv
  public :: wd_tl
\langle Declaration \ of \ propagators \rangle + \equiv
  public :: wd_run
\langle Declaration \ of \ propagators \rangle + \equiv
  public :: gauss
                                                          \Theta(p^2)\Gamma
                                                                                                                      (Y.20)
\langle Implementation \ of \ propagators \rangle \equiv
  pure function wd_tl (p, w) result (width)
    real(kind=default) :: width
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: w
    if (p*p > 0) then
        width = w
    else
        width = 0
    end if
  end function wd_tl
                                                                                                                      (Y.21)
\langle Implementation\ of\ propagators \rangle + \equiv
  pure function wd_run (p, m, w) result (width)
    real(kind=default) :: width
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m
    real(kind=default), intent(in) :: w
    if (p*p > 0) then
        width = w * (p*p) / m**2
    else
        width = 0
    end if
  end function wd_run
\langle Implementation \ of \ propagators \rangle + \equiv
  pure function gauss (x, mu, w) result (gg)
    real(kind=default) :: gg
    real(kind=default), intent(in) :: x, mu, w
    if (w > 0) then
       gg = exp(-(x - mu**2)**2/4.0_default/mu**2/w**2) * &
             sqrt(sqrt(PI/2)) / w / mu
       else
       gg = 1.0_default
    \quad \text{end if} \quad
  end function gauss
\langle Declaration \ of \ propagators \rangle + \equiv
  public :: pr_phi, pr_unitarity, pr_feynman, pr_gauge, pr_rxi
  public :: pr_vector_pure
  public :: pj_phi, pj_unitarity
  public :: pg_phi, pg_unitarity
```

$$\frac{\mathrm{i}}{p^2 - m^2 + \mathrm{i}m\Gamma} \phi \tag{Y.22}$$

(Implementation of propagators)+=
pure function pr_phi (p, m, w, phi) result (pphi)
 complex(kind=default) :: pphi
 type(momentum), intent(in) :: p
 real(kind=default), intent(in) :: m, w
 complex(kind=default), intent(in) :: phi
 pphi = (1 / cmplx (p*p - m**2, m*w, kind=default)) * phi
 end function pr_phi

$$\sqrt{\frac{\pi}{M\Gamma}}\phi$$
 (Y.23)

\langle Implementation of propagators\rangle +=
 pure function pj_phi (m, w, phi) result (pphi)
 complex(kind=default) :: pphi
 real(kind=default), intent(in) :: m, w
 complex(kind=default), intent(in) :: phi
 pphi = (0, -1) * sqrt (PI / m / w) * phi
 end function pj_phi

\(\langle Implementation of propagators \rangle +=
 pure function pg_phi (p, m, w, phi) result (pphi)
 complex(kind=default) :: pphi
 type(momentum), intent(in) :: p
 real(kind=default), intent(in) :: m, w
 complex(kind=default), intent(in) :: phi
 pphi = ((0, 1) * gauss (p*p, m, w)) * phi
 end function pg_phi

$$\frac{\mathrm{i}}{p^2 - m^2 + \mathrm{i}m\Gamma} \left(-g_{\mu\nu} + \frac{p_{\mu}p_{\nu}}{m^2} \right) \epsilon^{\nu}(p) \tag{Y.24}$$

NB: the explicit cast to vector is required here, because a specific complex_momentum procedure for operator (*) would introduce ambiguities. NB: we used to use the constructor vector (p%t, p%x) instead of the temporary variable, but the Intel Fortran Compiler choked on it.

```
\langle \mathit{Implementation of propagators} \rangle + \equiv
```

```
pure function pr_unitarity (p, m, w, cms, e) result (pe)
  type(vector) :: pe
  type(momentum), intent(in) :: p
  real(kind=default), intent(in) :: m, w
  type(vector), intent(in) :: e
  logical, intent(in) :: cms
  type(vector) :: pv
  complex(kind=default) :: c_mass2
  pv = p
  if (cms) then
     c_mass2 = cmplx (m**2, -m*w, kind=default)
     c_{mass2} = m**2
  end if
  pe = - (1 / cmplx (p*p - m**2, m*w, kind=default)) &
       * (e - (p*e / c_mass2) * pv)
end function pr_unitarity
```

$$\sqrt{\frac{\pi}{M\Gamma}} \left(-g_{\mu\nu} + \frac{p_{\mu}p_{\nu}}{m^2} \right) \epsilon^{\nu}(p) \tag{Y.25}$$

 $\langle Implementation \ of \ propagators \rangle + \equiv$

```
pure function pj_unitarity (p, m, w, e) result (pe)
  type(vector) :: pe
  type(momentum), intent(in) :: p
  real(kind=default), intent(in) :: m, w
  type(vector), intent(in) :: e
  type(vector) :: pv
```

```
pe = (0, 1) * sqrt (PI / m / w) * (e - (p*e / m**2) * pv)
  end function pj_unitarity
\langle Implementation \ of \ propagators \rangle + \equiv
  pure function pg_unitarity (p, m, w, e) result (pe)
    type(vector) :: pe
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(vector), intent(in) :: e
    type(vector) :: pv
    pv = p
    pe = - gauss (p*p, m, w) &
           * (e - (p*e / m**2) * pv)
  end function pg_unitarity
                                                           \frac{-i}{p^2}\epsilon^{\nu}(p)
                                                                                                                         (Y.26)
\langle Implementation \ of \ propagators \rangle + \equiv
  pure function pr_feynman (p, e) result (pe)
    type(vector) :: pe
    type(momentum), intent(in) :: p
    type(vector), intent(in) :: e
    pe = - (1 / (p*p)) * e
  end function pr_feynman
                                             \frac{\mathrm{i}}{p^2} \left( -g_{\mu\nu} + (1-\xi) \frac{p_{\mu}p_{\nu}}{p^2} \right) \epsilon^{\nu}(p)
                                                                                                                         (Y.27)
\langle Implementation \ of \ propagators \rangle + \equiv
  pure function pr_gauge (p, xi, e) result (pe)
    type(vector) :: pe
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: xi
    type(vector), intent(in) :: e
    real(kind=default) :: p2
    type(vector) :: pv
    p2 = p*p
    pv = p
    pe = -(1 / p2) * (e - ((1 - xi) * (p*e) / p2) * pv)
  end function pr_gauge
                                   \frac{\mathrm{i}}{p^2 - m^2 + \mathrm{i} m \Gamma} \left( -g_{\mu\nu} + (1 - \xi) \frac{p_{\mu} p_{\nu}}{p^2 - \xi m^2} \right) \epsilon^{\nu}(p)
                                                                                                                         (Y.28)
\langle Implementation \ of \ propagators \rangle + \equiv
  pure function pr_rxi (p, m, w, xi, e) result (pe)
    type(vector) :: pe
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w, xi
    type(vector), intent(in) :: e
    real(kind=default) :: p2
    type(vector) :: pv
    p2 = p*p
    pv = p
    pe = - (1 / cmplx (p2 - m**2, m*w, kind=default)) &
           * (e - ((1 - xi) * (p*e) / (p2 - xi * m**2)) * pv)
  end function pr_rxi
                                                \frac{\mathrm{i}}{p^2 - m^2 + \mathrm{i} m \Gamma} \left( -g_{\mu\nu} \right) \epsilon^{\nu}(p)
                                                                                                                         (Y.29)
\langle Implementation \ of \ propagators \rangle + \equiv
  pure function pr_vector_pure (p, m, w, e) result (pe)
    type(vector) :: pe
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
```

```
real(kind=default) :: p2  
type(vector) :: pv  
p2 = p*p  
pv = p  
pe = - (1 / cmplx (p2 - m**2, m*w, kind=default)) * e  
end function pr_vector_pure  
\langle Declaration \ of \ propagators \rangle + \equiv   
public :: pr_tensor, pr_tensor_pure  
\frac{\mathrm{i} P^{\mu\nu,\rho\sigma}(p,m)}{p^2 - m^2 + \mathrm{i} m\Gamma} T_{\rho\sigma} \tag{Y.30a}
```

with

$$\begin{split} P^{\mu\nu,\rho\sigma}(p,m) &= \frac{1}{2} \left(g^{\mu\rho} - \frac{p^{\mu}p^{\nu}}{m^2} \right) \left(g^{\nu\sigma} - \frac{p^{\nu}p^{\sigma}}{m^2} \right) + \frac{1}{2} \left(g^{\mu\sigma} - \frac{p^{\mu}p^{\sigma}}{m^2} \right) \left(g^{\nu\rho} - \frac{p^{\nu}p^{\rho}}{m^2} \right) \\ &- \frac{1}{3} \left(g^{\mu\nu} - \frac{p^{\mu}p^{\nu}}{m^2} \right) \left(g^{\rho\sigma} - \frac{p^{\rho}p^{\sigma}}{m^2} \right) \end{split} \tag{Y.30b}$$

Be careful with raising and lowering of indices:

type(vector), intent(in) :: e

$$g^{\mu\nu} - \frac{k^{\mu}k^{\nu}}{m^2} = \begin{pmatrix} 1 - k^0k^0/m^2 & -k^0\vec{k}/m^2 \\ -\vec{k}k^0/m^2 & -\mathbf{1} - \vec{k} \otimes \vec{k}/m^2 \end{pmatrix}$$
(Y.31a)

$$g^{\mu}_{\ \nu} - \frac{k^{\mu}k_{\nu}}{m^2} = \begin{pmatrix} 1 - k^0k^0/m^2 & k^0\vec{k}/m^2 \\ -\vec{k}k^0/m^2 & \mathbf{1} + \vec{k} \otimes \vec{k}/m^2 \end{pmatrix}$$
 (Y.31b)

```
\langle Implementation \ of \ propagators \rangle + \equiv
 pure function pr_tensor (p, m, w, t) result (pt)
   type(tensor) :: pt
   type(momentum), intent(in) :: p
   real(kind=default), intent(in) :: m, w
   type(tensor), intent(in) :: t
   complex(kind=default) :: p_dd_t
   real(kind=default), dimension(0:3,0:3) :: p_uu, p_ud, p_du, p_dd
   integer :: i, j
   p_uu(0,0) = 1 - p%t * p%t / m**2
   p_uu(0,1:3) = -p\%t * p\%x / m**2
   p_uu(1:3,0) = p_uu(0,1:3)
   do i = 1, 3
       do j = 1, 3
         p_u(i,j) = -p_x(i) * p_x(j) / m**2
   end do
   do i = 1, 3
      p_u(i,i) = -1 + p_u(i,i)
   end do
   p_ud(:,0) = p_uu(:,0)
   p_ud(:,1:3) = -p_uu(:,1:3)
   p_du = transpose (p_ud)
   p_dd(:,0) = p_du(:,0)
   p_dd(:,1:3) = -p_du(:,1:3)
   p_dd_t = 0
   do i = 0, 3
       do j = 0, 3
         p_dd_t = p_dd_t + p_dd(i,j) * t%t(i,j)
       end do
   end do
   pt%t = matmul (p_ud, matmul (0.5_default * (t%t + transpose (t%t)), p_du)) &
         - (p_dd_t / 3.0_default) * p_uu
   pt%t = pt%t / cmplx (p*p - m**2, m*w, kind=default)
  end function pr_tensor
```

$$\frac{\mathrm{i}P_p^{\mu\nu,\rho\sigma}}{p^2 - m^2 + \mathrm{i}m\Gamma} T_{\rho\sigma} \tag{Y.32a}$$

with

```
\frac{1}{2}g^{\mu\rho}g^{\nu\sigma} + \frac{1}{2}g^{\mu\sigma}g^{\nu\rho} - \frac{1}{2}g^{\mu\nu}g^{\rho\sigma} (Y.32b)
\langle Implementation \ of \ propagators \rangle + \equiv
  pure function pr_tensor_pure (p, m, w, t) result (pt)
    type(tensor) :: pt
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(tensor), intent(in) :: t
    complex(kind=default) :: p_dd_t
    real(kind=default), dimension(0:3,0:3) :: g_uu
    integer :: i, j
    g_uu(0,0) = 1
    g_uu(0,1:3) = 0
    g_uu(1:3,0) = g_uu(0,1:3)
    do i = 1, 3
        do j = 1, 3
           g_uu(i,j) = 0
        end do
    end do
    do i = 1, 3
        g_u(i,i) = -1
    p_dd_t = t\%t(0,0) - t\%t(1,1) - t\%t(2,2) - t\%t(3,3)
    pt\%t = 0.5_default * ((t\%t + transpose (t\%t)) &
          - p_dd_t * g_uu )
    pt\%t = pt\%t / cmplx (p*p - m**2, m*w, kind=default)
  end function pr_tensor_pure
                                         Y.9.1 Triple Gauge Couplings
\langle Declaration \ of \ couplings \rangle \equiv
  public :: g_gg
According to (9.6c)
  A^{a,\mu}(k_1+k_2) = -ig((k_1^{\mu}-k_2^{\mu})A^{a_1}(k_1)\cdot A^{a_2}(k_2)
                                          +(2k_2+k_1)\cdot A^{a_1}(k_1)A^{a_2,\mu}(k_2)-A^{a_1,\mu}(k_1)A^{a_2}(k_2)\cdot (2k_1+k_2) (Y.33)
\langle Implementation \ of \ couplings \rangle \equiv
  pure function g_gg (g, a1, k1, a2, k2) result (a)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: a1, a2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: a
    a = (0, -1) * g * ((k1 - k2) * (a1 * a2) &
                            + ((2*k2 + k1) * a1) * a2 - a1 * ((2*k1 + k2) * a2))
  end function g_gg
                                      Y.9.2 Quadruple Gauge Couplings
\langle Declaration \ of \ couplings \rangle + \equiv
  public :: x_gg, g_gx
                           T^{a,\mu\nu}(k_1+k_2) = g(A^{a_1,\mu}(k_1)A^{a_2,\nu}(k_2) - A^{a_1,\nu}(k_1)A^{a_2,\mu}(k_2))
                                                                                                                   (Y.34)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function x_gg (g, a1, a2) result (x)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: a1, a2
    type(tensor2odd) :: x
    x = g * (a1 .wedge. a2)
  end function x_gg
```

```
A^{a,\mu}(k_1+k_2)=qA^{a_1}_{\nu}(k_1)T^{a_2,\nu\mu}(k_2)
                                                                                                                   (Y.35)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function g_gx (g, a1, x) result (a)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: a1
    type(tensor2odd), intent(in) :: x
    type(vector) :: a
    a = g * (a1 * x)
  end function g_gx
                                               Y.9.3 Scalar Current
\langle Declaration \ of \ couplings \rangle + \equiv
  public :: v_ss, s_vs
                                       V^{\mu}(k_1 + k_2) = g(k_1^{\mu} - k_2^{\mu})\phi_1(k_1)\phi_2(k_2)
                                                                                                                   (Y.36)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function v_ss (g, phi1, k1, phi2, k2) result (v)
    complex(kind=default), intent(in) :: g, phi1, phi2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: v
    v = (k1 - k2) * (g * phi1 * phi2)
  end function v_ss
                                        \phi(k_1 + k_2) = g(k_1^{\mu} + 2k_2^{\mu})V_{\mu}(k_1)\phi(k_2)
                                                                                                                   (Y.37)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function s_vs (g, v1, k1, phi2, k2) result (phi)
    complex(kind=default), intent(in) :: g, phi2
    type(vector), intent(in) :: v1
    type(momentum), intent(in) :: k1, k2
    complex(kind=default) :: phi
    phi = g * ((k1 + 2*k2) * v1) * phi2
  end function s_vs
                                 Y.9.4
                                           Transversal Scalar-Vector Coupling
\langle Declaration \ of \ couplings \rangle + \equiv
  public :: s_vv_t, v_sv_t
                          phi(k_1 + k_2) = g((V_1(k_1)V_2(k_2))(k_1k_2) - (V_1(k_1)k_2)(V_2(k_2)k_1))
                                                                                                                   (Y.38)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function s_vv_t (g, v1, k1, v2, k2) result (phi)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    complex(kind=default) :: phi
    phi = g * ((v1*v2) * (k1*k2) - (v1*k2) * (v2*k1))
  end function s_vv_t
                            V_1^{\mu}(k_{\phi} + k_V) = gphi(((k_{\phi} + k_V)k_V)V_2^{\mu} - (k_{\phi} + k_V)V_2)k_V^{\mu})
                                                                                                                   (Y.39)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function v_sv_t (g, phi, kphi, v, kv) result (vout)
    complex(kind=default), intent(in) :: g, phi
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: kv, kphi
    type(momentum) :: kout
    type(vector) :: vout
    kout = - (kv + kphi)
    vout = g * phi * ((kout*kv) * v - (v * kout) * kv)
  end function v_sv_t
```

Y.9.5 Transversal TensorScalar-Vector Coupling

```
\langle Declaration \ of \ couplings \rangle + \equiv
  public :: tphi_vv, tphi_vv_cf, v_tphiv, v_tphiv_cf
                              phi(k_1 + k_2) = g(V_1(k_1)(k_1 + k_2)) * (V_2(k_2)(k_1 + k_2))
                                                                                                                (Y.40)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function tphi_vv (g, v1, k1, v2, k2) result (phi)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    complex(kind=default) :: phi
    type(momentum) :: k
    k = - (k1 + k2)
    phi = 2 * g * (v1*k) * (v2*k)
  end function tphi_vv
                                    phi(k_1 + k_2) = g((V_1(k_1)V_2(k_2))(k_1 + k_2)^2)
                                                                                                                (Y.41)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function tphi_vv_cf (g, v1, k1, v2, k2) result (phi)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    complex(kind=default) :: phi
    type(momentum) :: k
    k = - (k1 + k2)
    phi = - g/2 * (v1*v2) * (k*k)
  end function tphi_vv_cf
                                   V_1^{\mu}(k_{\phi} + k_V) = gphi((k_{\phi} + k_V)V_2)(k_{\phi} + k_V)^{\mu}
                                                                                                                (Y.42)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function v_tphiv (g, phi, kphi,v, kv) result (vout)
    complex(kind=default), intent(in) :: g, phi
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: kv, kphi
    type(momentum) :: kout
    type(vector) :: vout
    kout = - (kv + kphi)
    vout = 2 * g * phi * ((v * kout) * kout)
  end function v_tphiv
                                   V_1^{\mu}(k_{\phi} + k_V) = gphi((k_{\phi} + k_V)(k_{\phi} + k_V))V_2^{\mu}
                                                                                                                (Y.43)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function v_tphiv_cf (g, phi, kphi,v, kv) result (vout)
    {\tt complex(kind=default),\ intent(in)\ ::\ g,\ phi}
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: kv, kphi
    type(momentum) :: kout
    type(vector) :: vout
    kout = - (kv + kphi)
    vout = -g/2 * phi * (kout*kout) * v
  end function v_tphiv_cf
                                       Y.9.6 Triple Vector Couplings
\langle Declaration \ of \ couplings \rangle + \equiv
  public :: tkv_vv, lkv_vv, tv_kvv, lv_kvv, kg_kgkg
  public :: t5kv_vv, 15kv_vv, t5v_kvv, 15v_kvv, kg5_kgkg, kg_kg5kg
  public :: dv_vv, v_dvv, dv_vv_cf, v_dvv_cf
```

```
V^{\mu}(k_1 + k_2) = iq(k_1 - k_2)^{\mu}V_1^{\nu}(k_1)V_{2\nu}(k_2)
                                                                                                                         (Y.44)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function tkv_vv (g, v1, k1, v2, k2) result (v)
     complex(kind=default), intent(in) :: g
     type(vector), intent(in) :: v1, v2
     type(momentum), intent(in) :: k1, k2
     type(vector) :: v
     v = (k1 - k2) * ((0, 1) * g * (v1*v2))
  end function tkv_vv
                                    V^{\mu}(k_1 + k_2) = ig\epsilon^{\mu\nu\rho\sigma}(k_1 - k_2)_{\nu}V_{1,\rho}(k_1)V_{2,\sigma}(k_2)
                                                                                                                         (Y.45)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function t5kv_vv (g, v1, k1, v2, k2) result (v)
     complex(kind=default), intent(in) :: g
     type(vector), intent(in) :: v1, v2
     type(momentum), intent(in) :: k1, k2
     type(vector) :: v
     type(vector) :: k
     k = k1 - k2
     v = (0, 1) * g * pseudo_vector (k, v1, v2)
  end function t5kv_vv
                                       V^{\mu}(k_1 + k_2) = ig(k_1 + k_2)^{\mu}V_1^{\nu}(k_1)V_{2,\nu}(k_2)
                                                                                                                         (Y.46)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function lkv_vv (g, v1, k1, v2, k2) result (v)
     complex(kind=default), intent(in) :: g
     type(vector), intent(in) :: v1, v2
     type(momentum), intent(in) :: k1, k2
     type(vector) :: v
     v = (k1 + k2) * ((0, 1) * g * (v1*v2))
  end function lkv_vv
                                    V^{\mu}(k_1 + k_2) = ig\epsilon^{\mu\nu\rho\sigma}(k_1 + k_2)_{\nu}V_{1,\rho}(k_1)V_{2,\sigma}(k_2)
                                                                                                                         (Y.47)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function 15kv_vv (g, v1, k1, v2, k2) result (v)
     complex(kind=default), intent(in) :: g
     type(vector), intent(in) :: v1, v2
     type(momentum), intent(in) :: k1, k2
     type(vector) :: v
     type(vector) :: k
     k = k1 + k2
     v = (0, 1) * g * pseudo_vector (k, v1, v2)
  end function 15kv_vv
                      V^{\mu}(k_1 + k_2) = ig(k_2 - k)^{\nu} V_{1,\nu}(k_1) V_2^{\mu}(k_2) = ig(2k_2 + k_1)^{\nu} V_{1,\nu}(k_1) V_2^{\mu}(k_2)
                                                                                                                         (Y.48)
using k = -k_1 - k_2
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function tv_kvv (g, v1, k1, v2, k2) result (v)
     complex(kind=default), intent(in) :: g
     type(vector), intent(in) :: v1, v2
     type(momentum), intent(in) :: k1, k2
     type(vector) :: v
     v = v2 * ((0, 1) * g * ((2*k2 + k1)*v1))
  end function tv_kvv
                                   V^{\mu}(k_1 + k_2) = ig\epsilon^{\mu\nu\rho\sigma}(2k_2 + k_1)_{\nu}V_{1,\rho}(k_1)V_{2,\sigma}(k_2)
                                                                                                                         (Y.49)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function t5v_kvv (g, v1, k1, v2, k2) result (v)
     complex(kind=default), intent(in) :: g
```

```
type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: v
    type(vector) :: k
    k = k1 + 2*k2
    v = (0, 1) * g * pseudo_vector (k, v1, v2)
  end function t5v_kvv
                                         V^{\mu}(k_1 + k_2) = -igk_1^{\nu}V_{1,\nu}(k_1)V_2^{\mu}(k_2)
                                                                                                                  (Y.50)
using k = -k_1 - k_2
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function lv_kvv (g, v1, k1, v2) result (v)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1
    type(vector) :: v
    v = v2 * ((0, -1) * g * (k1*v1))
  end function lv_kvv
                                     V^{\mu}(k_1 + k_2) = -ig\epsilon^{\mu\nu\rho\sigma}k_{1,\nu}V_{1,\rho}(k_1)V_{2,\sigma}(k_2)
                                                                                                                  (Y.51)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function 15v_kvv (g, v1, k1, v2) result (v)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1
    type(vector) :: v
    type(vector) :: k
    k = k1
    v = (0, -1) * g * pseudo_vector (k, v1, v2)
  end function 15v_kvv
                          A^{\mu}(k_1 + k_2) = igk^{\nu} \left( F_{1,\nu}^{\rho}(k_1) F_{2,\rho\mu}(k_2) - F_{1,\mu}^{\rho}(k_1) F_{2,\rho\nu}(k_2) \right)
                                                                                                                  (Y.52)
with k = -k_1 - k_2, i. e.
  A^{\mu}(k_1 + k_2) = -ig([(k_2)(k_1 A_2) - (k_1 k_2)(k_2)]A_1^{\mu}
                                          + [(k_1k_2)(kA_1) - (kk_1)(k_2A_1)]A_2^{\mu}
                                          + [(k_2A_1)(kA_2) - (kk_2)(A_1A_2)]k_1^{\mu}
                                                                         + [(kk_1)(A_1A_2) - (kA_1)(k_1A_2)]k_2^{\mu}
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function kg_kgkg (g, a1, k1, a2, k2) result (a)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: a1, a2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: a
    real(kind=default) :: k1k1, k2k2, k1k2, kk1, kk2
    complex(kind=default) :: a1a2, k2a1, ka1, k1a2, ka2
    k1k1 = k1 * k1
    k1k2 = k1 * k2
    k2k2 = k2 * k2
    kk1 = k1k1 + k1k2
    kk2 = k1k2 + k2k2
    k2a1 = k2 * a1
    ka1 = k2a1 + k1 * a1
    k1a2 = k1 * a2
    ka2 = k1a2 + k2 * a2
    a1a2 = a1 * a2
    a = (0, -1) * g * (
                              (kk2 * k1a2 - k1k2 * ka2) * a1 &
                            + (k1k2 * ka1 - kk1 * k2a1) * a2 &
                            + (ka2 * k2a1 - kk2 * a1a2) * k1 &
                            + (kk1 * a1a2 - ka1 * k1a2) * k2 )
  end function kg_kgkg
```

$$A^{\mu}(k_1 + k_2) = ige^{\mu\nu\rho\sigma}k_{\nu}F_{1,\rho}{}^{\lambda}(k_1)F_{2,\lambda\sigma}(k_2)$$
 (Y.54)

with $k = -k_1 - k_2$, i. e.

$$A^{\mu}(k_1+k_2) = -2\mathrm{i}g\epsilon^{\mu\nu\rho\sigma}k_{\nu}\Big((k_2A_1)k_{1,\rho}A_{2,\sigma} + (k_1A_2)A_{1,\rho}k_{2,\sigma} - (A_1A_2)k_{1,\rho}k_{2,\sigma} - (k_1k_2)A_{1,\rho}A_{2,\sigma}\Big)$$

$$-(A_1A_2)k_{1,\rho}k_{2,\sigma} - (k_1k_2)A_{1,\rho}A_{2,\sigma}\Big)$$

$$(Y.55)$$

$$\langle Implementation of couplings \rangle + \equiv$$

end function kg5_kgkg

$$A^{\mu}(k_1 + k_2) = igk_{\nu} \left(\epsilon^{\mu\rho\lambda\sigma} F_{1,\ \rho}^{\ \nu} - \epsilon^{\nu\rho\lambda\sigma} F_{1,\ \rho}^{\ \mu} \right) \frac{1}{2} F_{1,\lambda\sigma}$$
 (Y.56)

with $k = -k_1 - k_2$, i. e.

$$A^{\mu}(k_1 + k_2) = -ig \Big(\epsilon^{\mu\rho\lambda\sigma}(kk_2) A_{2,\rho} - \epsilon^{\mu\rho\lambda\sigma}(kA_2) k_{2,\rho} - k_2^{\mu} \epsilon^{\nu\rho\lambda\sigma} k_n u A_{2,\rho} + A_2^{\mu} \epsilon^{\nu\rho\lambda\sigma} k_n u k_{2,\rho} \Big) k_{1,\lambda} A_{1,\sigma} \quad (Y.57)$$

+ a2 * pseudo_scalar (kv, k2v, k1v, a1))

This is not the most efficienct way of doing it: $\epsilon^{\mu\nu\rho\sigma}F_{1,\rho\sigma}$ should be cached!

end function kg_kg5kg

$$V^{\mu}(k_1 + k_2) = -g((k_1 + k_2)V_1)V_2^{\mu} + ((k_1 + k_2)V_2)V_1^{\mu}$$
(Y.58)

⟨Implementation of couplings⟩+≡
pure function dv_vv (g, v1, k1, v2, k2) result (v)
 complex(kind=default), intent(in) :: g
 type(vector), intent(in) :: v1, v2
 type(momentum), intent(in) :: k1, k2
 type(vector) :: v
 type(vector) :: k
 k = -(k1 + k2)
 v = g * ((k * v1) * v2 + (k * v2) * v1)

end function dv_vv

```
V^{\mu}(k_1 + k_2) = \frac{g}{2}(V_1(k_1)V_2(k_2))(k_1 + k_2)^{\mu}
                                                                                                                 (Y.59)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function dv_vv_cf (g, v1, k1, v2, k2) result (v)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: v
    type(vector) :: k
    k = -(k1 + k2)
    v = - g/2 * (v1 * v2) * k
  end function dv_vv_cf
                                           V_1^{\mu} = g * (kV_2)V(k) + (VV_2)k
                                                                                                                 (Y.60)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function v_dvv (g, v, k, v2) result (v1)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v, v2
    type(momentum), intent(in) :: k
    type(vector) :: v1
    v1 = g * ((v * v2) * k + (k * v2) * v)
  end function v_dvv
                                                V_1^{\mu} = -\frac{g}{2}(V(k)k)V_2^{\mu}
                                                                                                                 (Y.61)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function v_dvv_cf (g, v, k, v2) result (v1)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v, v2
    type(momentum), intent(in) :: k
    type(vector) :: v1
    v1 = -g/2 * (v * k) * v2
  end function v_dvv_cf
                               Y.10
                                          Tensorvector - Scalar coupling
\langle Declaration \ of \ couplings \rangle + \equiv
  public :: dv_phi2,phi_dvphi, dv_phi2_cf, phi_dvphi_cf
                   V^{\mu}(k_1 + k_2) = g * ((k_1k_2 + k_2k_2)k_1^{\mu} + (k_1k_2 + k_1k_1)k_2^{\mu}) * phi_1(k_1)phi_2(k_2)
                                                                                                                 (Y.62)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function dv_phi2 (g, phi1, k1, phi2, k2) result (v)
    complex(kind=default), intent(in) :: g, phi1, phi2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: v
    v = g * phi1 * phi2 * ( &
         (k1 * k2 + k2 * k2) * k1 + &
         (k1 * k2 + k1 * k1) * k2)
  end function dv_phi2
                            V^{\mu}(k_1 + k_2) = -\frac{g}{2} * (k_1 k_2) * (k_1 + k_2)^{\mu} * phi_1(k_1)phi_2(k_2)
                                                                                                                 (Y.63)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function dv_phi2_cf (g, phi1, k1, phi2, k2) result (v)
    complex(kind=default), intent(in) :: g, phi1, phi2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: v
    v = -g/2 * phi1 * phi2 * (k1 * k2) * (k1 + k2)
  end function dv_phi2_cf
```

```
phi_1(k_1) = g * ((k_1k_2 + k_2k_2)(k_1 * V(-k_1 - k_2)) + (k_1k_2 + k_1k_1)(k_2 * V(-k_1 - k_2))) * phi_2(k_2)
                                                                                                           (Y.64)
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function phi_dvphi (g, v, k, phi2, k2) result (phi1)
    complex(kind=default), intent(in) :: g, phi2
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: k, k2
    complex(kind=default) :: phi1
    type(momentum) :: k1
    k1 = - (k + k2)
    phi1 = g * phi2 * ( &
         (k1 * k2 + k2 * k2) * (k1 * V) + &
         (k1 * k2 + k1 * k1) * (k2 * V)
  end function phi_dvphi
                               phi_1(k_1) = -\frac{g}{2} * (k_1k_2) * ((k_1 + k_2)V(-k_1 - k_2))
                                                                                                           (Y.65)
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function phi_dvphi_cf (g, v, k, phi2, k2) result (phi1)
    complex(kind=default), intent(in) :: g, phi2
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: k, k2
    complex(kind=default) :: phi1
    type(momentum) :: k1
    k1 = -(k + k2)
    phi1 = -g/2 * phi2 * (k1 * k2) * ((k1 + k2) * v)
  end function phi_dvphi_cf
                             Y.11 Scalar-Vector Dim-5 Couplings
\langle Declaration \ of \ couplings \rangle + \equiv
 public :: phi_vv, v_phiv, phi_u_vv, v_u_phiv
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function phi_vv (g, k1, k2, v1, v2) result (phi)
    complex(kind=default), intent(in) :: g
    type(momentum), intent(in) :: k1, k2
    type(vector), intent(in) :: v1, v2
    complex(kind=default) :: phi
    phi = g * pseudo_scalar (k1, v1, k2, v2)
  end function phi_vv
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function v_phiv (g, phi, k1, k2, v) result (w)
    complex(kind=default), intent(in) :: g, phi
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: k1, k2
    type(vector) :: w
    w = g * phi * pseudo_vector (k1, k2, v)
  end function v_phiv
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function phi_u_vv (g, k1, k2, v1, v2) result (phi)
    complex(kind=default), intent(in) :: g
    type(momentum), intent(in) :: k1, k2
    type(vector), intent(in) :: v1, v2
    complex(kind=default) :: phi
    phi = g * ((k1*v2)*((-(k1+k2))*v1) + &
                (k2*v1)*((-(k1+k2))*v2) + &
                (((k1+k2)*(k1+k2))*(v1*v2)))
  end function phi_u_vv
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function v_u_phiv (g, phi, k1, k2, v) result (w)
    complex(kind=default), intent(in) :: g, phi
    type(vector), intent(in) :: v
```

Y.12 Dim-6 Anoumalous Couplings with Higgs

```
\langle Declaration \ of \ couplings \rangle + \equiv
 public :: s_vv_6D, v_sv_6DP, v_sv_6DP, a_hz_D, h_az_D, z_ah_D, &
       a_hz_DP, h_az_DP, z_ah_DP, h_hh_6
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function s_vv_6D (g, v1, k1, v2, k2) result (phi)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    complex(kind=default) :: phi
    phi = g * (-(k1 * v1) * (k1 * v2) - (k2 * v1) * (k2 * v2) &
         + ((k1 * k1) + (k2 * k2)) * (v1 * v2))
  end function s_vv_6D
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function v_sv_6D (g, phi, kphi, v, kv) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: kphi, kv
    type(vector) :: vout
    vout = g * ( - phi * (kv * v) * kv - phi * ((kphi + kv) * v) * (kphi + kv) &
          + phi * (kv * kv) * v + phi * ((kphi + kv)*(kphi + kv)) * v)
  end function v sv 6D
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function s_vv_6DP (g, v1, k1, v2, k2) result (phi)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    complex(kind=default) :: phi
    phi = g * ( (-(k1+k2)*v1) * (k1*v2) - ((k1+k2)*v2) * (k2*v1) + &
          ((k1+k2)*(k1+k2))*(v1*v2))
  end function s_vv_6DP
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function v_sv_6DP (g, phi, kphi, v, kv) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: kphi, kv
    type(vector) :: vout
    vout = g * phi * ((-(kphi + kv)*v) * kphi + (kphi * v) * kv + &
          (kphi*kphi) * v )
  end function v_sv_6DP
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function a_hz_D (g, h1, k1, v2, k2) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: vout
    vout = g * h1 * (((k1 + k2) * v2) * (k1 + k2) + &
          ((k1 + k2) * (k1 + k2)) * v2)
  end function a_hz_D
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_az_D (g, v1, k1, v2, k2) result (hout)
    complex(kind=default), intent(in) :: g
```

```
type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    complex(kind=default) :: hout
    hout = g * ((k1 * v1) * (k1 * v2) + (k1 * k1) * (v1 * v2))
  end function h_az_D
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function z_ah_D (g, v1, k1, h2, k2) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h2
    type(vector), intent(in) :: v1
    type(momentum), intent(in) :: k1, k2
    type(vector) :: vout
    vout = g * h2 * ((k1 * v1) * k1 + ((k1 * k1)) * v1)
  end function z_ah_D
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function a_hz_DP (g, h1, k1, v2, k2) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: vout
    vout = g * ((- h1 * (k1 + k2) * v2) * (k1) &
         + h1 * ((k1 + k2) * (k1)) *v2)
  end function a_hz_DP
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_az_DP (g, v1, k1, v2, k2) result (hout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    complex(kind=default) :: hout
    hout = g * (- (k1 * v2) * ((k1 + k2) * v1) + (k1 * (k1 + k2)) * (v1 * v2))
  end function h_az_DP
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function z_ah_DP (g, v1, k1, h2, k2) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h2
    type(vector), intent(in) :: v1
    type(momentum), intent(in) :: k1, k2
    type(vector) :: vout
    vout = g * h2* ((k2 * v1) * k1 - (k1 * k2) * v1)
  end function z_ah_DP
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_hh_6 (g, h1, k1, h2, k2) result (hout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1, h2
    type(momentum), intent(in) :: k1, k2
    complex(kind=default) :: hout
    hout = g * ((k1* k1) + (k2 * k2) + (k1* k2)) * h1 * h2
  end function h_hh_6
                   Y.13 Dim-6 Anoumalous Couplings without Higgs
\langle Declaration \ of \ couplings \rangle + \equiv
  public :: g_gg_13, g_gg_23, g_gg_6, kg_kgkg_i
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function g_gg_23 (g, v1, k1, v2, k2) result (vout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: vout
    vout = g * (v1 * (-2*(k1*v2)) + v2 * (2*k2 * v1) + (k1 - k2) * (v1*v2))
  end function g_gg_23
```

```
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function g_gg_13 (g, v1, k1, v2, k2) result (vout)
   complex(kind=default), intent(in) :: g
   type(vector), intent(in) :: v1, v2
   type(momentum), intent(in) :: k1, k2
   type(vector) :: vout
   vout = g * (v1 * (2*(k1 + k2)*v2) - v2 * ((k1 + 2*k2) * v1) + 2*k2 * (v1 * v2))
 end function g_gg_13
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function g_gg_6 (g, v1, k1, v2, k2) result (vout)
   complex(kind=default), intent(in) :: g
   type(vector), intent(in) :: v1, v2
   type(momentum), intent(in) :: k1, k2
   type(vector) :: vout
   vout = g * &
         (k1 * ((-(k1 + k2) * v2) * (k2 * v1) + ((k1 + k2) * k2) * (v1 * v2)) &
         + k2 * (((k1 + k2) * v1) * (k1 * v2) - ((k1 + k2) * k1) * (v1 * v2)) &
         + v1 * (-((k1 + k2) * k2) * (k1 * v2) + (k1 * k2) * ((k1 + k2) * v2)) &
         + v2 * (((k1 + k2) * k1) * (k2 * v1) - (k1 * k2) * ((k1 + k2) * v1)))
 end function g_gg_6
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function kg_kgkg_i (g, a1, k1, a2, k2) result (a)
   complex(kind=default), intent(in) :: g
   type(vector), intent(in) :: a1, a2
   type(momentum), intent(in) :: k1, k2
   type(vector) :: a
   real(kind=default) :: k1k1, k2k2, k1k2, kk1, kk2
   complex(kind=default) :: a1a2, k2a1, ka1, k1a2, ka2
   k1k1 = k1 * k1
   k1k2 = k1 * k2
   k2k2 = k2 * k2
   kk1 = k1k1 + k1k2
   kk2 = k1k2 + k2k2
   k2a1 = k2 * a1
   ka1 = k2a1 + k1 * a1
   k1a2 = k1 * a2
   ka2 = k1a2 + k2 * a2
   a1a2 = a1 * a2
   a = (-1) * g * (
                        (kk2 * k1a2 - k1k2 * ka2) * a1 &
         + (k1k2 * ka1 - kk1 * k2a1) * a2 &
         + (ka2 * k2a1 - kk2 * a1a2) * k1 &
         + (kk1 * a1a2 - ka1 * k1a2) * k2 )
 end function kg_kgkg_i
```

Y.14 Dim-6 Anoumalous Couplings with AWW

```
\langle Declaration \ of \ couplings \rangle + \equiv
  public ::a_ww_DP, w_aw_DP, a_ww_DW
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function a_ww_DP (g, v1, k1, v2, k2) result (vout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: vout
    vout = g * ( - ((k1 + k2) * v2) * v1 + ((k1 + k2) * v1) * v2)
  end function a_ww_DP
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function w_aw_DP (g, v1, k1, v2, k2) result (vout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: vout
    vout = g * ((k1 * v2) * v1 - (v1 * v2) * k1)
  end function w_aw_DP
```

```
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function a_ww_DW (g, v1, k1, v2, k2) result (vout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: vout
    vout = g * (v1 * (- (4*k1 + 2*k2) * v2) &
         + v2 * ((2*k1 + 4*k2) * v1) &
         + (k1 - k2) * (2*v1*v2))
  end function a_ww_DW
\langle Declaration \ of \ couplings \rangle + \equiv
 public :: w_wz_DPW, z_ww_DPW, w_wz_DW, z_ww_DW, w_wz_D, z_ww_D
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function w_wz_DPW (g, v1, k1, v2, k2) result (vout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: vout
    vout = g * (v1 * (-(k1+k2)*v2 - k1*v2) + v2 * ((k1+k2)*v1) + k1 * (v1*v2))
  end function w_wz_DPW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function z_ww_DPW (g, v1, k1, v2, k2) result (vout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: vout
    vout = g * (k1*(v1*v2) - k2*(v1*v2) - v1*(k1*v2) + v2*(k2*v1))
  end function z_ww_DPW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function w_wz_DW (g, v1, k1, v2, k2) result (vout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: vout
    vout = g * (v2 * (v1 * k2) - k2 * (v1 * v2))
  end function w_wz_DW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function z_ww_DW (g, v1, k1, v2, k2) result (vout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: vout
    vout = g * (v1 * ((-1)*(k1+k2) * v2) + v2 * ((k1+k2) * v1))
  end function z_ww_DW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function w_wz_D (g, v1, k1, v2, k2) result (vout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: vout
    vout = g * (v2 * (k2*v1) - k2 * (v1*v2))
  end function w_wz_D
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function z_ww_D (g, v1, k1, v2, k2) result (vout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: vout
    vout = g * (v1 * (- (k1 + k2) * v2) + v2 * ((k1 + k2) * v1))
  end function z_ww_D
```

Y.15 Dim-6 Quartic Couplings

```
\langle Declaration \ of \ couplings \rangle + \equiv
 public :: hhhh_p2, a_hww_DPB, h_aww_DPB, w_ahw_DPB, a_hww_DPW, h_aww_DPW, &
       w_ahw_DPW, a_hww_DW, h_aww_DW, w3_ahw_DW, w4_ahw_DW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function hhhh_p2 (g, h1, k1, h2, k2, h3, k3) result (hout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1, h2, h3
    type(momentum), intent(in) :: k1, k2, k3
    complex(kind=default) :: hout
    hout = g * h1*h2*h3* (k1*k1 + k2*k2 +k3*k3 + k1*k3 + k1*k2 + k2*k3)
  end function hhhh_p2
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function a_hww_DPB (g, h1, k1, v2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * (v3*((k1+k2+k3)*v2) - v2*((k1+k2+k3)*v3))
  end function a_hww_DPB
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_aww_DPB (g, v1, k1, v2, k2, v3, k3) result (hout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    complex(kind=default) :: hout
    hout = g * ((k1 * v3) * (v1 * v2) - (k1 * v2) * (v1 * v3))
  end function h_aww_DPB
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function w_ahw_DPB (g, v1, k1, h2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h2
    type(vector), intent(in) :: v1, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h2 * (v1 * (k1 * v3) - k1 * (v1 * v3))
  end function w_ahw_DPB
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function a_hww_DPW (g, h1, k1, v2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * (v3 * ((2*k1+k2+k3)*v2) - v2 * ((2*k1+k2+k3)*v3))
  end function a_hww_DPW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_aww_DPW (g, v1, k1, v2, k2, v3, k3) result (hout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    complex(kind=default) :: hout
    hout = g * ((-(2*k1+k2+k3)*v2)*(v1*v3)+((2*k1+k2+k3)*v3)*(v1*v2))
  end function h_aww_DPW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function w_ahw_DPW (g, v1, k1, h2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h2
    type(vector), intent(in) :: v1, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
```

```
vout = g * h2 * ((k2 - k1) * (v1 * v3) + v1 * ((k1 - k2) * v3))
  end function w_ahw_DPW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function a_hww_DW (g, h1, k1, v2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * (v2 * (-(3*k1 + 4*k2 + 4*k3) * v3) &
         + v3 * ((3*k1 + 2*k2 + 4*k3) * v2) &
         + (k2 - k3) *2*(v2 * v3))
  end function a_hww_DW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_aww_DW (g, v1, k1, v2, k2, v3, k3) result (hout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    complex(kind=default) :: hout
    hout = g * ((v1*v2) * ((3*k1 - k2 - k3)*v3) &
         + (v1*v3) * ((-3*k1 - k2 + k3)*v2) &
         + (v2*v3) * (2*(k2-k3)*v1))
  end function h_aww_DW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function w3_ahw_DW (g, v1, k1, h2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h2
    type(vector), intent(in) :: v1, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h2 * (v1 * ((4*k1 + k2) * v3) &
         +v3 * (-2*(k1 + k2 + 2*k3) * v1) &
         +(-2*k1 + k2 + 2*k3) * (v1*v3))
  end function w3_ahw_DW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function w4_ahw_DW (g, v1, k1, h2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h2
    type(vector), intent(in) :: v1, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h2 * (v1 * (-(4*k1 + k2 + 2*k3) * v3) &
         + v3 * (2*(k1 + k2 + 2*k3) * v1) &
         +(4*k1 + k2) * (v1*v3))
  end function w4_ahw_DW
\langle Declaration \ of \ couplings \rangle + \equiv
 public ::a_aww_DW, w_aaw_DW, a_aww_W, w_aaw_W
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function a_aww_DW (g, v1, k1, v2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * (2*v1*(v2*v3) - v2*(v1*v3) - v3*(v1*v2))
  end function a_aww_DW
  pure function w_aaw_DW (g, v1, k1, v2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * (2*v3*(v1*v2) - v2*(v1*v3) - v1*(v2*v3))
  end function w_aaw_DW
  pure function a_aww_W (g, v1, k1, v2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
```

```
type(vector), intent(in) :: v1, v2, v3
  type(momentum), intent(in) :: k1, k2, k3
  type(vector) :: vout
!!! Recalculated WK 2018-08-24
  type(momentum) :: k4
  k4 = -(k1+k2+k3)
!!! negative sign (-g) causes expected gauge cancellation
  vout = (-g) * ( &
       + (k1*v3)*(k3*v2)*v1 - (k3*v2)*(v1*v3)*k1 &
       -(k1*k3)*(v2*v3)*v1 + (k3*v1)*(v2*v3)*k1 &
       -(k1*v3)*(v1*v2)*k3 + (k1*v2)*(v1*v3)*k3 &
       + (k1*k3)*(v1*v2)*v3 - (k3*v1)*(k1*v2)*v3 &
       + (k3*v2)*(k4*v3)*v1 - (k3*v2)*(k4*v1)*v3 &
       -(k3*k4)*(v2*v3)*v1 + (k4*v1)*(v2*v3)*k3 &
       - (k3*v1)*(k4*v3)*v2 + (k3*v1)*(k4*v2)*v3 &
       + (k3*k4)*(v1*v3)*v2 - (k4*v2)*(v1*v3)*k3 &
       + (k1*v2)*(k2*v3)*v1 - (k2*v3)*(v1*v2)*k1 &
       -(k1*k2)*(v2*v3)*v1 + (k2*v1)*(v2*v3)*k1 &
       -(k1*v2)*(v1*v3)*k2 + (k1*v3)*(v1*v2)*k2 &
       + (k1*k2)*(v1*v3)*v2 - (k2*v1)*(k1*v3)*v2 &
       + (k2*v3)*(k4*v2)*v1 - (k2*v3)*(k4*v1)*v2 &
       -(k2*k4)*(v2*v3)*v1 + (k4*v1)*(v2*v3)*k2 &
       -(k2*v1)*(k4*v2)*v3 + (k2*v1)*(k4*v3)*v2 &
       + (k2*k4)*(v1*v2)*v3 - (k4*v3)*(v1*v2)*k2 &
!!! Original Version
    vout = g * (v1*((-(k2+k3)*v2)*(k2*v3) + (-(k2+k3)*v3)*(k3*v2)) &
        +v2*((-((k2-k3)*v1)*(k1+k2+k3)*v3) - (k1*v3)*(k2*v1) &
         + ((k1+k2+k3)*v1)*(k2*v3)) &
        +v3*((k2-k3)*v1)*((k1+k2+k3)*v2) - (k1*v2)*(k3*v1) &
         + ((k1+k2+k3)*v1)*(k3*v2)) &
         +(v1*v2)*(((2*k1+k2+k3)*v3)*k2 - (k2*v3)*k1 - (k1*v3)*k3) &
         +(v1*v3)*(((2*k1+k2+k3)*v2)*k3 - (k3*v2)*k1 - (k1*v2)*k3) &
         +(v2*v3)*((-(k1+k2+k3)*v1)*(k2+k3) + ((k2+k3)*v1)*k1) &
         +(-(k1+k2+k3)*k3 +k1*k2)*((v1*v3)*v2 - (v2*v3)*v1) &
         +(-(k1+k2+k3)*k2 + k1*k3)*((v1*v2)*v3 - (v2*v3)*v1))
end function a_aww_W
pure function w_aaw_w (g, v1, k1, v2, k2, v3, k3) result (vout)
  complex(kind=default), intent(in) :: g
  type(vector), intent(in) :: v1, v2, v3
  type(momentum), intent(in) :: k1, k2, k3
  type(vector) :: vout
!!! Recalculated WK 2018-08-25
  type(momentum) :: k4
  k4 = -(k1+k2+k3)
!!! negative sign (-g) causes expected gauge cancellation
  vout = (-g) * ( & 
       + (k3*v1)*(k1*v2)*v3 - (k1*v2)*(v3*v1)*k3 &
       -(k3*k1)*(v2*v1)*v3 + (k1*v3)*(v2*v1)*k3 &
       -(k3*v1)*(v3*v2)*k1 + (k3*v2)*(v3*v1)*k1 &
       + (k3*k1)*(v3*v2)*v1 - (k1*v3)*(k3*v2)*v1 &
       + (k1*v2)*(k4*v1)*v3 - (k1*v2)*(k4*v3)*v1 &
       -(k1*k4)*(v2*v1)*v3 + (k4*v3)*(v2*v1)*k1 &
       -(k1*v3)*(k4*v1)*v2 + (k1*v3)*(k4*v2)*v1 &
       + (k1*k4)*(v3*v1)*v2 - (k4*v2)*(v3*v1)*k1 &
       + (k3*v2)*(k2*v1)*v3 - (k2*v1)*(v3*v2)*k3 &
       -(k3*k2)*(v2*v1)*v3 + (k2*v3)*(v2*v1)*k3 &
       -(k3*v2)*(v3*v1)*k2 + (k3*v1)*(v3*v2)*k2 &
       + (k3*k2)*(v3*v1)*v2 - (k2*v3)*(k3*v1)*v2 &
       + (k2*v1)*(k4*v2)*v3 - (k2*v1)*(k4*v3)*v2 &
       -(k2*k4)*(v2*v1)*v3 + (k4*v3)*(v2*v1)*k2 &
       -(k2*v3)*(k4*v2)*v1 + (k2*v3)*(k4*v1)*v2 &
       + (k2*k4)*(v3*v2)*v1 - (k4*v1)*(v3*v2)*k2 &
!!! Original Version
   vout = g * (v1*((k1*v3)*(-(k1+k2+2*k3)*v2) + (k2*v3)*((k1+k2+k3)*v2) &
```

```
!
           + (k1*v2)*((k1+k2+k3)*v3)) &
           + v2*(((k1-k2)*v3)*((k1+k2+k3)*v1) - (k2*v3)*(k3*v1) &
  1
           + (k2*v1)*((k1+k2+k3)*v3)) &
           + v3*((k1*v2)*(-(k1+k2)*v1) + (k2*v1)*(-(k1+k2)*v2)) &
           + (v1*v2)*((k1+k2)*(-(k1+k2+k3)*v3) + k3*((k1+k2)*v3))&
           + (v1*v3)*(-k2*(k3*v2) - k3*(k1*v2) + k1*((k1+k2+2*k3)*v2)) &
           + (v2*v3)*(-k1*(k3*v1) - k3*(k2*v1) + k2*((k1+k2+2*k3)*v1)) &
           + (-k2*(k1+k2+k3) + k1*k3)*(v1*(v2*v3) - v3*(v1*v2)) &
           + (-k1*(k1+k2+k3) + k2*k3)*(v2*(v1*v3) - v3*(v1*v2)))
  end function w_aaw_W
\langle Declaration \ of \ couplings \rangle + \equiv
  public :: h_hww_D, w_hhw_D, h_hww_DP, w_hhw_DP, h_hvv_PB, v_hhv_PB
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_hww_D (g, h1, k1, v2, k2, v3, k3) result (hout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    complex(kind=default) :: hout
    hout = g * h1 * ((v2*v3)*((k2*k2)+(k3*k3)) - (k2*v2)*(k2*v3) &
          -(k3*v2)*(k3*v3))
  end function h_hww_D
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function w_hhw_D (g, h1, k1, h2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1, h2
    type(vector), intent(in) :: v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * h2 * (v3 * ((k1+k2+k3)*(k1+k2+k3)+(k3*k3)) &
         -(k1+k2+k3) * ((k1+k2+k3)*v3) - k3 * (k3*v3))
  end function w_hhw_D
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_hww_DP (g, h1, k1, v2, k2, v3, k3) result (hout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    complex(kind=default) :: hout
    hout = g * h1 * (-((k2+k3)*v2)*(k2*v3) - &
         ((k2+k3)*v3)*(k3*v2)+(v2*v3)*((k2+k3)*(k2+k3)))
  end function h_hww_DP
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function w_hw_DP (g, h1, k1, h2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1, h2
    type(vector), intent(in) :: v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * h2 * (k3*((k1+k2)*v3) + (k1+k2)*(-(k1+k2+k3)*v3) &
         + v3*((k1+k2)*(k1+k2)))
  end function w_hhw_DP
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_hvv_PB (g, h1, k1, v2, k2, v3, k3) result (hout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    complex(kind=default) :: hout
    hout = g * h1 * ((k2*v3)*(k3*v2) - (k2*k3)*(v2*v3))
  end function h_hvv_PB
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function v_hhv_PB (g, h1, k1, h2, k2, v3, k3) result (vout)
```

```
complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1, h2
    type(vector), intent(in) :: v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * h2 * ((-(k1+k2+k3)*v3)*k3 + ((k1+k2+k3)*k3)*v3)
  end function v_hhv_PB
\langle Declaration \ of \ couplings \rangle + \equiv
  public :: a_hhz_D, h_ahz_D, z_ahh_D, a_hhz_DP, h_ahz_DP, z_ahh_DP, &
       a_hhz_PB, h_ahz_PB, z_ahh_PB
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function a_hhz_D (g, h1, k1, h2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1, h2
    type(vector), intent(in) :: v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * h2 * ((k1+k2+k3) * ((k1+k2+k3)*v3) &
         - v3 * ((k1+k2+k3)*(k1+k2+k3)))
  end function a_hhz_D
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_ahz_D (g, v1, k1, h2, k2, v3, k3) result (hout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h2
    type(vector), intent(in) :: v1, v3
    type(momentum), intent(in) :: k1, k2, k3
    complex(kind=default) :: hout
    hout = g * h2 * ((k1*v1)*(k1*v3) - (k1*k1)*(v1*v3))
  end function h_ahz_D
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function z_ahh_D (g, v1, k1, h2, k2, h3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1
    complex(kind=default), intent(in) :: h2, h3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h2 * h3 * ((k1*v1)*k1 - (k1*k1)*v1)
  end function z_ahh_D
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function a_hhz_DP (g, h1, k1, h2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1, h2
    type(vector), intent(in) :: v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * h2 * ((-(k1+k2+k3)*v3)*(k1+k2) + ((k1+k2+k3)*(k1+k2))*v3)
  end function a_hhz_DP
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_ahz_DP (g, v1, k1, h2, k2, v3, k3) result (hout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h2
    type(vector), intent(in) :: v1, v3
    type(momentum), intent(in) :: k1, k2, k3
    complex(kind=default) :: hout
    hout = g * h2 * ((k1*v3)*(-(k1+k3)*v1) + (k1*(k1+k3))*(v1*v3))
  end function h_ahz_DP
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function z_ahh_DP (g, v1, k1, h2, k2, h3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1
    complex(kind=default), intent(in) :: h2, h3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
```

```
vout = g * h2 * h3 * (k1*((k2+k3)*v1) - v1*(k1*(k2+k3)))
  end function z_ahh_DP
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function a_hhz_PB (g, h1, k1, h2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1, h2
    type(vector), intent(in) :: v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * h2 * (k3*((k1+k2+k3)*v3) - v3*((k1+k2+k3)*k3))
  end function a_hhz_PB
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_ahz_PB (g, v1, k1, h2, k2, v3, k3) result (hout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h2
    type(vector), intent(in) :: v1, v3
    type(momentum), intent(in) :: k1, k2, k3
    complex(kind=default) :: hout
    hout = g * h2 * ((-k1*v3)*(k3*v1) + (k1*k3)*(v1*v3))
  end function h_ahz_PB
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function z_ahh_PB (g, v1, k1, h2, k2, h3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1
    complex(kind=default), intent(in) :: h2, h3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h2 * h3 * (k1*((k1+k2+k3)*v1) - v1*(k1*(k1+k2+k3)))
  end function z_ahh_PB
\langle Declaration \ of \ couplings \rangle + \equiv
 public :: h_wwz_DW, w_hwz_DW, z_hww_DW, h_wwz_DPB, w_hwz_DPB, z_hww_DPB
 public :: h_wwz_DDPW, w_hwz_DDPW, z_hww_DDPW, h_wwz_DPW, w_hwz_DPW, z_hww_DPW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_wwz_DW (g, v1, k1, v2, k2, v3, k3) result (hout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    complex(kind=default) :: hout
    hout = g * (((k1-k2)*v3)*(v1*v2)-((2*k1+k2)*v2)*(v1*v3) + &
         ((k1+2*k2)*v1)*(v2*v3))
  end function h_wwz_DW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function w_hwz_DW (g, h1, k1, v2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * (v2*(-(k1+2*k2+k3)*v3) + v3*((2*k1+k2+2*k3)*v2) - &
         (k1 - k2 + k3)*(v2*v3))
  end function w_hwz_DW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function z_hww_DW (g, h1, k1, v2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * ((k2-k3)*(v2*v3) - v2*((2*k2+k3)*v3) + v3*((k2+2*k3)*v2))
  end function z_hww_DW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_wwz_DPB (g, v1, k1, v2, k2, v3, k3) result (hout)
    complex(kind=default), intent(in) :: g
```

```
type(vector), intent(in) :: v1, v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    complex(kind=default) :: hout
    hout = g * ((k3*v1)*(v2*v3) - (k3*v2)*(v1*v3))
  end function h_wwz_DPB
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function w_hwz_DPB (g, h1, k1, v2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * (k3*(v2*v3) - v3*(k3*v2))
  end function w_hwz_DPB
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function z_hww_DPB (g, h1, k1, v2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * (((k1+k2+k3)*v3)*v2 - ((k1+k2+k3)*v2)*v3)
  end function z_hww_DPB
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_wwz_DDPW (g, v1, k1, v2, k2, v3, k3) result (hout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    complex(kind=default) :: hout
    hout = g * (((k1-k2)*v3)*(v1*v2)-((k1-k3)*v2)*(v1*v3)+((k2-k3)*v1)*(v2*v3))
  end function h_wwz_DDPW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function w_hwz_DDPW (g, h1, k1, v2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * ((-(k1+2*k2+k3)*v3)*v2 + ((k1+k2+2*k3)*v2)*v3 + &
         (v2*v3)*(k2-k3))
  end function w_hwz_DDPW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function z_hww_DDPW (g, h1, k1, v2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * ((v2*v3)*(k2-k3) - ((k1+2*k2+k3)*v3) *v2 + &
         ((k1+k2+2*k3)*v2)*v3)
  end function z_hww_DDPW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function h_wwz_DPW (g, v1, k1, v2, k2, v3, k3) result (hout)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    complex(kind=default) :: hout
    hout = g * (((k1-k2)*v3)*(v1*v2) + (-(2*k1+k2+k3)*v2)*(v1*v3) + &
         ((k1+2*k2+k3)*v1)*(v2*v3))
  end function h_wwz_DPW
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function w_hwz_DPW (g, h1, k1, v2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
```

```
complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * ((-(k1+2*k2+k3)*v3)*v2 + ((2*k1+k2+k3)*v2)*v3 + &
          (v2*v3)*(k2-k1))
  end function w_hwz_DPW
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function z_hww_DPW (g, h1, k1, v2, k2, v3, k3) result (vout)
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: h1
    type(vector), intent(in) :: v2, v3
    type(momentum), intent(in) :: k1, k2, k3
    type(vector) :: vout
    vout = g * h1 * ((v2*v3)*(k2-k3) + ((k1-k2)*v3)*v2 + ((k3-k1)*v2)*v3)
  end function z_hww_DPW
                                  Y.16 Scalar Dim-5 Couplings
\langle Declaration \ of \ couplings \rangle + \equiv
  public :: phi_dim5s2
                                         \phi_1(k_1) = g(k_2 \cdot k_3)\phi_2(k_2)\phi_3(k_3)
                                                                                                              (Y.66)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function phi_dim5s2 (g, phi2, k2, phi3, k3) result (phi1)
    complex(kind=default), intent(in) :: g, phi2, phi3
    type(momentum), intent(in) :: k2, k3
    complex(kind=default) :: phi1
    phi1 = g * phi2 * phi3 * (k2 * k3)
  end function phi_dim5s2
                               Y.17
                                         Tensorscalar-Scalar Couplings
\langle Declaration \ of \ couplings \rangle + \equiv
  public :: tphi_ss, tphi_ss_cf, s_tphis, s_tphis_cf
                     \phi(k_1 + k_2) = 2q((k_1 \cdot k_2) + (k_1 \cdot k_1))((k_1 \cdot k_2) + (k_2 \cdot k_2))\phi_1(k_1)\phi_2(k_2)
                                                                                                              (Y.67)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function tphi_ss (g, phi1, k1, phi2, k2) result (phi)
    complex(kind=default), intent(in) :: g, phi1, phi2
    type(momentum), intent(in) :: k1, k2
    complex(kind=default) :: phi
    phi = 2 * g * phi1 * phi2 * &
               ((k1 * k2) + (k1 * k1)) * &
               ((k1 * k2) + (k2 * k2))
  end function tphi_ss
                          \phi(k_1 + k_2) = -g/2(k_1 \cdot k_2)((k_1 + k_2) \cdot (k_1 + k_2))\phi_1(k_1)\phi_2(k_2)
                                                                                                              (Y.68)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function tphi_ss_cf (g, phi1, k1, phi2, k2) result (phi)
    complex(kind=default), intent(in) :: g, phi1, phi2
    type(momentum), intent(in) :: k1, k2
    complex(kind=default) :: phi
    phi = - g/2 * phi1 * phi2 * &
               (k1 * k2) * &
               ((k1 + k2) * (k1 + k2))
  end function tphi_ss_cf
```

```
\phi_1(k_1) = 2q((k_1 \cdot k_2) + (k_1 \cdot k_1))((k_1 \cdot k_2) + (k_2 \cdot k_2))\phi(k_2 - k_1)\phi_2(k_2)
                                                                                                                  (Y.69)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function s_tphis (g, phi, k, phi2, k2) result (phi1)
    complex(kind=default), intent(in) :: g, phi, phi2
    type(momentum), intent(in) :: k, k2
    complex(kind=default) :: phi1
    type(momentum) :: k1
    k1 = - (k + k2)
    phi1 = 2 * g * phi * phi2 * &
               ((k1 * k2) + (k1 * k1)) * &
               ((k1 * k2) + (k2 * k2))
  end function s_tphis
                           \phi_1(k_1) = -g/2(k_1 \cdot k_2)((k_1 + k_2) \cdot (k_1 + k_2))\phi(k_2 - k_1)\phi_2(k_2)
                                                                                                                  (Y.70)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function s_tphis_cf (g, phi, k, phi2, k2) result (phi1)
    complex(kind=default), intent(in) :: g, phi, phi2
    type(momentum), intent(in) :: k, k2
    complex(kind=default) :: phi1
    type(momentum) :: k1
    k1 = - (k + k2)
    phi1 = -g/2 * phi * phi2 * &
               (k1 * k2) * &
               ((k1 + k2) * (k1 + k2))
  end function s_tphis_cf
                             Y.18 Scalar2-Vector2 Dim-8 Couplings
\langle Declaration \ of \ couplings \rangle + \equiv
  public :: phi_phi2v_1, v_phi2v_1, phi_phi2v_2, v_phi2v_2
                              \phi_2(k_2) = g((k_1 \cdot V_1)(k_2 \cdot V_2) + (k_1 \cdot V_1)(k_1 \cdot V_2)) \phi_1(k_1)
                                                                                                                  (Y.71)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function phi_phi2v_1 (g, phi1, k1, v1, k_v1, v2, k_v2) result (phi2)
    complex(kind=default), intent(in) :: g, phi1
    type(momentum), intent(in) :: k1, k_v1, k_v2
    type(momentum) :: k2
    type(vector), intent(in) :: v1, v2
    complex(kind=default) :: phi2
    k2 = - k1 - k_v1 - k_v2
    phi2 = g * phi1 * &
           ((k1 * v1) * (k2 * v2) + (k1 * v2) * (k2 * v1))
  end function phi_phi2v_1
                                  V_2^{\mu} = g(k_1^{\mu}(k_2 \cdot V_1) + k_2^{\mu}(k_1 \cdot V_1))\phi_1(k_1)\phi_2(k_2)
                                                                                                                  (Y.72)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function v_phi2v_1 (g, phi1, k1, phi2, k2, v1) result (v2)
    complex(kind=default), intent(in) :: g, phi1, phi2
    type(momentum), intent(in) :: k1, k2
    type(vector), intent(in) :: v1
    type(vector) :: v2
    v2 = g * phi1 * phi2 * &
           (k1 * (k2 * v1) + k2 * (k1 * v1))
  end function v_phi2v_1
                                         \phi_2(k_2) = g(k_1 \cdot k_2) (V_1 \cdot V_2) \phi_1(k_1)
                                                                                                                  (Y.73)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function phi_phi2v_2 (g, phi1, k1, v1,k_v1, v2, k_v2) result (phi2)
    complex(kind=default), intent(in) :: g, phi1
```

```
type(momentum), intent(in) :: k1, k_v1, k_v2
          type(vector), intent(in) :: v1, v2
          type(momentum) :: k2
          complex(kind=default) :: phi2
          k2 = - k1 - k_v1 - k_v2
          phi2 = g * phi1 * (k1 * k2) * (v1 * v2)
     end function phi_phi2v_2
                                                                                                            V_2^{\mu} = gV_1^{\mu} (k_1 \cdot k_2) \phi_1 \phi_2
                                                                                                                                                                                                                                                                       (Y.74)
\langle Implementation \ of \ couplings \rangle + \equiv
    pure function v_phi2v_2 (g, phi1, k1, phi2, k2, v1) result (v2)
          {\tt complex(kind=default),\ intent(in)\ ::\ g,\ phi1,\ phi2}
          type(momentum), intent(in) :: k1, k2
          type(vector), intent(in) :: v1
          type(vector) :: v2
          v2 = g * phi1 * phi2 * &
                           (k1 * k2) * v1
     end function v_phi2v_2
                                                                                   Y.19 Scalar Dim-8 Couplings
\langle Declaration \ of \ couplings \rangle + \equiv
    public :: s_dim8s3
                                \phi(k_1) = g\left[ (k_1 \cdot k_2) (k_3 \cdot k_4) + (k_1 \cdot k_3) (k_2 \cdot k_4) + (k_1 \cdot k_4) (k_2 \cdot k_3) \right] \phi_2(k_2) \phi_3(k_3) \phi_4(k_4)
                                                                                                                                                                                                                                                                       (Y.75)
\langle Implementation \ of \ couplings \rangle + \equiv
    pure function s_dim8s3 (g, phi2, k2, phi3, k3, phi4, k4) result (phi1)
          complex(kind=default), intent(in) :: g, phi2, phi3, phi4
          type(momentum), intent(in) :: k2, k3, k4
          type(momentum) :: k1
          complex(kind=default) :: phi1
          k1 = - k2 - k3 - k4
          phi1 = g * ( (k1 * k2) * (k3 * k4) + (k1 * k3) * (k2 * k4) &
                                + (k1 * k4) * (k2 * k3) ) * phi2 * phi3 * phi4
     end function s_dim8s3
                                                        Y.20
                                                                                Mixed Scalar2-Vector2 Dim-8 Couplings
\langle Declaration \ of \ couplings \rangle + \equiv
     \texttt{public} :: \texttt{phi\_phi2v\_m\_0}, \ \texttt{v\_phi2v\_m\_0}, \ \texttt{phi\_phi2v\_m\_1}, \ \texttt{v\_phi2v\_m\_1}, \ \texttt{phi\_phi2v\_m\_1}, \ \texttt{v\_phi2v\_m\_1}, \ \texttt{v\_phi2
                                          \phi_2(k_2) = g\left( (V_1 \cdot k_{V_2}) \left( V_2 \cdot k_{V_1} \right) \left( k_1 \cdot k_2 \right) - \left( (V_1 \cdot V_2) \left( k_{V_1} \cdot k_{V_2} \right) \left( k_1 \cdot k_2 \right) \right) \phi_1(k_1)
                                                                                                                                                                                                                                                                       (Y.76)
\langle Implementation \ of \ couplings \rangle + \equiv
    pure function phi_phi2v_m_0 (g, phi1, k1, v1, k_v1, v2, k_v2) result (phi2)
          complex(kind=default), intent(in) :: g, phi1
          type(momentum), intent(in) :: k1, k_v1, k_v2
          type(momentum) :: k2
          type(vector), intent(in) :: v1, v2
          complex(kind=default) :: phi2
          k2 = - k1 - k_v1 - k_v2
          phi2 = g * phi1 * &
                                      (v1 * k_v2) * (v2 * k_v1) * (k1 * k2) &
                                      - (v1 * v2) * (k_v1 * k_v2) * (k1 * k2) )
     end function phi_phi2v_m_0
                                                   V_{2}^{\mu} = g\left(k_{V_{1}}^{\mu}\left(V_{1} \cdot k_{V_{2}}\right)\left(k_{1} \cdot k_{2}\right) - V_{1}^{\mu}\left(k_{V_{1}} \cdot k_{V_{2}}\right)\left(k_{1} \cdot k_{2}\right)\right)\phi_{1}(k_{1})\phi_{2}(k_{2})\right)
                                                                                                                                                                                                                                                                       (Y.77)
\langle Implementation \ of \ couplings \rangle + \equiv
    pure function v_phi2v_m_0 (g, phi1, k1, phi2, k2, v1, k_v1) result (v2)
          complex(kind=default), intent(in) :: g, phi1, phi2
          type(momentum), intent(in) :: k1, k2, k_v1
```

```
type(vector), intent(in) :: v1
                      type(momentum) :: k_v2
                      type(vector) :: v2
                      k_v2 = - k_v1 - k1 - k2
                      v2 = g * phi1 * phi2 * &
                                                                  (k_v1 * (v1 * k_v2) * (k1 * k2) &
                                                                     -v1*(k_v2*k_v1)*(k1*k2))
           end function v_phi2v_m_0
\phi_{2}(k_{2}) = g\left(\left(V_{1} \cdot V_{2}\right)\left(k_{1} \cdot k_{V_{2}}\right)\left(k_{2} \cdot k_{V_{1}}\right) + \left(\left(V_{1} \cdot V_{2}\right)\left(k_{1} \cdot k_{V_{1}}\right)\left(k_{2} \cdot k_{V_{2}}\right) + \left(\left(V_{1} \cdot k_{2}\right)\left(V_{2} \cdot k_{1}\right)\left(k_{V_{1}} \cdot k_{V_{2}}\right) + \left(\left(V_{1} \cdot k_{1}\right)\left(V_{2} \cdot k_{2}\right)\left(k_{V_{1}} \cdot k_{V_{2}}\right) + \left(\left(V_{1} \cdot k_{2}\right)\left(k_{V_{1}} \cdot k_{V_{2}}\right) + \left(\left(V_{1} \cdot k_{2}\right)\left(k_{V_{1}} \cdot k_{V_{2}}\right)\right) + \left(\left(V_{1} \cdot k_{2}\right)\left(k_{V_{1}} \cdot k_{V_{2}}\right) + \left(\left(V_{1} \cdot k_{2}\right)\left(k_{V_{1}} \cdot k_{V_{2}}\right)\right) + \left(\left(V_{1} \cdot k_{2}\right)\left(k_{V_{1}} \cdot k_{V_{2}}\right) + \left(\left(V_{1} \cdot k_{2}\right)\left(k_{V_{1}} \cdot k_{V_{2}}\right)\right) + \left(\left(V_{1} \cdot k_{V_{2}}\right)\left(k_{V_{1}} \cdot k_{V_{2}}\right)\right) + \left(\left(V_{1} \cdot k_{V_{2}}\right)\left(k_{V_{1}} \cdot k_{V_{2}}\right)\right) + \left(\left(V_{1} \cdot k_{V_{2}}\right)\left(k_{V_{1}} \cdot k_{V_{2}}\right)\right) + \left(\left(V_{1} \cdot k_{V_{1}} \cdot k_{V_{2}}\right)\right) + \left(\left(V_{1} \cdot k_{V_{2}}\right)\left(k_{V_{1}} \cdot k_{V_{2}}\right)\right) + \left(\left(V_{1} \cdot k_{V_{2}}\right)\left(k_{V_{2}} \cdot k_{V_{2}
\langle Implementation \ of \ couplings \rangle + \equiv
           pure function phi_phi2v_m_1 (g, phi1, k1, v1, k_v1, v2, k_v2) result (phi2)
                      complex(kind=default), intent(in) :: g, phi1
                      type(momentum), intent(in) :: k1, k_v1, k_v2
                      type(momentum) :: k2
                      type(vector), intent(in) :: v1, v2
                      complex(kind=default) :: phi2
                      k2 = - k1 - k_v1 - k_v2
                      phi2 = g * phi1 * &
                                                                            (v1 * v2) * (k1 * k_v2) * (k2 * k_v1) &
                                                                           + (v1 * v2) * (k1 * k_v1) * (k2 * k_v2) &
                                                                           + (v1 * k2) * (v2 * k1) * (k_v1 * k_v2) &
                                                                           + (v1 * k1) * (v2 * k2) * (k_v1 * k_v2) &
                                                                           - (v1 * k_v2) * (v2 * k2) * (k1 * k_v1) &
                                                                            - (v1 * k2) * (v2 * k_v1) * (k1 * k_v2) &
                                                                            - (v1 * k_v2) * (v2 * k1) * (k2 * k_v1) &
                                                                            - (v1 * k1) * (v2 * k_v1) * (k2 * k_v2) )
           end function phi_phi2v_m_1
V_{2}^{\mu} = g\left(k_{1}^{\mu}\left(V_{1}\cdot k_{2}\right)\left(k_{V_{1}}\cdot k_{V_{2}}\right) + k_{2}^{\mu}\left(V_{1}\cdot k_{1}\right)\left(k_{V_{1}}\cdot k_{V_{2}}\right) + V_{1}^{\mu}\left(k_{V_{1}}\cdot k_{1}\right)\left(k_{V_{2}}\cdot k_{2}\right) + V_{1}^{\mu}\left(k_{V_{1}}\cdot k_{2}\right)\left(k_{V_{2}}\cdot k_{1}\right) - k_{1}^{\mu}\left(V_{1}\cdot k_{V_{2}}\right)\left(k_{V_{1}}\cdot k_{V_{2}}\right) + V_{1}^{\mu}\left(k_{V_{1}}\cdot k_{V_{2}}\right) + V_{1}^{\mu}\left(k_{V_{1}}\cdot k_{V_{2}}\right)\left(k_{V_{1}}\cdot k_{V_{2}}\right) + V_{1}^{\mu}\left(k_{V_{1}}\cdot k_{V_{2}}\right) + V_{1}^{\mu}\left(k_
\langle Implementation \ of \ couplings \rangle + \equiv
           pure function v_phi2v_m_1 (g, phi1, k1, phi2, k2, v1, k_v1) result (v2)
                      complex(kind=default), intent(in) :: g, phi1, phi2
                      type(momentum), intent(in) :: k1, k2, k_v1
                      type(vector), intent(in) :: v1
                      type(momentum) :: k_v2
                      type(vector) :: v2
                      k_v2 = - k_v1 - k1 - k2
                      v2 = g * phi1 * phi2 * &
                                                                 (k1 * (v1 * k2) * (k_v1 * k_v2) &
                                                                + k2 * (v1 * k1) * (k_v1 * k_v2) &
                                                                + v1 * (k_v1 * k1) * (k_v2 * k2) &
                                                                + v1 * (k_v1 * k2) * (k_v2 * k1) &
                                                                 - k1 * (v1 * k_v2) * (k_v1 * k2) &
                                                                 - k2 * (v1 * k_v2) * (k_v1 * k1) &
                                                                 - k_v1 * (v1 * k1) * (k_v2 * k2) &
                                                                 - k_v1 * (v1 * k2) * (k_v2 * k1) )
           end function v_phi2v_m_1
\phi_{2}(k_{2}) = g\left(\left(V_{1} \cdot k_{V_{2}}\right)\left(k_{1} \cdot V_{2}\right)\left(k_{2} \cdot k_{V_{1}}\right) + \left(\left(V_{1} \cdot k_{V_{2}}\right)\left(k_{1} \cdot k_{V_{1}}\right)\left(k_{2} \cdot k_{V_{2}}\right) + \left(\left(V_{1} \cdot k_{1}\right)\left(V_{2} \cdot k_{V_{1}}\right)\left(k_{2} \cdot k_{V_{2}}\right) + \left(\left(V_{1} \cdot k_{2}\right)\left(V_{2} \cdot k_{V_{1}}\right)\left(k_{2} \cdot k_{V_{2}}\right) + \left(\left(V_{1} \cdot k_{2}\right)\left(V_{2} \cdot k_{V_{1}}\right)\left(k_{2} \cdot k_{V_{2}}\right) + \left(\left(V_{1} \cdot k_{V_{2}}\right)\left(k_{2} \cdot k_{V_{2}}\right) + \left(\left(V_{1} \cdot k_{V_{2}}\right)\left(k_{V_{2}} \cdot k_{V_{2}}\right)\right) + \left(\left(V_{1} \cdot k_{V_{2}}\right)\left(k_{V_{2}} \cdot k_{V_{2}}\right) + \left(\left(V_{1} \cdot k_{V_{2}}\right)\left(k_{V_{2}} \cdot k_{V_{2}}\right)\right) + 
 \langle Implementation \ of \ couplings \rangle + \equiv
           pure function phi_phi2v_m_7 (g, phi1, k1, v1, k_v1, v2, k_v2) result (phi2)
                      complex(kind=default), intent(in) :: g, phi1
                      type(momentum), intent(in) :: k1, k_v1, k_v2
                      type(momentum) :: k2
                      type(vector), intent(in) :: v1, v2
                      complex(kind=default) :: phi2
                      k2 = - k1 - k_v1 - k_v2
                      phi2 = g * phi1 * &
```

```
(v1 * k_v2) * (k1 * v2) * (k2 * k_v1) &
                                   + (v1 * k_v2) * (k1 * k_v1) * (k2 * v2) &
                                   + (v1 * k1) * (v2 * k_v1) * (k2 * k_v2) &
                                   + (v1 * k2) * (v2 * k_v1) * (k1 * k_v2) &
                                    - (v1 * v2) * (k1 * k_v2) * (k2 * k_v1) &
                                    - (v1 * v2) * (k1 * k_v1) * (k2 * k_v2) &
                                    - (v1 * k2) * (v2 * k1) * (k_v1 * k_v2) &
                                    - (v1 * k1) * (v2 * k2) * (k_v1 * k_v2) )
     end function phi_phi2v_m_7
V_{2}^{\mu} = g\left(k_{1}^{\mu}\left(V_{1}\cdot k_{V_{2}}\right)\left(k_{2}\cdot k_{V_{1}}\right) + k_{2}^{\mu}\left(V_{1}\cdot k_{V_{2}}\right)\left(k_{1}\cdot k_{V_{1}}\right) + k_{V_{1}}^{\mu}\left(V_{1}\cdot k_{1}\right)\left(k_{2}\cdot k_{V_{2}}\right) + k_{V_{1}}^{\mu}\left(V_{1}\cdot k_{2}\right)\left(k_{1}\cdot k_{V_{2}}\right) - k_{1}^{\mu}\left(V_{1}\cdot k_{2}\right)\left(k_{V_{1}}\cdot k_{V_{2}}\right) + k_{V_{1}}^{\mu}\left(V_{1}\cdot k_{V_{2}}\right) + k_{V_{1}}^{\mu}\left(V_{1}\cdot k_{V_{2}}\right)\left(k_{V_{1}}\cdot k_{V_{2}}\right) + k_{V_{1}}^{\mu}\left(V_{1}\cdot k_{V_{2}}\right) + k_{V_{1}}^{\mu}\left(V_{1}\cdot k_{V_{2}}\right)\left(k_{V_{1}}\cdot k_{V_{2}}\right) + k_{V_{1}}^{\mu}\left(V_{1}\cdot k_{V_{2
\langle Implementation \ of \ couplings \rangle + \equiv
     pure function v_phi2v_m_7 (g, phi1, k1, phi2, k2, v1, k_v1) result (v2)
          complex(kind=default), intent(in) :: g, phi1, phi2
          type(momentum), intent(in) :: k1, k2, k_v1
          type(vector), intent(in) :: v1
          type(momentum) :: k_v2
          type(vector) :: v2
          k_v2 = - k_v1 - k1 - k2
          v2 = g * phi1 * phi2 * &
                              (k1 * (v1 * k_v2) * (k2 * k_v1) &
                              + k2 * (v1 * k_v2) * (k1 * k_v1) &
                              + k_v1 * (v1 * k1) * (k2 * k_v2) &
                              + k_v1 * (v1 * k2) * (k1 * k_v2) &
                              -k1 * (v1 * k2) * (k_v1 * k_v2) &
                              - k2 * (v1 * k1) * (k_v1 * k_v2) &
                              -v1*(k1*k_v2)*(k2*k_v1) &
                              -v1*(k1*k_v1)*(k2*k_v2))
     end function v_phi2v_m_7
                                                                                Transversal Gauge 4 Dim-8 Couplings
\langle Declaration \ of \ couplings \rangle + \equiv
     public :: g_dim8g3_t_0, g_dim8g3_t_1, g_dim8g3_t_2
                                              V_1^{\mu} = g \left[ k_2^{\mu} \left( k_1 \cdot V_2 \right) - V_2^{\mu} \left( k_1 \cdot k_2 \right) \right] \left[ \left( k_3 \cdot V_4 \right) \left( k_4 \cdot V_3 \right) - \left( V_3 \cdot V_4 \right) \left( k_3 \cdot k_4 \right) \right]
                                                                                                                                                                                                                                                   (Y.82)
 \langle Implementation \ of \ couplings \rangle + \equiv
     pure function g_dim8g3_t_0 (g, v2, k2, v3, k3, v4, k4) result (v1)
          complex(kind=default), intent(in) :: g
          type(vector), intent(in) :: v2, v3, v4
          type(momentum), intent(in) :: k2, k3, k4
          type(vector) :: v1
          type(momentum) :: k1
          k1 = - k2 - k3 - k4
          v1 = g * (k2 * (k1 * v2) - v2 * (k1 * k2)) &
                            * ((k3 * v4) * (k4 * v3) - (v3 * v4) * (k3 * k4))
     end function g_dim8g3_t_0
                                              V_1^{\mu} = g \left[ k_2^{\mu} \left( k_1 \cdot V_2 \right) - V_2^{\mu} \left( k_1 \cdot k_2 \right) \right] \left[ \left( k_3 \cdot V_4 \right) \left( k_4 \cdot V_3 \right) - \left( V_3 \cdot V_4 \right) \left( k_3 \cdot k_4 \right) \right]
                                                                                                                                                                                                                                                   (Y.83)
 \langle Implementation \ of \ couplings \rangle + \equiv
     pure function g_dim8g3_t_1 (g, v2, k2, v3, k3, v4, k4) result (v1)
          complex(kind=default), intent(in) :: g
          type(vector), intent(in) :: v2, v3, v4
          type(momentum), intent(in) :: k2, k3, k4
          type(vector) :: v1
          type(momentum) :: k1
          k1 = - k2 - k3 - k4
          v1 = g * (v3 * (v2 * k4) * (k1 * k3) * (k2 * v4) &
                              + v4 * (v2 * k3) * (k1 * k4) * (k2 * v3) &
                              + k3 * (v2 * v4) * (k1 * v3) * (k2 * k4) &
```

```
+ k4 * (v2 * v3) * (k1 * v4) * (k2 * k3) &
                              - v3 * (v2 * v4) * (k1 * k3) * (k2 * k4) &
                              - v4 * (v2 * v3) * (k1 * k4) * (k2 * k3) &
                              - k3 * (v2 * k4) * (k1 * v3) * (k2 * v4) &
                              - k4 * (v2 * k3) * (k1 * v4) * (k2 * v3))
     end function g_dim8g3_t_1
V_{1}^{\mu} = g \left[ k_{2}^{\mu} \left( V_{2} \cdot k_{3} \right) \left( V_{3} \cdot k_{4} \right) \left( V_{4} \cdot k_{1} \right) \right. \\ \left. + k_{3}^{\mu} \left( V_{2} \cdot k_{1} \right) \left( V_{3} \cdot k_{4} \right) \left( V_{4} \cdot k_{2} \right) \right. \\ \left. + k_{2}^{\mu} \left( V_{2} \cdot k_{4} \right) \left( V_{3} \cdot k_{1} \right) \left( V_{4} \cdot k_{3} \right) \right. \\ \left. + k_{4}^{\mu} \left( V_{2} \cdot k_{1} \right) \left( V_{3} \cdot k_{2} \right) \left( V_{4} \cdot k_{2} \right) \right] \right] \left. + k_{3}^{\mu} \left( V_{2} \cdot k_{1} \right) \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2}
\langle Implementation \ of \ couplings \rangle + \equiv
    pure function g_dim8g3_t_2 (g, v2, k2, v3, k3, v4, k4) result (v1)
         complex(kind=default), intent(in) :: g
         type(vector), intent(in) :: v2, v3, v4
         type(momentum), intent(in) :: k2, k3, k4
         type(vector) :: v1
         type(momentum) :: k1
         k1 = - k2 - k3 - k4
         v1 = g * (k2 * (v2 * k3) * (v3 * k4) * (v4 * k1) &
                              + k3 * (v2 * k1) * (v3 * k4) * (v4 * k2) &
                              + k2 * (v2 * k4) * (v3 * k1) * (v4 * k3) &
                              + k4 * (v2 * k1) * (v3 * k2) * (v4 * k3) &
                              + k4 * (v2 * k3) * (v3 * v4) * (k1 * k2) &
                              + k3 * (v2 * k4) * (v3 * v4) * (k1 * k2) &
                              -k3*(v2*v4)*(v3*k4)*(k1*k2) &
                              -v4*(v2*k3)*(v3*k4)*(k1*k2) &
                              - k4 * (v2 * v3) * (v4 * k3) * (k1 * k2) &
                              - v3 * (v2 * k4) * (v4 * k3) * (k1 * k2) &
                              - k2 * (v2 * k4) * (v3 * v4) * (k1 * k3) &
                              + k2 * (v2 * v4) * (v3 * k4) * (k1 * k3) &
                              -v2*(v3*k4)*(v4*k2)*(k1*k3)&
                              - k2 * (v2 * k3) * (v3 * v4) * (k1 * k4) &
                              + k2 * (v2 * v3) * (v4 * k3) * (k1 * k4) &
                              -v2*(v3*k2)*(v4*k3)*(k1*k4)&
                              - k4 * (v2 * k1) * (v3 * v4) * (k2 * k3) &
                              + v4 * (v2 * k1) * (v3 * k4) * (k2 * k3) &
                              - v2 * (v3 * k4) * (v4 * k1) * (k2 * k3) &
                              + v2 * (v3 * v4) * (k1 * k4) * (k2 * k3) &
                              - k3 * (v2 * k1) * (v3 * v4) * (k2 * k4) &
                              + v3 * (v2 * k1) * (v4 * k3) * (k2 * k4) &
                              -v2*(v3*k1)*(v4*k3)*(k2*k4)&
                              + v2 * (v3 * v4) * (k1 * k3) * (k2 * k4) &
                              - k2 * (v2 * v4) * (v3 * k1) * (k3 * k4) &
                              - v4 * (v2 * k1) * (v3 * k2) * (k3 * k4) &
                              - k2 * (v2 * v3) * (v4 * k1) * (k3 * k4) &
                              + v2 * (v3 * k2) * (v4 * k1) * (k3 * k4) &
                              - v3 * (v2 * k1) * (v4 * k2) * (k3 * k4) &
                              + v2 * (v3 * k1) * (v4 * k2) * (k3 * k4) &
                              + v4 * (v2 * v3) * (k1 * k2) * (k3 * k4) &
                              + v3 * (v2 * v4) * (k1 * k2) * (k3 * k4))
     end function g_dim8g3_t_2
                                                                                       Mixed Gauge 1 Dim-8 Couplings
\langle Declaration \ of \ couplings \rangle + \equiv
    public :: g_dim8g3_m_0, g_dim8g3_m_1, g_dim8g3_m_7
  V_{1}^{\mu} = g_{1} \left[ V_{2}^{\mu} \left( V_{3} \cdot V_{4} \right) \left( k_{1} \cdot k_{2} \right) - k_{2}^{\mu} \left( V_{2} \cdot k_{1} \right) \left( V_{3} \cdot V_{4} \right) \right] + g_{2} \left[ V_{2}^{\mu} \left( V_{3} \cdot V_{4} \right) \left( k_{3} \cdot k_{4} \right) - V_{2}^{\mu} \left( V_{3} \cdot k_{4} \right) \left( V_{4} \cdot k_{3} \right) \right]  (Y.85)
\langle Implementation \ of \ couplings \rangle + \equiv
    pure function g_dim8g3_m_0 (g1, g2, v2, k2, v3, k3, v4, k4) result (v1)
         complex(kind=default), intent(in) :: g1, g2
```

type(vector), intent(in) :: v2, v3, v4

```
type(momentum), intent(in) :: k2, k3, k4
                 type(vector) :: v1
                 type(momentum) :: k1
                 k1 = - k2 - k3 - k4
                 v1 = g1 * (v2 * (v3 * v4) * (k1 * k2) &
                                                        - k2 * (v2 * k1) * (v3 * v4)) &
                              + g2 * (v2 * (v3 * v4) * (k3 * k4) &
                                                        - v2 * (v3 * k4) * (v4 * k3))
         end function g_dim8g3_m_0
V_{1}^{\mu}=g_{1}\left[k_{2}^{\mu}\left(V_{2}\cdot V_{4}\right)\left(V_{3}\cdot k_{1}\right)\right.\\ \left.+V_{4}^{\mu}\left(V_{2}\cdot k_{1}\right)\left(V_{3}\cdot k_{2}\right)\right.\\ \left.+k_{2}^{\mu}\left(V_{2}\cdot V_{3}\right)\left(V_{4}\cdot k_{1}\right)\right.\\ \left.+V_{3}^{\mu}\left(V_{2}\cdot k_{1}\right)\left(V_{4}\cdot k_{2}\right)\right.\\ \left.-V_{2}^{\mu}\left(V_{3}\cdot k_{2}\right)\left(V_{4}\cdot k_{1}\right)\right.\\ \left.+V_{3}^{\mu}\left(V_{2}\cdot V_{3}\right)\left(V_{4}\cdot k_{1}\right)\right.\\ \left.+V_{3}^{\mu}\left(V_{2}\cdot V_{3}\right)\left(V_{3}\cdot k_{2}\right)\right.\\ \left.+V_{4}^{\mu}\left(V_{2}\cdot V_{3}\right)\left(V_{3}\cdot V_{3}\right)\right.\\ \left.+V_{4}^{\mu}\left(V_{3}\cdot V_{3}\right)\left(V_{3}\cdot V_{3}\right)\right]
\langle Implementation \ of \ couplings \rangle + \equiv
        pure function g_{m} = 1 (g1, g2, v2, k2, v3, k3, v4, k4) result (v1)
                 complex(kind=default), intent(in) :: g1, g2
                 type(vector), intent(in) :: v2, v3, v4
                 type(momentum), intent(in) :: k2, k3, k4
                 type(vector) :: v1
                 type(momentum) :: k1
                 k1 = - k2 - k3 - k4
                 v1 = g1 * (k2 * (v2 * v4) * (v3 * k1)
                                                        + v4 * (v2 * k1) * (v3 * k2)
                                                        + k2 * (v2 * v3) * (v4 * k1)
                                                        + v3 * (v2 * k1) * (v4 * k2)
                                                        -v2*(v3*k2)*(v4*k1)
                                                        -v2*(v3*k1)*(v4*k2)
                                                        - v4 * (v2 * v3) * (k1 * k2)
                                                        - v3 * (v2 * v4) * (k1 * k2)) &
                              + g2 * (k3 * (v2 * v4) * (v3 * k4)
                                                        - k4 * (v2 * k3) * (v3 * v4)
                                                        - k3 * (v2 * k4) * (v3 * v4)
                                                        + v4 * (v2 * k3) * (v3 * k4)
                                                       + k4 * (v2 * v3) * (v4 * k3)
                                                        + v3 * (v2 * k4) * (v4 * k3)
                                                        - v4 * (v2 * v3) * (k3 * k4)
                                                        - v3 * (v2 * v4) * (k3 * k4))
         end function g_dim8g3_m_1
V_{1}^{\mu} = g_{1} \left[ V_{2}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{4} \cdot k_{1} \right) \right. \\ \left. + V_{2}^{\mu} \left( V_{4} \cdot k_{1} \right) \left( V_{4} \cdot k_{2} \right) \right. \\ \left. + V_{4}^{\mu} \left( V_{2} \cdot V_{3} \right) \left( k_{1} \cdot k_{2} \right) \right. \\ \left. + V_{3}^{\mu} \left( V_{2} \cdot V_{4} \right) \left( k_{1} \cdot k_{2} \right) \\ \left. - k_{2}^{\mu} \left( V_{2} \cdot V_{4} \right) \left( V_{3} \cdot k_{1} \right) \right] \right] \\ \left. - k_{2}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{4} \cdot k_{1} \right) \right] + V_{2}^{\mu} \left( V_{3} \cdot k_{2} \right) \\ \left. - k_{3}^{\mu} \left( V_{2} \cdot V_{3} \right) \left( V_{3} \cdot k_{2} \right) \right] + V_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \right] \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \right] \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \right] \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \right] \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \right] \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \right] \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \right] \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \right] \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \right] \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \right] \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \right] \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \right] \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \right] \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \right] \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right) \left( V_{3} \cdot k_{2} \right) \right] \\ \left. - k_{3}^{\mu} \left( V_{3} \cdot k_{2} \right)
\langle Implementation \ of \ couplings \rangle + \equiv
        pure function g_dim8g3_m_7 (g1, g2, g3, v2, k2, v3, k3, v4, k4) result (v1)
                 complex(kind=default), intent(in) :: g1, g2, g3
                 type(vector), intent(in) :: v2, v3, v4
                 type(momentum), intent(in) :: k2, k3, k4
                 type(vector) :: v1
                 type(momentum) :: k1
                 k1 = - k2 - k3 - k4
                 v1 = g1 * (v2 * (v3 * k2) * (v4 * k1)
                                                        + v2 * (v3 * k1) * (v4 * k2)
                                                        + v4 * (v2 * v3) * (k1 * k2)
                                                        + v3 * (v2 * v4) * (k1 * k2)
                                                        - k2 * (v2 * v4) * (v3 * k1)
                                                        - v4 * (v2 * k1) * (v3 * k2)
                                                        - k2 * (v2 * v3) * (v4 * k1)
                                                         -v3*(v2*k1)*(v4*k2)) &
                              + g2 * (k3 * (v2 * k1) * (v3 * v4)
                                                        + k4 * (v2 * k1) * (v3 * v4)
                                                        + k2 * (v2 * k3) * (v3 * v4)
                                                        + k2 * (v2 * k4) * (v3 * v4)
                                                        + v4 * (v2 * k4) * (v3 * k1)
                                                        + k4 * (v2 * v4) * (v3 * k2) &
                                                        + v3 * (v2 * k3) * (v4 * k1) &
```

```
+ v2 * (v3 * k4) * (v4 * k1) &
          + k3 * (v2 * v3) * (v4 * k2) &
          + v2 * (v3 * k4) * (v4 * k2) &
          + v2 * (v3 * k1) * (v4 * k3) &
          + v2 * (v3 * k2) * (v4 * k3)
          + v4 * (v2 * v3) * (k1 * k3)
          + v3 * (v2 * v4) * (k1 * k4)
          + v3 * (v2 * v4) * (k2 * k3)
          + v4 * (v2 * v3) * (k2 * k4)
          - k4 * (v2 * v4) * (v3 * k1)
          - v4 * (v2 * k3) * (v3 * k1)
          - k3 * (v2 * v4) * (v3 * k2)
          - v4 * (v2 * k4) * (v3 * k2)
          - k2 * (v2 * v4) * (v3 * k4)
          - v4 * (v2 * k1) * (v3 * k4)
          -k3 * (v2 * v3) * (v4 * k1) &
          -v3*(v2*k4)*(v4*k1) &
          - k4 * (v2 * v3) * (v4 * k2) &
          -v3*(v2*k3)*(v4*k2) &
          - k2 * (v2 * v3) * (v4 * k3) &
          -v3*(v2*k1)*(v4*k3) &
          -v2*(v3*v4)*(k1*k3) &
          - v2 * (v3 * v4) * (k1 * k4) &
          -v2*(v3*v4)*(k2*k3) &
          - v2 * (v3 * v4) * (k2 * k4)) &
    + g3 * (k4 * (v2 * k3) * (v3 * v4) &
          + k3 * (v2 * k4) * (v3 * v4)
          + v4 * (v2 * v3) * (k3 * k4)
          + v3 * (v2 * v4) * (k3 * k4)
          - k3 * (v2 * v4) * (v3 * k4)
          - v4 * (v2 * k3) * (v3 * k4)
          - k4 * (v2 * v3) * (v4 * k3)
          - v3 * (v2 * k4) * (v4 * k3))
end function g_dim8g3_m_7
```

 $\langle Declaration \ of \ couplings \rangle + \equiv$

Y.23 Graviton Couplings

```
public :: s_gravs, v_gravv, grav_ss, grav_vv
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function s_gravs (g, m, k1, k2, t, s) result (phi)
    {\tt complex(kind=default),\ intent(in)\ ::\ g,\ s}
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: k1, k2
    type(tensor), intent(in) :: t
    complex(kind=default) :: phi, t_tr
    t_{tr} = t\%t(0,0) - t\%t(1,1) - t\%t(2,2) - t\%t(3,3)
    phi = g * s * (((t*k1)*k2) + ((t*k2)*k1) &
         -g * (m**2 + (k1*k2))*t_tr)/2.0_default
  end function s_gravs
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function grav_ss (g, m, k1, k2, s1, s2) result (t)
    complex(kind=default), intent(in) :: g, s1, s2
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: k1, k2
    type(tensor) :: t_metric, t
    t_metric%t = 0
    t_metric\%t(0,0) = 1.0_default
    t_metric(t(1,1) = -1.0_default)
    t_metric\%t(2,2) = -1.0_default
    t_metric\%t(3,3) = -1.0_default
    t = g*s1*s2/2.0_default * (-(m**2 + (k1*k2)) * t_metric &
      + (k1.tprod.k2) + (k2.tprod.k1))
  end function grav_ss
```

```
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function v_gravv (g, m, k1, k2, t, v) result (vec)
   complex(kind=default), intent(in) :: g
   real(kind=default), intent(in) :: m
   type(momentum), intent(in) :: k1, k2
   {\tt type(vector),\ intent(in)\ ::\ v}
   type(tensor), intent(in) :: t
   complex(kind=default) :: t_tr
   real(kind=default) :: xi
   type(vector) :: vec
   xi = 1.0_default
   t_{tr} = t\%t(0,0) - t\%t(1,1) - t\%t(2,2) - t\%t(3,3)
   vec = (-g)/ 2.0_default * (((k1*k2) + m**2) * &
         (t*v + v*t - t_tr * v) + t_tr * (k1*v) * k2 &
         - (k1*v) * ((k2*t) + (t*k2)) &
         - ((k1*(t*v)) + (v*(t*k1))) * k2 &
         + ((k1*(t*k2)) + (k2*(t*k1))) * v)
  !!!
            Unitarity gauge: xi -> Infinity
  111
            + (1.0_default/xi) * (t_tr * ((k1*v)*k2) + &
  !!!
            (k2*v)*k2 + (k2*v)*k1 - (k1*(t*v))*k1 + &
  !!!
            (k2*v)*(k2*t) - (v*(t*k1))*k1 - (k2*v)*(t*k2)))
 end function v_gravv
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function grav_vv (g, m, k1, k2, v1, v2) result (t)
   complex(kind=default), intent(in) :: g
   type(momentum), intent(in) :: k1, k2
   real(kind=default), intent(in) :: m
   real(kind=default) :: xi
   type(vector), intent (in) :: v1, v2
   type(tensor) :: t_metric, t
   xi = 0.00001_default
   t_metric%t = 0
   t_metric\%t(0,0) = 1.0_default
   t_metric\%t(1,1) = -1.0_default
   t_metric\%t(2,2) = -1.0_default
   t_metric\%t(3,3) = -1.0_default
   t = (-g)/2.0_{default} * ( &
         ((k1*k2) + m**2) * ( &
         (v1.tprod.v2) + (v2.tprod.v1) - (v1*v2) * t_metric) &
         + (v1*k2)*(v2*k1)*t_metric &
         - (k2*v1)*((v2.tprod.k1) + (k1.tprod.v2)) &
         - (k1*v2)*((v1.tprod.k2) + (k2.tprod.v1)) &
         + (v1*v2)*((k1.tprod.k2) + (k2.tprod.k1)))
  !!!
            Unitarity gauge: xi -> Infinity
  111
            + (1.0_default/xi) * ( &
            ((k1*v1)*(k1*v2) + (k2*v1)*(k2*v2) + (k1*v1)*(k2*v2))* &
  !!!
  !!!
            t_metric) - (k1*v1) * ((k1.tprod.v2) + (v2.tprod.k1)) &
  !!!
            - (k2*v2) * ((k2.tprod.v1) + (v1.tprod.k2)))
 end function grav_vv
                                      Y.24 Tensor Couplings
\langle Declaration \ of \ couplings \rangle + \equiv
 public :: t2_vv, v_t2v, t2_vv_cf, v_t2v_cf, &
         t2_vv_1, v_t2v_1, t2_vv_t, v_t2v_t, &
         t2_phi2, phi_t2phi, t2_phi2_cf, phi_t2phi_cf
                                         T_{\mu\nu} = g * V_{1\,\mu} V_{2\,\nu} + V_{1\,\nu} V_{2\,\mu}
                                                                                                        (Y.88)
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function t2_vv (g, v1, v2) result (t)
   complex(kind=default), intent(in) :: g
   type(vector), intent(in) :: v1, v2
   type(tensor) :: t
    type(tensor) :: tmp
```

```
tmp = v1.tprod.v2
    t\%t = g * (tmp\%t + transpose (tmp\%t))
  end function t2_vv
                                               V_{1\,\mu} = g * T_{\mu\nu} V_2^{\nu} + T_{\nu\mu} V_2^{\nu}
                                                                                                                    (Y.89)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function v_t2v (g, t, v) result (tv)
    complex(kind=default), intent(in) :: g
    type(tensor), intent(in) :: t
    type(vector), intent(in) :: v
    type(vector) :: tv
    type(tensor) :: tmp
    tmp%t = t%t + transpose (t%t)
    tv = g * (tmp * v)
  end function v_t2v
                                                   T_{\mu\nu} = -\frac{g}{2} V_1^{\rho} V_{2\rho}
                                                                                                                    (Y.90)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function t2_vv_cf (g, v1, v2) result (t)
    complex(kind=default), intent(in) :: g
    complex(kind=default) :: tmp_s
    type(vector), intent(in) :: v1, v2
    type(tensor) :: t_metric, t
    t_metric%t = 0
    t_metric(0,0) = 1.0_default
    t_metric\%t(1,1) = -1.0_default
    t_metric\%t(2,2) = -1.0_default
    t_metric\%t(3,3) = -1.0_default
    tmp_s = v1 * v2
    t\%t = - (g /2.0_default) * tmp_s * t_metric\%t
  end function t2_vv_cf
                                                    V_{1\,\mu} = -\frac{g}{2} T^{\nu}_{\nu} V^{\mu}_{2}
                                                                                                                    (Y.91)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function v_t2v_cf (g, t, v) result (tv)
    complex(kind=default), intent(in) :: g
    type(tensor), intent(in) :: t
    type(vector), intent(in) :: v
    type(vector) :: tv, tmp_tv
    tmp_tv = (t\%t(0,0)-t\%t(1,1)-t\%t(2,2)-t\%t(3,3)) * v
    tv = - (g/2.0_default) * tmp_tv
  end function v_t2v_cf
                                     T_{\mu\nu} = g * (k_1 \mu k_2 \nu + k_1 \nu k_2 \mu) \phi_1 (k_1) \phi_1 (k_2)
                                                                                                                    (Y.92)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function t2_phi2 (g, phi1, k1, phi2, k2) result (t)
    {\tt complex(kind=default),\ intent(in)\ ::\ g,\ phi1,\ phi2}
    type(momentum), intent(in) :: k1, k2
    type(tensor) :: t
    type(tensor) :: tmp
    tmp = k1.tprod.k2
    t%t = g * (tmp%t + transpose (tmp%t)) * phi1 * phi2
  end function t2_phi2
                                      \phi_1(k_1) = g * (T_{\mu\nu}k_1^{\mu}k_2^{\nu} + T_{\nu\mu}k_2^{\mu}k_1^{\nu}) \phi_2(k_2)
                                                                                                                    (Y.93)
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function phi_t2phi (g, t, kt, phi2, k2) result (phi1)
    complex(kind=default), intent(in) :: g, phi2
    type(tensor), intent(in) :: t
    type(momentum), intent(in) :: kt, k2
    type(momentum) :: k1
```

```
complex(kind=default) :: phi1
    type(tensor) :: tmp
    k1 = -kt - k2
    tmp\%t = t\%t + transpose (t\%t)
    phi1 = g * ((tmp * k2) * k1) * phi2
  end function phi_t2phi
                                         T_{\mu\nu} = -\frac{g}{2} k_1^{\rho} k_2 \rho \phi_1 (k_1) \phi_2 (k_2)
                                                                                                            (Y.94)
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function t2_phi2_cf (g, phi1, k1, phi2, k2) result (t)
    complex(kind=default), intent(in) :: g, phi1, phi2
    complex(kind=default) :: tmp_s
    type(momentum), intent(in) :: k1, k2
    type(tensor) :: t_metric, t
    t_metric%t = 0
    t_{metric}(t(0,0)) = 1.0_{default}
    t_metric\%t(1,1) = -1.0_default
    t_metric\%t(2,2) = -1.0_default
    t_metric\%t(3,3) = -1.0_default
    tmp_s = (k1 * k2) * phi1 * phi2
    t\%t = - (g /2.0_default) * tmp_s * t_metric\%t
  end function t2_phi2_cf
                                        \phi_1(k_1) = -\frac{g}{2} T_{\nu}^{\nu} (k_1 \cdot k_2) \, \phi_2(k_2)
                                                                                                           (Y.95)
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function phi_t2phi_cf (g, t, kt, phi2, k2) result (phi1)
    complex(kind=default), intent(in) :: g, phi2
    type(tensor), intent(in) :: t
    type(momentum), intent(in) :: kt, k2
    type(momentum) :: k1
    complex(kind=default) :: tmp_ts, phi1
    k1 = - kt - k2
    tmp_ts = (t\%t(0,0)-t\%t(1,1)-t\%t(2,2)-t\%t(3,3))
    phi1 = - (g/2.0_default) * tmp_ts * (k1 * k2) * phi2
  end function phi_t2phi_cf
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function t2_vv_1 (g, v1, v2) result (t)
    complex(kind=default), intent(in) :: g
    complex(kind=default) :: tmp_s
    type(vector), intent(in) :: v1, v2
    type(tensor) :: tmp
    type(tensor) :: t_metric, t
    t_metric%t = 0
    t_metric\%t(0,0) = 1.0_default
    t_metric\%t(1,1) = -1.0_default
    t_metric\%t(2,2) = -1.0_default
    t_metric\%t(3,3) = -1.0_default
    tmp = v1.tprod.v2
    tmp_s = v1 * v2
    t\%t = g * (tmp\%t + transpose (tmp\%t) - tmp_s * t_metric\%t)
  end function t2_vv_1
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function v_t2v_1 (g, t, v) result (tv)
    complex(kind=default), intent(in) :: g
    type(tensor), intent(in) :: t
    type(vector), intent(in) :: v
    type(vector) :: tv, tmp_tv
    type(tensor) :: tmp
    tmp_tv = (t\%t(0,0)-t\%t(1,1)-t\%t(2,2)-t\%t(3,3)) * v
    tmp%t = t%t + transpose (t%t)
    tv = g * (tmp * v - tmp_tv)
  end function v_t2v_1
```

```
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function t2_vv_t (g, v1, k1, v2, k2) result (t)
    complex(kind=default), intent(in) :: g
    complex(kind=default) :: tmp_s
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(tensor) :: tmp, tmp_v1k2, tmp_v2k1, tmp_k1k2, tmp2
    type(tensor) :: t_metric, t
    t_metric%t = 0
    t_metric\%t(0,0) = 1.0_default
    t_metric(t(1,1) = -1.0_default)
    t_metric\%t(2,2) = -1.0_default
    t_metric\%t(3,3) = -1.0_default
    tmp = v1.tprod.v2
    tmp_s = v1 * v2
    tmp_v1k2 = (v2 * k1) * (v1.tprod.k2)
    tmp_v2k1 = (v1 * k2) * (v2.tprod.k1)
    tmp_k1k2 = tmp_s * (k1.tprod.k2)
    tmp2\%t = tmp_v1k2\%t + tmp_v2k1\%t - tmp_k1k2\%t
    t\%t = g * ( (k1*k2) * (tmp\%t + transpose (tmp\%t) - tmp_s * t_metric\%t ) &
         + ((v1 * k2) * (v2 * k1)) * t_metric%t &
         - tmp2%t - transpose(tmp2%t))
  end function t2_vv_t
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function v_t2v_t (g, t, kt, v, kv) result (tv)
    complex(kind=default), intent(in) :: g
    type(tensor), intent(in) :: t
    {\tt type(vector),\ intent(in)\ ::\ v}
    type(momentum), intent(in) :: kt, kv
    type(momentum) :: kout
    type(vector) :: tv, tmp_tv
    type(tensor) :: tmp
    kout = - (kt + kv)
    tmp_tv = (t%t(0,0)-t%t(1,1)-t%t(2,2)-t%t(3,3)) * v
    tmp%t = t%t + transpose (t%t)
    tv = g * ((tmp * v - tmp_tv) * (kv * kout)&
         + ( t\%t(0,0)-t\%t(1,1)-t\%t(2,2)-t\%t(3,3) ) * (kout * v ) * kv &
         - (kout * v) * ( tmp * kv) &
         - (v* (t * kout) + kout * (t * v)) * kv &
         + (kout* (t * kv) + kv * (t * kout)) * v)
  end function v_t2v_t
\langle Declaration \ of \ couplings \rangle + \equiv
 public :: t2_vv_d5_1, v_t2v_d5_1
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function t2_vv_d5_1 (g, v1, k1, v2, k2) result (t)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(tensor) :: t
    t = (g * (v1 * v2)) * (k1-k2).tprod.(k1-k2)
  end function t2_vv_d5_1
\langle Implementation \ of \ couplings \rangle + \equiv
 pure function v_t2v_d5_1 (g, t1, k1, v2, k2) result (tv)
    {\tt complex(kind=default),\ intent(in)\ ::\ g}
    {\tt type(tensor),\ intent(in)\ ::\ t1}
    type(vector), intent(in) :: v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: tv
    tv = (g * ((k1+2*k2).tprod.(k1+2*k2) * t1)) * v2
  end function v_t2v_d5_1
\langle Declaration \ of \ couplings \rangle + \equiv
  public :: t2_vv_d5_2, v_t2v_d5_2
\langle Implementation \ of \ couplings \rangle + \equiv
```

```
pure function t2_vv_d5_2 (g, v1, k1, v2, k2) result (t)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(tensor) :: t
    t = (g * (k2 * v1)) * (k2-k1).tprod.v2
    t\%t = t\%t + transpose (t\%t)
  end function t2_vv_d5_2
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function v_t2v_d5_2 (g, t1, k1, v2, k2) result (tv)
    complex(kind=default), intent(in) :: g
    type(tensor), intent(in) :: t1
    type(vector), intent(in) :: v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: tv
    type(tensor) :: tmp
    type(momentum) :: k1_k2, k1_2k2
    k1_k2 = k1 + k2
    k1_2k2 = k1_k2 + k2
    tmp%t = t1%t + transpose (t1%t)
    tv = (g * (k1_k2 * v2)) * (k1_2k2 * tmp)
  end function v_t2v_d5_2
\langle Declaration \ of \ couplings \rangle + \equiv
  public :: t2_vv_d7, v_t2v_d7
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function t2_vv_d7 (g, v1, k1, v2, k2) result (t)
    complex(kind=default), intent(in) :: g
    type(vector), intent(in) :: v1, v2
    type(momentum), intent(in) :: k1, k2
    type(tensor) :: t
    t = (g * (k2 * v1) * (k1 * v2)) * (k1-k2).tprod.(k1-k2)
  end function t2_vv_d7
\langle Implementation \ of \ couplings \rangle + \equiv
  pure function v_t2v_d7 (g, t1, k1, v2, k2) result (tv)
    complex(kind=default), intent(in) :: g
    type(tensor), intent(in) :: t1
    type(vector), intent(in) :: v2
    type(momentum), intent(in) :: k1, k2
    type(vector) :: tv
    type(vector) :: k1_k2, k1_2k2
    k1_k2 = k1 + k2
    k1_2k2 = k1_k2 + k2
    tv = (-g * (k1_k2 * v2) * (k1_2k2.tprod.k1_2k2 * t1)) * k2
  end function v_t2v_d7
                                        Y.25 Spinor Couplings
\langle omega\_spinor\_couplings.f90 \rangle \equiv
  \langle Copyleft \rangle
  module omega_spinor_couplings
    use kinds
    use constants
    use omega_spinors
    use omega_vectors
    use omega_tensors
    use omega_couplings
    implicit none
    private
    (Declaration of spinor on shell wave functions)
    \langle Declaration \ of \ spinor \ off \ shell \ wave \ functions \rangle
    \langle Declaration \ of \ spinor \ currents \rangle
    \langle Declaration \ of \ spinor \ propagators \rangle
    integer, parameter, public :: omega_spinor_cpls_2010_01_A = 0
```

$\bar{\psi}(g_V\gamma^\mu-g_A\gamma^\mu\gamma_5)\psi$	$ullet$ va_ff $(g_V,g_A,ar{\psi},\psi)$
$g_V \psi \gamma^\mu \psi$	$ $ v_ff (g_V, ψ, ψ)
$g_A ar{\psi} \gamma_5 \gamma^\mu \psi$	$ extstyle \mathtt{a_ff}(g_A, \psi, \psi)$
$g_L \bar{\psi} \gamma^\mu (1 - \gamma_5) \psi$	t t t t t t t t t t t t t
$g_R \bar{\psi} \gamma^\mu (1 + \gamma_5) \psi$	$ extsf{vr_ff}(g_R,ar{\psi},\psi)$
$V(g_V - g_A \gamma_5)\psi$	$\texttt{f_vaf}(g_V,g_A,V,\psi)$
$g_V V \psi$	$\texttt{f_vf}(g_V, V, \psi)$
$g_A\gamma_5 V\psi$	$ extsf{f_af}(g_A,V,\psi)$
$g_L V(1-\gamma_5)\psi$	$\texttt{f_vlf}(g_L, V, \psi)$
$g_R V(1+\gamma_5)\psi$	$\texttt{f_vrf}(g_R, V, \psi)$
$\bar{\psi}V(g_V-g_A\gamma_5)$	$\texttt{f_fva}(g_V,g_A,\bar{\psi},V)$
$g_V ar{\psi} V$	$f ext{-}fv(g_V,ar{\psi},V)$
$g_A ar{\psi} \gamma_5 V$	$\texttt{f_fa}(g_A, \bar{\psi}, V)$
$g_L \bar{\psi} V(1-\gamma_5)$	$\texttt{f_fvl}(g_L,\bar{\psi},V)$
$g_R \bar{\psi} V (1 + \gamma_5)$	$\texttt{f_fvr}(g_R,\bar{\psi},V)$

Mnemonically abbreviated names of Fortran functions implementing fermionic vector and axial currents.

```
contains
```

(Implementation of spinor on shell wave functions)

(Implementation of spinor off shell wave functions)

⟨Implementation of spinor currents⟩

(Implementation of spinor propagators)

end module omega_spinor_couplings

See table Y.1 for the names of Fortran functions. We could have used long names instead, but this would increase the chance of running past continuation line limits without adding much to the legibility.

Y.25.1 Fermionic Vector and Axial Couplings

There's more than one chiral representation. This one is compatible with HELAS [5].

$$\gamma^0 = \begin{pmatrix} 0 & \mathbf{1} \\ \mathbf{1} & 0 \end{pmatrix}, \ \gamma^i = \begin{pmatrix} 0 & \sigma^i \\ -\sigma^i & 0 \end{pmatrix}, \ \gamma_5 = i\gamma^0\gamma^1\gamma^2\gamma^3 = \begin{pmatrix} -\mathbf{1} & 0 \\ 0 & \mathbf{1} \end{pmatrix}$$
 (Y.96)

Therefore

$$g_S + g_P \gamma_5 = \begin{pmatrix} g_S - g_P & 0 & 0 & 0\\ 0 & g_S - g_P & 0 & 0\\ 0 & 0 & g_S + g_P & 0\\ 0 & 0 & 0 & g_S + g_P \end{pmatrix}$$
(Y.97a)

$$g_V \gamma^0 - g_A \gamma^0 \gamma_5 = \begin{pmatrix} 0 & 0 & g_V - g_A & 0\\ 0 & 0 & 0 & g_V - g_A\\ g_V + g_A & 0 & 0 & 0\\ 0 & g_V + g_A & 0 & 0 \end{pmatrix}$$
(Y.97b)

$$g_V \gamma^1 - g_A \gamma^1 \gamma_5 = \begin{pmatrix} 0 & 0 & 0 & g_V - g_A \\ 0 & 0 & g_V - g_A & 0 \\ 0 & -g_V - g_A & 0 & 0 \\ -g_V - g_A & 0 & 0 & 0 \end{pmatrix}$$
 (Y.97c)

$$g_V \gamma^2 - g_A \gamma^2 \gamma_5 = \begin{pmatrix} 0 & 0 & 0 & -\mathrm{i}(g_V - g_A) \\ 0 & 0 & \mathrm{i}(g_V - g_A) & 0 \\ 0 & \mathrm{i}(g_V + g_A) & 0 & 0 \\ -\mathrm{i}(g_V + g_A) & 0 & 0 & 0 \end{pmatrix}$$
(Y.97d)

$$g_S + g_P \gamma_5 = \begin{pmatrix} g_S - g_P & 0 & 0 & 0 \\ 0 & g_S - g_P & 0 & 0 \\ 0 & 0 & g_S + g_P & 0 \\ 0 & 0 & 0 & g_S + g_P \end{pmatrix}$$

$$g_V \gamma^0 - g_A \gamma^0 \gamma_5 = \begin{pmatrix} 0 & 0 & g_V - g_A & 0 \\ 0 & 0 & 0 & g_V - g_A \\ g_V + g_A & 0 & 0 & 0 \\ 0 & g_V + g_A & 0 & 0 \end{pmatrix}$$

$$g_V \gamma^1 - g_A \gamma^1 \gamma_5 = \begin{pmatrix} 0 & 0 & 0 & g_V - g_A \\ 0 & 0 & g_V - g_A & 0 \\ 0 & -g_V - g_A & 0 & 0 \\ -g_V - g_A & 0 & 0 & 0 \end{pmatrix}$$

$$g_V \gamma^2 - g_A \gamma^2 \gamma_5 = \begin{pmatrix} 0 & 0 & 0 & -i(g_V - g_A) \\ 0 & 0 & i(g_V + g_A) & 0 & 0 \\ -i(g_V + g_A) & 0 & 0 & 0 \\ -i(g_V + g_A) & 0 & 0 & 0 \end{pmatrix}$$

$$g_V \gamma^3 - g_A \gamma^3 \gamma_5 = \begin{pmatrix} 0 & 0 & g_V - g_A & 0 \\ 0 & 0 & i(g_V - g_A) & 0 \\ 0 & 0 & 0 & -g_V + g_A \\ 0 & 0 & 0 & -g_V + g_A \end{pmatrix}$$

$$g_V \gamma^3 - g_A \gamma^3 \gamma_5 = \begin{pmatrix} 0 & 0 & g_V - g_A & 0 \\ 0 & 0 & 0 & -g_V + g_A \\ -g_V - g_A & 0 & 0 & 0 \end{pmatrix}$$

$$(Y.97e)$$

 $\langle Declaration \ of \ spinor \ currents \rangle \equiv$

$\bar{\psi}(g_S + g_P \gamma_5)\psi$	$sp_ff(g_S,g_P,ar{\psi},\psi)$
$g_S ar{\psi} \psi$	$\texttt{s_ff}(g_S,\bar{\psi},\psi)$
$g_P ar{\psi} \gamma_5 \psi$	$\texttt{p_ff}(g_P,\bar{\psi},\psi)$
$g_L \bar{\psi} (1 - \gamma_5) \psi$	$\mathtt{sl_ff}(g_L,\bar{\psi},\psi)$
$g_R \bar{\psi}(1+\gamma_5)\psi$	$\mathtt{sr_ff}(g_R,ar{\psi},\psi)$
$\phi(g_S + g_P \gamma_5)\psi$	$\texttt{f_spf}(g_S, g_P, \phi, \psi)$
$g_S\phi\psi$	$\texttt{f_sf}(g_S,\phi,\psi)$
$g_P\phi\gamma_5\psi$	$\texttt{f_pf}(g_P,\phi,\psi)$
$g_L\phi(1-\gamma_5)\psi$	$\texttt{f_slf}(g_L,\phi,\psi)$
$g_R\phi(1+\gamma_5)\psi$	$\texttt{f_srf}(g_R,\phi,\psi)$
$\bar{\psi}\phi(g_S+g_P\gamma_5)$	$\texttt{f_fsp}(g_S,g_P,\bar{\psi},\phi)$
$g_Sar{\psi}\phi$	$\texttt{f_fs}(g_S,\bar{\psi},\phi)$
$g_P ar{\psi} \phi \gamma_5$	$\texttt{f_fp}(g_P,\bar{\psi},\phi)$
$g_L \bar{\psi} \phi (1 - \gamma_5)$	$\texttt{f_fsl}(g_L,\bar{\psi},\phi)$
$g_R \bar{\psi} \phi (1 + \gamma_5)$	$\texttt{f_fsr}(g_R,\bar{\psi},\phi)$

Table Y.2: Mnemonically abbreviated names of Fortran functions implementing fermionic scalar and pseudo scalar "currents".

```
public :: va_ff, v_ff, a_ff, vl_ff, vr_ff, vlr_ff, grav_ff, va2_ff, &
            tva_ff, tlr_ff, trl_ff, tvam_ff, tlrm_ff, trlm_ff, va3_ff
\langle Implementation \ of \ spinor \ currents \rangle \equiv
 pure function va_ff (gv, ga, psibar, psi) result (j)
   type(vector) :: j
   complex(kind=default), intent(in) :: gv, ga
   type(conjspinor), intent(in) :: psibar
   type(spinor), intent(in) :: psi
   complex(kind=default) :: gl, gr
   complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
   gl = gv + ga
   gr = gv - ga
   g13 = psibar%a(1)*psi%a(3)
   g14 = psibar%a(1)*psi%a(4)
   g23 = psibar%a(2)*psi%a(3)
   g24 = psibar%a(2)*psi%a(4)
   g31 = psibar%a(3)*psi%a(1)
   g32 = psibar%a(3)*psi%a(2)
   g41 = psibar%a(4)*psi%a(1)
   g42 = psibar%a(4)*psi%a(2)
   j\%t = gr * (g13 + g24) + g1 * (
                                              g31 + g42)
   j\%x(1) = gr * (g14 + g23) - g1 * (
                                              g32 + g41)
   j\%x(2) = (gr * ( - g14 + g23) + g1 * ( g32 - g41)) * (0, 1)
   j\%x(3) = gr * (g13 - g24) + g1 * (-g31 + g42)
 end function va_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function va2_ff (gva, psibar, psi) result (j)
   type(vector) :: j
   complex(kind=default), intent(in), dimension(2) :: gva
   type(conjspinor), intent(in) :: psibar
   type(spinor), intent(in) :: psi
   complex(kind=default) :: gl, gr
   complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
   gl = gva(1) + gva(2)
   gr = gva(1) - gva(2)
   g13 = psibar%a(1)*psi%a(3)
   g14 = psibar%a(1)*psi%a(4)
   g23 = psibar%a(2)*psi%a(3)
   g24 = psibar%a(2)*psi%a(4)
   g31 = psibar%a(3)*psi%a(1)
   g32 = psibar%a(3)*psi%a(2)
   g41 = psibar%a(4)*psi%a(1)
   g42 = psibar%a(4)*psi%a(2)
```

```
= gr * (g13 + g24) + g1 * (
                                                g31 + g42)
    j\%x(1) = gr * ( g14 + g23) - g1 * ( g32 + g41)
    j\%x(2) = (gr * ( - g14 + g23) + g1 * ( g32 - g41)) * (0, 1)
    j\%x(3) = gr * (g13 - g24) + g1 * (-g31 + g42)
  end function va2_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function va3_ff (gv, ga, psibar, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gv, ga
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    j = va_ff (gv, ga, psibar, psi)
    j\%t = 0.0_default
  end function va3_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function tva_ff (gv, ga, psibar, psi) result (t)
    type(tensor2odd) :: t
    complex(kind=default), intent(in) :: gv, ga
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    complex(kind=default) :: gl, gr
    complex(kind=default) :: g12, g21, g1m2, g34, g43, g3m4
    gr
           = gv + ga
           = gv - ga
    gl
    g12
           = psibar%a(1)*psi%a(2)
    g21
           = psibar%a(2)*psi%a(1)
    g1m2
          = psibar%a(1)*psi%a(1) - psibar%a(2)*psi%a(2)
           = psibar%a(3)*psi%a(4)
    g34
    g43
           = psibar%a(4)*psi%a(3)
    g3m4
          = psibar%a(3)*psi%a(3) - psibar%a(4)*psi%a(4)
    t\%e(1) = (gl * ( - g12 - g21) + gr * (
                                                g34 + g43)) * (0, 1)
    t\%e(2) = gl * ( - g12 + g21) + gr * (
                                                g34 - g43)
    t\%e(3) = (gl * ( - g1m2)
                              ) + gr * (
                                                g3m4 )) * (0, 1)
    t\%b(1) = gl * ( g12 + g21) + gr * (
                                                g34 + g43)
    t\%b(2) = (g1 * ( - g12 + g21) + gr * ( - g34 + g43)) * (0, 1)
    t\%b(3) = gl * (
                       g1m2
                                 ) + gr * (
                                                g3m4
  end function tva_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function tlr_ff (gl, gr, psibar, psi) result (t)
    type(tensor2odd) :: t
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    t = tva_ff (gr+gl, gr-gl, psibar, psi)
  end function tlr_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function trl_ff (gr, gl, psibar, psi) result (t)
    type(tensor2odd) :: t
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    t = tva_ff (gr+gl, gr-gl, psibar, psi)
  end function trl_ff
\langle \mathit{Implementation of spinor currents} \rangle + \equiv
 pure function tvam_ff (gv, ga, psibar, psi, p) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gv, ga
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    type(momentum), intent(in) :: p
    j = (tva_ff(gv, ga, psibar, psi) * p) * (0,1)
  end function tvam_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
```

```
pure function tlrm_ff (gl, gr, psibar, psi, p) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    type(momentum), intent(in) :: p
    j = tvam_ff (gr+gl, gr-gl, psibar, psi, p)
  end function tlrm_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
  pure function trlm_ff (gr, gl, psibar, psi, p) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    type(momentum), intent(in) :: p
    j = tvam_ff (gr+gl, gr-gl, psibar, psi, p)
  end function trlm_ff
Special cases that avoid some multiplications
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
  pure function v_ff (gv, psibar, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gv
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
    g13 = psibar%a(1)*psi%a(3)
    g14 = psibar%a(1)*psi%a(4)
    g23 = psibar%a(2)*psi%a(3)
    g24 = psibar%a(2)*psi%a(4)
    g31 = psibar%a(3)*psi%a(1)
    g32 = psibar%a(3)*psi%a(2)
    g41 = psibar%a(4)*psi%a(1)
    g42 = psibar%a(4)*psi%a(2)
               gv * ( g13 + g24 + g31 + g42)
    j\%x(1) =
               gv * ( g14 + g23 - g32 - g41)
    j\%x(2) =
               gv * ( - g14 + g23 + g32 - g41) * (0, 1)
    j\%x(3) =
               gv * (g13 - g24 - g31 + g42)
  end function v_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
  pure function a_ff (ga, psibar, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: ga
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
    g13 = psibar%a(1)*psi%a(3)
    g14 = psibar%a(1)*psi%a(4)
    g23 = psibar%a(2)*psi%a(3)
    g24 = psibar%a(2)*psi%a(4)
    g31 = psibar%a(3)*psi%a(1)
    g32 = psibar%a(3)*psi%a(2)
    g41 = psibar%a(4)*psi%a(1)
    g42 = psibar%a(4)*psi%a(2)
          = ga * ( - g13 - g24 + g31 + g42)
    j\%x(1) = - ga * ( g14 + g23 + g32 + g41)
    j\%x(2) = ga * (g14 - g23 + g32 - g41) * (0, 1)
    j\%x(3) =
              ga * ( - g13 + g24 - g31 + g42)
  end function a_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
  pure function vl_ff (gl, psibar, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
```

```
complex(kind=default) :: gl2
    complex(kind=default) :: g31, g32, g41, g42
    g12 = 2 * g1
    g31 = psibar%a(3)*psi%a(1)
    g32 = psibar%a(3)*psi%a(2)
    g41 = psibar%a(4)*psi%a(1)
    g42 = psibar%a(4)*psi%a(2)
         = gl2 * ( g31 + g42)
    j%t
    j\%x(1) = - gl2 * (
                          g32 + g41)
    j\%x(2) = g12 * ( g32 - g41) * (0, 1)

j\%x(3) = g12 * ( - g31 + g42)
  end function vl_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
  pure function vr_ff (gr, psibar, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    complex(kind=default) :: gr2
    complex(kind=default) :: g13, g14, g23, g24
    gr2 = 2 * gr
    g13 = psibar%a(1)*psi%a(3)
    g14 = psibar%a(1)*psi%a(4)
    g23 = psibar%a(2)*psi%a(3)
    g24 = psibar%a(2)*psi%a(4)
    j%t
           = gr2 * ( g13 + g24)
    j\%x(1) = gr2 * (g14 + g23)
    j\%x(2) = gr2 * ( - g14 + g23) * (0, 1)
    j\%x(3) = gr2 * (
                        g13 - g24)
  end function vr_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
  pure function grav_ff (g, m, kb, k, psibar, psi) result (j)
    type(tensor) :: j
    complex(kind=default), intent(in) :: g
    real(kind=default), intent(in) :: m
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    type(momentum), intent(in) :: kb, k
    complex(kind=default) :: g2, g8, c_dum
    type(vector) :: v_dum
    type(tensor) :: t_metric
    t_metric%t = 0
    t_metric\%t(0,0) = 1.0_default
    t_metric\%t(1,1) = -1.0_default
    t_metric\%t(2,2) = -1.0_default
    t_metric\%t(3,3) = -1.0_default
    g2 = g/2.0_default
    g8 = g/8.0_default
    v_dum = v_ff(g8, psibar, psi)
    c_{dum} = (-m) * s_{ff} (g2, psibar, psi) - (kb+k)*v_{dum}
    j = c_dum*t_metric - (((kb+k).tprod.v_dum) + &
          (v_dum.tprod.(kb+k)))
  end function grav_ff
                g_L \gamma_\mu (1 - \gamma_5) + g_R \gamma_\mu (1 + \gamma_5) = (g_L + g_R) \gamma_\mu - (g_L - g_R) \gamma_\mu \gamma_5 = g_V \gamma_\mu - g_A \gamma_\mu \gamma_5
                                                                                                            (Y.98)
... give the compiler the benefit of the doubt that it will optimize the function all. If not, we could inline it ...
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
  pure function vlr_ff (gl, gr, psibar, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    j = va_ff (gl+gr, gl-gr, psibar, psi)
  end function vlr_ff
```

and

$$\psi - \phi \gamma_5 = \begin{pmatrix} 0 & 0 & v_- - a_- & -v^* + a^* \\ 0 & 0 & -v + a & v_+ - a_+ \\ v_+ + a_+ & v^* + a^* & 0 & 0 \\ v + a & v_- + a_- & 0 & 0 \end{pmatrix}$$
(Y.99)

with $v_{\pm} = v_0 \pm v_3$, $a_{\pm} = a_0 \pm a_3$, $v = v_1 + iv_2$, $v^* = v_1 - iv_2$, $a = a_1 + ia_2$, and $a^* = a_1 - ia_2$. But note that \cdot^* is not complex conjugation for complex v_{μ} or a_{μ} .

```
\langle Declaration \ of \ spinor \ currents \rangle + \equiv
 public :: f_vaf, f_vf, f_af, f_vlf, f_vrf, f_vlrf, f_va2f, &
            f_tvaf, f_tlrf, f_trlf, f_tvamf, f_tlrmf, f_trlmf, f_va3f
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_vaf (gv, ga, v, psi) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: gv, ga
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    complex(kind=default) :: gl, gr
    complex(kind=default) :: vp, vm, v12, v12s
    gl = gv + ga
    gr = gv - ga
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v%x(1) + (0,1)*v%x(2)
    v12s = v\%x(1) - (0,1)*v\%x(2)
    vpsi\%a(1) = gr * ( vm * psi\%a(3) - v12s * psi\%a(4))
    vpsi\%a(2) = gr * ( - v12 * psi\%a(3) + vp * psi\%a(4))
    vpsi%a(3) = gl * (     vp * psi%a(1) + v12s * psi%a(2))
    vpsi\%a(4) = gl * ( v12 * psi\%a(1) + vm * psi\%a(2))
  end function f_vaf
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_va2f (gva, v, psi) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in), dimension(2) :: gva
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    complex(kind=default) :: gl, gr
    complex(kind=default) :: vp, vm, v12, v12s
    gl = gva(1) + gva(2)
    gr = gva(1) - gva(2)
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v\%x(1) + (0,1)*v\%x(2)
    v12s = v\%x(1) - (0,1)*v\%x(2)
    vpsi\%a(1) = gr * ( vm * psi\%a(3) - v12s * psi\%a(4))
    vpsi\%a(2) = gr * ( - v12 * psi\%a(3) + vp * psi\%a(4))
    vpsi\%a(3) = gl * ( vp * psi\%a(1) + v12s * psi\%a(2))
    vpsi\%a(4) = gl * ( v12 * psi\%a(1) + vm * psi\%a(2))
  end function f_va2f
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_va3f (gv, ga, v, psi) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: gv, ga
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    complex(kind=default) :: gl, gr
    complex(kind=default) :: vp, vm, v12, v12s
    gl = gv + ga
    gr = gv - ga
    vp = v\%x(3) !+ v\%t
    vm = - v\%x(3) !+ v\%t
    v12 = v\%x(1) + (0,1)*v\%x(2)
    v12s = v%x(1) - (0,1)*v%x(2)
    vpsi\%a(1) = gr * ( vm * psi\%a(3) - v12s * psi\%a(4))
```

vpsi%a(2) = gr * (- v12 * psi%a(3) + vp * psi%a(4))

```
vpsi\%a(3) = gl * ( vp * psi\%a(1) + v12s * psi\%a(2))
    vpsi\%a(4) = gl * (
                         v12 * psi%a(1) + vm * psi%a(2))
  end function f_va3f
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_tvaf (gv, ga, t, psi) result (tpsi)
    type(spinor) :: tpsi
    complex(kind=default), intent(in) :: gv, ga
    type(tensor2odd), intent(in) :: t
    type(spinor), intent(in) :: psi
    complex(kind=default) :: gl, gr
    complex(kind=default) :: e21, e21s, b12, b12s, be3, be3s
         = gv + ga
    gl
         = gv - ga
    e21 = t\%e(2) + t\%e(1)*(0,1)
    e21s = t\%e(2) - t\%e(1)*(0,1)
    b12 = t\%b(1) + t\%b(2)*(0,1)
    b12s = t\%b(1) - t\%b(2)*(0,1)
    be3 = t\%b(3) + t\%e(3)*(0,1)
    be3s = t\%b(3) - t\%e(3)*(0,1)
    tpsi%a(1) =
                   2*gl * ( psi%a(1) * be3 + psi%a(2) * ( e21 +b12s))
                   2*gl * ( -psi%a(2) * be3 + psi%a(1) * (-e21s+b12 ))
    tpsi%a(2) =
                   2*gr * ( psi%a(3) * be3s + psi%a(4) * (-e21 +b12s))
    tpsi\%a(3) =
    tpsi\%a(4) =
                   2*gr * ( -psi%a(4) * be3s + psi%a(3) * ( e21s+b12 ))
  end function f_tvaf
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_tlrf (gl, gr, t, psi) result (tpsi)
    type(spinor) :: tpsi
    complex(kind=default), intent(in) :: gl, gr
    type(tensor2odd), intent(in) :: t
    type(spinor), intent(in) :: psi
    tpsi = f_tvaf (gr+gl, gr-gl, t, psi)
  end function f_tlrf
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_trlf (gr, gl, t, psi) result (tpsi)
    type(spinor) :: tpsi
    complex(kind=default), intent(in) :: gl, gr
    type(tensor2odd), intent(in) :: t
    type(spinor), intent(in) :: psi
    tpsi = f_tvaf (gr+gl, gr-gl, t, psi)
  end function f_trlf
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_tvamf (gv, ga, v, psi, k) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: gv, ga
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    type(tensor2odd) :: t
    t = (v.wedge.k) * (0, 0.5)
    vpsi = f_tvaf(gv, ga, t, psi)
  end function f_tvamf
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_tlrmf (gl, gr, v, psi, k) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: gl, gr
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    vpsi = f_tvamf (gr+gl, gr-gl, v, psi, k)
  end function f_tlrmf
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_trlmf (gr, gl, v, psi, k) result (vpsi)
    type(spinor) :: vpsi
```

```
complex(kind=default), intent(in) :: gl, gr
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    vpsi = f_tvamf (gr+gl, gr-gl, v, psi, k)
  end function f_trlmf
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_vf (gv, v, psi) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: gv
    {\tt type(vector),\ intent(in)\ ::\ v}
    type(spinor), intent(in) :: psi
    complex(kind=default) :: vp, vm, v12, v12s
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v\%x(1) + (0,1)*v\%x(2)
    v12s = v\%x(1) - (0,1)*v\%x(2)
    vpsi%a(1) = gv * (    vm * psi%a(3) - v12s * psi%a(4))
    vpsi\%a(2) = gv * ( - v12 * psi\%a(3) + vp * psi\%a(4))
    vpsi%a(3) = gv * (     vp * psi%a(1) + v12s * psi%a(2))
    vpsi\%a(4) = gv * ( v12 * psi\%a(1) + vm * psi\%a(2))
  end function f_vf
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_af (ga, v, psi) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: ga
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    complex(kind=default) :: vp, vm, v12, v12s
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v%x(1) + (0,1)*v%x(2)
    v12s = v%x(1) - (0,1)*v%x(2)
    vpsi\%a(1) = ga * ( - vm * psi\%a(3) + v12s * psi\%a(4))
                         v12 * psi%a(3) - vp * psi%a(4))
    vpsi\%a(2) = ga * (
    vpsi\%a(3) = ga * (
                          vp * psi%a(1) + v12s * psi%a(2))
    vpsi\%a(4) = ga * ( v12 * psi\%a(1) + vm * psi\%a(2))
  end function f_af
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_vlf (gl, v, psi) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: gl
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    complex(kind=default) :: gl2
    complex(kind=default) :: vp, vm, v12, v12s
    g12 = 2 * g1
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v%x(1) + (0,1)*v%x(2)
    v12s = v\%x(1) - (0,1)*v\%x(2)
    vpsi\%a(1) = 0
    vpsi%a(2) = 0
    vpsi\%a(3) = gl2 * (
                           vp * psi%a(1) + v12s * psi%a(2))
    vpsi\%a(4) = gl2 * ( v12 * psi\%a(1) + vm * psi\%a(2))
  end function f_vlf
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
  pure function f_vrf (gr, v, psi) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: gr
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    complex(kind=default) :: gr2
    complex(kind=default) :: vp, vm, v12, v12s
```

```
gr2 = 2 * gr
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v\%x(1) + (0,1)*v\%x(2)
    v12s = v\%x(1) - (0,1)*v\%x(2)
    vpsi\%a(1) = gr2 * ( vm * psi\%a(3) - v12s * psi\%a(4))
    vpsi\%a(2) = gr2 * ( - v12 * psi\%a(3) + vp * psi\%a(4))
    vpsi\%a(3) = 0
    vpsi\%a(4) = 0
  end function f_vrf
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_vlrf (gl, gr, v, psi) result (vpsi)
    type(spinor) :: vpsi
    complex(kind=default), intent(in) :: gl, gr
    type(vector), intent(in) :: v
    type(spinor), intent(in) :: psi
    vpsi = f_vaf (gl+gr, gl-gr, v, psi)
  end function f_vlrf
\langle Declaration \ of \ spinor \ currents \rangle + \equiv
 public :: f_fva, f_fv, f_fa, f_fvl, f_fvr, f_fvlr, f_fva2, &
            f_ftva, f_ftlr, f_ftrl, f_ftvam, f_ftlrm, f_ftrlm, f_fva3
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
  pure function f_fva (gv, ga, psibar, v) result (psibarv)
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in) :: gv, ga
    type(conjspinor), intent(in) :: psibar
    type(vector), intent(in) :: v
    complex(kind=default) :: gl, gr
    complex(kind=default) :: vp, vm, v12, v12s
    gl = gv + ga
    gr = gv - ga
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v%x(1) + (0,1)*v%x(2)
    v12s = v\%x(1) - (0,1)*v\%x(2)
    psibarv%a(1) = gl * ( psibar%a(3) * vp + psibar%a(4) * v12)
    psibarv\%a(2) = gl * ( psibar\%a(3) * v12s + psibar\%a(4) * vm )
    psibarv%a(3) = gr * ( psibar%a(1) * vm - psibar%a(2) * v12)
    psibarv\%a(4) = gr * ( - psibar\%a(1) * v12s + psibar\%a(2) * vp )
  end function f_fva
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_fva2 (gva, psibar, v) result (psibarv)
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in), dimension(2) :: gva
    type(conjspinor), intent(in) :: psibar
    type(vector), intent(in) :: v
    complex(kind=default) :: gl, gr
    complex(kind=default) :: vp, vm, v12, v12s
    gl = gva(1) + gva(2)
    gr = gva(1) - gva(2)
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v\%x(1) + (0,1)*v\%x(2)
    v12s = v\%x(1) - (0,1)*v\%x(2)
    psibarv%a(1) = gl * ( psibar%a(3) * vp + psibar%a(4) * v12)
    psibarv%a(2) = gl * (
                             psibar%a(3) * v12s + psibar%a(4) * vm)
    psibarv%a(3) = gr * ( psibar%a(1) * vm - psibar%a(2) * v12)
    psibarv%a(4) = gr * ( - psibar%a(1) * v12s + psibar%a(2) * vp )
  end function f_fva2
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_fva3 (gv, ga, psibar, v) result (psibarv)
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in) :: gv, ga
    type(conjspinor), intent(in) :: psibar
```

```
type(vector), intent(in) :: v
   complex(kind=default) :: gl, gr
   complex(kind=default) :: vp, vm, v12, v12s
   gl = gv + ga
   gr = gv - ga
   vp = v\%x(3) !+ v\%t
   vm = - v\%x(3) !+ v\%t
   v12 = v\%x(1) + (0,1)*v\%x(2)
   v12s = v\%x(1) - (0,1)*v\%x(2)
   psibarv%a(1) = gl * ( psibar%a(3) * vp + psibar%a(4) * v12)
                            psibar%a(3) * v12s + psibar%a(4) * vm )
   psibarv%a(2) = gl * (
   psibarv%a(3) = gr * ( psibar%a(1) * vm - psibar%a(2) * v12)
   psibarv%a(4) = gr * ( - psibar%a(1) * v12s + psibar%a(2) * vp )
 end function f_fva3
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_ftva (gv, ga, psibar, t) result (psibart)
   type(conjspinor) :: psibart
   complex(kind=default), intent(in) :: gv, ga
   type(conjspinor), intent(in) :: psibar
   type(tensor2odd), intent(in) :: t
   complex(kind=default) :: gl, gr
   complex(kind=default) :: e21, e21s, b12, b12s, be3, be3s
       = gv + ga
   gl
       = gv - ga
   e21 = t\%e(2) + t\%e(1)*(0,1)
   e21s = t\%e(2) - t\%e(1)*(0,1)
   b12 = t\%b(1) + t\%b(2)*(0,1)
   b12s = t\%b(1) - t\%b(2)*(0,1)
   be3 = t\%b(3) + t\%e(3)*(0,1)
   be3s = t\%b(3) - t\%e(3)*(0,1)
   psibart%a(1) = 2*gl * ( psibar%a(1) * be3 + psibar%a(2) * (-e21s+b12 ))
   psibart%a(4) = 2*gr * ( - psibar%a(4) * be3s + psibar%a(3) * (-e21 +b12s))
 end function f_ftva
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_{ftlr} (gl, gr, psibar, t) result (psibart)
   type(conjspinor) :: psibart
   complex(kind=default), intent(in) :: gl, gr
   type(conjspinor), intent(in) :: psibar
   type(tensor2odd), intent(in) :: t
   psibart = f_ftva (gr+gl, gr-gl, psibar, t)
 end function f_ftlr
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_ftrl (gr, gl, psibar, t) result (psibart)
   type(conjspinor) :: psibart
   complex(kind=default), intent(in) :: gl, gr
   type(conjspinor), intent(in) :: psibar
   type(tensor2odd), intent(in) :: t
   psibart = f_ftva (gr+gl, gr-gl, psibar, t)
 end function f_ftrl
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_ftvam (gv, ga, psibar, v, k) result (psibarv)
   type(conjspinor) :: psibarv
   complex(kind=default), intent(in) :: gv, ga
   type(conjspinor), intent(in) :: psibar
   type(vector), intent(in) :: v
   type(momentum), intent(in) :: k
   type(tensor2odd) :: t
   t = (v.wedge.k) * (0, 0.5)
   psibarv = f_ftva(gv, ga, psibar, t)
 end function f_ftvam
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_ftlrm (gl, gr, psibar, v, k) result (psibarv)
```

```
type(conjspinor) :: psibarv
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: k
    psibarv = f_ftvam (gr+gl, gr-gl, psibar, v, k)
  end function f_ftlrm
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_ftrlm (gr, gl, psibar, v, k) result (psibarv)
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: k
    psibarv = f_ftvam (gr+gl, gr-gl, psibar, v, k)
  end function f_ftrlm
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_fv (gv, psibar, v) result (psibarv)
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in) :: gv
    {\tt type(conjspinor),\ intent(in)\ ::\ psibar}
    type(vector), intent(in) :: v
    complex(kind=default) :: vp, vm, v12, v12s
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v\%x(1) + (0,1)*v\%x(2)
    v12s = v\%x(1) - (0,1)*v\%x(2)
    psibarv%a(1) = gv * ( psibar%a(3) * vp + psibar%a(4) * v12)
    psibarv%a(2) = gv * ( psibar%a(3) * v12s + psibar%a(4) * vm )
    psibarv%a(3) = gv * ( psibar%a(1) * vm - psibar%a(2) * v12)
    psibarv%a(4) = gv * ( - psibar%a(1) * v12s + psibar%a(2) * vp )
  end function f_fv
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_fa (ga, psibar, v) result (psibarv)
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in) :: ga
    type(vector), intent(in) :: v
    type(conjspinor), intent(in) :: psibar
    complex(kind=default) :: vp, vm, v12, v12s
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v\%x(1) + (0,1)*v\%x(2)
    v12s = v\%x(1) - (0,1)*v\%x(2)
    psibarv%a(1) = ga * ( psibar%a(3) * vp + psibar%a(4) * v12)
    psibarv%a(2) = ga * ( psibar%a(3) * v12s + psibar%a(4) * vm )
    psibarv%a(3) = ga * ( - psibar%a(1) * vm + psibar%a(2) * v12)
    psibarv%a(4) = ga * ( psibar%a(1) * v12s - psibar%a(2) * vp )
  end function f_fa
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
  pure function f_fvl (gl, psibar, v) result (psibarv)
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in) :: gl
    type(conjspinor), intent(in) :: psibar
    type(vector), intent(in) :: v
    complex(kind=default) :: gl2
    complex(kind=default) :: vp, vm, v12, v12s
    g12 = 2 * g1
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v\%x(1) + (0,1)*v\%x(2)
    v12s = v\%x(1) - (0,1)*v\%x(2)
    psibarv%a(1) = gl2 * ( psibar%a(3) * vp + psibar%a(4) * v12)
    psibarv%a(2) = gl2 * ( psibar%a(3) * v12s + psibar%a(4) * vm )
    psibarv%a(3) = 0
```

```
psibarv%a(4) = 0
  end function f_fvl
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_fvr (gr, psibar, v) result (psibarv)
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in) :: gr
    {\tt type(conjspinor),\ intent(in)\ ::\ psibar}
    type(vector), intent(in) :: v
    complex(kind=default) :: gr2
    complex(kind=default) :: vp, vm, v12, v12s
    gr2 = 2 * gr
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v%x(1) + (0,1)*v%x(2)
    v12s = v%x(1) - (0,1)*v%x(2)
    psibarv%a(1) = 0
    psibarv%a(2) = 0
    psibarv%a(3) = gr2 * (    psibar%a(1) * vm - psibar%a(2) * v12)
    psibarv%a(4) = gr2 * ( - psibar%a(1) * v12s + psibar%a(2) * vp )
  end function f_fvr
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_fvlr (gl, gr, psibar, v) result (psibarv)
    type(conjspinor) :: psibarv
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(vector), intent(in) :: v
    psibarv = f_fva (gl+gr, gl-gr, psibar, v)
  end function f_fvlr
                       Y.25.2 Fermionic Scalar and Pseudo Scalar Couplings
\langle Declaration \ of \ spinor \ currents \rangle + \equiv
 public :: sp_ff, s_ff, p_ff, sl_ff, sr_ff, slr_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function sp_ff (gs, gp, psibar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gs, gp
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
            (gs - gp) * (psibar%a(1)*psi%a(1) + psibar%a(2)*psi%a(2)) &
         + (gs + gp) * (psibar%a(3)*psi%a(3) + psibar%a(4)*psi%a(4))
  end function sp_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function s_ff (gs, psibar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gs
    type(conjspinor), intent(in) :: psibar
    {\tt type(spinor),\ intent(in)\ ::\ psi}
    j = gs * (psibar * psi)
  end function s_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function p_ff (gp, psibar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gp
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    j = gp * ( psibar%a(3)*psi%a(3) + psibar%a(4)*psi%a(4) &
               - psibar%a(1)*psi%a(1) - psibar%a(2)*psi%a(2))
  end function p_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function sl_ff (gl, psibar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl
```

```
type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    j = 2 * gl * (psibar%a(1)*psi%a(1) + psibar%a(2)*psi%a(2))
  end function sl_ff
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function sr_ff (gr, psibar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    j = 2 * gr * (psibar%a(3)*psi%a(3) + psibar%a(4)*psi%a(4))
  end function sr_ff
                        g_L(1-\gamma_5) + g_R(1+\gamma_5) = (g_R+g_L) + (g_R-g_L)\gamma_5 = g_S + g_P\gamma_5
                                                                                                           (Y.100)
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function slr_ff (gl, gr, psibar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    type(spinor), intent(in) :: psi
    j = sp_ff (gr+gl, gr-gl, psibar, psi)
  end function slr_ff
\langle Declaration \ of \ spinor \ currents \rangle + \equiv
 public :: f_spf, f_sf, f_pf, f_slf, f_srf, f_slrf
\langle Implementation\ of\ spinor\ currents \rangle + \equiv
 pure function f\_spf (gs, gp, phi, psi) result (phipsi)
    type(spinor) :: phipsi
    complex(kind=default), intent(in) :: gs, gp
    complex(kind=default), intent(in) :: phi
    type(spinor), intent(in) :: psi
    phipsi\%a(1:2) = ((gs - gp) * phi) * psi\%a(1:2)
    phipsi\%a(3:4) = ((gs + gp) * phi) * psi\%a(3:4)
  end function f_spf
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_sf (gs, phi, psi) result (phipsi)
    type(spinor) :: phipsi
    complex(kind=default), intent(in) :: gs
    complex(kind=default), intent(in) :: phi
    type(spinor), intent(in) :: psi
    phipsi%a = (gs * phi) * psi%a
  end function f_sf
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_pf (gp, phi, psi) result (phipsi)
    type(spinor) :: phipsi
    complex(kind=default), intent(in) :: gp
    complex(kind=default), intent(in) :: phi
    type(spinor), intent(in) :: psi
    phipsi%a(1:2) = (- gp * phi) * psi%a(1:2)
    phipsi%a(3:4) = (gp * phi) * psi%a(3:4)
  end function f_pf
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_slf (gl, phi, psi) result (phipsi)
    type(spinor) :: phipsi
    complex(kind=default), intent(in) :: gl
    complex(kind=default), intent(in) :: phi
    type(spinor), intent(in) :: psi
    phipsi\%a(1:2) = (2 * gl * phi) * psi\%a(1:2)
    phipsi\%a(3:4) = 0
  end function f_slf
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_srf (gr, phi, psi) result (phipsi)
    type(spinor) :: phipsi
```

```
complex(kind=default), intent(in) :: gr
    complex(kind=default), intent(in) :: phi
    type(spinor), intent(in) :: psi
    phipsi\%a(1:2) = 0
    phipsi\%a(3:4) = (2 * gr * phi) * psi\%a(3:4)
  end function f_srf
\langle \mathit{Implementation of spinor currents} \rangle + \equiv
 pure function f_slrf (gl, gr, phi, psi) result (phipsi)
    type(spinor) :: phipsi
    {\tt complex(kind=default),\ intent(in)\ ::\ gl,\ gr}
    complex(kind=default), intent(in) :: phi
    type(spinor), intent(in) :: psi
    phipsi = f_spf (gr+gl, gr-gl, phi, psi)
  end function f_slrf
\langle Declaration \ of \ spinor \ currents \rangle + \equiv
 public :: f_fsp, f_fs, f_fp, f_fsl, f_fsr, f_fslr
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_fsp (gs, gp, psibar, phi) result (psibarphi)
    type(conjspinor) :: psibarphi
    complex(kind=default), intent(in) :: gs, gp
    type(conjspinor), intent(in) :: psibar
    complex(kind=default), intent(in) :: phi
    psibarphi\%a(1:2) = ((gs - gp) * phi) * psibar\%a(1:2)
    psibarphi%a(3:4) = ((gs + gp) * phi) * psibar%a(3:4)
  end function f_fsp
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_fs (gs, psibar, phi) result (psibarphi)
    type(conjspinor) :: psibarphi
    complex(kind=default), intent(in) :: gs
    type(conjspinor), intent(in) :: psibar
    complex(kind=default), intent(in) :: phi
    psibarphi%a = (gs * phi) * psibar%a
  end function f_fs
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_fp (gp, psibar, phi) result (psibarphi)
    type(conjspinor) :: psibarphi
    complex(kind=default), intent(in) :: gp
    type(conjspinor), intent(in) :: psibar
    complex(kind=default), intent(in) :: phi
    psibarphi\%a(1:2) = (-gp * phi) * psibar\%a(1:2)
    psibarphi%a(3:4) = ( gp * phi) * psibar%a(3:4)
  end function f_fp
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_fsl (gl, psibar, phi) result (psibarphi)
    type(conjspinor) :: psibarphi
    complex(kind=default), intent(in) :: gl
    type(conjspinor), intent(in) :: psibar
    complex(kind=default), intent(in) :: phi
    psibarphi%a(1:2) = (2 * gl * phi) * psibar%a(1:2)
    psibarphi\%a(3:4) = 0
  end function f_fsl
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
 pure function f_fsr (gr, psibar, phi) result (psibarphi)
    type(conjspinor) :: psibarphi
    complex(kind=default), intent(in) :: gr
    type(conjspinor), intent(in) :: psibar
    complex(kind=default), intent(in) :: phi
    psibarphi\%a(1:2) = 0
    psibarphi\%a(3:4) = (2 * gr * phi) * psibar\%a(3:4)
  end function f_fsr
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
  pure function f_fslr (gl, gr, psibar, phi) result (psibarphi)
```

```
type(conjspinor) :: psibarphi
    complex(kind=default), intent(in) :: gl, gr
    type(conjspinor), intent(in) :: psibar
    complex(kind=default), intent(in) :: phi
    psibarphi = f_fsp (gr+gl, gr-gl, psibar, phi)
  end function f_fslr
\langle Declaration \ of \ spinor \ currents \rangle + \equiv
  public :: f_gravf, f_fgrav
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
  pure function f_gravf (g, m, kb, k, t, psi) result (tpsi)
    type(spinor) :: tpsi
    complex(kind=default), intent(in) :: g
    real(kind=default), intent(in) :: m
    type(spinor), intent(in) :: psi
    type(tensor), intent(in) :: t
    type(momentum), intent(in) :: kb, k
    complex(kind=default) :: g2, g8, t_tr
    type(vector) :: kkb
    kkb = k + kb
    g2 = g / 2.0_default
    g8 = g / 8.0_default
    t_t = t\%t(0,0) - t\%t(1,1) - t\%t(2,2) - t\%t(3,3)
    tpsi = (- f_sf (g2, cmplx (m,0.0, kind=default), psi) &
              - f_vf ((g8*m), kkb, psi)) * t_tr - &
    f_vf (g8,(t*kkb + kkb*t),psi)
  end function f_gravf
\langle Implementation \ of \ spinor \ currents \rangle + \equiv
  pure function f_fgrav (g, m, kb, k, psibar, t) result (psibart)
    type(conjspinor) :: psibart
    complex(kind=default), intent(in) :: g
    real(kind=default), intent(in) :: m
    type(conjspinor), intent(in) :: psibar
    type(tensor), intent(in) :: t
    type(momentum), intent(in) :: kb, k
    type(vector) :: kkb
    complex(kind=default) :: g2, g8, t_tr
    kkb = k + kb
    g2 = g / 2.0_default
    g8 = g / 8.0_default
    t_{tr} = t\%t(0,0) - t\%t(1,1) - t\%t(2,2) - t\%t(3,3)
    psibart = (- f_fs (g2, psibar, cmplx (m, 0.0, kind=default)) &
         - f_fv ((g8 * m), psibar, kkb)) * t_tr - &
           f_fv (g8,psibar,(t*kkb + kkb*t))
  end function f_fgrav
                                       Y.25.3 On Shell Wave Functions
\langle Declaration \ of \ spinor \ on \ shell \ wave \ functions \rangle \equiv
  public :: u, ubar, v, vbar
  private :: chi_plus, chi_minus
                                       \chi_{+}(\vec{p}) = \frac{1}{\sqrt{2|\vec{p}|(|\vec{p}| + p_3)}} \begin{pmatrix} |\vec{p}| + p_3 \\ p_1 + \mathrm{i} p_2 \end{pmatrix}
                                                                                                                 (Y.101a)
                                       \chi_{-}(\vec{p}) = \frac{1}{\sqrt{2|\vec{p}|(|\vec{p}| + p_3)}} \begin{pmatrix} -p_1 + ip_2 \\ |\vec{p}| + p_3 \end{pmatrix}
                                                                                                                 (Y.101b)
\langle Implementation \ of \ spinor \ on \ shell \ wave \ functions \rangle \equiv
  pure function chi_plus (p) result (chi)
    complex(kind=default), dimension(2) :: chi
    type(momentum), intent(in) :: p
    real(kind=default) :: pabs
    pabs = sqrt (dot_product (p%x, p%x))
```

```
if (pabs + p%x(3) \le 1000 * epsilon (pabs) * pabs) then
         chi = (/ cmplx ( 0.0, 0.0, kind=default), &
                     cmplx (1.0, 0.0, kind=default) /)
     else
         chi = 1 / sqrt (2*pabs*(pabs + p%x(3))) &
               * (/ cmplx (pabs + p%x(3), kind=default), &
                      cmplx (p%x(1), p%x(2), kind=default) /)
     end if
  end function chi_plus
\langle Implementation \ of \ spinor \ on \ shell \ wave \ functions \rangle + \equiv
  pure function chi_minus (p) result (chi)
     complex(kind=default), dimension(2) :: chi
     type(momentum), intent(in) :: p
     real(kind=default) :: pabs
     pabs = sqrt (dot_product (p%x, p%x))
     if (pabs + p%x(3) \le 1000 * epsilon (pabs) * pabs) then
         chi = (/ cmplx (-1.0, 0.0, kind=default), &
                     cmplx ( 0.0, 0.0, kind=default) /)
     else
         chi = 1 / sqrt (2*pabs*(pabs + p%x(3))) &
                * (/ cmplx (-p%x(1), p%x(2), kind=default), &
                      cmplx (pabs + p%x(3), kind=default) /)
     end if
  end function chi_minus
                    u_{\pm}(p,|m|) = \begin{pmatrix} \sqrt{p_0 \mp |\vec{p}|} \cdot \chi_{\pm}(\vec{p}) \\ \sqrt{p_0 \pm |\vec{p}|} \cdot \chi_{+}(\vec{p}) \end{pmatrix} \qquad u_{\pm}(p,-|m|) = \begin{pmatrix} -i\sqrt{p_0 \mp |\vec{p}|} \cdot \chi_{\pm}(\vec{p}) \\ +i\sqrt{p_0 \pm |\vec{p}|} \cdot \chi_{+}(\vec{p}) \end{pmatrix}
                                                                                                                               (Y.102)
```

Determining the mass from the momenta is a numerically haphazardous for light particles. Therefore, we accept some redundancy and pass the mass explicitely. Even if the mass is not used in the chiral representation, we do so for symmetry with polarization vectors and to be prepared for other representations.

```
\langle Implementation \ of \ spinor \ on \ shell \ wave \ functions \rangle + \equiv
 pure function u (mass, p, s) result (psi)
    type(spinor) :: psi
    real(kind=default), intent(in) :: mass
    type(momentum), intent(in) :: p
    integer, intent(in) :: s
    complex(kind=default), dimension(2) :: chi
    real(kind=default) :: pabs, delta, m
    m = abs(mass)
    pabs = sqrt (dot_product (p%x, p%x))
    if (m < epsilon (m) * pabs) then
        delta = 0
    else
        delta = sqrt (max (p%t - pabs, 0._default))
    end if
    select case (s)
    case (1)
       chi = chi_plus (p)
       psi\%a(1:2) = delta * chi
       psi\%a(3:4) = sqrt (p\%t + pabs) * chi
    case (-1)
       chi = chi_minus (p)
       psi\%a(1:2) = sqrt (p\%t + pabs) * chi
       psi\%a(3:4) = delta * chi
    case default
       pabs = m ! make the compiler happy and use m
       psi%a = 0
    end select
    if (mass < 0) then
       psi\%a(1:2) = - imago * psi\%a(1:2)
       psi\%a(3:4) = + imago * psi\%a(3:4)
  end function u
```

 $\langle Implementation \ of \ spinor \ on \ shell \ wave \ functions \rangle + \equiv$

```
pure function ubar (m, p, s) result (psibar)
    type(conjspinor) :: psibar
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: p
    integer, intent(in) :: s
    type(spinor) :: psi
    psi = u (m, p, s)
    psibar%a(1:2) = conjg (psi%a(3:4))
    psibar%a(3:4) = conjg (psi%a(1:2))
  end function ubar
                                          v_{\pm}(p) = \begin{pmatrix} \mp \sqrt{p_0 \pm |\vec{p}|} \cdot \chi_{\mp}(\vec{p}) \\ \pm \sqrt{p_0 \mp |\vec{p}|} \cdot \chi_{\mp}(\vec{p}) \end{pmatrix}
                                                                                                              (Y.103)
\langle Implementation \ of \ spinor \ on \ shell \ wave \ functions \rangle + \equiv
  pure function v (mass, p, s) result (psi)
    type(spinor) :: psi
    real(kind=default), intent(in) :: mass
    type(momentum), intent(in) :: p
    integer, intent(in) :: s
    complex(kind=default), dimension(2) :: chi
    real(kind=default) :: pabs, delta, m
    m = abs(mass)
    pabs = sqrt (dot_product (p%x, p%x))
    if (m < epsilon (m) * pabs) then
         delta = 0
    else
         delta = sqrt (max (p%t - pabs, 0._default))
    end if
    select case (s)
    case (1)
       chi = chi_minus (p)
       psi\%a(1:2) = - sqrt (p\%t + pabs) * chi
       psi\%a(3:4) = delta * chi
    case (-1)
       chi = chi_plus (p)
       psi\%a(1:2) = delta * chi
       psi\%a(3:4) = - sqrt (p\%t + pabs) * chi
    case default
       pabs = m ! make the compiler happy and use m
       psi\%a = 0
    end select
    if (mass < 0) then
       psi\%a(1:2) = - imago * psi\%a(1:2)
       psi\%a(3:4) = + imago * psi\%a(3:4)
     end if
  end function v
\langle Implementation \ of \ spinor \ on \ shell \ wave \ functions \rangle + \equiv
  pure function vbar (m, p, s) result (psibar)
    type(conjspinor) :: psibar
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: p
    integer, intent(in) :: s
    type(spinor) :: psi
    psi = v (m, p, s)
    psibar%a(1:2) = conjg (psi%a(3:4))
    psibar%a(3:4) = conjg (psi%a(1:2))
  end function vbar
```

Y.25.4 Off Shell Wave Functions

I've just taken this over from Christian Schwinn's version.

```
\langle Declaration \ of \ spinor \ off \ shell \ wave \ functions \rangle \equiv public :: brs_u, brs_ubar, brs_v, brs_vbar
```

The off-shell wave functions needed for gauge checking are obtained from the LSZ-formulas:

$$\langle \text{Out} | d^{\dagger} | \text{In} \rangle = i \int d^4 x \bar{v} e^{-ikx} (i\partial \!\!/ - m) \langle \text{Out} | \psi | \text{In} \rangle$$
 (Y.104a)

$$\langle \text{Out}|b|\text{In}\rangle = -i\int d^4x \bar{u}e^{ikx}(i\partial \!\!\!/ - m) \langle \text{Out}|\psi|\text{In}\rangle$$
 (Y.104b)

$$\langle \text{Out} | d | \text{In} \rangle = i \int d^4x \left\langle \text{Out} | \bar{\psi} | \text{In} \right\rangle (-i \overleftarrow{\partial} - m) v e^{ikx}$$
 (Y.104c)

$$\langle \text{Out} | b^{\dagger} | \text{In} \rangle = -i \int d^4x \langle \text{Out} | \bar{\psi} | \text{In} \rangle (-i \overleftarrow{\phi} - m) u e^{-ikx}$$
 (Y.104d)

Since the relative sign between fermions and antifermions is ignored for on-shell amplitudes we must also ignore it here, so all wavefunctions must have a (-i) factor. In momentum space we have:

$$brsu(p) = (-i)(\not p - m)u(p) \tag{Y.105}$$

```
(Y.105)
\langle Implementation \ of \ spinor \ off \ shell \ wave \ functions \rangle \equiv
  pure function brs_u (m, p, s) result (dpsi)
      type(spinor) :: dpsi,psi
       real(kind=default), intent(in) :: m
       type(momentum), intent(in) :: p
       integer, intent(in) :: s
       type (vector)::vp
       complex(kind=default), parameter :: one = (1, 0)
       vp=p
      psi=u(m,p,s)
       dpsi=cmplx(0.0,-1.0)*(f_vf(one,vp,psi)-m*psi)
  end function brs_u
                                              brsv(p) = i(\not p + m)v(p)
                                                                                                              (Y.106)
\langle Implementation \ of \ spinor \ off \ shell \ wave \ functions \rangle + \equiv
  pure function brs_v (m, p, s) result (dpsi)
      type(spinor) :: dpsi, psi
      real(kind=default), intent(in) :: m
       type(momentum), intent(in) :: p
       integer, intent(in) ::
       type (vector)::vp
       complex(kind=default), parameter :: one = (1, 0)
       vp=p
       psi=v(m,p,s)
       dpsi=cmplx(0.0,1.0)*(f_vf(one,vp,psi)+m*psi)
  end function brs_v
                                            brs\bar{u}(p) = (-i)\bar{u}(p)(\not p - m)
```

$$brs\bar{u}(p) = (-i)\bar{u}(p)(\not p - m) \tag{Y.107}$$

 $\langle Implementation \ of \ spinor \ off \ shell \ wave \ functions \rangle + \equiv$ pure function brs_ubar (m, p, s)result (dpsibar) type(conjspinor) :: dpsibar, psibar real(kind=default), intent(in) :: m type(momentum), intent(in) :: p integer, intent(in) :: s type (vector)::vp complex(kind=default), parameter :: one = (1, 0) psibar=ubar(m,p,s) dpsibar=cmplx(0.0,-1.0)*(f_fv(one,psibar,vp)-m*psibar) end function brs_ubar

$$brs\bar{v}(p) = (i)\bar{v}(p)(\not p + m) \tag{Y.108}$$

 $\langle Implementation \ of \ spinor \ off \ shell \ wave \ functions \rangle + \equiv$ pure function brs_vbar (m, p, s) result (dpsibar) type(conjspinor) :: dpsibar,psibar real(kind=default), intent(in) :: m type(momentum), intent(in) :: p

```
integer, intent(in) :: s
  type(vector)::vp
  complex(kind=default), parameter :: one = (1, 0)
  vp=p
  psibar=vbar(m,p,s)
  dpsibar=cmplx(0.0,1.0)*(f_fv(one,psibar,vp)+m*psibar)
end function brs_vbar
```

NB: The remarks on momentum flow in the propagators don't apply here since the incoming momenta are flipped for the wave functions.

Y.25.5 Propagators

NB: the common factor of i is extracted:

```
⟨Declaration of spinor propagators⟩≡
public :: pr_psi, pr_psibar
public :: pj_psi, pj_psibar
public :: pg_psi, pg_psibar
```

$$\frac{i(-\not p+m)}{p^2-m^2+\mathrm{i}m\Gamma}\psi\tag{Y.109}$$

NB: the sign of the momentum comes about because all momenta are treated as *outgoing* and the particle charge flow is therefore opposite to the momentum.

```
\langle Implementation \ of \ spinor \ propagators \rangle \equiv
 pure function pr_psi (p, m, w, cms, psi) result (ppsi)
    type(spinor) :: ppsi
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(spinor), intent(in) :: psi
    logical, intent(in) :: cms
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
    complex(kind=default) :: num_mass
    q = qv
    if (cms) then
       num_mass = sqrt(cmplx(m**2, -m*w, kind=default))
    else
       num_mass = cmplx (m, 0, kind=default)
    end if
    ppsi = (1 / cmplx (p*p - m**2, m*w, kind=default)) &
         * (- f_vf (one, vp, psi) + num_mass * psi)
  end function pr_psi
```

$$\sqrt{\frac{\pi}{M\Gamma}}(-\not p + m)\psi \tag{Y.110}$$

```
\langle Implementation \ of \ spinor \ propagators \rangle + \equiv
 pure function pj_psi (p, m, w, psi) result (ppsi)
    type(spinor) :: ppsi
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(spinor), intent(in) :: psi
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
    ppsi = (0, -1) * sqrt (PI / m / w) * (- f_vf (one, vp, psi) + m * psi)
  end function pj_psi
\langle Implementation \ of \ spinor \ propagators \rangle + \equiv
 pure function pg_psi (p, m, w, psi) result (ppsi)
    type(spinor) :: ppsi
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(spinor), intent(in) :: psi
    type(vector) :: vp
```

complex(kind=default), parameter :: one = (1, 0)

(Y.113)

NB: the sign of the momentum comes about because all momenta are treated as *outgoing* and the antiparticle charge flow is therefore parallel to the momentum.

(Implementation of spinor propagators)+=
pure function pr_psibar (p, m, w, cms, psibar) result (ppsibar)
 type(conjspinor) :: ppsibar
 type(momentum), intent(in) :: p
 real(kind=default), intent(in) :: m, w
 type(conjspinor), intent(in) :: psibar
 logical, intent(in) :: cms
 type(vector) :: vp
 complex(kind=default), parameter :: one = (1, 0)
 complex(kind=default) :: num_mass
 vp = p
 if (cms) then
 num_mass = sqrt(cmplx(m**2, -m*w, kind=default))
 else
 num_mass = cmplx (m, 0, kind=default)
 end if

ppsibar = (1 / cmplx (p*p - m**2, m*w, kind=default)) &
 * (f_fv (one, psibar, vp) + num_mass * psibar)

end function pr_psibar

$$\sqrt{\frac{\pi}{M\Gamma}}\bar{\psi}(\not p+m) \tag{Y.112}$$

NB: the sign of the momentum comes about because all momenta are treated as *outgoing* and the antiparticle charge flow is therefore parallel to the momentum.

```
\langle Implementation \ of \ spinor \ propagators \rangle + \equiv
  pure function pj_psibar (p, m, w, psibar) result (ppsibar)
    type(conjspinor) :: ppsibar
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(conjspinor), intent(in) :: psibar
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
    ppsibar = (0, -1) * sqrt (PI / m / w) * (f_fv (one, psibar, vp) + m * psibar)
  end function pj_psibar
\langle Implementation \ of \ spinor \ propagators \rangle + \equiv
  pure function pg_psibar (p, m, w, psibar) result (ppsibar)
    type(conjspinor) :: ppsibar
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(conjspinor), intent(in) :: psibar
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
    ppsibar = gauss (p*p, m, w) * (f_fv (one, psibar, vp) + m * psibar)
  end function pg_psibar
                                           \frac{i(-\not\!p+m)}{p^2-m^2+\mathrm{i} m\Gamma}\sum_n\psi_n\otimes\bar\psi_n
```

NB: the temporary variables psi(1:4) are not nice, but the compilers should be able to optimize the unnecessary copies away. In any case, even if the copies are performed, they are (probably) negligible compared to the floating point multiplications anyway . . .

```
⟨(Not used yet) Declaration of operations for spinors⟩≡
type, public :: spinordyad
```

```
! private (omegalib needs access, but DON'T TOUCH IT!)
     complex(kind=default), dimension(4,4) :: a
  end type spinordyad
\langle (Not \ used \ yet) \ Implementation \ of \ spinor \ propagators \rangle \equiv
  pure function pr_dyadleft (p, m, w, psipsibar) result (psipsibarp)
    type(spinordyad) :: psipsibarp
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(spinordyad), intent(in) :: psipsibar
    integer :: i
    type(vector) :: vp
    type(spinor), dimension(4) :: psi
    complex(kind=default) :: pole
    complex(kind=default), parameter :: one = (1, 0)
    pole = 1 / cmplx (p*p - m**2, m*w, kind=default)
    do i = 1, 4
       psi(i)%a = psipsibar%a(:,i)
       psi(i) = pole * (-f_vf (one, vp, psi(i)) + m * psi(i))
       psipsibarp%a(:,i) = psi(i)%a
    end do
  end function pr_dyadleft
                                          \sum_{n} \psi_{n} \otimes \bar{\psi}_{n} \frac{i(\not p + m)}{p^{2} - m^{2} + \mathrm{i} m \Gamma}
                                                                                                           (Y.114)
\langle (Not \ used \ yet) \ Implementation \ of \ spinor \ propagators \rangle + \equiv
  pure function pr_dyadright (p, m, w, psipsibar) result (psipsibarp)
    type(spinordyad) :: psipsibarp
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(spinordyad), intent(in) :: psipsibar
    integer :: i
    type(vector) :: vp
    type(conjspinor), dimension(4) :: psibar
    complex(kind=default) :: pole
    complex(kind=default), parameter :: one = (1, 0)
    vp = p
    pole = 1 / cmplx (p*p - m**2, m*w, kind=default)
    do i = 1, 4
       psibar(i)%a = psipsibar%a(i,:)
       psibar(i) = pole * (f_fv (one, psibar(i), vp) + m * psibar(i))
       psipsibarp%a(i,:) = psibar(i)%a
    end do
  end function pr_dyadright
                                 Y.26 Spinor Couplings Revisited
\langle omega\_bispinor\_couplings.f90 \rangle \equiv
  \langle Copyleft \rangle
  module omega_bispinor_couplings
    use kinds
    use constants
    use omega_bispinors
    use omega_vectorspinors
    use omega_vectors
    use omega_couplings
    implicit none
    private
    (Declaration of bispinor on shell wave functions)
    ⟨Declaration of bispinor off shell wave functions⟩
    ⟨Declaration of bispinor currents⟩
    ⟨Declaration of bispinor propagators⟩
    integer, parameter, public :: omega_bispinor_cpls_2010_01_A = 0
```

contains

```
(Implementation of bispinor on shell wave functions)
  (Implementation of bispinor off shell wave functions)
  ⟨Implementation of bispinor currents⟩
  ⟨Implementation of bispinor propagators⟩
end module omega_bispinor_couplings
```

See table Y.1 for the names of Fortran functions. We could have used long names instead, but this would increase the chance of running past continuation line limits without adding much to the legibility.

Y.26.1 Fermionic Vector and Axial Couplings

There's more than one chiral representation. This one is compatible with HELAS [5].

$$\gamma^0 = \begin{pmatrix} 0 & \mathbf{1} \\ \mathbf{1} & 0 \end{pmatrix}, \ \gamma^i = \begin{pmatrix} 0 & \sigma^i \\ -\sigma^i & 0 \end{pmatrix}, \ \gamma_5 = i\gamma^0\gamma^1\gamma^2\gamma^3 = \begin{pmatrix} -\mathbf{1} & 0 \\ 0 & \mathbf{1} \end{pmatrix}, \tag{Y.115a}$$

$$C = \begin{pmatrix} \epsilon & 0 \\ 0 & -\epsilon \end{pmatrix} , \qquad \epsilon = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} . \tag{Y.115b}$$

Therefore

$$g_S + g_P \gamma_5 = \begin{pmatrix} g_S - g_P & 0 & 0 & 0\\ 0 & g_S - g_P & 0 & 0\\ 0 & 0 & g_S + g_P & 0\\ 0 & 0 & 0 & g_S + g_P \end{pmatrix}$$
(Y.116a)

$$g_V \gamma^0 - g_A \gamma^0 \gamma_5 = \begin{pmatrix} 0 & 0 & g_V - g_A & 0\\ 0 & 0 & 0 & g_V - g_A\\ g_V + g_A & 0 & 0 & 0\\ 0 & g_V + g_A & 0 & 0 \end{pmatrix}$$
(Y.116b)

$$g_V \gamma^1 - g_A \gamma^1 \gamma_5 = \begin{pmatrix} 0 & 0 & 0 & g_V - g_A \\ 0 & 0 & g_V - g_A & 0 \\ 0 & -g_V - g_A & 0 & 0 \\ -g_V - g_A & 0 & 0 & 0 \end{pmatrix}$$
(Y.116c)

$$g_V \gamma^2 - g_A \gamma^2 \gamma_5 = \begin{pmatrix} 0 & 0 & 0 & -i(g_V - g_A) \\ 0 & 0 & i(g_V - g_A) & 0 \\ 0 & i(g_V + g_A) & 0 & 0 \\ -i(g_V + g_A) & 0 & 0 & 0 \end{pmatrix}$$
(Y.116d)

$$g_{S} + g_{P}\gamma_{5} = \begin{pmatrix} g_{S} - g_{P} & 0 & 0 & 0 \\ 0 & g_{S} - g_{P} & 0 & 0 \\ 0 & 0 & g_{S} + g_{P} & 0 \\ 0 & 0 & 0 & g_{S} + g_{P} \end{pmatrix}$$

$$g_{V}\gamma^{0} - g_{A}\gamma^{0}\gamma_{5} = \begin{pmatrix} 0 & 0 & g_{V} - g_{A} & 0 \\ 0 & 0 & 0 & g_{V} - g_{A} \\ g_{V} + g_{A} & 0 & 0 & 0 \\ 0 & g_{V} + g_{A} & 0 & 0 \end{pmatrix}$$

$$g_{V}\gamma^{1} - g_{A}\gamma^{1}\gamma_{5} = \begin{pmatrix} 0 & 0 & 0 & g_{V} - g_{A} \\ 0 & 0 & g_{V} - g_{A} & 0 \\ 0 & -g_{V} - g_{A} & 0 & 0 \\ -g_{V} - g_{A} & 0 & 0 & 0 \end{pmatrix}$$

$$g_{V}\gamma^{2} - g_{A}\gamma^{2}\gamma_{5} = \begin{pmatrix} 0 & 0 & 0 & -i(g_{V} - g_{A}) \\ 0 & 0 & i(g_{V} + g_{A}) & 0 & 0 \\ 0 & i(g_{V} + g_{A}) & 0 & 0 \\ -i(g_{V} + g_{A}) & 0 & 0 & 0 \end{pmatrix}$$

$$g_{V}\gamma^{3} - g_{A}\gamma^{3}\gamma_{5} = \begin{pmatrix} 0 & 0 & g_{V} - g_{A} & 0 \\ 0 & 0 & 0 & -g_{V} + g_{A} \\ 0 & 0 & 0 & -g_{V} + g_{A} \end{pmatrix}$$

$$g_{V}\gamma^{3} - g_{A}\gamma^{3}\gamma_{5} = \begin{pmatrix} 0 & 0 & g_{V} - g_{A} & 0 \\ 0 & 0 & 0 & -g_{V} + g_{A} \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$(Y.116e)$$

and

$$C(g_S + g_P \gamma_5) = \begin{pmatrix} 0 & g_S - g_P & 0 & 0\\ -g_S + g_P & 0 & 0 & 0\\ 0 & 0 & 0 & -g_S - g_P\\ 0 & 0 & g_S + g_P & 0 \end{pmatrix}$$
(Y.117a)

$$C(g_V \gamma^0 - g_A \gamma^0 \gamma_5) = \begin{pmatrix} 0 & 0 & 0 & g_V - g_A \\ 0 & 0 & -g_V + g_A & 0 \\ 0 & -g_V - g_A & 0 & 0 \\ g_V + g_A & 0 & 0 & 0 \end{pmatrix}$$
(Y.117b)

$$C(g_V \gamma^1 - g_A \gamma^1 \gamma_5) = \begin{pmatrix} 0 & 0 & g_V - g_A & 0\\ 0 & 0 & 0 & -g_V + g_A\\ g_V + g_A & 0 & 0 & 0\\ 0 & -g_V - g_A & 0 & 0 \end{pmatrix}$$
(Y.117c)

$$C(g_{V}\gamma^{0} - g_{A}\gamma^{0}\gamma_{5}) = \begin{pmatrix} 0 & 0 & 0 & g_{V} - g_{A} \\ 0 & 0 & -g_{V} + g_{A} & 0 \\ 0 & -g_{V} - g_{A} & 0 & 0 \\ g_{V} + g_{A} & 0 & 0 & 0 \end{pmatrix}$$

$$C(g_{V}\gamma^{1} - g_{A}\gamma^{1}\gamma_{5}) = \begin{pmatrix} 0 & 0 & g_{V} - g_{A} & 0 \\ 0 & 0 & g_{V} - g_{A} & 0 \\ 0 & 0 & 0 & -g_{V} + g_{A} \\ g_{V} + g_{A} & 0 & 0 & 0 \\ 0 & -g_{V} - g_{A} & 0 & 0 \end{pmatrix}$$

$$C(g_{V}\gamma^{2} - g_{A}\gamma^{2}\gamma_{5}) = \begin{pmatrix} 0 & 0 & i(g_{V} - g_{A}) & 0 \\ 0 & 0 & 0 & i(g_{V} - g_{A}) & 0 \\ i(g_{V} + g_{A}) & 0 & 0 & 0 \\ 0 & i(g_{V} + g_{A}) & 0 & 0 & 0 \end{pmatrix}$$

$$(Y.117c)$$

```
C(g_V \gamma^3 - g_A \gamma^3 \gamma_5) = \begin{pmatrix} 0 & 0 & 0 & -g_V + g_A \\ 0 & 0 & -g_V + g_A & 0 \\ 0 & -g_V - g_A & 0 & 0 \\ -g_V - g_A & 0 & 0 & 0 \end{pmatrix} (Y.117e)
```

```
\langle Declaration \ of \ bispinor \ currents \rangle \equiv
 public :: va_ff, v_ff, a_ff, vl_ff, vr_ff, vlr_ff, va2_ff, tva_ff, tvam_ff, &
            tlr_ff, tlrm_ff
\langle Implementation \ of \ bispinor \ currents \rangle \equiv
 pure function va_ff (gv, ga, psil, psir) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gv, ga
    type(bispinor), intent(in) :: psil, psir
    complex(kind=default) :: gl, gr
    complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
    gl = gv + ga
    gr = gv - ga
    g13 = psil%a(1)*psir%a(3)
    g14 = psil%a(1)*psir%a(4)
    g23 = psil%a(2)*psir%a(3)
    g24 = psil%a(2)*psir%a(4)
    g31 = psil%a(3)*psir%a(1)
    g32 = psil%a(3)*psir%a(2)
    g41 = psil%a(4)*psir%a(1)
    g42 = psil\%a(4)*psir\%a(2)
    j\%t = gr * ( g14 - g23) + g1 * ( - g32 + g41)
    j\%x(1) = gr * ( g13 - g24) + g1 * ( g31 - g42)
    j\%x(2) = (gr * ( g13 + g24) + g1 * ( g31 + g42)) * (0, 1)
    j\%x(3) = gr * ( - g14 - g23) + g1 * ( - g32 - g41)
  end function va_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function va2_ff (gva, psil, psir) result (j)
    type(vector) :: j
    complex(kind=default), intent(in), dimension(2) :: gva
    type(bispinor), intent(in) :: psil, psir
    complex(kind=default) :: gl, gr
    complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
    gl = gva(1) + gva(2)
    gr = gva(1) - gva(2)
    g13 = psil%a(1)*psir%a(3)
    g14 = psil%a(1)*psir%a(4)
    g23 = psi1\%a(2)*psir\%a(3)
    g24 = psil%a(2)*psir%a(4)
    g31 = psil%a(3)*psir%a(1)
    g32 = psil%a(3)*psir%a(2)
    g41 = psil%a(4)*psir%a(1)
    g42 = psil%a(4)*psir%a(2)
    j\%t = gr * ( g14 - g23) + g1 * ( - g32 + g41)
    j\%x(1) = gr * ( g13 - g24) + g1 * ( g31 - g42)
    j\%x(2) = (gr * ( g13 + g24) + g1 * ( g31 + g42)) * (0, 1)
    j\%x(3) = gr * ( - g14 - g23) + g1 * ( - g32 - g41)
  end function va2_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function v_ff (gv, psil, psir) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gv
    type(bispinor), intent(in) :: psil, psir
    complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
    g13 = psil%a(1)*psir%a(3)
    g14 = psil%a(1)*psir%a(4)
    g23 = psil%a(2)*psir%a(3)
    g24 = psil%a(2)*psir%a(4)
    g31 = psil%a(3)*psir%a(1)
    g32 = psil%a(3)*psir%a(2)
    g41 = psil%a(4)*psir%a(1)
```

```
g42 = psil%a(4)*psir%a(2)
    j%t =
               gv * (g14 - g23 - g32 + g41)
               gv * ( g13 - g24 + g31 - g42)
    j\%x(1) =
    j\%x(2) =
               gv * (g13 + g24 + g31 + g42) * (0, 1)
    j\%x(3) = gv * ( - g14 - g23 - g32 - g41)
  end function v_ff
\langle \mathit{Implementation of bispinor currents} \rangle + \equiv
 pure function a_ff (ga, psil, psir) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: ga
    type(bispinor), intent(in) :: psil, psir
    complex(kind=default) :: g13, g14, g23, g24, g31, g32, g41, g42
    g13 = psil%a(1)*psir%a(3)
    g14 = psil%a(1)*psir%a(4)
    g23 = psil%a(2)*psir%a(3)
    g24 = psil%a(2)*psir%a(4)
    g31 = psil%a(3)*psir%a(1)
    g32 = psil%a(3)*psir%a(2)
    g41 = psil%a(4)*psir%a(1)
    g42 = psil%a(4)*psir%a(2)
         = -ga * ( g14 - g23 + g32 - g41)
    j\%x(1) = -ga * ( g13 - g24 - g31 + g42)
    j\%x(2) = -ga * ( g13 + g24 - g31 - g42) * (0, 1)
    j\%x(3) = -ga * ( - g14 - g23 + g32 + g41)
  end function a_ff
\langle Implementation of bispinor currents \rangle + \equiv
  pure function vl_ff (gl, psil, psir) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl
    type(bispinor), intent(in) :: psil, psir
    complex(kind=default) :: gl2
    complex(kind=default) :: g31, g32, g41, g42
    g12 = 2 * g1
    g31 = psil%a(3)*psir%a(1)
    g32 = psil%a(3)*psir%a(2)
    g41 = psil%a(4)*psir%a(1)
    g42 = psil%a(4)*psir%a(2)
               g12 * ( - g32 + g41)
    j%t
               gl2 * ( g31 - g42)
    j\%x(1) =
               gl2 * ( g31 + g42) * (0, 1)
    j\%x(2) =
    i\%x(3) =
               gl2 * ( - g32 - g41)
  end function vl_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function vr_ff (gr, psil, psir) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gr
    type(bispinor), intent(in) :: psil, psir
    complex(kind=default) :: gr2
    complex(kind=default) :: g13, g14, g23, g24
    gr2 = 2 * gr
    g13 = psil%a(1)*psir%a(3)
    g14 = psil%a(1)*psir%a(4)
    g23 = psil%a(2)*psir%a(3)
    g24 = psil%a(2)*psir%a(4)
        = gr2 * ( g14 - g23)
    j\%x(1) = gr2 * ( g13 - g24)
    j\%x(2) = gr2 * ( g13 + g24) * (0, 1)
    j\%x(3) = gr2 * ( - g14 - g23)
  end function vr_ff
\langle Implementation of bispinor currents \rangle + \equiv
 pure function vlr_ff (gl, gr, psibar, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(bispinor), intent(in) :: psibar
```

```
type(bispinor), intent(in) :: psi
    j = va_ff (gl+gr, gl-gr, psibar, psi)
  end function vlr_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function tva_ff (gv, ga, psibar, psi) result (t)
    type(tensor2odd) :: t
    complex(kind=default), intent(in) :: gv, ga
    type(bispinor), intent(in) :: psibar
    type(bispinor), intent(in) :: psi
    complex(kind=default) :: gl, gr
    complex(kind=default) :: g11, g22, g33, g44, g1p2, g3p4
            = gv + ga
    gl
            = gv - ga
    g11
            = psibar%a(1)*psi%a(1)
    g22
            = psibar%a(2)*psi%a(2)
    g1p2 = psibar%a(1)*psi%a(2) + psibar%a(2)*psi%a(1)
           = psibar%a(3)*psi%a(4) + psibar%a(4)*psi%a(3)
    g3p4
            = psibar%a(3)*psi%a(3)
    g33
          = psibar%a(4)*psi%a(4)
    t\%e(1) = (g1 * ( - g11 + g22) + gr * ( - g33 + g44)) * (0, 1)
    t\%e(2) = g1 * ( g11 + g22) + gr * ( g33 + g44)
    t\%e(3) = (gl * ( g1p2 ) + gr * ( g3p4 )) * (0, 1)
    t\%b(1) = gl * ( g11 - g22) + gr * ( - g33 + g44)
    t\%b(2) = (g1 * ( g11 + g22) + gr * ( - g33 - g44)) * (0, 1)
    t\%b(3) = g1 * ( - g1p2 ) + gr * ( g3p4
  end function tva_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function tlr_ff (gl, gr, psibar, psi) result (t)
    type(tensor2odd) :: t
    complex(kind=default), intent(in) :: gl, gr
    type(bispinor), intent(in) :: psibar
    type(bispinor), intent(in) :: psi
    t = tva_ff (gr+gl, gr-gl, psibar, psi)
  end function tlr_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function tvam_ff (gv, ga, psibar, psi, p) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gv, ga
    type(bispinor), intent(in) :: psibar
    type(bispinor), intent(in) :: psi
    type(momentum), intent(in) :: p
    j = (tva_ff(gv, ga, psibar, psi) * p) * (0,1)
  end function tvam_ff
\langle Implementation\ of\ bispinor\ currents \rangle + \equiv
  pure function tlrm_ff (gl, gr, psibar, psi, p) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(bispinor), intent(in) :: psibar
    type(bispinor), intent(in) :: psi
    type(momentum), intent(in) :: p
    j = tvam_ff (gr+gl, gr-gl, psibar, psi, p)
  end function tlrm_ff
and

\psi - \phi \gamma_5 = \begin{pmatrix} 0 & 0 & v_- - a_- & -v^* + a^* \\ 0 & 0 & -v + a & v_+ - a_+ \\ v_+ + a_+ & v^* + a^* & 0 & 0 \\ v + a & v_- + a_- & 0 & 0 \end{pmatrix}

                                                                                                             (Y.118)
with v_{\pm} = v_0 \pm v_3, a_{\pm} = a_0 \pm a_3, v = v_1 + iv_2, v^* = v_1 - iv_2, a = a_1 + ia_2, and a^* = a_1 - ia_2. But note that \cdot^*
is not complex conjugation for complex v_{\mu} or a_{\mu}.
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
  public :: f_vaf, f_vf, f_af, f_vlf, f_vrf, f_vlrf, f_va2f, &
              f_tvaf, f_tlrf, f_tvamf, f_tlrmf
```

```
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function f_vaf (gv, ga, v, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: gv, ga
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    complex(kind=default) :: gl, gr
    complex(kind=default) :: vp, vm, v12, v12s
    gl = gv + ga
    gr = gv - ga
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v\%x(1) + (0,1)*v\%x(2)
    v12s = v\%x(1) - (0,1)*v\%x(2)
    vpsi\%a(1) = gr * ( vm * psi\%a(3) - v12s * psi\%a(4))
    vpsi\%a(2) = gr * ( - v12 * psi\%a(3) + vp * psi\%a(4))
    vpsi\%a(3) = gl * ( vp * psi\%a(1) + v12s * psi\%a(2))
    vpsi%a(4) = gl * ( v12 * psi%a(1) + vm * psi%a(2))
  end function f_vaf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_va2f (gva, v, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in), dimension(2) :: gva
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    complex(kind=default) :: gl, gr
    complex(kind=default) :: vp, vm, v12, v12s
    gl = gva(1) + gva(2)
    gr = gva(1) - gva(2)
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v%x(1) + (0,1)*v%x(2)
    v12s = v%x(1) - (0,1)*v%x(2)
    vpsi\%a(1) = gr * ( vm * psi\%a(3) - v12s * psi\%a(4))
    vpsi\%a(2) = gr * ( - v12 * psi\%a(3) + vp * psi\%a(4))
    vpsi%a(3) = gl * ( vp * psi%a(1) + v12s * psi%a(2))
    vpsi\%a(4) = gl * (
                         v12 * psi%a(1) + vm * psi%a(2))
  end function f_va2f
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_vf (gv, v, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: gv
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    complex(kind=default) :: vp, vm, v12, v12s
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v%x(1) + (0,1)*v%x(2)
    v12s = v%x(1) - (0,1)*v%x(2)
    vpsi\%a(1) = gv * ( vm * psi\%a(3) - v12s * psi\%a(4))
    vpsi\%a(2) = gv * ( - v12 * psi\%a(3) + vp * psi\%a(4))
    vpsi\%a(3) = gv * ( vp * psi\%a(1) + v12s * psi\%a(2))
    vpsi\%a(4) = gv * ( v12 * psi\%a(1) + vm * psi\%a(2))
  end function f_vf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function f_af (ga, v, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: ga
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    complex(kind=default) :: vp, vm, v12, v12s
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v%x(1) + (0,1)*v%x(2)
    v12s = v%x(1) - (0,1)*v%x(2)
```

```
vpsi\%a(1) = ga * ( - vm * psi\%a(3) + v12s * psi\%a(4))
    vpsi\%a(2) = ga * (
                        v12 * psi%a(3) - vp * psi%a(4))
                         vp * psi%a(1) + v12s * psi%a(2))
    vpsi\%a(3) = ga * (
    vpsi\%a(4) = ga * ( v12 * psi\%a(1) + vm * psi\%a(2))
  end function f_af
\langle Implementation\ of\ bispinor\ currents \rangle + \equiv
 pure function f_vlf (gl, v, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: gl
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    complex(kind=default) :: gl2
    complex(kind=default) :: vp, vm, v12, v12s
    g12 = 2 * g1
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v\%x(1) + (0,1)*v\%x(2)
    v12s = v\%x(1) - (0,1)*v\%x(2)
    vpsi%a(1) = 0
    vpsi\%a(2) = 0
    vpsi\%a(3) = gl2 * ( vp * psi\%a(1) + v12s * psi\%a(2))
    vpsi\%a(4) = gl2 * ( v12 * psi\%a(1) + vm * psi\%a(2))
  end function f_vlf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_vrf (gr, v, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: gr
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    complex(kind=default) :: gr2
    complex(kind=default) :: vp, vm, v12, v12s
    gr2 = 2 * gr
    vp = v\%t + v\%x(3)
    vm = v\%t - v\%x(3)
    v12 = v\%x(1) + (0,1)*v\%x(2)
    v12s = v%x(1) - (0,1)*v%x(2)
    vpsi\%a(1) = gr2 * ( vm * psi\%a(3) - v12s * psi\%a(4))
    vpsi\%a(2) = gr2 * ( - v12 * psi\%a(3) + vp * psi\%a(4))
    vpsi%a(3) = 0
    vpsi\%a(4) = 0
  end function f_vrf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_vlrf (gl, gr, v, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: gl, gr
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    vpsi = f_vaf (gl+gr, gl-gr, v, psi)
  end function f_vlrf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_tvaf (gv, ga, t, psi) result (tpsi)
    type(bispinor) :: tpsi
    complex(kind=default), intent(in) :: gv, ga
    type(tensor2odd), intent(in) :: t
    type(bispinor), intent(in) :: psi
    complex(kind=default) :: gl, gr
    complex(kind=default) :: e21, e21s, b12, b12s, be3, be3s
        = gv + ga
    gr
       = gv - ga
    gl
    e21 = t\%e(2) + t\%e(1)*(0,1)
    e21s = t\%e(2) - t\%e(1)*(0,1)
    b12 = t\%b(1) + t\%b(2)*(0,1)
    b12s = t\%b(1) - t\%b(2)*(0,1)
    be3 = t\%b(3) + t\%e(3)*(0,1)
```

```
be3s = t\%b(3) - t\%e(3)*(0,1)
                   2*gl*(psi%a(1)*be3+psi%a(2)*(e21+b12s))
    tpsi%a(1) =
                   2*gl * ( - psi%a(2) * be3 + psi%a(1) * (-e21s+b12 ))
    tpsi%a(2) =
                   2*gr * ( psi\%a(3) * be3s + psi\%a(4) * (-e21 +b12s))
    tpsi\%a(3) =
    tpsi%a(4) =
                   2*gr * ( -psi%a(4) * be3s + psi%a(3) * ( e21s+b12 ))
  end function f_tvaf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_tlrf (gl, gr, t, psi) result (tpsi)
    type(bispinor) :: tpsi
    complex(kind=default), intent(in) :: gl, gr
    type(tensor2odd), intent(in) :: t
    type(bispinor), intent(in) :: psi
    tpsi = f_tvaf (gr+gl, gr-gl, t, psi)
  end function f_tlrf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_tvamf (gv, ga, v, psi, k) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: gv, ga
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    type(tensor2odd) :: t
    t = (v.wedge.k) * (0, 0.5)
    vpsi = f_tvaf(gv, ga, t, psi)
  end function f_tvamf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_tlrmf (gl, gr, v, psi, k) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: gl, gr
    type(vector), intent(in) :: v
    type(bispinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    vpsi = f_tvamf (gr+gl, gr-gl, v, psi, k)
  end function f_tlrmf
                       Y.26.2 Fermionic Scalar and Pseudo Scalar Couplings
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
 public :: sp_ff, s_ff, p_ff, sl_ff, sr_ff, slr_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function sp_ff (gs, gp, psil, psir) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gs, gp
    type(bispinor), intent(in) :: psil, psir
            (gs - gp) * (psil%a(1)*psir%a(2) - psil%a(2)*psir%a(1)) &
         + (gs + gp) * (- psil%a(3)*psir%a(4) + psil%a(4)*psir%a(3))
  end function sp_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function s_ff (gs, psil, psir) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gs
    type(bispinor), intent(in) :: psil, psir
    j = gs * (psil * psir)
  end function s_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function p_ff (gp, psil, psir) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gp
    type(bispinor), intent(in) :: psil, psir
    j = gp * (-psil\%a(1)*psir\%a(2) + psil\%a(2)*psir\%a(1) &
                psil%a(3)*psir%a(4) + psil%a(4)*psir%a(3))
  end function p_ff
```

```
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function sl_ff (gl, psil, psir) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl
    type(bispinor), intent(in) :: psil, psir
    j = 2 * gl * (psil%a(1)*psir%a(2) - psil%a(2)*psir%a(1))
  end function sl_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function sr_ff (gr, psil, psir) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gr
    type(bispinor), intent(in) :: psil, psir
    j = 2 * gr * (- psil%a(3)*psir%a(4) + psil%a(4)*psir%a(3))
  end function sr_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function slr_ff (gl, gr, psibar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(bispinor), intent(in) :: psibar
    type(bispinor), intent(in) :: psi
    j = sp_ff (gr+gl, gr-gl, psibar, psi)
  end function slr_ff
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
 public :: f_spf, f_sf, f_pf, f_slf, f_srf, f_slrf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_spf (gs, gp, phi, psi) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: gs, gp
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    phipsi\%a(1:2) = ((gs - gp) * phi) * psi\%a(1:2)
    phipsi\%a(3:4) = ((gs + gp) * phi) * psi\%a(3:4)
  end function f_spf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_sf (gs, phi, psi) result (phipsi)
    type(bispinor) :: phipsi
    {\tt complex(kind=default),\ intent(in)\ ::\ gs}
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    phipsi%a = (gs * phi) * psi%a
  end function f_sf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_pf (gp, phi, psi) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: gp
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    phipsi%a(1:2) = (-gp * phi) * psi%a(1:2)
    phipsi%a(3:4) = (gp * phi) * psi%a(3:4)
  end function f_pf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_slf (gl, phi, psi) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: gl
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    phipsi\%a(1:2) = (2 * gl * phi) * psi\%a(1:2)
    phipsi\%a(3:4) = 0
  end function f_slf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_srf (gr, phi, psi) result (phipsi)
    type(bispinor) :: phipsi
```

```
complex(kind=default), intent(in) :: gr
complex(kind=default), intent(in) :: phi
type(bispinor), intent(in) :: psi
phipsi%a(1:2) = 0
phipsi%a(3:4) = (2 * gr * phi) * psi%a(3:4)
end function f_srf

⟨Implementation of bispinor currents⟩+=
pure function f_slrf (gl, gr, phi, psi) result (phipsi)
type(bispinor) :: phipsi
complex(kind=default), intent(in) :: gl, gr
complex(kind=default), intent(in) :: phi
type(bispinor), intent(in) :: psi
phipsi = f_spf (gr+gl, gr-gl, phi, psi)
end function f_slrf
```

Y.26.3 Couplings for BRST Transformations

3-Couplings

The lists of needed gamma matrices can be found in the next subsection with the gravitino couplings.

```
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
 private :: vv_ff, f_vvf
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
 public :: vmom_ff, mom_ff, mom5_ff, moml_ff, momr_ff, lmom_ff, rmom_ff
\langle Implementation\ of\ bispinor\ currents \rangle + \equiv
 pure function vv_ff (psibar, psi, k) result (psibarpsi)
    type(vector) :: psibarpsi
    type(bispinor), intent(in) :: psibar, psi
    type(vector), intent(in) :: k
    complex(kind=default) :: kp, km, k12, k12s
    type(bispinor) :: kgpsi1, kgpsi2, kgpsi3, kgpsi4
    kp = k\%t + k\%x(3)
    km = k\%t - k\%x(3)
    k12 = k%x(1) + (0,1)*k%x(2)
    k12s = k\%x(1) - (0,1)*k\%x(2)
    kgpsi1\%a(1) = -k\%x(3) * psi\%a(1) - k12s * psi\%a(2)
    kgpsi1\%a(2) = -k12 * psi\%a(1) + k\%x(3) * psi\%a(2)
    kgpsi1\%a(3) = k\%x(3) * psi\%a(3) + k12s * psi\%a(4)
    kgpsi1\%a(4) = k12 * psi\%a(3) - k\%x(3) * psi\%a(4)
    kgpsi2\%a(1) = ((0,-1) * k\%x(2)) * psi\%a(1) - km * psi\%a(2)
    kgpsi2\%a(2) = - kp * psi\%a(1) + ((0,1) * k\%x(2)) * psi\%a(2)
    kgpsi2\%a(3) = ((0,-1) * k\%x(2)) * psi\%a(3) + kp * psi\%a(4)
    kgpsi2\%a(4) = km * psi\%a(3) + ((0,1) * k\%x(2)) * psi\%a(4)
    kgpsi3\%a(1) = (0,1) * (k\%x(1) * psi\%a(1) + km * psi\%a(2))
    kgpsi3\%a(2) = (0,-1) * (kp * psi\%a(1) + k\%x(1) * psi\%a(2))
    kgpsi3\%a(3) = (0,1) * (k\%x(1) * psi\%a(3) - kp * psi\%a(4))
    kgpsi3\%a(4) = (0,1) * (km * psi\%a(3) - k\%x(1) * psi\%a(4))
    kgpsi4\%a(1) = -k\%t * psi\%a(1) - k12s * psi\%a(2)
    kgpsi4\%a(2) = k12 * psi\%a(1) + k\%t * psi\%a(2)
    kgpsi4%a(3) = k%t * psi%a(3) - k12s * psi%a(4)
    kgpsi4\%a(4) = k12 * psi\%a(3) - k\%t * psi\%a(4)
                    = 2 * (psibar * kgpsi1)
    psibarpsi%t
    psibarpsi%x(1) = 2 * (psibar * kgpsi2)
    psibarpsi%x(2) = 2 * (psibar * kgpsi3)
    psibarpsi%x(3) = 2 * (psibar * kgpsi4)
  end function vv_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_vvf (v, psi, k) result (kvpsi)
    type(bispinor) :: kvpsi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: k, v
      complex(kind=default) :: kv30, kv21, kv01, kv31, kv02, kv32
    complex(kind=default) :: ap, am, bp, bm, bps, bms
    kv30 = k%x(3) * v%t - k%t * v%x(3)
```

```
kv21 = (0,1) * (k%x(2) * v%x(1) - k%x(1) * v%x(2))
    kv01 = k\%t * v\%x(1) - k\%x(1) * v\%t
    kv31 = k\%x(3) * v\%x(1) - k\%x(1) * v\%x(3)
    kv02 = (0,1) * (k\%t * v\%x(2) - k\%x(2) * v\%t)
    kv32 = (0,1) * (k%x(3) * v%x(2) - k%x(2) * v%x(3))
    ap = 2 * (kv30 + kv21)
    am = 2 * (-kv30 + kv21)
    bp = 2 * (kv01 + kv31 + kv02 + kv32)
    bm = 2 * (kv01 - kv31 + kv02 - kv32)
    bps = 2 * (kv01 + kv31 - kv02 - kv32)
    bms = 2 * (kv01 - kv31 - kv02 + kv32)
    kvpsi\%a(1) = am * psi\%a(1) + bms * psi\%a(2)
    kvpsi%a(2) = bp * psi%a(1) - am * psi%a(2)
    kvpsi\%a(3) = ap * psi\%a(3) - bps * psi\%a(4)
     kvpsi\%a(4) = -bm * psi\%a(3) - ap * psi\%a(4) 
  end function f_vvf
\langle Implementation of bispinor currents \rangle + \equiv
 pure function vmom_ff (g, psibar, psi, k) result (psibarpsi)
    type(vector) :: psibarpsi
    complex(kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psibar, psi
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    psibarpsi = g * vv_ff (psibar, psi, vk)
  end function vmom_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function mom_ff (g, m, psibar, psi, k) result (psibarpsi)
    complex(kind=default) :: psibarpsi
    type(bispinor), intent(in) :: psibar, psi
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: g, m
    type(bispinor) :: kmpsi
    complex(kind=default) :: kp, km, k12, k12s
    kp = k\%t + k\%x(3)
    km = k\%t - k\%x(3)
    k12 = k\%x(1) + (0,1)*k\%x(2)
    k12s = k\%x(1) - (0,1)*k\%x(2)
    kmpsi\%a(1) = km * psi\%a(3) - k12s * psi\%a(4)
    kmpsi%a(2) = kp * psi%a(4) - k12 * psi%a(3)
    kmpsi\%a(3) = kp * psi\%a(1) + k12s * psi\%a(2)
    kmpsi%a(4) = k12 * psi%a(1) + km * psi%a(2)
    psibarpsi = g * (psibar * kmpsi) + s_ff (m, psibar, psi)
  end function mom_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function mom5_ff (g, m, psibar, psi, k) result (psibarpsi)
    complex(kind=default) :: psibarpsi
    type(bispinor), intent(in) :: psibar, psi
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: g, m
    type(bispinor) :: g5psi
    g5psi%a(1:2) = - psi%a(1:2)
    g5psi%a(3:4) = psi%a(3:4)
    psibarpsi = mom_ff (g, m, psibar, g5psi, k)
  end function mom5_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function moml_ff (g, m, psibar, psi, k) result (psibarpsi)
    complex(kind=default) :: psibarpsi
    type(bispinor), intent(in) :: psibar, psi
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: g, m
    type(bispinor) :: leftpsi
    leftpsi%a(1:2) = 2 * psi%a(1:2)
    leftpsi%a(3:4) = 0
```

```
psibarpsi = mom_ff (g, m, psibar, leftpsi, k)
  end function moml_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function momr_ff (g, m, psibar, psi, k) result (psibarpsi)
    complex(kind=default) :: psibarpsi
    type(bispinor), intent(in) :: psibar, psi
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: g, m
    type(bispinor) :: rightpsi
    rightpsi%a(1:2) = 0
    rightpsi%a(3:4) = 2 * psi%a(3:4)
    psibarpsi = mom_ff (g, m, psibar, rightpsi, k)
  end function momr_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function lmom_ff (g, m, psibar, psi, k) result (psibarpsi)
    complex(kind=default) :: psibarpsi
    type(bispinor), intent(in) :: psibar, psi
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: g, m
    psibarpsi = mom_ff (g, m, psibar, psi, k) + &
                 mom5_ff (g,-m, psibar, psi, k)
  end function lmom_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function rmom_ff (g, m, psibar, psi, k) result (psibarpsi)
    complex(kind=default) :: psibarpsi
    type(bispinor), intent(in) :: psibar, psi
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: g, m
    psibarpsi = mom_ff (g, m, psibar, psi, k) - &
                 mom5_ff (g,-m, psibar, psi, k)
  end function rmom_ff
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
  public :: f_vmomf, f_momf, f_mom5f, f_mom1f, f_momf, f_lmomf, f_rmomf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_vmomf (g, v, psi, k) result (kvpsi)
    type(bispinor) :: kvpsi
    type(bispinor), intent(in) :: psi
    complex(kind=default), intent(in) :: g
    type(momentum), intent(in) :: k
    type(vector), intent(in) :: v
    type(vector) :: vk
    vk = k
    kvpsi = g * f_vvf (v, psi, vk)
  end function f_vmomf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function f_momf (g, m, phi, psi, k) result (kmpsi)
    type(bispinor) :: kmpsi
    type(bispinor), intent(in) :: psi
    {\tt complex(kind=default),\ intent(in)\ ::\ phi,\ g,\ m}
    type(momentum), intent(in) :: k
    complex(kind=default) :: kp, km, k12, k12s
    kp = k\%t + k\%x(3)
    km = k\%t - k\%x(3)
    k12 = k\%x(1) + (0,1)*k\%x(2)
    k12s = k\%x(1) - (0,1)*k\%x(2)
    kmpsi%a(1) = km * psi%a(3) - k12s * psi%a(4)
    kmpsi%a(2) = -k12 * psi%a(3) + kp * psi%a(4)
    kmpsi\%a(3) = kp * psi\%a(1) + k12s * psi\%a(2)
    kmpsi\%a(4) = k12 * psi\%a(1) + km * psi\%a(2)
    kmpsi = g * (phi * kmpsi) + f_sf (m, phi, psi)
  end function f_momf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_mom5f (g, m, phi, psi, k) result (kmpsi)
```

```
type(bispinor) :: kmpsi
    type(bispinor), intent(in) :: psi
    complex(kind=default), intent(in) :: phi, g, m
    type(momentum), intent(in) :: k
    type(bispinor) :: g5psi
    g5psi%a(1:2) = - psi%a(1:2)
g5psi%a(3:4) = psi%a(3:4)
    kmpsi = f_momf (g, m, phi, g5psi, k)
  end function f_mom5f
\langle Implementation\ of\ bispinor\ currents \rangle + \equiv
 pure function f_momlf (g, m, phi, psi, k) result (kmpsi)
    type(bispinor) :: kmpsi
    type(bispinor), intent(in) :: psi
    complex(kind=default), intent(in) :: phi, g, m
    type(momentum), intent(in) :: k
    type(bispinor) :: leftpsi
    leftpsi%a(1:2) = 2 * psi%a(1:2)
    leftpsi%a(3:4) = 0
    kmpsi = f_momf (g, m, phi, leftpsi, k)
  end function f_momlf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_momrf (g, m, phi, psi, k) result (kmpsi)
    type(bispinor) :: kmpsi
    type(bispinor), intent(in) :: psi
    complex(kind=default), intent(in) :: phi, g, m
    type(momentum), intent(in) :: k
    type(bispinor) :: rightpsi
    rightpsi%a(1:2) = 0
    rightpsi%a(3:4) = 2 * psi%a(3:4)
    kmpsi = f_momf (g, m, phi, rightpsi, k)
  end function f_momrf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_lmomf (g, m, phi, psi, k) result (kmpsi)
    type(bispinor) :: kmpsi
    type(bispinor), intent(in) :: psi
    complex(kind=default), intent(in) :: phi, g, m
    type(momentum), intent(in) :: k
    kmpsi = f_momf (g, m, phi, psi, k) + &
             f_mom5f (g,-m, phi, psi, k)
  end function f_lmomf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_rmomf (g, m, phi, psi, k) result (kmpsi)
    type(bispinor) :: kmpsi
    type(bispinor), intent(in) :: psi
    complex(kind=default), intent(in) :: phi, g, m
    type(momentum), intent(in) :: k
    kmpsi = f_momf (g, m, phi, psi, k) - &
             f_mom5f (g,-m, phi, psi, k)
  end function f_rmomf
                                                   4-Couplings
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
 public :: v2_ff, sv1_ff, sv2_ff, pv1_ff, pv2_ff, sv11_ff, sv12_ff, &
       svr1_ff, svr2_ff, svlr1_ff, svlr2_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function v2_ff (g, psibar, v, psi) result (v2)
    type(vector) :: v2
    complex (kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psibar, psi
    {\tt type(vector),\ intent(in)\ ::\ v}
    v2 = (-g) * vv_ff (psibar, psi, v)
  end function v2_ff
```

```
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function sv1_ff (g, psibar, v, psi) result (phi)
    complex(kind=default) :: phi
    type(bispinor), intent(in) :: psibar, psi
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: g
    phi = psibar * f_vf (g, v, psi)
  end function sv1_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function sv2_ff (g, psibar, phi, psi) result (v)
    type(vector) :: v
    complex(kind=default), intent(in) :: phi, g
    type(bispinor), intent(in) :: psibar, psi
    v = phi * v_ff (g, psibar, psi)
  end function sv2_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function pv1_ff (g, psibar, v, psi) result (phi)
    complex(kind=default) :: phi
    type(bispinor), intent(in) :: psibar, psi
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: g
    phi = - (psibar * f_af (g, v, psi))
  end function pv1_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function pv2_ff (g, psibar, phi, psi) result (v)
    type(vector) :: v
    complex(kind=default), intent(in) :: phi, g
    type(bispinor), intent(in) :: psibar, psi
    v = -(phi * a_ff (g, psibar, psi))
  end function pv2_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function svl1_ff (g, psibar, v, psi) result (phi)
    complex(kind=default) :: phi
    type(bispinor), intent(in) :: psibar, psi
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: g
    phi = psibar * f_vlf (g, v, psi)
  end function svl1_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function svl2_ff (g, psibar, phi, psi) result (v)
    type(vector) :: v
    complex(kind=default), intent(in) :: phi, g
    type(bispinor), intent(in) :: psibar, psi
    v = phi * vl_ff (g, psibar, psi)
  end function svl2_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function svr1_ff (g, psibar, v, psi) result (phi)
    complex(kind=default) :: phi
    type(bispinor), intent(in) :: psibar, psi
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: g
    phi = psibar * f_vrf (g, v, psi)
  end function svr1_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function svr2_ff (g, psibar, phi, psi) result (v)
    type(vector) :: v
    complex(kind=default), intent(in) :: phi, g
    type(bispinor), intent(in) :: psibar, psi
    v = phi * vr_ff (g, psibar, psi)
  end function svr2_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function svlr1_ff (gl, gr, psibar, v, psi) result (phi)
```

```
complex(kind=default) :: phi
    type(bispinor), intent(in) :: psibar, psi
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: gl, gr
    phi = psibar * f_vlrf (gl, gr, v, psi)
  end function svlr1_ff
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function svlr2_ff (gl, gr, psibar, phi, psi) result (v)
    type(vector) :: v
    complex(kind=default), intent(in) :: phi, gl, gr
    type(bispinor), intent(in) :: psibar, psi
    v = phi * vlr_ff (gl, gr, psibar, psi)
  end function svlr2_ff
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
 public :: f_v2f, f_svf, f_pvf, f_svlf, f_svrf, f_svlrf
\langle Implementation of bispinor currents \rangle + \equiv
 pure function f_v2f (g, v1, v2, psi) result (vpsi)
    type(bispinor) :: vpsi
    complex(kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v1, v2
    vpsi = g * f_vvf (v2, psi, v1)
  end function f_v2f
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_svf (g, phi, v, psi) result (pvpsi)
    type(bispinor) :: pvpsi
    complex(kind=default), intent(in) :: g, phi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    pvpsi = phi * f_vf (g, v, psi)
  end function f_svf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_pvf (g, phi, v, psi) result (pvpsi)
    type(bispinor) :: pvpsi
    complex(kind=default), intent(in) :: g, phi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    pvpsi = -(phi * f_af (g, v, psi))
  end function f_pvf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_svlf (g, phi, v, psi) result (pvpsi)
    type(bispinor) :: pvpsi
    complex(kind=default), intent(in) :: g, phi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    pvpsi = phi * f_vlf (g, v, psi)
  end function f_svlf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_svrf (g, phi, v, psi) result (pvpsi)
    type(bispinor) :: pvpsi
    complex(kind=default), intent(in) :: g, phi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    pvpsi = phi * f_vrf (g, v, psi)
  end function f_svrf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_svlrf (gl, gr, phi, v, psi) result (pvpsi)
    type(bispinor) :: pvpsi
    complex(kind=default), intent(in) :: gl, gr, phi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    pvpsi = phi * f_vlrf (gl, gr, v, psi)
  end function f_svlrf
```

Y.26.4 Gravitino Couplings

```
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
  public :: pot_grf, pot_fgr, s_grf, s_fgr, p_grf, p_fgr, &
       sl_grf, sl_fgr, sr_grf, sr_fgr, slr_grf, slr_fgr
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
 private :: fgvgr, fgvg5gr, fggvvgr, grkgf, grkkggf, &
       fgkgr, fg5gkgr, grvgf, grg5vgf, grkgggf, fggkggr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function pot_grf (g, gravbar, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(vectorspinor) :: gamma_psi
    gamma_psi\%psi(1)\%a(1) = psi\%a(3)
    gamma_psi\%psi(1)\%a(2) = psi\%a(4)
    gamma_psi\%psi(1)\%a(3) = psi\%a(1)
    gamma_psi\%psi(1)\%a(4) = psi\%a(2)
    gamma_psi\%psi(2)\%a(1) = psi\%a(4)
    gamma_psi\%psi(2)\%a(2) = psi\%a(3)
    gamma_psi\%psi(2)\%a(3) = - psi\%a(2)
    gamma_psi\%psi(2)\%a(4) = - psi\%a(1)
    gamma_psi\%psi(3)\%a(1) = (0,-1) * psi\%a(4)
    gamma_psi\%psi(3)\%a(2) = (0,1) * psi\%a(3)
    gamma_psi\%psi(3)\%a(3) = (0,1) * psi\%a(2)
    gamma_psi\%psi(3)\%a(4) = (0,-1) * psi\%a(1)
    gamma_psi\%psi(4)\%a(1) = psi\%a(3)
    gamma_psi\%psi(4)\%a(2) = - psi\%a(4)
    gamma_psi\%psi(4)\%a(3) = - psi\%a(1)
    gamma_psi\%psi(4)\%a(4) = psi\%a(2)
    j = g * (gravbar * gamma_psi)
  end function pot_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function pot_fgr (g, psibar, grav) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    type(bispinor) :: gamma_grav
    ((0,1)*grav%psi(3)%a(4)) - grav%psi(4)%a(3)
    gamma_grav%a(2) = grav%psi(1)%a(4) - grav%psi(2)%a(3) - &
           ((0,1)*grav%psi(3)%a(3)) + grav%psi(4)%a(4)
    gamma_grav%a(3) = grav%psi(1)%a(1) + grav%psi(2)%a(2) - &
           ((0,1)*grav%psi(3)%a(2)) + grav%psi(4)%a(1)
    gamma_grav%a(4) = grav%psi(1)%a(2) + grav%psi(2)%a(1) + &
           ((0,1)*grav%psi(3)%a(1)) - grav%psi(4)%a(2)
    j = g * (psibar * gamma_grav)
  end function pot_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function grvgf (gravbar, psi, k) result (j)
    complex(kind=default) :: j
    complex(kind=default) :: kp, km, k12, k12s
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: k
    type(vectorspinor) :: kg_psi
    kp = k\%t + k\%x(3)
    km = k\%t - k\%x(3)
    k12 = k%x(1) + (0,1)*k%x(2)
    k12s = k%x(1) - (0,1)*k%x(2)
    !!! Since we are taking the spinor product here, NO explicit
    !!! charge conjugation matrix is needed!
    kg_psi%psi(1)%a(1) = km * psi%a(1) - k12s * psi%a(2)
```

```
kg_psi\%psi(1)\%a(2) = (-k12) * psi\%a(1) + kp * psi\%a(2)
    kg_psi%psi(1)%a(3) = kp * psi%a(3) + k12s * psi%a(4)
    kg_psi\%psi(1)\%a(4) = k12 * psi\%a(3) + km * psi\%a(4)
    kg_psi%psi(2)%a(1) = k12s * psi%a(1) - km * psi%a(2)
    kg_psi\%psi(2)\%a(2) = (-kp) * psi\%a(1) + k12 * psi\%a(2)
    kg_psi\%psi(2)\%a(3) = k12s * psi\%a(3) + kp * psi\%a(4)
    kg_psi%psi(2)%a(4) = km * psi%a(3) + k12 * psi%a(4)
    kg_psi\%psi(3)\%a(1) = (0,1) * (k12s * psi\%a(1) + km * psi\%a(2))
    \label{eq:kg_psi_psi} $$ kg_psi_psi_3)_a(2) = (0,1) * (- kp * psi_a(1) - k12 * psi_a(2))$
    kg_psi\%psi(3)\%a(3) = (0,1) * (k12s * psi\%a(3) - kp * psi\%a(4))
    kg_psi\%psi(3)\%a(4) = (0,1) * (km * psi\%a(3) - k12 * psi\%a(4))
    kg_psi%psi(4)%a(1) = (-km) * psi%a(1) - k12s * psi%a(2)
    kg_psi\%psi(4)\%a(2) = k12 * psi\%a(1) + kp * psi\%a(2)
    kg_psi%psi(4)%a(3) = kp * psi%a(3) - k12s * psi%a(4)
    kg_psi_psi_4)_a(4) = k12 * psi_a(3) - km * psi_a(4)
    j = gravbar * kg_psi
  end function grvgf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function grg5vgf (gravbar, psi, k) result (j)
    complex(kind=default) :: j
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: k
    type(bispinor) :: g5_psi
    g5_psi%a(1:2) = - psi%a(1:2)
g5_psi%a(3:4) = psi%a(3:4)
    j = grvgf (gravbar, g5_psi, k)
  end function grg5vgf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function s_grf (g, gravbar, psi, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    j = g * grvgf (gravbar, psi, vk)
  end function s_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function sl_grf (gl, gravbar, psi, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_1
    type(momentum), intent(in) :: k
    psi_1%a(1:2) = psi%a(1:2)
    psi_1\%a(3:4) = 0
    j = s_grf (gl, gravbar, psi_l, k)
  end function sl_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function sr_grf (gr, gravbar, psi, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gr
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_r
    type(momentum), intent(in) :: k
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    j = s_grf (gr, gravbar, psi_r, k)
  end function sr_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
```

```
pure function slr_grf (gl, gr, gravbar, psi, k) result (j)
   complex(kind=default) :: j
   complex(kind=default), intent(in) :: gl, gr
   type(vectorspinor), intent(in) :: gravbar
   type(bispinor), intent(in) :: psi
   type(momentum), intent(in) :: k
   j = sl_grf (gl, gravbar, psi, k) + sr_grf (gr, gravbar, psi, k)
 end function slr_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function fgkgr (psibar, grav, k) result (j)
   complex(kind=default) :: j
   complex(kind=default) :: kp, km, k12, k12s
   type(bispinor), intent(in) :: psibar
   type(vectorspinor), intent(in) :: grav
   type(vector), intent(in) :: k
   type(bispinor) :: gk_grav
   kp = k\%t + k\%x(3)
   km = k\%t - k\%x(3)
   k12 = k\%x(1) + (0,1)*k\%x(2)
   k12s = k\%x(1) - (0,1)*k\%x(2)
   !!! Since we are taking the spinor product here, NO explicit
   !!! charge conjugation matrix is needed!
   gk_grav%a(1) = kp * grav%psi(1)%a(1) + k12s * grav%psi(1)%a(2) &
                 - k12 * grav%psi(2)%a(1) - km * grav%psi(2)%a(2) &
                 + (0,1) * k12 * grav%psi(3)%a(1)
                 + (0,1) * km * grav%psi(3)%a(2) &
                 - kp * grav%psi(4)%a(1) - k12s * grav%psi(4)%a(2)
   - kp * grav%psi(2)%a(1) - k12s * grav%psi(2)%a(2) &
                 - (0,1) * kp * grav%psi(3)%a(1) &
                 - (0,1) * k12s * grav%psi(3)%a(2) &
                 + k12 * grav%psi(4)%a(1) + km * grav%psi(4)%a(2)
   gk_grav%a(3) = km * grav%psi(1)%a(3) - k12s * grav%psi(1)%a(4) &
                 - k12 * grav%psi(2)%a(3) + kp * grav%psi(2)%a(4) &
                 + (0,1) * k12 * grav%psi(3)%a(3) &
                 - (0,1) * kp * grav%psi(3)%a(4) &
                 + km * grav%psi(4)%a(3) - k12s * grav%psi(4)%a(4)
   gk_grav%a(4) = - k12 * grav%psi(1)%a(3) + kp * grav%psi(1)%a(4) &
                 + km * grav%psi(2)%a(3) - k12s * grav%psi(2)%a(4) &
                 + (0,1) * km * grav%psi(3)%a(3) &
                 -(0,1) * k12s * grav%psi(3)%a(4) &
                 + k12 * grav%psi(4)%a(3) - kp * grav%psi(4)%a(4)
   j = psibar * gk_grav
 end function fgkgr
\langle Implementation of bispinor currents \rangle + \equiv
 pure function fg5gkgr (psibar, grav, k) result (j)
   complex(kind=default) :: j
   type(bispinor), intent(in) :: psibar
   type(vectorspinor), intent(in) :: grav
   type(vector), intent(in) :: k
   type(bispinor) :: psibar_g5
   psibar_g5\%a(1:2) = - psibar\%a(1:2)
   psibar_g5\%a(3:4) = psibar\%a(3:4)
   j = fgkgr (psibar_g5, grav, k)
 end function fg5gkgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function s_fgr (g, psibar, grav, k) result (j)
   complex(kind=default) :: j
   complex(kind=default), intent(in) :: g
   type(bispinor), intent(in) :: psibar
   type(vectorspinor), intent(in) :: grav
   type(momentum), intent(in) :: k
   type(vector) :: vk
   vk = k
   j = g * fgkgr (psibar, grav, vk)
```

```
end function s_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function sl_fgr (gl, psibar, grav, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_l
    type(vectorspinor), intent(in) :: grav
    type(momentum), intent(in) :: k
    psibar_1\%a(1:2) = psibar\%a(1:2)
    psibar_1\%a(3:4) = 0
    j = s_fgr (gl, psibar_l, grav, k)
  end function sl_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function sr_fgr (gr, psibar, grav, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gr
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_r
    type(vectorspinor), intent(in) :: grav
    type(momentum), intent(in) :: k
    psibar_r%a(1:2) = 0
    psibar_r%a(3:4) = psibar%a(3:4)
    j = s_fgr (gr, psibar_r, grav, k)
  end function sr\_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function slr_fgr (gl, gr, psibar, grav, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    type(momentum), intent(in) :: k
    j = sl_fgr (gl, psibar, grav, k) + sr_fgr (gr, psibar, grav, k)
  end function slr_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function p_grf (g, gravbar, psi, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    j = g * grg5vgf (gravbar, psi, vk)
  end function p_grf
\langle Implementation\ of\ bispinor\ currents \rangle + \equiv
 pure function p_fgr (g, psibar, grav, k) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    j = g * fg5gkgr (psibar, grav, vk)
  end function p_fgr
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
 public :: f_potgr, f_sgr, f_pgr, f_vgr, f_vlrgr, f_slgr, f_srgr, f_slrgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_potgr (g, phi, psi) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi
```

type(vectorspinor), intent(in) :: psi phipsi%a(1) = (g * phi) * (psi%psi(1)%a(3) - psi%psi(2)%a(4) + &((0,1)*psi%psi(3)%a(4)) - psi%psi(4)%a(3))phipsi%a(2) = (g * phi) * (psi%psi(1)%a(4) - psi%psi(2)%a(3) - &((0,1)*psi%psi(3)%a(3)) + psi%psi(4)%a(4))phipsi%a(3) = (g * phi) * (psi%psi(1)%a(1) + psi%psi(2)%a(2) - &((0,1)*psi%psi(3)%a(2)) + psi%psi(4)%a(1))phipsi%a(4) = (g * phi) * (psi%psi(1)%a(2) + psi%psi(2)%a(1) + &((0,1)*psi%psi(3)%a(1)) - psi%psi(4)%a(2))end function f_potgr

The slashed notation:

$$\not k = \begin{pmatrix} 0 & 0 & k_{-} & -k^{*} \\ 0 & 0 & -k & k_{+} \\ k_{+} & k^{*} & 0 & 0 \\ k & k_{-} & 0 & 0 \end{pmatrix}, \qquad \not k \gamma_{5} = \begin{pmatrix} 0 & 0 & k_{-} & -k^{*} \\ 0 & 0 & -k & k_{+} \\ -k_{+} & -k^{*} & 0 & 0 \\ -k_{-} & -k_{-} & 0 & 0 \end{pmatrix}$$
 (Y.119)

with $k_{\pm} = k_0 \pm k_3$, $k = k_1 + ik_2$, $k^* = k_1 - ik_2$. But note that \cdot^* is not complex conjugation for complex k_{μ} .

$$\gamma^{0} \not k = \begin{pmatrix} k_{+} & k^{*} & 0 & 0 \\ k & k_{-} & 0 & 0 \\ 0 & 0 & k_{-} & -k^{*} \\ 0 & 0 & -k & k_{+} \end{pmatrix}, \qquad \gamma^{0} \not k \gamma^{5} = \begin{pmatrix} -k_{+} & -k^{*} & 0 & 0 \\ -k & -k_{-} & 0 & 0 \\ 0 & 0 & k_{-} & -k^{*} \\ 0 & 0 & -k & k_{+} \end{pmatrix}$$
(Y.120a)

$$\gamma^{1} \not k = \begin{pmatrix} k & k_{-} & 0 & 0 \\ k_{+} & k^{*} & 0 & 0 \\ 0 & 0 & k & -k_{+} \\ 0 & 0 & -k_{-} & k^{*} \end{pmatrix}, \qquad \gamma^{1} \not k \gamma^{5} = \begin{pmatrix} -k & -k_{-} & 0 & 0 \\ -k_{+} & -k^{*} & 0 & 0 \\ 0 & 0 & k & -k_{+} \\ 0 & 0 & -k_{-} & k^{*} \end{pmatrix}$$
(Y.120b)

$$\gamma^{1} \not k = \begin{pmatrix} k & k_{-} & 0 & 0 \\ k_{+} & k^{*} & 0 & 0 \\ 0 & 0 & k & -k_{+} \\ 0 & 0 & -k_{-} & k^{*} \end{pmatrix}, \qquad \gamma^{1} \not k \gamma^{5} = \begin{pmatrix} -k & -k_{-} & 0 & 0 \\ -k_{+} & -k^{*} & 0 & 0 \\ 0 & 0 & k & -k_{+} \\ 0 & 0 & -k_{-} & k^{*} \end{pmatrix}$$

$$\gamma^{2} \not k = \begin{pmatrix} -ik & -ik_{-} & 0 & 0 \\ ik_{+} & ik^{*} & 0 & 0 \\ 0 & 0 & -ik & ik_{+} \\ 0 & 0 & -ik_{-} & ik^{*} \end{pmatrix}, \qquad \gamma^{2} \not k \gamma^{5} = \begin{pmatrix} ik & ik_{-} & 0 & 0 \\ -ik_{+} & -ik^{*} & 0 & 0 \\ 0 & 0 & -ik_{-} & ik^{*} \end{pmatrix}$$

$$\gamma^{3} \not k = \begin{pmatrix} k_{+} & k^{*} & 0 & 0 \\ -k_{-} & -k_{-} & 0 & 0 \\ 0 & 0 & -k_{-} & k^{*} \end{pmatrix}, \qquad \gamma^{3} \not k \gamma^{5} = \begin{pmatrix} -k_{+} & -k^{*} & 0 & 0 \\ k_{-} & k_{-} & 0 & 0 \\ 0 & 0 & -k_{-} & k^{*} \\ 0 & 0 & -k_{-} & k^{*} \\ 0 & 0 & -k_{-} & k^{*} \end{pmatrix}$$

$$(Y.120c)$$

$$\gamma^{3} \not k = \begin{pmatrix} k_{+} & k^{*} & 0 & 0 \\ -k & -k_{-} & 0 & 0 \\ 0 & 0 & -k_{-} & k^{*} \\ 0 & 0 & -k & k_{+} \end{pmatrix}, \qquad \gamma^{3} \not k \gamma^{5} = \begin{pmatrix} -k_{+} & -k^{*} & 0 & 0 \\ k & k_{-} & 0 & 0 \\ 0 & 0 & -k_{-} & k^{*} \\ 0 & 0 & -k & k_{+} \end{pmatrix}$$
(Y.120d)

and

$$k \gamma^{1} = \begin{pmatrix} k^{*} & -k_{-} & 0 & 0 \\ -k_{+} & k & 0 & 0 \\ 0 & 0 & k^{*} & k_{+} \\ 0 & 0 & k_{-} & k \end{pmatrix}, \qquad k \gamma^{1} \gamma^{5} = \begin{pmatrix} -k^{*} & k_{-} & 0 & 0 \\ k_{+} & -k & 0 & 0 \\ 0 & 0 & k^{*} & k_{+} \\ 0 & 0 & k_{-} & k \end{pmatrix}$$
(Y.121b)

$$k \gamma^2 = \begin{pmatrix} ik^* & ik_- & 0 & 0\\ -ik_+ & -ik & 0 & 0\\ 0 & 0 & ik^* & -ik_+\\ 0 & 0 & ik_- & -ik \end{pmatrix}, \qquad k \gamma^2 \gamma^5 = \begin{pmatrix} -ik^* & -ik_- & 0 & 0\\ ik_+ & ik & 0 & 0\\ 0 & 0 & ik^* & -ik_+\\ 0 & 0 & ik_- & -ik \end{pmatrix}$$
(Y.121c)

$$k \gamma^{3} = \begin{pmatrix} -k_{-} & -k^{*} & 0 & 0 \\ k & k_{+} & 0 & 0 \\ 0 & 0 & k_{+} & -k^{*} \\ 0 & 0 & k & -k_{-} \end{pmatrix}, \qquad k \gamma^{3} \gamma^{5} = \begin{pmatrix} k_{-} & k^{*} & 0 & 0 \\ -k & -k_{+} & 0 & 0 \\ 0 & 0 & k_{+} & -k^{*} \\ 0 & 0 & k & -k_{-} \end{pmatrix}$$
(Y.121d)

and

$$C\gamma^{0} \not k = \begin{pmatrix} k & k_{-} & 0 & 0 \\ -k_{+} & -k^{*} & 0 & 0 \\ 0 & 0 & k & -k_{+} \\ 0 & 0 & k_{-} & -k^{*} \end{pmatrix}, \qquad C\gamma^{0} \not k \gamma^{5} = \begin{pmatrix} -k & -k_{-} & 0 & 0 \\ k_{+} & k^{*} & 0 & 0 \\ 0 & 0 & k & -k_{+} \\ 0 & 0 & k_{-} & -k^{*} \end{pmatrix}$$
(Y.122a)

$$C\gamma^{1} \not k = \begin{pmatrix} k_{+} & k^{*} & 0 & 0 \\ -k & -k_{-} & 0 & 0 \\ 0 & 0 & k_{-} & -k^{*} \\ 0 & 0 & k & -k_{+} \end{pmatrix}, \qquad C\gamma^{1} \not k \gamma^{5} = \begin{pmatrix} -k_{+} & -k^{*} & 0 & 0 \\ k & k_{-} & 0 & 0 \\ 0 & 0 & k_{-} & -k^{*} \\ 0 & 0 & k & -k_{+} \end{pmatrix}$$

$$C\gamma^{2} \not k = \begin{pmatrix} ik_{+} & ik^{*} & 0 & 0 \\ ik_{-} & ik_{-} & 0 & 0 \\ 0 & 0 & ik_{-} & -ik^{*} \\ 0 & 0 & -ik_{-} & ik_{+} \end{pmatrix}, \qquad C\gamma^{2} \not k \gamma^{5} = \begin{pmatrix} -ik_{+} & -ik^{*} & 0 & 0 \\ -ik_{-} & -ik_{-} & 0 & 0 \\ 0 & 0 & ik_{-} & -ik^{*} \\ 0 & 0 & -ik_{-} & ik_{+} \end{pmatrix}$$

$$C\gamma^{3} \not k = \begin{pmatrix} -k_{-} & -k_{-} & 0 & 0 \\ -k_{+} & -k^{*} & 0 & 0 \\ 0 & 0 & k_{-} & -k_{+} \\ 0 & 0 & -k_{-} & k^{*} \end{pmatrix}, \qquad C\gamma^{3} \not k \gamma^{5} = \begin{pmatrix} k_{-} & 0 & 0 \\ k_{+} & k^{*} & 0 & 0 \\ 0 & 0 & k_{-} & -k_{+} \\ 0 & 0 & -k_{-} & k^{*} \end{pmatrix}$$

$$(Y.122c)$$

$$C\gamma^{2} \not k = \begin{pmatrix} \mathrm{i}k_{+} & \mathrm{i}k^{*} & 0 & 0 \\ \mathrm{i}k_{-} & \mathrm{i}k_{-} & 0 & 0 \\ 0 & 0 & \mathrm{i}k_{-} & -\mathrm{i}k^{*} \\ 0 & 0 & -\mathrm{i}k & \mathrm{i}k_{+} \end{pmatrix}, \qquad C\gamma^{2} \not k \gamma^{5} = \begin{pmatrix} -\mathrm{i}k_{+} & -\mathrm{i}k^{*} & 0 & 0 \\ -\mathrm{i}k_{-} & -\mathrm{i}k_{-} & 0 & 0 \\ 0 & 0 & \mathrm{i}k_{-} & -\mathrm{i}k^{*} \\ 0 & 0 & -\mathrm{i}k & \mathrm{i}k_{+} \end{pmatrix}$$
 (Y.122c)

$$C\gamma^{3} \not k = \begin{pmatrix} -k & -k_{-} & 0 & 0 \\ -k_{+} & -k^{*} & 0 & 0 \\ 0 & 0 & k & -k_{+} \\ 0 & 0 & -k_{-} & k^{*} \end{pmatrix}, \qquad C\gamma^{3} \not k \gamma^{5} = \begin{pmatrix} k & k_{-} & 0 & 0 \\ k_{+} & k^{*} & 0 & 0 \\ 0 & 0 & k & -k_{+} \\ 0 & 0 & -k_{-} & k^{*} \end{pmatrix}$$
(Y.122d)

and

$$Ck\gamma^{0} = \begin{pmatrix} -k & k^{+} & 0 & 0\\ -k_{-} & k^{*} & 0 & 0\\ 0 & 0 & -k & -k_{-}\\ 0 & 0 & k_{+} & k^{*} \end{pmatrix}, \qquad Ck\gamma^{0}\gamma^{5} = \begin{pmatrix} k & -k_{+} & 0 & 0\\ k_{-} & -k^{*} & 0 & 0\\ 0 & 0 & -k & -k_{-}\\ 0 & 0 & k_{+} & k^{*} \end{pmatrix}$$
(Y.123a)

$$C k \gamma^{1} = \begin{pmatrix} -k_{+} & k & 0 & 0 \\ -k^{*} & k_{-} & 0 & 0 \\ 0 & 0 & -k_{-} & -k \\ 0 & 0 & k^{*} & k_{+} \end{pmatrix}, \qquad C k \gamma^{1} \gamma^{5} = \begin{pmatrix} k_{+} & -k & 0 & 0 \\ k^{*} & -k_{-} & 0 & 0 \\ 0 & 0 & -k_{-} & -k \\ 0 & 0 & k^{*} & k_{+} \end{pmatrix}$$
(Y.123b)

$$Ck\gamma^{0} = \begin{pmatrix} -k & k^{+} & 0 & 0 \\ -k_{-} & k^{*} & 0 & 0 \\ 0 & 0 & -k & -k_{-} \\ 0 & 0 & k_{+} & k^{*} \end{pmatrix}, \qquad Ck\gamma^{0}\gamma^{5} = \begin{pmatrix} k & -k_{+} & 0 & 0 \\ k_{-} & -k^{*} & 0 & 0 \\ 0 & 0 & -k & -k_{-} \\ 0 & 0 & k_{+} & k^{*} \end{pmatrix}$$

$$Ck\gamma^{1} = \begin{pmatrix} -k_{+} & k & 0 & 0 \\ -k^{*} & k_{-} & 0 & 0 \\ 0 & 0 & -k_{-} & -k \\ 0 & 0 & k^{*} & k_{+} \end{pmatrix}, \qquad Ck\gamma^{1}\gamma^{5} = \begin{pmatrix} k_{+} & -k & 0 & 0 \\ k^{*} & -k_{-} & 0 & 0 \\ 0 & 0 & -k_{-} & -k \\ 0 & 0 & k^{*} & k_{+} \end{pmatrix}$$

$$Ck\gamma^{2} = \begin{pmatrix} -ik_{+} & -ik & 0 & 0 \\ -ik^{*} & -ik_{-} & 0 & 0 \\ 0 & 0 & -ik_{-} & ik \\ 0 & 0 & ik^{*} & -ik_{+} \end{pmatrix}, \qquad Ck\gamma^{2}\gamma^{5} = \begin{pmatrix} ik_{+} & ik & 0 & 0 \\ ik^{*} & ik_{-} & 0 & 0 \\ 0 & 0 & -ik_{-} & ik \\ 0 & 0 & ik^{*} & -ik_{+} \end{pmatrix}$$

$$(Y.123c)$$

$$C \not k \gamma^3 = \begin{pmatrix} k & k_+ & 0 & 0 \\ k_- & k^* & 0 & 0 \\ 0 & 0 & -k & k_- \\ 0 & 0 & k_+ & -k^* \end{pmatrix}, \qquad C \not k \gamma^3 \gamma^5 = \begin{pmatrix} -k & -k_+ & 0 & 0 \\ -k_- & -k^* & 0 & 0 \\ 0 & 0 & -k & k_- \\ 0 & 0 & k_+ & -k^* \end{pmatrix}$$
(Y.123d)

```
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function fgvgr (psi, k) result (kpsi)
    type(bispinor) :: kpsi
    complex(kind=default) :: kp, km, k12, k12s
    type(vector), intent(in) :: k
    type(vectorspinor), intent(in) :: psi
    kp = k\%t + k\%x(3)
    km = k\%t - k\%x(3)
    k12 = k%x(1) + (0,1)*k%x(2)
    k12s = k\%x(1) - (0,1)*k\%x(2)
    kpsi\%a(1) = kp * psi\%psi(1)\%a(1) + k12s * psi\%psi(1)\%a(2) &
               - k12 * psi%psi(2)%a(1) - km * psi%psi(2)%a(2) &
              + (0,1) * k12 * psi%psi(3)%a(1) + (0,1) * km * psi%psi(3)%a(2) &
               - kp * psi%psi(4)%a(1) - k12s * psi%psi(4)%a(2)
    kpsi\%a(2) = k12 * psi\%psi(1)\%a(1) + km * psi\%psi(1)\%a(2) &
               - kp * psi%psi(2)%a(1) - k12s * psi%psi(2)%a(2) &
               - (0,1) * kp * psi%psi(3)%a(1) - (0,1) * k12s * psi%psi(3)%a(2) &
               + k12 * psi%psi(4)%a(1) + km * psi%psi(4)%a(2)
    kpsi\%a(3) = km * psi\%psi(1)\%a(3) - k12s * psi\%psi(1)\%a(4) &
               - k12 * psi%psi(2)%a(3) + kp * psi%psi(2)%a(4) &
              + (0,1) * k12 * psi%psi(3)%a(3) - (0,1) * kp * psi%psi(3)%a(4) &
               + km * psi%psi(4)%a(3) - k12s * psi%psi(4)%a(4)
    kpsi\%a(4) = - k12 * psi\%psi(1)\%a(3) + kp * psi\%psi(1)\%a(4) &
               + km * psi%psi(2)%a(3) - k12s * psi%psi(2)%a(4) &
               + (0,1) * km * psi%psi(3)%a(3) - (0,1) * k12s * psi%psi(3)%a(4) &
              + k12 * psi%psi(4)%a(3) - kp * psi%psi(4)%a(4)
  end function fgvgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_sgr (g, phi, psi, k) result (phipsi)
```

type(bispinor) :: phipsi

```
complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi
    type(momentum), intent(in) :: k
    type(vectorspinor), intent(in) :: psi
    type(vector) :: vk
    vk = k
    phipsi = (g * phi) * fgvgr (psi, vk)
  end function f_sgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_slgr (gl, phi, psi, k) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: gl
    complex(kind=default), intent(in) :: phi
    type(momentum), intent(in) :: k
    type(vectorspinor), intent(in) :: psi
    phipsi = f_sgr (gl, phi, psi, k)
    phipsi\%a(3:4) = 0
  end function f_slgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_srgr (gr, phi, psi, k) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: gr
    complex(kind=default), intent(in) :: phi
    type(momentum), intent(in) :: k
    type(vectorspinor), intent(in) :: psi
    phipsi = f_sgr (gr, phi, psi, k)
    phipsi\%a(1:2) = 0
  end function f_srgr
\langle Implementation\ of\ bispinor\ currents \rangle + \equiv
 pure function f_slrgr (gl, gr, phi, psi, k) result (phipsi)
    type(bispinor) :: phipsi, phipsi_1, phipsi_r
    complex(kind=default), intent(in) :: gl, gr
    complex(kind=default), intent(in) :: phi
    type(momentum), intent(in) :: k
    type(vectorspinor), intent(in) :: psi
    phipsi_l = f_slgr (gl, phi, psi, k)
    phipsi_r = f_srgr (gr, phi, psi, k)
    phipsi%a(1:2) = phipsi_1%a(1:2)
    phipsi\%a(3:4) = phipsi_r\%a(3:4)
  end function f_slrgr
\langle Implementation of bispinor currents \rangle + \equiv
 pure function fgvg5gr (psi, k) result (kpsi)
    type(bispinor) :: kpsi
    type(vector), intent(in) :: k
    type(vectorspinor), intent(in) :: psi
    type(bispinor) :: kpsi_dum
    kpsi_dum = fgvgr (psi, k)
    kpsi\%a(1:2) = - kpsi_dum\%a(1:2)
    kpsi\%a(3:4) = kpsi_dum\%a(3:4)
  end function fgvg5gr
\langle Implementation of bispinor currents \rangle + \equiv
 pure function f_pgr (g, phi, psi, k) result (phipsi)
    type(bispinor) :: phipsi
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi
    type(momentum), intent(in) :: k
    type(vectorspinor), intent(in) :: psi
    type(vector) :: vk
    phipsi = (g * phi) * fgvg5gr (psi, vk)
  end function f_pgr
```

The needed construction of gamma matrices involving the commutator of two gamma matrices. For the slashed terms we use as usual the abbreviations $k_{\pm} = k_0 \pm k_3$, $k = k_1 + \mathrm{i}k_2$, $k^* = k_1 - \mathrm{i}k_2$ and analogous expressions for

the vector v^{μ} . We remind you that \cdot^* is not complex conjugation for complex k_{μ} . Furthermore we introduce (in what follows the brackets around the vector indices have the usual meaning of antisymmetrizing with respect to the indices inside the brackets, here without a factor two in the denominator)

$$a_{+} = k_{+}v_{-} + kv^{*} - k_{-}v_{+} - k^{*}v = 2(k_{[3}v_{0]} + ik_{[2}v_{1]})$$
(Y.124a)

$$a_{-} = k_{-}v_{+} + kv^{*} - k_{+}v_{-} - k^{*}v = 2(-k_{[3}v_{0]} + ik_{[2}v_{1]})$$
 (Y.124b)

$$b_{+} = 2(k_{+}v - kv_{+}) = 2(k_{[0}v_{1]} + k_{[3}v_{1]} + ik_{[0}v_{2]} + ik_{[3}v_{2]})$$
(Y.124c)

$$b_{-} = 2(k_{-}v - kv_{-}) = 2(k_{[0}v_{1]} - k_{[3}v_{1]} + ik_{[0}v_{2]} - ik_{[3}v_{2]})$$
(Y.124d)

$$b_{+} = 2(k_{+}v - kv_{+})$$

$$b_{-} = 2(k_{-}v - kv_{-})$$

$$b_{+*} = 2(k_{+}v^{*} - k^{*}v_{+})$$

$$= 2(k_{[0}v_{1]} + k_{[3}v_{1]} + ik_{[0}v_{2]} + ik_{[3}v_{2]})$$

$$= 2(k_{[0}v_{1]} - k_{[3}v_{1]} + ik_{[0}v_{2]} - ik_{[3}v_{2]})$$

$$= 2(k_{[0}v_{1]} + k_{[3}v_{1]} - ik_{[0}v_{2]} - ik_{[3}v_{2]})$$

$$= 2(k_{[0}v_{1]} + k_{[3}v_{1]} - ik_{[0}v_{2]} - ik_{[3}v_{2]})$$

$$= 2(k_{[0}v_{1]} + k_{[0}v_{2]} - ik_{[0}v_{2]} - ik_{[0}v_{2]})$$

$$= 2(k_{[0}v_{1]} + k_{[0}v_{2]} - ik_{[0}v_{2]} - ik_$$

$$b_{-*} = 2(k_{-}v^* - k^*v_{-}) = 2(k_{[0}v_{1]} - k_{[3}v_{1]} - ik_{[0}v_{2]} + ik_{[3}v_{2]})$$
 (Y.124f)

Of course, one could introduce a more advanced notation, but we don't want to become confused.

$$[k, \gamma^{0}] = \begin{pmatrix} -2k_{3} & -2k^{*} & 0 & 0\\ -2k & 2k_{3} & 0 & 0\\ 0 & 0 & 2k_{3} & 2k^{*}\\ 0 & 0 & 2k & -2k_{3} \end{pmatrix}$$
(Y.125a)

$$[\not k, \gamma^1] = \begin{pmatrix} -2ik_2 & -2k_- & 0 & 0\\ -2k_+ & 2ik_2 & 0 & 0\\ 0 & 0 & -2ik_2 & 2k_+\\ 0 & 0 & 2k_- & 2ik_2 \end{pmatrix}$$
(Y.125b)

$$[k, \gamma^{0}] = \begin{pmatrix} -2k_{3} & -2k^{*} & 0 & 0 \\ -2k & 2k_{3} & 0 & 0 \\ 0 & 0 & 2k_{3} & 2k^{*} \\ 0 & 0 & 2k & -2k_{3} \end{pmatrix}$$

$$[k, \gamma^{1}] = \begin{pmatrix} -2ik_{2} & -2k_{-} & 0 & 0 \\ -2k_{+} & 2ik_{2} & 0 & 0 \\ 0 & 0 & -2ik_{2} & 2k_{+} \\ 0 & 0 & 2k_{-} & 2ik_{2} \end{pmatrix}$$

$$[k, \gamma^{2}] = \begin{pmatrix} 2ik_{1} & 2ik_{-} & 0 & 0 \\ -2ik_{+} & -2ik_{1} & 0 & 0 \\ 0 & 0 & 2ik_{1} & -2ik_{+} \\ 0 & 0 & 2ik_{-} & -2ik_{1} \end{pmatrix}$$

$$(Y.125a)$$

$$[k, \gamma^{2}] = \begin{pmatrix} 2ik_{1} & 2ik_{-} & 0 & 0 \\ -2ik_{+} & -2ik_{1} & 0 & 0 \\ 0 & 0 & 2ik_{-} & -2ik_{1} \end{pmatrix}$$

$$(Y.125c)$$

$$[k, \gamma^3] = \begin{pmatrix} -2k_0 & -2k^* & 0 & 0\\ 2k & 2k_0 & 0 & 0\\ 0 & 0 & 2k_0 & -2k^*\\ 0 & 0 & 2k & -2k_0 \end{pmatrix}$$
(Y.125d)

$$[\not k, \not V] = \begin{pmatrix} a_- & b_{-*} & 0 & 0 \\ b_+ & -a_- & 0 & 0 \\ 0 & 0 & a_+ & -b_{+*} \\ 0 & 0 & -b_- & -a_+ \end{pmatrix}$$
 (Y.125e)

$$[k, V] = \begin{pmatrix} a_{-} & b_{-*} & 0 & 0 \\ b_{+} & -a_{-} & 0 & 0 \\ 0 & 0 & a_{+} & -b_{+*} \\ 0 & 0 & -b_{-} & -a_{+} \end{pmatrix}$$

$$\gamma^{5} \gamma^{0} [k, V] = \begin{pmatrix} 0 & 0 & -a_{+} & b_{+*} \\ 0 & 0 & b_{-} & a_{+} \\ a_{-} & b_{-*} & 0 & 0 \\ b_{+} & -a_{-} & 0 & 0 \end{pmatrix}$$

$$\gamma^{5} \gamma^{1} [k, V] = \begin{pmatrix} 0 & 0 & b_{-} & a_{+} \\ 0 & 0 & -a_{+} & b_{+*} \\ -b_{+} & a_{-} & 0 & 0 \\ -a_{-} & -b_{-*} & 0 & 0 \end{pmatrix}$$

$$(Y.125e)$$

$$(Y.125e)$$

$$\gamma^{5}\gamma^{1}[\not k, \not V] = \begin{pmatrix} 0 & 0 & b_{-} & a_{+} \\ 0 & 0 & -a_{+} & b_{+*} \\ -b_{+} & a_{-} & 0 & 0 \\ -a_{-} & -b_{-*} & 0 & 0 \end{pmatrix}$$
(Y.125g)

$$\gamma^{5} \gamma^{2} [\not k, V] = \begin{pmatrix}
0 & 0 & -ib_{-} & -ia_{+} \\
0 & 0 & -ia_{+} & ib_{+*} \\
ib_{+} & -ia_{-} & 0 & 0 \\
-ia_{-} & -ib_{-*} & 0 & 0
\end{pmatrix}$$

$$\gamma^{5} \gamma^{3} [\not k, V] = \begin{pmatrix}
0 & 0 & -a_{+} & b_{+*} \\
0 & 0 & -b_{-} & -a_{+} \\
-a_{-} & -b_{-*} & 0 & 0 \\
b_{+} & -a_{-} & 0 & 0
\end{pmatrix}$$
(Y.125h)

$$\gamma^{5}\gamma^{3}[k,V] = \begin{pmatrix} 0 & 0 & -a_{+} & b_{+*} \\ 0 & 0 & -b_{-} & -a_{+} \\ -a_{-} & -b_{-*} & 0 & 0 \\ b_{+} & -a_{-} & 0 & 0 \end{pmatrix}$$
(Y.125i)

and

$$[\not k, \not V] \gamma^0 \gamma^5 = \begin{pmatrix} 0 & 0 & a_- & b_{-*} \\ 0 & 0 & b_+ & -a_- \\ -a_+ & b_{+*} & 0 & 0 \\ b_- & a_+ & 0 & 0 \end{pmatrix}$$

$$(Y.126a)$$

$$[\not k, \not V] \gamma^1 \gamma^5 = \begin{pmatrix} 0 & 0 & b_{-*} & a_{-} \\ 0 & 0 & -a_{-} & b_{+} \\ -b_{+*} & a_{+} & 0 & 0 \\ -a_{+} & -b_{-} & 0 & 0 \end{pmatrix}$$
 (Y.126b)

$$[\not k, \not V] \gamma^2 \gamma^5 = \begin{pmatrix} 0 & 0 & ib_{-*} & -ia_{-} \\ 0 & 0 & -ia_{-} & -ib_{+} \\ -ib_{+*} & -ia_{+} & 0 & 0 \\ -ia_{+} & ib_{-} & 0 & 0 \end{pmatrix}$$
(Y.126c)

$$[\not k, \not V]\gamma^3 \gamma^5 = \begin{pmatrix} 0 & 0 & a_- & -b_{-*} \\ 0 & 0 & b_+ & a_- \\ a_+ & b_{+*} & 0 & 0 \\ -b_- & a_+ & 0 & 0 \end{pmatrix}$$
(Y.126d)

In what follows l always means twice the value of k, e.g. $l_+=2k_+$. We use the abbreviation $C^{\mu\nu}\equiv C[k,\gamma^\mu]\gamma^\nu\gamma^5$.

$$C^{00} = \begin{pmatrix} 0 & 0 & -l & -l_3 \\ 0 & 0 & l_3 & l^* \\ l & -l_3 & 0 & 0 \\ -l_3 & -l^* & 0 & 0 \end{pmatrix}, \qquad C^{20} = \begin{pmatrix} 0 & 0 & -\mathrm{i}l_+ & -\mathrm{i}l_1 \\ 0 & 0 & -\mathrm{i}l_1 & -\mathrm{i}l_- \\ \mathrm{i}l_- & -\mathrm{i}l_1 & 0 & 0 \\ -\mathrm{i}l_1 & \mathrm{i}l_+ & 0 & 0 \end{pmatrix}$$
(Y.127a)

$$C^{00} = \begin{pmatrix} 0 & 0 & -l & -l_3 \\ 0 & 0 & l_3 & l^* \\ l & -l_3 & 0 & 0 \\ -l_3 & -l^* & 0 & 0 \end{pmatrix}, \qquad C^{20} = \begin{pmatrix} 0 & 0 & -il_+ & -il_1 \\ 0 & 0 & -il_1 & -il_- \\ il_- & -il_1 & 0 & 0 \\ -il_1 & il_+ & 0 & 0 \end{pmatrix}$$

$$C^{01} = \begin{pmatrix} 0 & 0 & l_3 & -l \\ 0 & 0 & l^* & l_3 \\ l_3 & -l & 0 & 0 \\ l^* & l_3 & 0 & 0 \end{pmatrix}, \qquad C^{21} = \begin{pmatrix} 0 & 0 & -il_1 & -il_+ \\ 0 & 0 & -il_- & -il_1 \\ il_1 & -il_- & 0 & 0 \\ -il_+ & il_1 & 0 & 0 \end{pmatrix}$$

$$(Y.127a)$$

$$C^{02} = \begin{pmatrix} 0 & 0 & \mathrm{i}l_3 & \mathrm{i}l \\ 0 & 0 & \mathrm{i}l^* & -\mathrm{i}l_3 \\ \mathrm{i}l_3 & \mathrm{i}l & 0 & 0 \\ \mathrm{i}l^* & -\mathrm{i}l_3 & 0 & 0 \end{pmatrix}, \qquad C^{22} = \begin{pmatrix} 0 & 0 & l_1 & -l_+ \\ 0 & 0 & l_- & -l_1 \\ -l_1 & -l_- & 0 & 0 \\ l_+ & l_1 & 0 & 0 \end{pmatrix}$$
(Y.127c)

$$C^{03} = \begin{pmatrix} 0 & 0 & -l & -l_3 \\ 0 & 0 & l_3 & -l^* \\ -l & -l_3 & 0 & 0 \\ l_3 & -l^* & 0 & 0 \end{pmatrix}, \qquad C^{23} = \begin{pmatrix} 0 & 0 & -il_+ & il_1 \\ 0 & 0 & -il_1 & il_- \\ -il_- & -il_1 & 0 & 0 \\ il_1 & il_+ & 0 & 0 \end{pmatrix}$$

$$C^{10} = \begin{pmatrix} 0 & 0 & -l_+ & il_2 \\ 0 & 0 & il_2 & l_- \\ l_- & il_2 & 0 & 0 \\ il_2 & -l_+ & 0 & 0 \end{pmatrix}, \qquad C^{30} = \begin{pmatrix} 0 & 0 & l & l_0 \\ 0 & 0 & l_0 & l^* \\ l & -l_0 & 0 & 0 \\ -l_0 & l^* & 0 & 0 \end{pmatrix}$$

$$(Y.127d)$$

$$C^{10} = \begin{pmatrix} 0 & 0 & -l_{+} & il_{2} \\ 0 & 0 & il_{2} & l_{-} \\ l_{-} & il_{2} & 0 & 0 \\ il_{2} & -l_{+} & 0 & 0 \end{pmatrix}, \qquad C^{30} = \begin{pmatrix} 0 & 0 & l & l_{0} \\ 0 & 0 & l_{0} & l^{*} \\ l & -l_{0} & 0 & 0 \\ -l_{0} & l^{*} & 0 & 0 \end{pmatrix}$$
(Y.127e)

$$C^{11} = \begin{pmatrix} 0 & 0 & il_2 & -l_+ \\ 0 & 0 & l_- & il_2 \\ -il_2 & -l_- & 0 & 0 \\ l_+ & -il_2 & 0 & 0 \end{pmatrix}, \qquad C^{31} = \begin{pmatrix} 0 & 0 & l_0 & l \\ 0 & 0 & l^* & l_0 \\ l_0 & -l & 0 & 0 \\ -l^* & l_0 & 0 & 0 \end{pmatrix}$$
(Y.127f)

$$C^{12} = \begin{pmatrix} 0 & 0 & -l_2 & il_+ \\ 0 & 0 & il_- & l_2 \\ l_2 & il_- & 0 & 0 \\ il_+ & -l_2 & 0 & 0 \end{pmatrix}, \qquad C^{32} = \begin{pmatrix} 0 & 0 & il_0 & -il \\ 0 & 0 & il^* & -il_0 \\ il_0 & il & 0 & 0 \\ -il^* & -il_0 & 0 & 0 \end{pmatrix}$$

$$C^{13} = \begin{pmatrix} 0 & 0 & -l_+ & -il_2 \\ 0 & 0 & il_2 & -l_- \\ -l_- & il_2 & 0 & 0 \\ -il^* & -l_0 & 0 & 0 \end{pmatrix}, \qquad C^{33} = \begin{pmatrix} 0 & 0 & l & -l_0 \\ 0 & 0 & l_0 & -l^* \\ -l & -l_0 & 0 & 0 \\ l_0 & l^* & 0 & 0 \end{pmatrix}$$

$$(Y.127g)$$

$$C^{13} = \begin{pmatrix} 0 & 0 & -l_{+} & -il_{2} \\ 0 & 0 & il_{2} & -l_{-} \\ -l_{-} & il_{2} & 0 & 0 \\ -il_{2} & -l_{+} & 0 & 0 \end{pmatrix}, \qquad C^{33} = \begin{pmatrix} 0 & 0 & l & -l_{0} \\ 0 & 0 & l_{0} & -l^{*} \\ -l & -l_{0} & 0 & 0 \\ l_{0} & l^{*} & 0 & 0 \end{pmatrix}$$
(Y.127h)

and, with the abbreviation $\tilde{C}^{\mu\nu} \equiv C \gamma^5 \gamma^{\nu} [k, \gamma^{\mu}]$ (note the reversed order of the indices!)

$$\tilde{C}^{00} = \begin{pmatrix} 0 & 0 & -l & l_3 \\ 0 & 0 & l_3 & l^* \\ l & -l_3 & 0 & 0 \\ -l_3 & -l^* & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{20} = \begin{pmatrix} 0 & 0 & -il_- & il_1 \\ 0 & 0 & il_1 & -il_+ \\ il_+ & il_1 & 0 & 0 \\ il_1 & il_- & 0 & 0 \end{pmatrix} \tag{Y.128a}$$

$$\tilde{C}^{00} = \begin{pmatrix} 0 & 0 & -l & l_3 \\ 0 & 0 & l_3 & l^* \\ l & -l_3 & 0 & 0 \\ -l_3 & -l^* & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{20} = \begin{pmatrix} 0 & 0 & -il_- & il_1 \\ 0 & 0 & il_1 & -il_+ \\ il_+ & il_1 & 0 & 0 \\ il_1 & il_- & 0 & 0 \end{pmatrix} \tag{Y.128a}$$

$$\tilde{C}^{01} = \begin{pmatrix} 0 & 0 & -l_3 & -l^* \\ 0 & 0 & l & -l_3 \\ -l_3 & -l^* & 0 & 0 \\ l & -l_3 & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{21} = \begin{pmatrix} 0 & 0 & -il_1 & il_+ \\ 0 & 0 & il_- & -il_1 \\ il_1 & il_- & 0 & 0 \\ il_+ & il_1 & 0 & 0 \end{pmatrix} \tag{Y.128b}$$

$$\tilde{C}^{02} = \begin{pmatrix}
0 & 0 & -il_3 & -il^* \\
0 & 0 & -il & il_3 \\
-il_3 & -il^* & 0 & 0 \\
-il & il_3 & 0 & 0
\end{pmatrix}, \qquad \tilde{C}^{22} = \begin{pmatrix}
0 & 0 & l_1 & -l_+ \\
0 & 0 & l_- & -l_1 \\
-l_1 & -l_- & 0 & 0 \\
l_+ & l_1 & 0 & 0
\end{pmatrix}$$
(Y.128c)

$$\tilde{C}^{03} = \begin{pmatrix} 0 & 0 & l & -l_3 \\ 0 & 0 & l_3 & l^* \\ l & -l_3 & 0 & 0 \\ l_3 & l^* & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{23} = \begin{pmatrix} 0 & 0 & il_- & -il_1 \\ 0 & 0 & il_1 & -il_+ \\ il_+ & il_1 & 0 & 0 \\ -il_1 & -il_- & 0 & 0 \end{pmatrix}$$

$$\tilde{C}^{10} = \begin{pmatrix} 0 & 0 & -l_- & -il_2 \\ 0 & 0 & -il_2 & l_+ \\ l_+ & -il_2 & 0 & 0 \\ -il_2 & -l_- & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{30} = \begin{pmatrix} 0 & 0 & -l & l_0 \\ 0 & 0 & l_0 & -l^* \\ -l & -l_0 & 0 & 0 \\ -l_0 & -l^* & 0 & 0 \end{pmatrix}$$

$$\tilde{C}^{11} = \begin{pmatrix} 0 & 0 & il_2 & -l_+ \\ 0 & 0 & l_- & il_2 \\ -il_2 & -l_- & 0 & 0 \\ l_+ & -il_2 & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{31} = \begin{pmatrix} 0 & 0 & -l_0 & l^* \\ 0 & 0 & l_- & -l_0 \\ -l_0 & -l^* & 0 & 0 \\ -l & -l_0 & 0 & 0 \end{pmatrix}$$

$$\tilde{C}^{12} = \begin{pmatrix} 0 & 0 & -l_2 & -il_+ \\ 0 & 0 & -il_- & l_2 \\ l_2 & -il_- & 0 & 0 \\ -il_+ & -l_2 & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{32} = \begin{pmatrix} 0 & 0 & -il_0 & il^* \\ 0 & 0 & -il & il_0 \\ -il_0 & -il^* & 0 & 0 \\ il & il_0 & 0 & 0 \end{pmatrix}$$

$$\tilde{C}^{13} = \begin{pmatrix} 0 & 0 & l_- & il_2 \\ 0 & 0 & -il_2 & l_+ \\ l_+ & -il_2 & 0 & 0 \\ il_2 & l_- & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{33} = \begin{pmatrix} 0 & 0 & l_- & -l_0 \\ 0 & 0 & l_0 & -l^* \\ -l_- & -l_0 & 0 & 0 \\ l_0 & l^* & 0 & 0 \end{pmatrix}$$

$$(Y.128b)$$

$$\tilde{C}^{10} = \begin{bmatrix} 0 & 0 & -ll_2 & l_+ \\ l_+ & -il_2 & 0 & 0 \\ -il_2 & -l_- & 0 & 0 \end{bmatrix}, \qquad \tilde{C}^{30} = \begin{bmatrix} 0 & 0 & l_0 & -l \\ -l & -l_0 & 0 & 0 \\ -l_0 & -l^* & 0 & 0 \end{bmatrix}$$

$$\begin{pmatrix} 0 & 0 & il_2 & -l_+ \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & -l_0 & l^* \end{pmatrix}$$
(Y.128e)

$$\tilde{C}^{11} = \begin{pmatrix}
0 & 0 & il_2 & -l_+ \\
0 & 0 & l_- & il_2 \\
-il_2 & -l_- & 0 & 0 \\
l_+ & -il_2 & 0 & 0
\end{pmatrix}, \qquad \tilde{C}^{31} = \begin{pmatrix}
0 & 0 & -l_0 & l^* \\
0 & 0 & l & -l_0 \\
-l_0 & -l^* & 0 & 0 \\
-l & -l_0 & 0 & 0
\end{pmatrix} \tag{Y.128f}$$

$$\tilde{C}^{12} = \begin{pmatrix} 0 & 0 & -l_2 & -il_+ \\ 0 & 0 & -il_- & l_2 \\ l_2 & -il_- & 0 & 0 \\ -il_+ & -l_2 & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{32} = \begin{pmatrix} 0 & 0 & -il_0 & il^* \\ 0 & 0 & -il & il_0 \\ -il_0 & -il^* & 0 & 0 \\ il & il_0 & 0 & 0 \end{pmatrix}$$
(Y.128g)

$$\tilde{C}^{13} = \begin{pmatrix} 0 & 0 & l_{-} & il_{2} \\ 0 & 0 & -il_{2} & l_{+} \\ l_{+} & -il_{2} & 0 & 0 \\ il_{2} & l_{-} & 0 & 0 \end{pmatrix}, \qquad \tilde{C}^{33} = \begin{pmatrix} 0 & 0 & l & -l_{0} \\ 0 & 0 & l_{0} & -l^{*} \\ -l & -l_{0} & 0 & 0 \\ l_{0} & l^{*} & 0 & 0 \end{pmatrix} \tag{Y.128h}$$

```
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function fggvvgr (v, psi, k) result (psikv)
    type(bispinor) :: psikv
    type(vectorspinor), intent(in) :: psi
    type(vector), intent(in) :: v, k
    complex(kind=default) :: kv30, kv21, kv01, kv31, kv02, kv32
    complex(kind=default) :: ap, am, bp, bm, bps, bms
    kv30 = k\%x(3) * v\%t - k\%t * v\%x(3)
    kv21 = (0,1) * (k%x(2) * v%x(1) - k%x(1) * v%x(2))
    kv01 = k\%t * v\%x(1) - k\%x(1) * v\%t
    kv31 = k%x(3) * v%x(1) - k%x(1) * v%x(3)
    kv02 = (0,1) * (k\%t * v\%x(2) - k\%x(2) * v\%t)
    kv32 = (0,1) * (k%x(3) * v%x(2) - k%x(2) * v%x(3))
    ap = 2 * (kv30 + kv21)
    am = 2 * (-kv30 + kv21)
    bp = 2 * (kv01 + kv31 + kv02 + kv32)
    bm = 2 * (kv01 - kv31 + kv02 - kv32)
    bps = 2 * (kv01 + kv31 - kv02 - kv32)
    bms = 2 * (kv01 - kv31 - kv02 + kv32)
    psikv%a(1) = (-ap) * psi%psi(1)%a(3) + bps * psi%psi(1)%a(4) &
               + (-bm) * psi%psi(2)%a(3) + (-ap) * psi%psi(2)%a(4) &
               + (0,1) * (bm * psi%psi(3)%a(3) + ap * psi%psi(3)%a(4)) &
               + ap * psi%psi(4)%a(3) + (-bps) * psi%psi(4)%a(4)
    psikv\%a(2) = bm * psi\%psi(1)\%a(3) + ap * psi\%psi(1)\%a(4) &
               + ap * psi%psi(2)%a(3) + (-bps) * psi%psi(2)%a(4) &
               + (0,1) * (ap * psi%psi(3)%a(3) - bps * psi%psi(3)%a(4)) &
               + bm * psi%psi(4)%a(3) + ap * psi%psi(4)%a(4)
    psikv\%a(3) = am * psi\%psi(1)\%a(1) + bms * psi\%psi(1)\%a(2) &
               + bp * psi%psi(2)%a(1) + (-am) * psi%psi(2)%a(2) &
               + (0,-1) * (bp * psi\%psi(3)\%a(1) + (-am) * psi\%psi(3)\%a(2)) &
               + am * psi%psi(4)%a(1) + bms * psi%psi(4)%a(2)
    psikv%a(4) = bp * psi%psi(1)%a(1) + (-am) * psi%psi(1)%a(2) &
               + am * psi%psi(2)%a(1) + bms * psi%psi(2)%a(2) &
               + (0,1) * (am * psi\%psi(3)\%a(1) + bms * psi\%psi(3)\%a(2)) &
               + (-bp) * psi%psi(4)%a(1) + am * psi%psi(4)%a(2)
  end function fggvvgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_vgr (g, v, psi, k) result (psikkkv)
    type(bispinor) :: psikkkv
    type(vectorspinor), intent(in) :: psi
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: k
```

```
complex(kind=default), intent(in) :: g
    type(vector) :: vk
    vk = k
    psikkkv = g * (fggvvgr (v, psi, vk))
  end function f_vgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_vlrgr (gl, gr, v, psi, k) result (psikv)
    type(bispinor) :: psikv
    type(vectorspinor), intent(in) :: psi
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: gl, gr
    type(vector) :: vk
    vk = k
    psikv = fggvvgr (v, psi, vk)
    psikv%a(1:2) = gl * psikv%a(1:2)
    psikv\%a(3:4) = gr * psikv\%a(3:4)
  end function f_vlrgr
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
 public :: gr_potf, gr_sf, gr_pf, gr_vf, gr_vlrf, gr_slf, gr_srf, gr_slrf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function gr_potf (g, phi, psi) result (phipsi)
    type(vectorspinor) :: phipsi
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    phipsi\%psi(1)\%a(1) = (g * phi) * psi\%a(3)
    phipsi\%psi(1)\%a(2) = (g * phi) * psi\%a(4)
    phipsi\%psi(1)\%a(3) = (g * phi) * psi\%a(1)
    phipsi\%psi(1)\%a(4) = (g * phi) * psi\%a(2)
    phipsi\%psi(2)\%a(1) = (g * phi) * psi\%a(4)
    phipsi%psi(2)%a(2) = (g * phi) * psi%a(3)
    phipsi\%psi(2)\%a(3) = ((-g) * phi) * psi\%a(2)
    phipsi%psi(2)%a(4) = ((-g) * phi) * psi%a(1)
    phipsi\%psi(3)\%a(1) = ((0,-1) * g * phi) * psi\%a(4)
    phipsi\%psi(3)\%a(2) = ((0,1) * g * phi) * psi\%a(3)
    phipsi\%psi(3)\%a(3) = ((0,1) * g * phi) * psi\%a(2)
    phipsi\%psi(3)\%a(4) = ((0,-1) * g * phi) * psi\%a(1)
    phipsi\%psi(4)\%a(1) = (g * phi) * psi\%a(3)
    phipsi\%psi(4)\%a(2) = ((-g) * phi) * psi\%a(4)
    phipsi\%psi(4)\%a(3) = ((-g) * phi) * psi\%a(1)
    phipsi%psi(4)%a(4) = (g * phi) * psi%a(2)
  end function gr_potf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function grkgf (psi, k) result (kpsi)
    type(vectorspinor) :: kpsi
    complex(kind=default) :: kp, km, k12, k12s
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: k
    kp = k\%t + k\%x(3)
    km = k\%t - k\%x(3)
    k12 = k\%x(1) + (0,1)*k\%x(2)
    k12s = k\%x(1) - (0,1)*k\%x(2)
    kpsi\%psi(1)\%a(1) = km * psi\%a(1) - k12s * psi\%a(2)
    kpsi\%psi(1)\%a(2) = (-k12) * psi\%a(1) + kp * psi\%a(2)
    kpsi\%psi(1)\%a(3) = kp * psi\%a(3) + k12s * psi\%a(4)
    kpsi\%psi(1)\%a(4) = k12 * psi\%a(3) + km * psi\%a(4)
    kpsi\%psi(2)\%a(1) = k12s * psi\%a(1) - km * psi\%a(2)
    kpsi\%psi(2)\%a(2) = (-kp) * psi\%a(1) + k12 * psi\%a(2)
    kpsi\%psi(2)\%a(3) = k12s * psi\%a(3) + kp * psi\%a(4)
    kpsi\%psi(2)\%a(4) = km * psi\%a(3) + k12 * psi\%a(4)
    kpsi\%psi(3)\%a(1) = (0,1) * (k12s * psi\%a(1) + km * psi\%a(2))
    kpsi\%psi(3)\%a(2) = (0,-1) * (kp * psi\%a(1) + k12 * psi\%a(2))
    kpsi\%psi(3)\%a(3) = (0,1) * (k12s * psi\%a(3) - kp * psi\%a(4))
```

```
kpsi\%psi(3)\%a(4) = (0,1) * (km * psi\%a(3) - k12 * psi\%a(4))
    kpsi\%psi(4)\%a(1) = -(km * psi\%a(1) + k12s * psi\%a(2))
    kpsi\%psi(4)\%a(2) = k12 * psi\%a(1) + kp * psi\%a(2)
    kpsi\%psi(4)\%a(3) = kp * psi\%a(3) - k12s * psi\%a(4)
    kpsi\%psi(4)\%a(4) = k12 * psi\%a(3) - km * psi\%a(4)
  end function grkgf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function gr_sf (g, phi, psi, k) result (phipsi)
    \verb"type"(vectorspinor") :: phipsi"
    {\tt complex(kind=default),\ intent(in)\ ::\ g}
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    phipsi = (g * phi) * grkgf (psi, vk)
  end function gr_sf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function gr_slf (gl, phi, psi, k) result (phipsi)
    type(vectorspinor) :: phipsi
    complex(kind=default), intent(in) :: gl
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l
    type(momentum), intent(in) :: k
    psi_1\%a(1:2) = psi\%a(1:2)
    psi_1\%a(3:4) = 0
    phipsi = gr_sf (gl, phi, psi_l, k)
  end function gr_slf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function gr_srf (gr, phi, psi, k) result (phipsi)
    type(vectorspinor) :: phipsi
    complex(kind=default), intent(in) :: gr
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_r
    type(momentum), intent(in) :: k
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    phipsi = gr_sf (gr, phi, psi_r, k)
  end function gr_srf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function gr_slrf (gl, gr, phi, psi, k) result (phipsi)
    type(vectorspinor) :: phipsi
    complex(kind=default), intent(in) :: gl, gr
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    phipsi = gr_slf (gl, phi, psi, k) + gr_srf (gr, phi, psi, k)
  end function gr_slrf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function grkggf (psi, k) result (kpsi)
    type(vectorspinor) :: kpsi
    complex(kind=default) :: kp, km, k12, k12s
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: k
    kp = k\%t + k\%x(3)
    km = k\%t - k\%x(3)
    k12 = k%x(1) + (0,1)*k%x(2)
    k12s = k%x(1) - (0,1)*k%x(2)
    kpsi\%psi(1)\%a(1) = (-km) * psi\%a(1) + k12s * psi\%a(2)
    kpsi\%psi(1)\%a(2) = k12 * psi\%a(1) - kp * psi\%a(2)
    kpsi\%psi(1)\%a(3) = kp * psi\%a(3) + k12s * psi\%a(4)
    kpsi\%psi(1)\%a(4) = k12 * psi\%a(3) + km * psi\%a(4)
```

```
kpsi\%psi(2)\%a(1) = (-k12s) * psi\%a(1) + km * psi\%a(2)
    kpsi\%psi(2)\%a(2) = kp * psi\%a(1) - k12 * psi\%a(2)
    kpsi\%psi(2)\%a(3) = k12s * psi\%a(3) + kp * psi\%a(4)
    kpsi\%psi(2)\%a(4) = km * psi\%a(3) + k12 * psi\%a(4)
    kpsi\%psi(3)\%a(1) = (0,-1) * (k12s * psi\%a(1) + km * psi\%a(2))
    kpsi\%psi(3)\%a(2) = (0,1) * (kp * psi\%a(1) + k12 * psi\%a(2))
    kpsi\%psi(3)\%a(3) = (0,1) * (k12s * psi\%a(3) - kp * psi\%a(4))
    kpsi\%psi(3)\%a(4) = (0,1) * (km * psi\%a(3) - k12 * psi\%a(4))
    kpsi\%psi(4)\%a(1) = km * psi\%a(1) + k12s * psi\%a(2)
    kpsi\%psi(4)\%a(2) = -(k12 * psi\%a(1) + kp * psi\%a(2))
    kpsi\%psi(4)\%a(3) = kp * psi\%a(3) - k12s * psi\%a(4)
    kpsi\%psi(4)\%a(4) = k12 * psi\%a(3) - km * psi\%a(4)
  end function grkggf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function gr_pf (g, phi, psi, k) result (phipsi)
    type(vectorspinor) :: phipsi
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi
    type(bispinor), intent(in) :: psi
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    phipsi = (g * phi) * grkggf (psi, vk)
  end function gr_pf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function grkkggf (v, psi, k) result (psikv)
    type(vectorspinor) :: psikv
    type(bispinor), intent(in) :: psi
    \label{eq:type} \mbox{type(vector), intent(in) :: v, k}
    complex(kind=default) :: kv30, kv21, kv01, kv31, kv02, kv32
    complex(kind=default) :: ap, am, bp, bm, bps, bms, imago
    imago = (0.0_default,1.0_default)
    kv30 = k%x(3) * v%t - k%t * v%x(3)
    kv21 = imago * (k%x(2) * v%x(1) - k%x(1) * v%x(2))
    kv01 = k\%t * v\%x(1) - k\%x(1) * v\%t
    kv31 = k%x(3) * v%x(1) - k%x(1) * v%x(3)
    kv02 = imago * (k%t * v%x(2) - k%x(2) * v%t)
    kv32 = imago * (k%x(3) * v%x(2) - k%x(2) * v%x(3))
    ap = 2 * (kv30 + kv21)
    am = 2 * ((-kv30) + kv21)
    bp = 2 * (kv01 + kv31 + kv02 + kv32)
    bm = 2 * (kv01 - kv31 + kv02 - kv32)
    bps = 2 * (kv01 + kv31 - kv02 - kv32)
    bms = 2 * (kv01 - kv31 - kv02 + kv32)
    psikv%psi(1)%a(1) = am * psi%a(3) + bms * psi%a(4)
    psikv%psi(1)%a(2) = bp * psi%a(3) + (-am) * psi%a(4)
    psikv%psi(1)%a(3) = (-ap) * psi%a(1) + bps * psi%a(2)
    psikv%psi(1)%a(4) = bm * psi%a(1) + ap * psi%a(2)
    psikv%psi(2)%a(1) = bms * psi%a(3) + am * psi%a(4)
    psikv\%psi(2)\%a(2) = (-am) * psi\%a(3) + bp * psi\%a(4)
    psikv\%psi(2)\%a(3) = (-bps) * psi\%a(1) + ap * psi\%a(2)
    psikv%psi(2)%a(4) = (-ap) * psi%a(1) + (-bm) * psi%a(2)
    psikv\%psi(3)\%a(1) = imago * (bms * psi\%a(3) - am * psi\%a(4))
    psikv\%psi(3)\%a(2) = (-imago) * (am * psi\%a(3) + bp * psi\%a(4))
    psikv%psi(3)%a(3) = (-imago) * (bps * psi%a(1) + ap * psi%a(2))
    psikv%psi(3)%a(4) = imago * ((-ap) * psi%a(1) + bm * psi%a(2))
    psikv%psi(4)%a(1) = am * psi%a(3) + (-bms) * psi%a(4)
    psikv%psi(4)%a(2) = bp * psi%a(3) + am * psi%a(4)
    psikv%psi(4)%a(3) = ap * psi%a(1) + bps * psi%a(2)
    psikv%psi(4)%a(4) = (-bm) * psi%a(1) + ap * psi%a(2)
  end function grkkggf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function gr_vf (g, v, psi, k) result (psikv)
    type(vectorspinor) :: psikv
    type(bispinor), intent(in) :: psi
```

```
type(vector), intent(in) :: v
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: g
    type(vector) :: vk
    vk = k
    psikv = g * (grkkggf (v, psi, vk))
  end function gr_vf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function gr_vlrf (gl, gr, v, psi, k) result (psikv)
    type(vectorspinor) :: psikv
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l, psi_r
    type(vector), intent(in) :: v
    type(momentum), intent(in) :: k
    complex(kind=default), intent(in) :: gl, gr
    type(vector) :: vk
    vk = k
    psi_1%a(1:2) = psi%a(1:2)
    psi_1\%a(3:4) = 0
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    psikv = gl * grkkggf (v, psi_l, vk) + gr * grkkggf (v, psi_r, vk)
  end function gr_vlrf
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
  public :: v_grf, v_fgr
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
  public :: vlr_grf, vlr_fgr
V^{\mu} = \psi_{\rho}^T C^{\mu\rho} \psi
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function grkgggf (psil, psir, k) result (j)
    type(vector) :: j
    type(vectorspinor), intent(in) :: psil
    type(bispinor), intent(in) :: psir
    type(vector), intent(in) :: k
    type(vectorspinor) :: c_psir0, c_psir1, c_psir2, c_psir3
    complex(kind=default) :: kp, km, k12, k12s, ik2
    kp = k\%t + k\%x(3)
    km = k\%t - k\%x(3)
    k12 = (k\%x(1) + (0,1)*k\%x(2))
    k12s = (k\%x(1) - (0,1)*k\%x(2))
    ik2 = (0,1) * k%x(2)
    !!! New version:
    c_psir0\%psi(1)\%a(1) = (-k\%x(3)) * psir\%a(3) + (-k12s) * psir\%a(4)
    c_psir0\%psi(1)\%a(2) = (-k12) * psir\%a(3) + k\%x(3) * psir\%a(4)
    c_psir0\%psi(1)\%a(3) = (-k\%x(3)) * psir\%a(1) + (-k12s) * psir\%a(2)
    c_psir0\%psi(1)\%a(4) = (-k12) * psir\%a(1) + k\%x(3) * psir\%a(2)
    c_psir0%psi(2)%a(1) = (-k12s) * psir%a(3) + (-k%x(3)) * psir%a(4)
    c_psir0\%psi(2)\%a(2) = k\%x(3) * psir\%a(3) + (-k12) * psir\%a(4)
    c_psir0\%psi(2)\%a(3) = k12s * psir\%a(1) + k\%x(3) * psir\%a(2)
    c_psir0\%psi(2)\%a(4) = (-k\%x(3)) * psir\%a(1) + k12 * psir\%a(2)
    c_psir0\%psi(3)\%a(1) = (0,1) * ((-k12s) * psir\%a(3) + k\%x(3) * psir\%a(4))
    c_psir0\%psi(3)\%a(2) = (0,1) * (k\%x(3) * psir\%a(3) + k12 * psir\%a(4))
    c_psir(3)\%a(3) = (0,1) * (k12s * psir\%a(1) + (-k\%x(3)) * psir\%a(2))
    c_psir(3)\%a(4) = (0,1) * ((-k\%x(3)) * psir\%a(1) + (-k12) * psir\%a(2))
    c_psir0\%psi(4)\%a(1) = (-k\%x(3)) * psir\%a(3) + k12s * psir\%a(4)
    c_psir0\%psi(4)\%a(2) = (-k12) * psir\%a(3) + (-k\%x(3)) * psir\%a(4)
    c_psir0\%psi(4)\%a(3) = k\%x(3) * psir\%a(1) + (-k12s) * psir\%a(2)
    c_psir0\%psi(4)\%a(4) = k12 * psir\%a(1) + k\%x(3) * psir\%a(2)
    !!!
    c_psir1\%psi(1)\%a(1) = (-ik2) * psir\%a(3) + (-km) * psir\%a(4)
    c_psir1\%psi(1)\%a(2) = (-kp) * psir\%a(3) + ik2 * psir\%a(4)
    c_psir1\%psi(1)\%a(3) = ik2 * psir\%a(1) + (-kp) * psir\%a(2)
    c_psir1%psi(1)%a(4) = (-km) * psir%a(1) + (-ik2) * psir%a(2)
    c_psir1\%psi(2)\%a(1) = (-km) * psir\%a(3) + (-ik2) * psir\%a(4)
```

```
c_psir1\%psi(2)\%a(2) = ik2 * psir\%a(3) + (-kp) * psir\%a(4)
    c_psir1\%psi(2)\%a(3) = kp * psir\%a(1) + (-ik2) * psir\%a(2)
    c_psir1\%psi(2)\%a(4) = ik2 * psir\%a(1) + km * psir\%a(2)
    c_psir1\%psi(3)\%a(1) = ((0,-1) * km) * psir\%a(3) + (-k\%x(2)) * psir\%a(4)
    c_psir1\%psi(3)\%a(2) = (-k\%x(2)) * psir\%a(3) + ((0,1) * kp) * psir\%a(4)
     c_p sir1\% psi(3)\% a(3) = ((0,1) * kp) * psir\% a(1) + (-k\% x(2)) * psir\% a(2) 
     c_psir1\%psi(3)\%a(4) = (-k\%x(2)) * psir\%a(1) + ((0,-1) * km) * psir\%a(2) 
    c_psir1\%psi(4)\%a(1) = (-ik2) * psir\%a(3) + km * psir\%a(4)
    c_psir1\%psi(4)\%a(2) = (-kp) * psir\%a(3) + (-ik2) * psir\%a(4)
    c_psir1\%psi(4)\%a(3) = (-ik2) * psir\%a(1) + (-kp) * psir\%a(2)
    c_psir1\%psi(4)\%a(4) = km * psir\%a(1) + (-ik2) * psir\%a(2)
    c_psir2\%psi(1)\%a(1) = (0,1) * (k\%x(1) * psir\%a(3) + km * psir\%a(4))
    c_psir2\%psi(1)\%a(2) = (0,-1) * (kp * psir\%a(3) + k\%x(1) * psir\%a(4))
    c_psir2\%psi(1)\%a(3) = (0,1) * ((-k\%x(1)) * psir\%a(1) + kp * psir\%a(2))
    c_psir2\%psi(1)\%a(4) = (0,1) * ((-km) * psir\%a(1) + k\%x(1) * psir\%a(2))
    c_psir2%psi(2)%a(1) = (0,1) * (km * psir%a(3) + k%x(1) * psir%a(4))
    c_psir2\%psi(2)\%a(2) = (0,-1) * (k\%x(1) * psir\%a(3) + kp * psir\%a(4))
    c_psir2\%psi(2)\%a(3) = (0,-1) * (kp * psir\%a(1) + (-k\%x(1)) * psir\%a(2))
     c_psir2\%psi(2)\%a(4) = (0,-1) * (k\%x(1) * psir\%a(1) + (-km) * psir\%a(2) ) 
    c_psir2\%psi(3)\%a(1) = (-km) * psir\%a(3) + k\%x(1) * psir\%a(4)
    c_psir2\%psi(3)\%a(2) = k\%x(1) * psir\%a(3) + (-kp) * psir\%a(4)
    c_psir2\%psi(3)\%a(3) = kp * psir\%a(1) + k\%x(1) * psir\%a(2)
    c_psir2\%psi(3)\%a(4) = k\%x(1) * psir\%a(1) + km * psir\%a(2)
    c_psir2\%psi(4)\%a(1) = (0,1) * (k\%x(1) * psir\%a(3) + (-km) * psir\%a(4))
    c_psir2\%psi(4)\%a(2) = (0,1) * ((-kp) * psir\%a(3) + k\%x(1) * psir\%a(4))
    c_psir2\%psi(4)\%a(3) = (0,1) * (k\%x(1) * psir\%a(1) + kp * psir\%a(2))
    c_psir2\%psi(4)\%a(4) = (0,1) * (km * psir\%a(1) + k\%x(1) * psir\%a(2))
    c_psir3\%psi(1)\%a(1) = (-k\%t) * psir\%a(3) - k12s * psir\%a(4)
    c_psir3\%psi(1)\%a(2) = k12 * psir\%a(3) + k\%t * psir\%a(4)
    c_psir3\%psi(1)\%a(3) = (-k\%t) * psir\%a(1) + k12s * psir\%a(2)
    c_psir3\%psi(1)\%a(4) = (-k12) * psir\%a(1) + k\%t * psir\%a(2)
    c_psir3\%psi(2)\%a(1) = (-k12s) * psir\%a(3) + (-k\%t) * psir\%a(4)
    c_psir3\%psi(2)\%a(2) = k\%t * psir\%a(3) + k12 * psir\%a(4)
    c_psir3\%psi(2)\%a(3) = (-k12s) * psir\%a(1) + k\%t * psir\%a(2)
    c_psir3\%psi(2)\%a(4) = (-k\%t) * psir\%a(1) + k12 * psir\%a(2)
    c_psir3%psi(3)%a(1) = (0,-1) * (k12s * psir%a(3) + (-k%t) * psir%a(4))
    c_psir3\%psi(3)\%a(2) = (0,1) * (k\%t * psir\%a(3) + (-k12) * psir\%a(4))
    c_psir3\%psi(3)\%a(3) = (0,-1) * (k12s * psir\%a(1) + k\%t * psir\%a(2))
    c_psir3\%psi(3)\%a(4) = (0,-1) * (k\%t * psir\%a(1) + k12 * psir\%a(2))
    c_psir3\%psi(4)\%a(1) = (-k\%t) * psir\%a(3) + k12s * psir\%a(4)
    c_psir3\%psi(4)\%a(2) = k12 * psir\%a(3) + (-k\%t) * psir\%a(4)
    c_psir3\%psi(4)\%a(3) = k\%t * psir\%a(1) + k12s * psir\%a(2)
    c_psir3\%psi(4)\%a(4) = k12 * psir\%a(1) + k\%t * psir\%a(2)
    j%t
              2 * (psil * c_psir0)
    j\%x(1) =
              2 * (psil * c_psir1)
               2 * (psil * c_psir2)
    j\%x(2) =
    j\%x(3) =
               2 * (psil * c_psir3)
  end function grkgggf
\langle Implementation\ of\ bispinor\ currents \rangle + \equiv
 pure function v_grf (g, psil, psir, k) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: psil
    type(bispinor), intent(in) :: psir
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    j = g * grkgggf (psil, psir, vk)
  end function v_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function vlr_grf (gl, gr, psil, psir, k) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr
```

```
type(vectorspinor), intent(in) :: psil
       type(bispinor), intent(in) :: psir
       type(bispinor) :: psir_l, psir_r
       type(momentum), intent(in) :: k
       type(vector) :: vk
       vk = k
       psir_1%a(1:2) = psir%a(1:2)
       psir_1\%a(3:4) = 0
       psir_r%a(1:2) = 0
       psir_r%a(3:4) = psir%a(3:4)
       j = gl * grkgggf (psil, psir_l, vk) + gr * grkgggf (psil, psir_r, vk)
   end function vlr_grf
V^{\mu} = \psi^T \tilde{C}^{\mu\rho} \psi_{\rho}; remember the reversed index order in C.
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
   pure function fggkggr (psil, psir, k) result (j)
       type(vector) :: j
       type(vectorspinor), intent(in) :: psir
       type(bispinor), intent(in) :: psil
       type(vector), intent(in) :: k
       type(bispinor) :: c_psir0, c_psir1, c_psir2, c_psir3
       \verb|complex(kind=default)| :: kp, km, k12, k12s, ik1, ik2|
       kp = k\%t + k\%x(3)
       km = k\%t - k\%x(3)
       k12 = k%x(1) + (0,1)*k%x(2)
       k12s = k\%x(1) - (0,1)*k\%x(2)
       ik1 = (0,1) * k%x(1)
       ik2 = (0,1) * k%x(2)
       c_psir0\%a(1) = k\%x(3) * (psir\%psi(1)\%a(4) + psir\%psi(4)\%a(4) &
                              + psir%psi(2)%a(3) + (0,1) * psir%psi(3)%a(3)) &
                              - k12 * (psir%psi(1)%a(3) + psir%psi(4)%a(3)) &
                              + k12s * (psir%psi(2)%a(4) + (0,1) * psir%psi(3)%a(4))
       c_psir0%a(2) = k%x(3) * (psir%psi(1)%a(3) - psir%psi(4)%a(3) + &
                                 psir%psi(2)%a(4) - (0,1) * psir%psi(3)%a(4)) + &
                                 k12s * (psir%psi(1)%a(4) - psir%psi(4)%a(4)) - &
                                 k12 * (psir%psi(2)%a(3) - (0,1) * psir%psi(3)%a(3))
       c_psir0\%a(3) = k\%x(3) * (-psir\%psi(1)\%a(2) + psir\%psi(4)\%a(2) + &
                                 psir%psi(2)%a(1) + (0,1) * psir%psi(3)%a(1)) + &
                                 k12 * (psir%psi(1)%a(1) - psir%psi(4)%a(1)) + &
                                 k12s * (psir%psi(2)%a(2) + (0,1) * psir%psi(3)%a(2))
       c_psir0\%a(4) = k\%x(3) * (-psir\%psi(1)\%a(1) - psir\%psi(4)\%a(1) + &
                                 psir%psi(2)%a(2) - (0,1) * psir%psi(3)%a(2)) - &
                                 k12s * (psir%psi(1)%a(2) + psir%psi(4)%a(2)) - &
                                 k12 * (psir%psi(2)%a(1) - (0,1) * psir%psi(3)%a(1))
       c_psir1\%a(1) = ik2 * (-psir\%psi(1)\%a(4) - psir\%psi(4)\%a(4) - &
                                 psir%psi(2)%a(3) - (0,1) * psir%psi(3)%a(3)) - &
                                 km * (psir%psi(1)%a(3) + psir%psi(4)%a(3)) + &
                                 kp * (psir%psi(2)%a(4) + (0,1) * psir%psi(3)%a(4))
       c_psir1\%a(2) = ik2 * (-psir\%psi(1)\%a(3) - psir\%psi(2)\%a(4) + &
                                 psir%psi(4)%a(3) + (0,1) * psir%psi(3)%a(4)) + &
                                 kp * (psir%psi(1)%a(4) - psir%psi(4)%a(4)) - &
                                 km * (psir%psi(2)%a(3) - (0,1) * psir%psi(3)%a(3))
       c_psir1\%a(3) = ik2 * (-psir\%psi(1)\%a(2) + psir\%psi(2)\%a(1) + &
                                 psir%psi(4)%a(2) + (0,1) * psir%psi(3)%a(1)) + &
                                 \label{eq:kp * (psir%psi(1)%a(1) - psir%psi(4)%a(1)) + & (psir%psi(4)%a(1)) + & (psir%psi
                                 km * (psir%psi(2)%a(2) + (0,1) * psir%psi(3)%a(2))
       c_psir1\%a(4) = ik2 * (-psir\%psi(1)\%a(1) + psir\%psi(2)\%a(2) - &
                                 psir%psi(4)%a(1) - (0,1) * psir%psi(3)%a(2)) - &
                                 km * (psir%psi(1)%a(2) + psir%psi(4)%a(2)) - &
                                 kp * (psir%psi(2)%a(1) - (0,1) * psir%psi(3)%a(1))
       !!!
       c_psir2\%a(1) = ik1 * (psir\%psi(2)\%a(3) + psir\%psi(1)\%a(4) &
                                 + psir%psi(4)%a(4) + (0,1) * psir%psi(3)%a(3)) - &
                                 ((0,1)*km) * (psir%psi(1)%a(3) + psir%psi(4)%a(3)) &
                                 + kp * (psir%psi(3)%a(4) - (0,1) * psir%psi(2)%a(4))
       c_psir2\%a(2) = ik1 * (psir\%psi(1)\%a(3) + psir\%psi(2)\%a(4) - &
```

```
psir%psi(4)%a(3) - (0,1) * psir%psi(3)%a(4)) - &
                    ((0,1)*kp) * (psir%psi(1)%a(4) - psir%psi(4)%a(4)) &
                    - km * (psir%psi(3)%a(3) + (0,1) * psir%psi(2)%a(3))
    c_psir2\%a(3) = ik1 * (psir\%psi(1)\%a(2) - psir\%psi(2)\%a(1) - \& 
                   psir%psi(4)%a(2) - (0,1) * psir%psi(3)%a(1)) + &
                    ((0,1)*kp) * (psir%psi(1)%a(1) - psir%psi(4)%a(1)) &
                    + km * (psir%psi(3)%a(2) - (0,1) * psir%psi(2)%a(2))
    c_psir2\%a(4) = ik1 * (psir\%psi(1)\%a(1) - psir\%psi(2)\%a(2) + \&
                   psir%psi(4)%a(1) + (0,1) * psir%psi(3)%a(2)) + &
                    ((0,1)*km) * (psir%psi(1)%a(2) + psir%psi(4)%a(2)) - &
                   kp * (psir%psi(3)%a(1) + (0,1) * psir%psi(2)%a(1))
    111
    c_psir3\%a(1) = k\%t * (psir\%psi(1)\%a(4) + psir\%psi(4)\%a(4) + &
                   psir%psi(2)%a(3) + (0,1) * psir%psi(3)%a(3)) - &
                   k12 * (psir%psi(1)%a(3) + psir%psi(4)%a(3)) - &
                   k12s * (psir%psi(2)%a(4) + (0,1) * psir%psi(3)%a(4))
    c_psir3\%a(2) = k\%t * (psir\%psi(1)\%a(3) - psir\%psi(4)\%a(3) + \&
                   psir%psi(2)%a(4) - (0,1) * psir%psi(3)%a(4)) - &
                   k12s * (psir%psi(1)%a(4) - psir%psi(4)%a(4)) - &
                   k12 * (psir%psi(2)%a(3) - (0,1) * psir%psi(3)%a(3))
    c_psir3\%a(3) = k\%t * (-psir\%psi(1)\%a(2) + psir\%psi(2)\%a(1) + &
                   psir%psi(4)%a(2) + (0,1) * psir%psi(3)%a(1)) - &
                   k12 * (psir%psi(1)%a(1) - psir%psi(4)%a(1)) + &
                   k12s * (psir%psi(2)%a(2) + (0,1) * psir%psi(3)%a(2))
    c_psir3\%a(4) = k\%t * (-psir\%psi(1)\%a(1) + psir\%psi(2)\%a(2) - &
                   psir%psi(4)%a(1) - (0,1) * psir%psi(3)%a(2)) - &
                   k12s * (psir%psi(1)%a(2) + psir%psi(4)%a(2)) + &
                   k12 * (psir%psi(2)%a(1) - (0,1) * psir%psi(3)%a(1))
    !!! Because we explicitly multiplied the charge conjugation matrix
    !!! we have to omit it from the spinor product and take the
    !!! ordinary product!
               2 * dot_product (conjg (psil%a), c_psir0%a)
    j\%x(1) =
               2 * dot_product (conjg (psil%a), c_psir1%a)
               2 * dot_product (conjg (psil%a), c_psir2%a)
    j%x(3) = 2 * dot_product (conjg (psil%a), c_psir3%a)
  end function fggkggr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function v_fgr (g, psil, psir, k) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: psir
    type(bispinor), intent(in) :: psil
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    j = g * fggkggr (psil, psir, vk)
  end function v_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function vlr_fgr (gl, gr, psil, psir, k) result (j)
    type(vector) :: j
    {\tt complex(kind=default),\ intent(in)\ ::\ gl,\ gr}
    {\tt type(vectorspinor),\ intent(in)\ ::\ psir}
    type(bispinor), intent(in) :: psil
    type(bispinor) :: psil_l
    type(bispinor) :: psil_r
    type(momentum), intent(in) :: k
    type(vector) :: vk
    vk = k
    psil_1\%a(1:2) = psil\%a(1:2)
    psil_1%a(3:4) = 0
    psil_r\%a(1:2) = 0
    psil_r%a(3:4) = psil%a(3:4)
    j = gl * fggkggr (psil_l, psir, vk) + gr * fggkggr (psil_r, psir, vk)
  end function vlr_fgr
```

Y.26.5 Gravitino 4-Couplings

```
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
 public :: f_s2gr, f_svgr, f_slvgr, f_srvgr, f_slrvgr, f_pvgr, f_v2gr, f_v2lrgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_s2gr (g, phi1, phi2, psi) result (phipsi)
    type(bispinor) :: phipsi
    type(vectorspinor), intent(in) :: psi
    complex(kind=default), intent(in) :: g
    {\tt complex(kind=default),\ intent(in)\ ::\ phi1,\ phi2}
    phipsi = phi2 * f_potgr (g, phi1, psi)
  end function f_s2gr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_svgr (g, phi, v, grav) result (phigrav)
    type(bispinor) :: phigrav
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: g, phi
    phigrav = (g * phi) * fgvg5gr (grav, v)
  end function f_svgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_slvgr (gl, phi, v, grav) result (phigrav)
    type(bispinor) :: phigrav, phidum
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: gl, phi
    phidum = (gl * phi) * fgvg5gr (grav, v)
    phigrav%a(1:2) = phidum%a(1:2)
    phigrav%a(3:4) = 0
  end function f_slvgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_srvgr (gr, phi, v, grav) result (phigrav)
    type(bispinor) :: phigrav, phidum
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: gr, phi
    phidum = (gr * phi) * fgvg5gr (grav, v)
    phigrav%a(1:2) = 0
    phigrav%a(3:4) = phidum%a(3:4)
  end function f_srvgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_slrvgr (gl, gr, phi, v, grav) result (phigrav)
    type(bispinor) :: phigrav
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: gl, gr, phi
    phigrav = f_slvgr (gl, phi, v, grav) + f_srvgr (gr, phi, v, grav)
  end function f_slrvgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_pvgr (g, phi, v, grav) result (phigrav)
    type(bispinor) :: phigrav
    type(vectorspinor), intent(in) :: grav
    {\tt type(vector),\ intent(in) :: \ v}
    complex(kind=default), intent(in) :: g, phi
    phigrav = (g * phi) * fgvgr (grav, v)
  end function f_pvgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function f_v2gr (g, v1, v2, grav) result (psi)
    type(bispinor) :: psi
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v1, v2
    psi = g * fggvvgr (v2, grav, v1)
  end function f_v2gr
```

```
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function f_v2lrgr (gl, gr, v1, v2, grav) result (psi)
    type(bispinor) :: psi
    complex(kind=default), intent(in) :: gl, gr
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v1, v2
    psi = fggvvgr (v2, grav, v1)
    psi\%a(1:2) = gl * psi\%a(1:2)
    psi\%a(3:4) = gr * psi\%a(3:4)
  end function f_v2lrgr
\langle Declaration \ of \ bispinor \ currents \rangle + \equiv
  public :: gr_s2f, gr_svf, gr_pvf, gr_slvf, gr_srvf, gr_slrvf, gr_v2f, gr_v2lrf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function gr_s2f (g, phi1, phi2, psi) result (phipsi)
    type(vectorspinor) :: phipsi
    type(bispinor), intent(in) :: psi
    complex(kind=default), intent(in) :: g
    complex(kind=default), intent(in) :: phi1, phi2
    phipsi = phi2 * gr_potf (g, phi1, psi)
  end function gr_s2f
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function gr_svf (g, phi, v, psi) result (phipsi)
    type(vectorspinor) :: phipsi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: g, phi
    phipsi = (g * phi) * grkggf (psi, v)
  end function gr_svf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function gr_slvf (gl, phi, v, psi) result (phipsi)
    type(vectorspinor) :: phipsi
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: gl, phi
    psi_1\%a(1:2) = psi\%a(1:2)
    psi_1\%a(3:4) = 0
    phipsi = (gl * phi) * grkggf (psi_l, v)
  end function gr_slvf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function gr_srvf (gr, phi, v, psi) result (phipsi)
    type(vectorspinor) :: phipsi
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_r
    type(vector), intent(in) :: v
    complex(kind=default), intent(in) :: gr, phi
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    phipsi = (gr * phi) * grkggf (psi_r, v)
  end function gr_srvf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function gr_slrvf (gl, gr, phi, v, psi) result (phipsi)
    type(vectorspinor) :: phipsi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    {\tt complex(kind=default),\ intent(in)\ ::\ gl,\ gr,\ phi}
    phipsi = gr_slvf (gl, phi, v, psi) + gr_srvf (gr, phi, v, psi)
  end function gr_slrvf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function gr_pvf (g, phi, v, psi) result (phipsi)
    type(vectorspinor) :: phipsi
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
```

```
complex(kind=default), intent(in) :: g, phi
    phipsi = (g * phi) * grkgf (psi, v)
  end function gr_pvf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function gr_v2f (g, v1, v2, psi) result (vvpsi)
    type(vectorspinor) :: vvpsi
    complex(kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v1, v2
    vvpsi = g * grkkggf (v2, psi, v1)
  end function gr_v2f
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function gr_v2lrf (gl, gr, v1, v2, psi) result (vvpsi)
    type(vectorspinor) :: vvpsi
    complex(kind=default), intent(in) :: gl, gr
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l, psi_r
    type(vector), intent(in) :: v1, v2
    psi_1\%a(1:2) = psi\%a(1:2)
    psi_1\%a(3:4) = 0
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    vvpsi = gl * grkkggf (v2, psi_l, v1) + gr * grkkggf (v2, psi_r, v1)
  end function gr_v2lrf
\langle Declaration\ of\ bispinor\ currents \rangle + \equiv
 public :: s2\_grf, s2\_fgr, sv1\_grf, sv2\_grf, sv1\_fgr, sv2\_fgr, \& \\
             slv1_grf, slv2_grf, slv1_fgr, slv2_fgr, &
             srv1_grf, srv2_grf, srv1_fgr, srv2_fgr, &
             slrv1_grf, slrv2_grf, slrv1_fgr, slrv2_fgr, &
             pv1_grf, pv2_grf, pv1_fgr, pv2_fgr, v2_grf, v2_fgr, &
             v2lr_grf, v2lr_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function s2_grf (g, gravbar, phi, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g, phi
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    j = phi * pot_grf (g, gravbar, psi)
  end function s2_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function s2_fgr (g, psibar, phi, grav) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g, phi
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    j = phi * pot_fgr (g, psibar, grav)
  end function s2_fgr
\langle Implementation\ of\ bispinor\ currents \rangle + \equiv
 pure function sv1\_grf (g, gravbar, v, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    j = g * grg5vgf (gravbar, psi, v)
  end function sv1_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function slv1_grf (gl, gravbar, v, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l
```

```
type(vector), intent(in) :: v
    psi_1\%a(1:2) = psi\%a(1:2)
    psi_1\%a(3:4) = 0
    j = gl * grg5vgf (gravbar, psi_l, v)
  end function slv1_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function srv1_grf (gr, gravbar, v, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gr
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_r
    type(vector), intent(in) :: v
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    j = gr * grg5vgf (gravbar, psi_r, v)
  \verb"end function srv1_grf"
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function slrv1_grf (gl, gr, gravbar, v, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l, psi_r
    type(vector), intent(in) :: v
    psi_1\%a(1:2) = psi\%a(1:2)
    psi_1\%a(3:4) = 0
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    j = gl * grg5vgf (gravbar, psi_l, v) + gr * grg5vgf (gravbar, psi_r, v)
  end function slrv1_grf
```

$$C\gamma^{0}\gamma^{0} = -C\gamma^{1}\gamma^{1} = -C\gamma^{2}\gamma^{2} = C\gamma^{3}\gamma^{3} = C = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$
(Y.129a)

$$C\gamma^{0}\gamma^{1} = -C\gamma^{1}\gamma^{0} = \begin{pmatrix} -1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & -1 & 0\\ 0 & 0 & 0 & 1 \end{pmatrix}$$
(Y.129b)

$$C\gamma^{0}\gamma^{2} = -C\gamma^{2}\gamma^{0} = \begin{pmatrix} -i & 0 & 0 & 0\\ 0 & -i & 0 & 0\\ 0 & 0 & -i & 0\\ 0 & 0 & 0 & -i \end{pmatrix}$$
(Y.129c)

$$C\gamma^{0}\gamma^{3} = -C\gamma^{3}\gamma^{0} = \begin{pmatrix} 0 & 1 & 0 & 0\\ 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0 \end{pmatrix}$$
 (Y.129d)

$$C\gamma^{1}\gamma^{2} = -C\gamma^{2}\gamma^{1} = \begin{pmatrix} 0 & i & 0 & 0\\ i & 0 & 0 & 0\\ 0 & 0 & 0 & -i\\ 0 & 0 & -i & 0 \end{pmatrix}$$
(Y.129e)

$$C\gamma^{1}\gamma^{3} = -C\gamma^{3}\gamma^{1} = \begin{pmatrix} -1 & 0 & 0 & 0\\ 0 & -1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$C\gamma^{2}\gamma^{3} = -C\gamma^{3}\gamma^{2} = \begin{pmatrix} -i & 0 & 0 & 0\\ 0 & i & 0 & 0\\ 0 & 0 & i & 0\\ 0 & 0 & 0 & -i \end{pmatrix}$$

$$(Y.129f)$$

$$C\gamma^2\gamma^3 = -C\gamma^3\gamma^2 = \begin{pmatrix} -i & 0 & 0 & 0\\ 0 & i & 0 & 0\\ 0 & 0 & i & 0\\ 0 & 0 & 0 & -i \end{pmatrix}$$
 (Y.129g)

```
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function sv2_grf (g, gravbar, phi, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: g, phi
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(vectorspinor) :: g0_psi, g1_psi, g2_psi, g3_psi
    g0_psi%psi(1)%a(1:2) = - psi%a(1:2)
    g0_psi\%psi(1)\%a(3:4) = psi\%a(3:4)
    g0_psi\%psi(2)\%a(1) = psi\%a(2)
    g0_psi\%psi(2)\%a(2) = psi\%a(1)
    g0_psi\%psi(2)\%a(3) = psi\%a(4)
    g0_psi\%psi(2)\%a(4) = psi\%a(3)
    g0_psi\%psi(3)\%a(1) = (0,-1) * psi\%a(2)
    g0_psi\%psi(3)\%a(2) = (0,1) * psi\%a(1)
    g0_psi\%psi(3)\%a(3) = (0,-1) * psi\%a(4)
    g0_psi\%psi(3)\%a(4) = (0,1) * psi\%a(3)
    g0_psi%psi(4)%a(1) = psi%a(1)
    g0_psi\%psi(4)\%a(2) = - psi\%a(2)
    g0_psi\%psi(4)\%a(3) = psi\%a(3)
    g0_psi\%psi(4)\%a(4) = - psi\%a(4)
    g1_psi\%psi(1)\%a(1:4) = - g0_psi\%psi(2)\%a(1:4)
    g1_psi%psi(2)%a(1:4) = - g0_psi%psi(1)%a(1:4)
    g1_psi\%psi(3)\%a(1) = (0,1) * psi\%a(1)
    g1_psi\%psi(3)\%a(2) = (0,-1) * psi\%a(2)
    g1_psi\%psi(3)\%a(3) = (0,-1) * psi\%a(3)
    g1_psi\%psi(3)\%a(4) = (0,1) * psi\%a(4)
    g1_psi\%psi(4)\%a(1) = - psi\%a(2)
    g1_psi\%psi(4)\%a(2) = psi\%a(1)
    g1_psi\%psi(4)\%a(3) = psi\%a(4)
    g1_psi%psi(4)%a(4) = - psi%a(3)
    g2_psi\%psi(1)\%a(1:4) = - g0_psi\%psi(3)\%a(1:4)
    g2_psi\%psi(2)\%a(1:4) = - g1_psi\%psi(3)\%a(1:4)
    g2_psi\%psi(3)\%a(1:4) = - g0_psi\%psi(1)\%a(1:4)
    g2_psi\%psi(4)\%a(1) = (0,1) * psi\%a(2)
    g2_psi\%psi(4)\%a(2) = (0,1) * psi\%a(1)
    g2_psi\%psi(4)\%a(3) = (0,-1) * psi\%a(4)
    g2_psi\%psi(4)\%a(4) = (0,-1) * psi\%a(3)
    g3_psi\%psi(1)\%a(1:4) = - g0_psi\%psi(4)\%a(1:4)
    g3_psi%psi(2)%a(1:4) = - g1_psi%psi(4)%a(1:4)
    g3_psi\%psi(3)\%a(1:4) = - g2_psi\%psi(4)\%a(1:4)
    g3_psi%psi(4)%a(1:4) = - g0_psi%psi(1)%a(1:4)
               (g * phi) * (gravbar * g0_psi)
    j\%x(1) =
               (g * phi) * (gravbar * g1_psi)
    j\%x(2) =
               (g * phi) * (gravbar * g2_psi)
    j\%x(3) =
               (g * phi) * (gravbar * g3_psi)
 end function sv2_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function slv2_grf (gl, gravbar, phi, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, phi
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l
    psi_1\%a(1:2) = psi\%a(1:2)
    psi_1\%a(3:4) = 0
    j = sv2_grf (gl, gravbar, phi, psi_l)
 end function slv2_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function srv2_grf (gr, gravbar, phi, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gr, phi
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_r
```

```
psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    j = sv2_grf (gr, gravbar, phi, psi_r)
  end function srv2_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function slrv2_grf (gl, gr, gravbar, phi, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr, phi
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l, psi_r
    psi_1%a(1:2) = psi%a(1:2)
    psi_1\%a(3:4) = 0
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    j = sv2_grf (gl, gravbar, phi, psi_l) + sv2_grf (gr, gravbar, phi, psi_r)
  end function slrv2_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function sv1_fgr (g, psibar, v, grav) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    j = g * fg5gkgr (psibar, grav, v)
  end function sv1_fgr
\langle Implementation of bispinor currents \rangle + \equiv
 pure function slv1_fgr (gl, psibar, v, grav) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_l
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    psibar_1\%a(1:2) = psibar\%a(1:2)
    psibar_1\%a(3:4) = 0
    j = gl * fg5gkgr (psibar_l, grav, v)
  end function slv1_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function srv1_fgr (gr, psibar, v, grav) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gr
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_r
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    psibar_r%a(1:2) = 0
    psibar_r%a(3:4) = psibar%a(3:4)
    j = gr * fg5gkgr (psibar_r, grav, v)
  end function srv1_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function slrv1_fgr (gl, gr, psibar, v, grav) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_1, psibar_r
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    psibar_1%a(1:2) = psibar%a(1:2)
    psibar_1\%a(3:4) = 0
    psibar_r%a(1:2) = 0
    psibar_r\%a(3:4) = psibar\%a(3:4)
    j = gl * fg5gkgr (psibar_l, grav, v) + gr * fg5gkgr (psibar_r, grav, v)
  end function slrv1_fgr
```

```
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function sv2_fgr (g, psibar, phi, grav) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: g, phi
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    {\tt type(bispinor)} \ :: \ {\tt g0\_grav}, \ {\tt g1\_grav}, \ {\tt g2\_grav}, \ {\tt g3\_grav}
    g0_grav%a(1) = -grav%psi(1)%a(1) + grav%psi(2)%a(2) - &
                   (0,1) * grav%psi(3)%a(2) + grav%psi(4)%a(1)
    g0_grav%a(2) = -grav%psi(1)%a(2) + grav%psi(2)%a(1) + &
                   (0,1) * grav%psi(3)%a(1) - grav%psi(4)%a(2)
    g0_grav\%a(3) = grav\%psi(1)\%a(3) + grav\%psi(2)\%a(4) - &
                   (0,1) * grav%psi(3)%a(4) + grav%psi(4)%a(3)
    g0_grav%a(4) = grav%psi(1)%a(4) + grav%psi(2)%a(3) + &
                   (0,1) * grav%psi(3)%a(3) - grav%psi(4)%a(4)
    111
    g1_grav%a(1) = grav%psi(1)%a(2) - grav%psi(2)%a(1) + &
                   (0,1) * grav%psi(3)%a(1) - grav%psi(4)%a(2)
    g1_grav%a(2) = grav%psi(1)%a(1) - grav%psi(2)%a(2) - &
                   (0,1) * grav%psi(3)%a(2) + grav%psi(4)%a(1)
    g1_grav%a(3) = grav%psi(1)%a(4) + grav%psi(2)%a(3) - &
                   (0,1) * grav%psi(3)%a(3) + grav%psi(4)%a(4)
    g1_grav%a(4) = grav%psi(1)%a(3) + grav%psi(2)%a(4) + &
                   (0,1) * grav%psi(3)%a(4) - grav%psi(4)%a(3)
    111
    g2_grav\%a(1) = (0,1) * (-grav\%psi(1)\%a(2) - grav\%psi(2)\%a(1) + &
                   grav%psi(4)%a(2)) - grav%psi(3)%a(1)
    g2_grav\%a(2) = (0,1) * (grav\%psi(1)\%a(1) + grav\%psi(2)\%a(2) + &
                   grav%psi(4)%a(1)) - grav%psi(3)%a(2)
    g2_grav\%a(3) = (0,1) * (-grav\%psi(1)\%a(4) + grav\%psi(2)\%a(3) - &
                   grav%psi(4)%a(4)) + grav%psi(3)%a(3)
    g2_grav%a(4) = (0,1) * (grav%psi(1)%a(3) - grav%psi(2)%a(4) - &
                   grav%psi(4)%a(3)) + grav%psi(3)%a(4)
    g3_grav%a(1) = -grav%psi(1)%a(2) + grav%psi(2)%a(2) - &
                   (0,1) * grav%psi(3)%a(2) - grav%psi(4)%a(1)
    g3_grav%a(2) = grav%psi(1)%a(1) - grav%psi(2)%a(1) - &
                   (0,1) * grav%psi(3)%a(1) - grav%psi(4)%a(2)
    g3_grav%a(3) = -grav%psi(1)%a(2) - grav%psi(2)%a(4) + &
                   (0,1) * grav%psi(3)%a(4) + grav%psi(4)%a(3)
    g3_grav%a(4) = -grav%psi(1)%a(4) + grav%psi(2)%a(3) + &
                   (0,1) * grav%psi(3)%a(3) + grav%psi(4)%a(4)
    j%t
                (g * phi) * (psibar * g0_grav)
    j\%x(1) =
                (g * phi) * (psibar * g1_grav)
    j\%x(2) =
               (g * phi) * (psibar * g2_grav)
    j\%x(3) =
               (g * phi) * (psibar * g3_grav)
  end function sv2_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function slv2_fgr (gl, psibar, phi, grav) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, phi
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_l
    type(vectorspinor), intent(in) :: grav
    psibar_1%a(1:2) = psibar%a(1:2)
    psibar_1\%a(3:4) = 0
    j = sv2_fgr (gl, psibar_l, phi, grav)
  end function slv2_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function srv2_fgr (gr, psibar, phi, grav) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gr, phi
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_r
    type(vectorspinor), intent(in) :: grav
```

```
psibar_r%a(1:2) = 0
    psibar_r%a(3:4) = psibar%a(3:4)
    j = sv2_fgr (gr, psibar_r, phi, grav)
  end function srv2_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function slrv2_fgr (gl, gr, psibar, phi, grav) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr, phi
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_l, psibar_r
    type(vectorspinor), intent(in) :: grav
    psibar_1\%a(1:2) = psibar\%a(1:2)
    psibar_1\%a(3:4) = 0
    psibar_r%a(1:2) = 0
    psibar_r\%a(3:4) = psibar\%a(3:4)
    j = sv2_fgr (gl, psibar_l, phi, grav) + sv2_fgr (gr, psibar_r, phi, grav)
  end function slrv2_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function pv1_grf (g, gravbar, v, psi) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(vector), intent(in) :: v
    j = g * grvgf (gravbar, psi, v)
  end function pv1_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function pv2_grf (g, gravbar, phi, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: g, phi
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
    type(bispinor) :: g5_psi
    g5_psi\%a(1:2) = - psi\%a(1:2)
    g5_psi%a(3:4) = psi%a(3:4)
    j = sv2_grf (g, gravbar, phi, g5_psi)
  end function pv2_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
  pure function pv1_fgr (g, psibar, v, grav) result (j)
    complex(kind=default) :: j
    complex(kind=default), intent(in) :: g
    type(bispinor), intent(in) :: psibar
    type(vectorspinor), intent(in) :: grav
    type(vector), intent(in) :: v
    j = g * fgkgr (psibar, grav, v)
  end function pv1_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function pv2_fgr (g, psibar, phi, grav) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: g, phi
    type(vectorspinor), intent(in) :: grav
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_g5
    psibar_g5\%a(1:2) = - psibar\%a(1:2)
    psibar_g5\%a(3:4) = psibar\%a(3:4)
    j = sv2_fgr (g, psibar_g5, phi, grav)
  end function pv2_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function v2_grf (g, gravbar, v, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: gravbar
    type(bispinor), intent(in) :: psi
```

```
type(vector), intent(in) :: v
    j = -g * grkgggf (gravbar, psi, v)
 end function v2_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function v2lr_grf (gl, gr, gravbar, v, psi) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr
    {\tt type(vectorspinor),\ intent(in)\ ::\ gravbar}
    type(bispinor), intent(in) :: psi
    type(bispinor) :: psi_l, psi_r
    type(vector), intent(in) :: v
    psi_1\%a(1:2) = psi\%a(1:2)
    psi_1\%a(3:4) = 0
    psi_r%a(1:2) = 0
    psi_r%a(3:4) = psi%a(3:4)
    j = -(gl * grkgggf (gravbar, psi_l, v) + gr * grkgggf (gravbar, psi_r, v))
  end function v2lr_grf
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function v2_fgr (g, psibar, v, grav) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: g
    type(vectorspinor), intent(in) :: grav
    type(bispinor), intent(in) :: psibar
    type(vector), intent(in) :: v
    j = -g * fggkggr (psibar, grav, v)
  end function v2_fgr
\langle Implementation \ of \ bispinor \ currents \rangle + \equiv
 pure function v2lr_fgr (gl, gr, psibar, v, grav) result (j)
    type(vector) :: j
    complex(kind=default), intent(in) :: gl, gr
    type(vectorspinor), intent(in) :: grav
    type(bispinor), intent(in) :: psibar
    type(bispinor) :: psibar_1, psibar_r
    type(vector), intent(in) :: v
    psibar_1%a(1:2) = psibar%a(1:2)
    psibar_1\%a(3:4) = 0
    psibar_r%a(1:2) = 0
    psibar_r%a(3:4) = psibar%a(3:4)
    j = -(gl * fggkggr (psibar_l, grav, v) + gr * fggkggr (psibar_r, grav, v))
  end function v2lr_fgr
```

Y.26.6 On Shell Wave Functions

⟨Declaration of bispinor on shell wave functions⟩≡
public :: u, v, ghost

$$\chi_{+}(\vec{p}) = \frac{1}{\sqrt{2|\vec{p}|(|\vec{p}| + p_3)}} \begin{pmatrix} |\vec{p}| + p_3 \\ p_1 + ip_2 \end{pmatrix}$$
(Y.130a)

$$\chi_{-}(\vec{p}) = \frac{1}{\sqrt{2|\vec{p}|(|\vec{p}| + p_3)}} \begin{pmatrix} -p_1 + ip_2 \\ |\vec{p}| + p_3 \end{pmatrix}$$
(Y.130b)

$$u_{\pm}(p) = \begin{pmatrix} \sqrt{p_0 \mp |\vec{p}|} \cdot \chi_{\pm}(\vec{p}) \\ \sqrt{p_0 \pm |\vec{p}|} \cdot \chi_{\pm}(\vec{p}) \end{pmatrix}$$
(Y.131)

```
⟨Implementation of bispinor on shell wave functions⟩≡
pure function u (mass, p, s) result (psi)
  type(bispinor) :: psi
  real(kind=default), intent(in) :: mass
  type(momentum), intent(in) :: p
  integer, intent(in) :: s
  complex(kind=default), dimension(2) :: chip, chim
  real(kind=default) :: pabs, norm, delta, m
```

```
m = abs(mass)
    pabs = sqrt (dot_product (p%x, p%x))
    if (m < epsilon (m) * pabs) then
        delta = 0
    else
        delta = sqrt (max (p%t - pabs, 0._default))
    end if
    if (pabs + p%x(3) \le 1000 * epsilon (pabs) * pabs) then
       chip = (/ cmplx ( 0.0, 0.0, kind=default), &
                   cmplx ( 1.0, 0.0, kind=default) /)
       chim = (/ cmplx (-1.0, 0.0, kind=default), &
                   cmplx ( 0.0, 0.0, kind=default) /)
    else
       norm = 1 / sqrt (2*pabs*(pabs + p%x(3)))
       chip = norm * (/ cmplx (pabs + p%x(3), kind=default), &
                          cmplx (p%x(1), p%x(2), kind=default) /)
       chim = norm * (/ cmplx (-p%x(1), p%x(2), kind=default), &
                           cmplx (pabs + p%x(3), kind=default) /)
    end if
    if (s > 0) then
       psi\%a(1:2) = delta * chip
       psi\%a(3:4) = sqrt (p\%t + pabs) * chip
    else
       psi\%a(1:2) = sqrt (p\%t + pabs) * chim
       psi\%a(3:4) = delta * chim
    end if
    pabs = m ! make the compiler happy and use m
    if (mass < 0) then
       psi\%a(1:2) = - imago * psi\%a(1:2)
       psi\%a(3:4) = + imago * psi\%a(3:4)
    end if
  end function u
                                        v_{\pm}(p) = \begin{pmatrix} \mp \sqrt{p_0 \pm |\vec{p}|} \cdot \chi_{\mp}(\vec{p}) \\ \pm \sqrt{p_0 \mp |\vec{p}|} \cdot \chi_{\mp}(\vec{p}) \end{pmatrix}
                                                                                                           (Y.132)
\langle Implementation \ of \ bispinor \ on \ shell \ wave \ functions \rangle + \equiv
 pure function v (mass, p, s) result (psi)
    type(bispinor) :: psi
    real(kind=default), intent(in) :: mass
    type(momentum), intent(in) :: p
    integer, intent(in) :: s
    complex(kind=default), dimension(2) :: chip, chim
    real(kind=default) :: pabs, norm, delta, m
    pabs = sqrt (dot_product (p%x, p%x))
    m = abs(mass)
    if (m < epsilon (m) * pabs) then
        delta = 0
    else
        delta = sqrt (max (p%t - pabs, 0._default))
    end if
    if (pabs + p%x(3) \le 1000 * epsilon (pabs) * pabs) then
       chip = (/ cmplx ( 0.0, 0.0, kind=default), &
                   cmplx ( 1.0, 0.0, kind=default) /)
       chim = (/ cmplx (-1.0, 0.0, kind=default), &
                   cmplx ( 0.0, 0.0, kind=default) /)
    else
       norm = 1 / sqrt (2*pabs*(pabs + p%x(3)))
       chip = norm * (/ cmplx (pabs + p%x(3), kind=default), &
                          cmplx (p%x(1), p%x(2), kind=default) /)
       \label{eq:chim} \mbox{chim = norm * (/ cmplx (-p%x(1), p%x(2), kind=default), \& }
                          cmplx (pabs + p%x(3), kind=default) /)
    end if
    if (s > 0) then
       psi\%a(1:2) = - sqrt (p\%t + pabs) * chim
       psi\%a(3:4) = delta * chim
```

```
else
       psi\%a(1:2) = delta * chip
       psi\%a(3:4) = - sqrt (p\%t + pabs) * chip
    end if
    pabs = m ! make the compiler happy and use m
    if (mass < 0) then
       psi\%a(1:2) = - imago * psi\%a(1:2)
       psi\%a(3:4) = + imago * psi\%a(3:4)
    end if
  end function v
\langle Implementation \ of \ bispinor \ on \ shell \ wave \ functions \rangle + \equiv
 pure function ghost (m, p, s) result (psi)
      type(bispinor) :: psi
      real(kind=default), intent(in) :: m
      type(momentum), intent(in) :: p
      integer, intent(in) :: s
      psi\%a(:) = 0
      select case (s)
      case (1)
         psi%a(1)
         psi\%a(2:4) = 0
      case (2)
         psi%a(1)
                    = 0
         psi%a(2)
                    = 1
         psi\%a(3:4) = 0
      case (3)
         psi%a(1:2) = 0
                   = 1
         psi%a(3)
         psi%a(4)
                    = 0
      case (4)
         psi%a(1:3) = 0
         psi%a(4)
      case (5)
         psi%a(1) =
         psi\%a(2) = - 2.3
         psi\%a(3) = -71.5
         psi\%a(4) =
                        0.1
      end select
 end function ghost
```

Y.26.7 Off Shell Wave Functions

This is the same as for the Dirac fermions except that the expressions for [ubar] and [vbar] are missing.

```
⟨Declaration of bispinor off shell wave functions⟩≡
public :: brs_u, brs_v
```

In momentum space we have:

$$brsu(p) = (-i)(\not p - m)u(p) \tag{Y.133}$$

```
\langle Implementation\ of\ bispinor\ off\ shell\ wave\ functions\rangle\equiv
 pure\ function\ brs\_u\ (m,\ p,\ s)\ result\ (dpsi)
 type(bispinor)\ ::\ dpsi,\ psi
 real(kind=default),\ intent(in)\ ::\ m
 type(momentum),\ intent(in)\ ::\ p
 integer,\ intent(in)\ ::\ s
 type\ (vector)::vp
 complex(kind=default),\ parameter\ ::\ one\ =\ (1,\ 0)
 vp=p
 psi=u(m,p,s)
 dpsi=cmplx(0.0,-1.0)*(f_vf(one,vp,psi)-m*psi)
 end\ function\ brs\_u
 brsv(p)=i(p\!\!/+m)v(p) 
 (Y.134)
```

⟨Implementation of bispinor off shell wave functions⟩+≡
pure function brs_v (m, p, s) result (dpsi)

```
type(bispinor) :: dpsi, psi
       real(kind=default), intent(in) :: m
       type(momentum), intent(in) :: p
       integer, intent(in) ::
       type (vector)::vp
       complex(kind=default), parameter :: one = (1, 0)
      vp=p
      psi=v(m,p,s)
      dpsi=cmplx(0.0,1.0)*(f_vf(one,vp,psi)+m*psi)
  end function brs_v
                                             Y.26.8 Propagators
\langle Declaration \ of \ bispinor \ propagators \rangle \equiv
  public :: pr_psi, pr_grav
  public :: pj_psi, pg_psi
                                                \frac{\mathrm{i}(-\not\!p+m)}{p^2-m^2+\mathrm{i}m\Gamma}\psi
                                                                                                           (Y.135)
NB: the sign of the momentum comes about because all momenta are treated as outgoing and the particle
charge flow is therefore opposite to the momentum.
\langle Implementation \ of \ bispinor \ propagators \rangle \equiv
  pure function pr_psi (p, m, w, cms, psi) result (ppsi)
    type(bispinor) :: ppsi
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(bispinor), intent(in) :: psi
    logical, intent(in) :: cms
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
    complex(kind=default) :: num_mass
    vp = p
    if (cms) then
        num_mass = sqrt(cmplx(m**2, -m*w, kind=default))
    else
       num_mass = cmplx (m, 0, kind=default)
    end if
    ppsi = (1 / cmplx (p*p - m**2, m*w, kind=default)) &
          * (- f_vf (one, vp, psi) + num_mass * psi)
  end function pr_psi
                                                \sqrt{\frac{\pi}{M\Gamma}}(-\not p+m)\psi
                                                                                                           (Y.136)
\langle Implementation \ of \ bispinor \ propagators \rangle + \equiv
  pure function pj_psi (p, m, w, psi) result (ppsi)
    type(bispinor) :: ppsi
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(bispinor), intent(in) :: psi
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
    vp = p
    ppsi = (0, -1) * sqrt (PI / m / w) * (- f_vf (one, vp, psi) + m * psi)
  end function pj_psi
\langle Implementation\ of\ bispinor\ propagators \rangle + \equiv
  pure function pg_psi (p, m, w, psi) result (ppsi)
    type(bispinor) :: ppsi
    type(momentum), intent(in) :: p
    real(kind=default), intent(in) :: m, w
    type(bispinor), intent(in) :: psi
    type(vector) :: vp
    complex(kind=default), parameter :: one = (1, 0)
    vp = p
```

ppsi = gauss (p*p, m, w) * (- f_vf (one, vp, psi) + m * psi) end function pg_psi

$$\frac{\mathrm{i}\left\{\left(-\not p+m\right)\left(-\eta_{\mu\nu}+\frac{p_{\mu}p_{\nu}}{m^{2}}\right)+\frac{1}{3}\left(\gamma_{\mu}-\frac{p_{\mu}}{m}\right)\left(\not p+m\right)\left(\gamma_{\nu}-\frac{p_{\nu}}{m}\right)\right\}}{p^{2}-m^{2}+\mathrm{i}m\Gamma}\psi^{\nu}$$
(Y.137)

```
\langle Implementation \ of \ bispinor \ propagators \rangle + \equiv
 pure function pr_grav (p, m, w, grav) result (propgrav)
   type(vectorspinor) :: propgrav
   type(momentum), intent(in) :: p
   real(kind=default), intent(in) :: m, w
    type(vectorspinor), intent(in) :: grav
   type(vector) :: vp
   type(bispinor) :: pgrav, ggrav1, ggrav2, ppgrav
    type(vectorspinor) :: etagrav_dum, etagrav, pppgrav, &
                          gg_grav_dum, gg_grav
   complex(kind=default), parameter :: one = (1, 0)
   real(kind=default) :: minv
   integer :: i
   vp = p
   minv = 1/m
   pgrav = p%t
                   * grav%psi(1) - p%x(1) * grav%psi(2) - &
            p%x(2) * grav%psi(3) - p%x(3) * grav%psi(4)
    ggrav%a(1) = grav%psi(1)%a(3) - grav%psi(2)%a(4) + (0,1) * &
                 grav%psi(3)%a(4) - grav%psi(4)%a(3)
   ggrav%a(2) = grav%psi(1)%a(4) - grav%psi(2)%a(3) - (0,1) * &
                grav%psi(3)%a(3) + grav%psi(4)%a(4)
   ggrav%a(3) = grav%psi(1)%a(1) + grav%psi(2)%a(2) - (0,1) * &
                 grav%psi(3)%a(2) + grav%psi(4)%a(1)
   ggrav%a(4) = grav%psi(1)%a(2) + grav%psi(2)%a(1) + (0,1) * &
                 grav%psi(3)%a(1) - grav%psi(4)%a(2)
   ggrav1 = ggrav - minv * pgrav
   ggrav2 = f_vf (one, vp, ggrav1) + m * ggrav - pgrav
   ppgrav = (-minv**2) * f_vf (one, vp, pgrav) + minv * pgrav
   do i = 1, 4
   etagrav_dum%psi(i) = f_vf (one, vp, grav%psi(i))
   end do
   etagrav = etagrav_dum - m * grav
   pppgrav%psi(1) = p%t
                          * ppgrav
   pppgrav%psi(2) = p%x(1) * ppgrav
   pppgrav psi(3) = px(2) * ppgrav
   pppgrav%psi(4) = p%x(3) * ppgrav
   gg_grav_dum%psi(1) = p%t * ggrav2
   gg_grav_dum\%psi(2) = p\%x(1) * ggrav2
   gg_grav_dum\%psi(3) = p\%x(2) * ggrav2
   gg_grav_dum\%psi(4) = p\%x(3) * ggrav2
   gg_grav = gr_potf (one, one, ggrav2) - minv * gg_grav_dum
   propgrav = (1 / cmplx (p*p - m**2, m*w, kind=default)) * &
         (etagrav + pppgrav + (1/3.0_default) * gg_grav)
 end function pr_grav
```

Y.27 Polarization vectorspinors

Here we construct the wavefunctions for (massive) gravitinos out of the wavefunctions of (massive) vectorbosons and (massive) Majorana fermions.

$$\psi^{\mu}_{(u;3/2)}(k) = \epsilon^{\mu}_{+}(k) \cdot u(k,+) \tag{Y.138a}$$

$$\psi^{\mu}_{(u;1/2)}(k) = \sqrt{\frac{1}{3}} \, \epsilon^{\mu}_{+}(k) \cdot u(k,-) + \sqrt{\frac{2}{3}} \, \epsilon^{\mu}_{0}(k) \cdot u(k,+) \tag{Y.138b}$$

$$\psi^{\mu}_{(u;-1/2)}(k) = \sqrt{\frac{2}{3}} \, \epsilon^{\mu}_{0}(k) \cdot u(k,-) + \sqrt{\frac{1}{3}} \, \epsilon^{\mu}_{-}(k) \cdot u(k,+) \tag{Y.138c}$$

$$\psi^{\mu}_{(u;-3/2)}(k) = \epsilon^{\mu}_{-}(k) \cdot u(k,-) \tag{Y.138d}$$

and in the same manner for $\psi^{\mu}_{(v;s)}$ with u replaced by v and with the conjugated polarization vectors. These gravitino wavefunctions obey the Dirac equation, they are transverse and they fulfill the irreducibility condition

$$\gamma_{\mu}\psi^{\mu}_{(u/v;s)} = 0. \tag{Y.139}$$

```
\langle \mathtt{omega\_vspinor\_polarizations.f90} \rangle \equiv
  \langle Copyleft \rangle
  module omega_vspinor_polarizations
    use kinds
    use constants
    use omega_vectors
    use omega_bispinors
    use omega_bispinor_couplings
    use omega_vectorspinors
    implicit none
     \langle Declaration \ of \ polarization \ vectorspinors \rangle
    integer, parameter, public :: omega_vspinor_pols_2010_01_A = 0
  contains
     \langle Implementation\ of\ polarization\ vectorspinors \rangle
  end module omega_vspinor_polarizations
\langle Declaration \ of \ polarization \ vectorspinors \rangle \equiv
  public :: ueps, veps
  private :: eps
  private :: outer_product
```

Here we implement the polarization vectors for vectorbosons with trigonometric functions, without the rotating of components done in HELAS [5]. These are only used for generating the polarization vectorspinors.

$$\epsilon_{+}^{\mu}(k) = \frac{-e^{+i\phi}}{\sqrt{2}} (0; \cos\theta\cos\phi - i\sin\phi, \cos\theta\sin\phi + i\cos\phi, -\sin\theta)$$
 (Y.140a)

$$\epsilon_{-}^{\mu}(k) = \frac{e^{-i\phi}}{\sqrt{2}} \left(0; \cos\theta\cos\phi + i\sin\phi, \cos\theta\sin\phi - i\cos\phi, -\sin\theta \right) \tag{Y.140b}$$

$$\epsilon_0^{\mu}(k) = \frac{1}{m} \left(|\vec{k}|; k^0 \sin \theta \cos \phi, k^0 \sin \theta \sin \phi, k^0 \cos \theta \right) \tag{Y.140c}$$

Determining the mass from the momenta is a numerically haphazardous for light particles. Therefore, we accept some redundancy and pass the mass explicitly. For the case that the momentum lies totally in the z-direction we take the convention $\cos \phi = 1$ and $\sin \phi = 0$.

```
\langle Implementation \ of \ polarization \ vectorspinors \rangle \equiv
  pure function eps (mass, k, s) result (e)
    type(vector) :: e
    real(kind=default), intent(in) :: mass
    type(momentum), intent(in) :: k
    integer, intent(in) :: s
    real(kind=default) :: kabs, kabs2, sqrt2, m
    real(kind=default) :: cos_phi, sin_phi, cos_th, sin_th
    complex(kind=default) :: epiphi, emiphi
    sqrt2 = sqrt (2.0_default)
    kabs2 = dot_product (k%x, k%x)
    m = abs(mass)
    if (kabs2 > 0) then
       kabs = sqrt (kabs2)
       if ((k\%x(1) == 0) .and. (k\%x(2) == 0)) then
          cos_phi = 1
          sin_phi = 0
       else
          cos_phi = k%x(1) / sqrt(k%x(1)**2 + k%x(2)**2)
          sin_phi = k%x(2) / sqrt(k%x(1)**2 + k%x(2)**2)
       cos_th = k%x(3) / kabs
       sin_th = sqrt(1 - cos_th**2)
       epiphi = cos_phi + (0,1) * sin_phi
       emiphi = cos_phi - (0,1) * sin_phi
       e%t = 0
       e%x = 0
```

```
select case (s)
       case (1)
          e%x(1) = epiphi * (-cos_th * cos_phi + (0,1) * sin_phi) / sqrt2
          e%x(2) = epiphi * (-cos_th * sin_phi - (0,1) * cos_phi) / sqrt2
          e%x(3) = epiphi * (sin_th / sqrt2)
       case (-1)
          e%x(1) = emiphi * (cos_th * cos_phi + (0,1) * sin_phi) / sqrt2
          e%x(2) = emiphi * (cos_th * sin_phi - (0,1) * cos_phi) / sqrt2
          e%x(3) = emiphi * (-sin_th / sqrt2)
       case (0)
          if (m > 0) then
             e\%t = kabs / m
             e%x = k%t / (m*kabs) * k%x
          end if
       case (4)
          if (m > 0) then
             e = (1 / m) * k
             e = (1 / k\%t) * k
          end if
       end select
          !!! for particles in their rest frame defined to be
           !!! polarized along the 3-direction
       e\%t = 0
       e%x = 0
       select case (s)
       case (1)
          e%x(1) = cmplx (-1, 0, kind=default) / sqrt2
          e%x(2) = cmplx ( 0,
                                 1, kind=default) / sqrt2
       case (-1)
          e%x(1) = cmplx (
                            1,
                                  0, kind=default) / sqrt2
          e%x(2) = cmplx (
                            Ο,
                                  1, kind=default) / sqrt2
       case (0)
          if (m > 0) then
             e%x(3) = 1
          end if
       case (4)
          if (m > 0) then
             e = (1 / m) * k
          else
             e = (1 / k\%t) * k
          end if
       end select
   end if
 end function eps
\langle Implementation\ of\ polarization\ vectorspinors \rangle + \equiv
 pure function ueps (m, k, s) result (t)
   type(vectorspinor) :: t
   real(kind=default), intent(in) :: m
   type(momentum), intent(in) :: k
   integer, intent(in) :: s
   integer :: i
   type(vector) :: ep, e0, em
   type(bispinor) :: up, um
   do i = 1, 4
     t\%psi(i)\%a = 0
   end do
   select case (s)
   case (2)
       ep = eps (m, k, 1)
      up = u (m, k, 1)
      t = outer_product (ep, up)
   case (1)
       ep = eps (m, k, 1)
       e0 = eps (m, k, 0)
       up = u (m, k, 1)
```

```
um = u (m, k, -1)
       t = (1 / sqrt (3.0_default)) * (outer_product (ep, um) &
            + sqrt (2.0_default) * outer_product (e0, up))
    case (-1)
       e0 = eps (m, k, 0)
       em = eps (m, k, -1)
       up = u (m, k, 1)
       um = u (m, k, -1)
       t = (1 / sqrt (3.0_default)) * (sqrt (2.0_default) * &
            outer_product (e0, um) + outer_product (em, up))
    case (-2)
       em = eps (m, k, -1)
       um = u (m, k, -1)
       t = outer_product (em, um)
    end select
  end function ueps
\langle Implementation \ of \ polarization \ vectorspinors \rangle + \equiv
 pure function veps (m, k, s) result (t)
    type(vectorspinor) :: t
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: k
    integer, intent(in) :: s
    integer :: i
    type(vector) :: ep, e0, em
    type(bispinor) :: vp, vm
    do i = 1, 4
      t\%psi(i)\%a = 0
    end do
    select case (s)
    case (2)
       ep = conjg(eps (m, k, 1))
       vp = v (m, k, 1)
       t = outer_product (ep, vp)
    case (1)
       ep = conjg(eps (m, k, 1))
       e0 = conjg(eps (m, k, 0))
       vp = v (m, k, 1)
       vm = v (m, k, -1)
       t = (1 / sqrt (3.0_default)) * (outer_product (ep, vm) &
            + sqrt (2.0_default) * outer_product (e0, vp))
    case (-1)
       e0 = conjg(eps (m, k, 0))
       em = conjg(eps (m, k, -1))
       vp = v (m, k, 1)
       vm = v (m, k, -1)
       t = (1 / sqrt (3.0_default)) * (sqrt (2.0_default) &
            * outer_product (e0, vm) + outer_product (em, vp))
    case (-2)
       em = conjg(eps (m, k, -1))
       vm = v (m, k, -1)
       t = outer_product (em, vm)
    end select
  end function veps
\langle Implementation \ of \ polarization \ vectorspinors \rangle + \equiv
 pure function outer_product (ve, sp) result (vs)
    type(vectorspinor) :: vs
    type(vector), intent(in) :: ve
    type(bispinor), intent(in) :: sp
    integer :: i
    vs\%psi(1)\%a(1:4) = ve\%t * sp\%a(1:4)
    do i = 1, 3
       vs\%psi((i+1))\%a(1:4) = ve\%x(i) * sp\%a(1:4)
    end do
  end function outer_product
```

Y.28 Color

```
\langle omega\_color.f90 \rangle \equiv
  \langle Copyleft \rangle
  module omega_color
    use kinds
    implicit none
    private
     \langle Declaration \ of \ color \ types \rangle
     \langle Declaration \ of \ color \ functions \rangle
    integer, parameter, public :: omega_color_2010_01_A = 0
  contains
     ⟨Implementation of color functions⟩
  end module omega_color
                                                         Color Sum
                                               Y.28.1
\langle Declaration \ of \ color \ types \rangle \equiv
  public :: omega_color_factor
  type omega_color_factor
     integer :: i1, i2
     real(kind=default) :: factor
  end type omega_color_factor
\langle Declaration \ of \ color \ functions \rangle \equiv
  public :: omega_color_sum
The !$omp instruction will result in parallel code if compiled with support for OpenMP otherwise it is ignored.
\langle Implementation \ of \ color \ functions \rangle \equiv
  ⟨pure unless OpenMP⟩
  function omega_color_sum (flv, hel, amp, cf) result (amp2)
    complex(kind=default) :: amp2
    integer, intent(in) :: flv, hel
    complex(kind=default), dimension(:,:,:), intent(in) :: amp
    type(omega_color_factor), dimension(:), intent(in) :: cf
    integer :: n
    amp2 = 0
    !$omp parallel do reduction(+:amp2)
    do n = 1, size (cf)
        amp2 = amp2 + cf(n)\%factor * &
                         amp(flv,cf(n)%i1,hel) * conjg (amp(flv,cf(n)%i2,hel))
    end do
     !$omp end parallel do
  end function omega_color_sum
In the bytecode for the OVM, we only save the symmetric part of the color factor table. This almost halves
the size of n gluon amplitudes for n > 6. For 2 \to (5,6) g the reduced color factor table still amounts for
\sim (75,93)\% of the bytecode, making it desirable to omit it completely by computing it dynamically to reduce
memory requirements. Note that 2\text{Re}(A_{i_1}A_{i_2}^*) = A_{i_1}A_{i_2}^* + A_{i_2}A_{i_1}^*.
\langle Declaration \ of \ color \ functions \rangle + \equiv
  public :: ovm_color_sum
\langle Implementation \ of \ color \ functions \rangle + \equiv
  ⟨pure unless OpenMP⟩
  function ovm_color_sum (flv, hel, amp, cf) result (amp2)
    real(kind=default) :: amp2
    integer, intent(in) :: flv, hel
    complex(kind=default), dimension(:,:,:), intent(in) :: amp
    type(omega_color_factor), dimension(:), intent(in) :: cf
    integer :: n
    amp2 = 0
    !$omp parallel do reduction(+:amp2)
    do n = 1, size (cf)
        if (cf(n)\%i1 == cf(n)\%i2) then
            amp2 = amp2 + cf(n)\%factor * &
                    real(amp(flv,cf(n)%i1,hel) * conjg(amp(flv,cf(n)%i2,hel)))
        else
```

```
amp2 = amp2 + cf(n)\%factor * 2 * &
                   real(amp(flv,cf(n)%i1,hel) * conjg(amp(flv,cf(n)%i2,hel)))
       end if
    end do
    !$omp end parallel do
  end function ovm_color_sum
                                               Y.29
                                                        Utilities
\langle \mathtt{omega\_utils.f90} \rangle \equiv
  \langle Copyleft \rangle
 module omega_utils
    use kinds
    use omega_vectors
    use omega_polarizations
    implicit none
    private
    \langle Declaration \ of \ utility \ functions \rangle
    \langle Numerical\ tolerances \rangle
    integer, parameter, public :: omega_utils_2010_01_A = 0
  contains
    \langle Implementation \ of \ utility \ functions \rangle
  end module omega_utils
                                Y.29.1 Helicity Selection Rule Heuristics
\langle Declaration \ of \ utility \ functions \rangle \equiv
 public :: omega_update_helicity_selection
\langle Implementation \ of \ utility \ functions \rangle \equiv
 pure subroutine omega_update_helicity_selection &
                 (count, amp, max_abs, sum_abs, mask, threshold, cutoff, mask_dirty)
    integer, intent(inout) :: count
    complex(kind=default), dimension(:,:,:), intent(in) :: amp
    real(kind=default), dimension(:), intent(inout) :: max_abs
    real(kind=default), intent(inout) :: sum_abs
    logical, dimension(:), intent(inout) :: mask
    real(kind=default), intent(in) :: threshold
    integer, intent(in) :: cutoff
    logical, intent(out) :: mask_dirty
    integer :: h
    real(kind=default) :: avg
    mask\_dirty = .false.
    if (threshold > 0) then
       count = count + 1
       if (count <= cutoff) then
           forall (h = lbound (amp, 3) : ubound (amp, 3))
              \max_{abs(h)} = \max_{abs(h), maxval (abs (amp(:,:,h))))
           end forall
           sum_abs = sum_abs + sum (abs (amp))
           if (count == cutoff) then
              avg = sum_abs / size (amp) / cutoff
              mask = max_abs >= threshold * epsilon (avg) * avg
              mask_dirty = .true.
           end if
       end if
    end if
  end subroutine omega_update_helicity_selection
                                             Y.29.2 Diagnostics
\langle Declaration \ of \ utility \ functions \rangle + \equiv
 public :: omega_report_helicity_selection
```

We shoul try to use msg_message from WHIZARD's diagnostics module, but this would spoil independent builds.

```
\langle Implementation \ of \ utility \ functions \rangle + \equiv
  subroutine omega_report_helicity_selection (mask, spin_states, threshold, unit)
    logical, dimension(:), intent(in) :: mask
    integer, dimension(:,:), intent(in) :: spin_states
    real(kind=default), intent(in) :: threshold
    integer, intent(in), optional :: unit
    integer :: u
    integer :: h, i
    if (present(unit)) then
       u = unit
    else
       u = 6
    end if
    if (u \ge 0) then
       write (unit = u, &
             fmt = "('| ','Contributing Helicity Combinations: ', I5, ' of ', I5)") &
             count (mask), size (mask)
       write (unit = u, &
            fmt = "('| ','Threshold: amp / avg > ', E9.2, ' = ', E9.2, ' * epsilon()')") &
             threshold * epsilon (threshold), threshold
       do h = 1, size (mask)
          if (mask(h)) then
              i = i + 1
             write (unit = u, fmt = "('| ',I4,': ',20I4)") i, spin_states (:, h)
          end if
       end do
    end if
  end subroutine omega_report_helicity_selection
\langle Declaration \ of \ utility \ functions \rangle + \equiv
 public :: omega_ward_warn, omega_ward_panic
```

The O'Mega amplitudes have only one particle off shell and are the sum of all possible diagrams with the other particles on-shell.

\$

The problem with these gauge checks is that are numerically very small amplitudes that vanish analytically and that violate transversality. The hard part is to determine the thresholds that make threse tests usable.

```
\langle Implementation \ of \ utility \ functions \rangle + \equiv
  subroutine omega_ward_warn (name, m, k, e)
    character(len=*), intent(in) :: name
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: k
    type(vector), intent(in) :: e
    type(vector) :: ek
    real(kind=default) :: abs_eke, abs_ek_abs_e
    ek = eps (m, k, 4)
    abs_eke = abs (ek * e)
    abs_ek_abs_e = abs (ek) * abs (e)
    print *, name, ":", abs_eke / abs_ek_abs_e, abs (ek), abs (e)
    if (abs_eke > 1000 * epsilon (abs_ek_abs_e)) then
       print *, "0'Mega: warning: non-transverse vector field: ", &
            name, ":", abs_eke / abs_ek_abs_e, abs (e)
  end subroutine omega_ward_warn
\langle Implementation \ of \ utility \ functions \rangle + \equiv
  subroutine omega_ward_panic (name, m, k, e)
    character(len=*), intent(in) :: name
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: k
    type(vector), intent(in) :: e
    type(vector) :: ek
    real(kind=default) :: abs_eke, abs_ek_abs_e
    ek = eps (m, k, 4)
```

```
abs_eke = abs (ek * e)
    abs_ek_abs_e = abs (ek) * abs (e)
    if (abs_eke > 1000 * epsilon (abs_ek_abs_e)) then
       print *, "O'Mega: panic: non-transverse vector field: ", &
            name, ":", abs_eke / abs_ek_abs_e, abs (e)
       stop
    end if
  end subroutine omega_ward_panic
\langle Declaration \ of \ utility \ functions \rangle + \equiv
 public :: omega_slavnov_warn, omega_slavnov_panic
\langle Implementation \ of \ utility \ functions \rangle + \equiv
  subroutine omega_slavnov_warn (name, m, k, e, phi)
    character(len=*), intent(in) :: name
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: k
    type(vector), intent(in) :: e
    complex(kind=default), intent(in) :: phi
    type(vector) :: ek
    real(kind=default) :: abs_eke, abs_ek_abs_e
    ek = eps (m, k, 4)
    abs_eke = abs (ek * e - phi)
    abs_ek_abs_e = abs (ek) * abs (e)
    print *, name, ":", abs_eke / abs_ek_abs_e, abs (ek), abs (e)
    if (abs_eke > 1000 * epsilon (abs_ek_abs_e)) then
       print *, "O'Mega: warning: non-transverse vector field: ", &
             name, ":", abs_eke / abs_ek_abs_e, abs (e)
    end if
  end subroutine omega_slavnov_warn
\langle Implementation \ of \ utility \ functions \rangle + \equiv
  subroutine omega_slavnov_panic (name, m, k, e, phi)
    {\tt character(len=*),\ intent(in)\ ::\ name}
    real(kind=default), intent(in) :: m
    type(momentum), intent(in) :: k
    type(vector), intent(in) :: e
    complex(kind=default), intent(in) :: phi
    type(vector) :: ek
    real(kind=default) :: abs_eke, abs_ek_abs_e
    ek = eps (m, k, 4)
    abs_eke = abs (ek * e - phi)
    abs_ek_abs_e = abs(ek) * abs(e)
    if (abs\_eke > 1000 * epsilon (abs\_ek\_abs\_e)) then
       print *, "O'Mega: panic: non-transverse vector field: ", &
            name, ":", abs_eke / abs_ek_abs_e, abs (e)
       stop
    end if
  end subroutine omega_slavnov_panic
\langle Declaration \ of \ utility \ functions \rangle + \equiv
 public :: omega_check_arguments_warn, omega_check_arguments_panic
\langle Implementation \ of \ utility \ functions \rangle + \equiv
  subroutine omega_check_arguments_warn (n, k)
    integer, intent(in) :: n
    real(kind=default), dimension(0:,:), intent(in) :: k
    integer :: i
    i = size(k,dim=1)
    if (i /= 4) then
       print *, "O'Mega: warning: wrong # of dimensions:", i
    end if
    i = size(k,dim=2)
    if (i /= n) then
       print *, "O'Mega: warning: wrong # of momenta:", i, &
             ", expected", n
    end if
  end subroutine omega_check_arguments_warn
```

```
\langle Implementation \ of \ utility \ functions \rangle + \equiv
  subroutine omega_check_arguments_panic (n, k)
    integer, intent(in) :: n
    real(kind=default), dimension(0:,:), intent(in) :: k
    logical :: error
    integer :: i
    error = .false.
    i = size(k,dim=1)
    if (i /= n) then
       print *, "O'Mega: warning: wrong # of dimensions:", i
    end if
    i = size(k,dim=2)
    if (i /= n) then
       print *, "O'Mega: warning: wrong # of momenta:", i, &
             ", expected", n
       error = .true.
    end if
    if (error) then
       stop
  end subroutine omega_check_arguments_panic
\langle Declaration \ of \ utility \ functions \rangle + \equiv
 public :: omega_check_helicities_warn, omega_check_helicities_panic
 private :: omega_check_helicity
\langle Implementation \ of \ utility \ functions \rangle + \equiv
 function omega_check_helicity (m, smax, s) result (error)
    real(kind=default), intent(in) :: m
    integer, intent(in) :: smax, s
    logical :: error
    select case (smax)
    case (0)
       error = (s /= 0)
    case (1)
       error = (abs (s) /= 1)
    case (2)
       if (m == 0.0_default) then
          error = .not. (abs (s) == 1 .or. abs (s) == 4)
       else
           error = .not. (abs (s) \leq 1 .or. abs (s) == 4)
       end if
    case (4)
       error = .true.
    case default
       error = .true.
    end select
  end function omega_check_helicity
\langle Implementation \ of \ utility \ functions \rangle + \equiv
  subroutine omega_check_helicities_warn (m, smax, s)
    real(kind=default), dimension(:), intent(in) :: m
    integer, dimension(:), intent(in) :: smax, s
    integer :: i
    do i = 1, size (m)
       if (omega_check_helicity (m(i), smax(i), s(i))) then
          print *, "O'Mega: warning: invalid helicity", s(i)
       end if
    end do
  end subroutine omega_check_helicities_warn
\langle Implementation \ of \ utility \ functions \rangle + \equiv
 subroutine omega_check_helicities_panic (m, smax, s)
    real(kind=default), dimension(:), intent(in) :: m
    integer, dimension(:), intent(in) :: smax, s
    logical :: error
    logical :: error1
```

```
integer :: i
    error = .false.
    do i = 1, size (m)
       error1 = omega_check_helicity (m(i), smax(i), s(i))
       if (error1) then
          print *, "O'Mega: panic: invalid helicity", s(i)
           error = .true.
       end if
    end do
    if (error) then
       stop
    end if
  end subroutine omega_check_helicities_panic
\langle Declaration \ of \ utility \ functions \rangle + \equiv
 public :: omega_check_momenta_warn, omega_check_momenta_panic
 \verb"private":: check_momentum_conservation, check_mass\_shell"
\langle Numerical\ tolerances \rangle \equiv
  integer, parameter, private :: MOMENTUM_TOLERANCE = 10000
\langle Implementation \ of \ utility \ functions \rangle + \equiv
 function check_momentum_conservation (k) result (error)
    real(kind=default), dimension(0:,:), intent(in) :: k
    logical :: error
    error = any (abs (sum (k(:,3:), dim = 2) - k(:,1) - k(:,2)) > &
         MOMENTUM_TOLERANCE * epsilon (maxval (abs (k), dim = 2)))
    if (error) then
       print *, sum (k(:,3:), dim = 2) - k(:,1) - k(:,2)
       print *, MOMENTUM_TOLERANCE * epsilon (maxval (abs (k), dim = 2)), &
             maxval (abs (k), dim = 2)
    end if
  end function check_momentum_conservation
\langle Numerical\ tolerances \rangle + \equiv
  integer, parameter, private :: ON_SHELL_TOLERANCE = 1000000
\langle Implementation \ of \ utility \ functions \rangle + \equiv
 function check_mass_shell (m, k) result (error)
    real(kind=default), intent(in) :: m
    real(kind=default), dimension(0:), intent(in) :: k
    real(kind=default) :: e2
    logical :: error
    e2 = k(1)**2 + k(2)**2 + k(3)**2 + m**2
    error = abs (k(0)**2 - e2) > ON_SHELL_TOLERANCE * epsilon (max <math>(k(0)**2, e2))
    if (error) then
       print *, k(0)**2 - e2
       print *, ON_SHELL_TOLERANCE * epsilon (max (k(0)**2, e2)), max (k(0)**2, e2)
    end if
  end function check_mass_shell
\langle Implementation \ of \ utility \ functions \rangle + \equiv
  subroutine omega_check_momenta_warn (m, k)
    real(kind=default), dimension(:), intent(in) :: m
    real(kind=default), dimension(0:,:), intent(in) :: k
    integer :: i
    if (check_momentum_conservation (k)) then
       print *, "O'Mega: warning: momentum not conserved"
    end if
    do i = 1, size(m)
       if (check_mass_shell (m(i), k(:,i))) then
           print *, "O'Mega: warning: particle #", i, "not on-shell"
       end if
    end do
  end subroutine omega_check_momenta_warn
\langle Implementation \ of \ utility \ functions \rangle + \equiv
  subroutine omega_check_momenta_panic (m, k)
    real(kind=default), dimension(:), intent(in) :: m
    real(kind=default), dimension(0:,:), intent(in) :: k
```

```
logical :: error
 logical :: error1
 integer :: i
 error = check_momentum_conservation (k)
 if (error) then
    print *, "O'Mega: panic: momentum not conserved"
 end if
 do i = 1, size(m)
     error1 = check_mass_shell (m(i), k(0:,i))
     if (error1) then
       print *, "O'Mega: panic: particle #", i, "not on-shell"
     end if
 end do
 if (error) then
     stop
 end if
end subroutine omega_check_momenta_panic
                                           Obsolete Summation
                                  Y.29.3
```

Spin/Helicity Summation

```
\langle Declaration \ of \ obsolete \ utility \ functions \rangle \equiv
  public :: omega_sum, omega_sum_nonzero, omega_nonzero
  private :: state_index
\langle Implementation \ of \ obsolete \ utility \ functions \rangle \equiv
  pure function omega_sum (omega, p, states, fixed) result (sigma)
    real(kind=default) :: sigma
    real(kind=default), dimension(0:,:), intent(in) :: p
    integer, dimension(:), intent(in), optional :: states, fixed
    ⟨interface for O'Mega Amplitude⟩
    integer, dimension(size(p,dim=2)) :: s, nstates
    integer :: j
    complex(kind=default) :: a
    if (present (states)) then
        nstates = states
    else
        nstates = 2
    end if
    sigma = 0
    s = -1
    sum_spins: do
        if (present (fixed)) then
            !!! print *, 's = ', s, ', fixed = ', fixed, ', nstates = ', nstates, &
                 ', fixed|s = ', merge (fixed, s, mask = nstates == 0)
           a = omega (p, merge (fixed, s, mask = nstates == 0))
        else
           a = omega (p, s)
        end if
        sigma = sigma + a * conjg(a)
        \langle Step \ s \ like \ a \ n-ary \ number \ and \ terminate \ when \ all \ (s == -1) \rangle
    end do sum_spins
    sigma = sigma / num_states (2, nstates(1:2))
  end function omega_sum
We're looping over all spins like a n-ary numbers (-1, \ldots, -1, -1), (-1, \ldots, -1, 0), (-1, \ldots, -1, 1), (-1, \ldots, 0, -1), (-1, \ldots, -1, 0)
\ldots, (1,\ldots,1,0), (1,\ldots,1,1):
\langle Step \ s \ like \ a \ n-ary \ number \ and \ terminate \ when \ all \ (s == -1) \rangle \equiv
  do j = size (p, dim = 2), 1, -1
     select case (nstates (j))
     case (3) ! massive vectors
         s(j) = modulo (s(j) + 2, 3) - 1
      case (2) ! spinors, massless vectors
         s(j) = - s(j)
      case (1) ! scalars
```

```
s(j) = -1
     case (0) ! fized spin
        s(j) = -1
     case default ! ???
        s(j) = -1
     end select
     if (s(j) /= -1) then
         cycle sum_spins
     end if
  end do
  exit sum_spins
The dual operation evaluates an n-number:
\langle Implementation \ of \ obsolete \ utility \ functions \rangle + \equiv
  pure function state_index (s, states) result (n)
    integer, dimension(:), intent(in) :: s
    integer, dimension(:), intent(in), optional :: states
    integer :: n
    integer :: j, p
    n = 1
    p = 1
    if (present (states)) then
       do j = size (s), 1, -1
           select case (states(j))
           case (3)
              n = n + p * (s(j) + 1)
           case (2)
              n = n + p * (s(j) + 1) / 2
           end select
           p = p * states(j)
       end do
    else
       do j = size (s), 1, -1
           n = n + p * (s(j) + 1) / 2
           p = p * 2
        end do
    end if
  end function state_index
\langle interface for O'Mega Amplitude \rangle \equiv
  interface
     pure function omega (p, s) result (me)
       use kinds
       implicit none
       complex(kind=default) :: me
       real(kind=default), dimension(0:,:), intent(in) :: p
       integer, dimension(:), intent(in) :: s
     end function omega
  end interface
\langle Declaration \ of \ obsolete \ utility \ functions \rangle + \equiv
  public :: num_states
\langle Implementation\ of\ obsolete\ utility\ functions \rangle + \equiv
  pure function num_states (n, states) result (ns)
    integer, intent(in) :: n
    integer, dimension(:), intent(in), optional :: states
    integer :: ns
    if (present (states)) then
       ns = product (states, mask = states == 2 .or. states == 3)
    else
       ns = 2**n
    end if
  end function num_states
```

Y.30 omega95

 $\langle omega95.f90 \rangle \equiv$

```
⟨Copyleft⟩
module omega95
use constants
use omega_spinors
use omega_vectors
use omega_vectors
use omega_tensors
use omega_tensors
use omega_tensor_polarizations
use omega_couplings
use omega_spinor_couplings
use omega_color
use omega_utils
public
end module omega95
```

Y.31 omega95 Revisited

```
⟨omega95_bispinors.f90⟩≡
⟨Copyleft⟩
module omega95_bispinors
use constants
use omega_bispinors
use omega_vectors
use omega_vectorspinors
use omega_vectorspinors
use omega_polarizations
use omega_vspinor_polarizations
use omega_couplings
use omega_couplings
use omega_bispinor_couplings
use omega_color
use omega_utils
public
end module omega95_bispinors
```

Y.32 Testing

```
(omega_testtools.f90) =
  (Copyleft)
module omega_testtools
  use kinds
  implicit none
  private
  real(kind=default), parameter, private :: ABS_THRESHOLD_DEFAULT = 1E-17
  real(kind=default), parameter, private :: THRESHOLD_DEFAULT = 0.6
  real(kind=default), parameter, private :: THRESHOLD_WARN = 0.8
   (Declaration of test support functions)
  contains
   (Implementation of test support functions)
  end module omega_testtools
```

Quantify the agreement of two real or complex numbers

$$\operatorname{agreement}(x,y) = \frac{\ln \Delta(x,y)}{\ln \epsilon} \in [0,1]$$
 (Y.141)

with

$$\Delta(x,y) = \frac{|x-y|}{\max(|x|,|y|)}$$
 (Y.142)

and values outside [0,1] replaced the closed value in the interval. In other words

- 1 for $x y = \max(|x|, |y|) \cdot \mathcal{O}(\epsilon)$ and
- 0 for $x y = \max(|x|, |y|) \cdot \mathcal{O}(1)$

```
with logarithmic interpolation. The cases x = 0 and y = 0 must be treated separately.
\langle Declaration \ of \ test \ support \ functions \rangle \equiv
  public :: agreement
  interface agreement
     module procedure agreement_real, agreement_complex, &
             agreement_real_complex, agreement_complex_real, &
             agreement_integer_complex, agreement_complex_integer, &
             agreement_integer_real, agreement_real_integer
  end interface
  private :: agreement_real, agreement_complex, &
       agreement_real_complex, agreement_complex_real, &
       agreement_integer_complex, agreement_complex_integer, &
       agreement_integer_real, agreement_real_integer
\langle Implementation \ of \ test \ support \ functions \rangle \equiv
  elemental function agreement_real (x, y, base) result (a)
    real(kind=default) :: a
    real(kind=default), intent(in) :: x, y
    real(kind=default), intent(in), optional :: base
    real(kind=default) :: scale, dxy
    if (present (base)) then
       scale = max (abs (x), abs (y), abs (base))
    else
       scale = max (abs (x), abs (y))
    end if
    if (ieee_is_nan (x) .or. ieee_is_nan (y)) then
    else if (scale <= 0) then
       a = -1
    else
       dxy = abs (x - y) / scale
       if (dxy \le 0.0_default) then
          a = 1
       else
          a = log (dxy) / log (epsilon (scale))
          a = max (0.0_default, min (1.0_default, a))
           if (ieee_is_nan (a)) then
              a = 0
           end if
       end if
    end if
    if (ieee_is_nan (a)) then
       a = 0
    end if
  end function agreement_real
Poor man's replacement
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  elemental function ieee_is_nan (x) result (yorn)
    logical :: yorn
    real (kind=default), intent(in) :: x
    yorn = (x /= x)
  end function ieee_is_nan
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  elemental function agreement_complex (x, y, base) result (a)
    real(kind=default) :: a
    complex(kind=default), intent(in) :: x, y
    real(kind=default), intent(in), optional :: base
    real(kind=default) :: scale, dxy
    if (present (base)) then
       scale = max (abs (x), abs (y), abs (base))
    else
       scale = max (abs (x), abs (y))
    end if
               ieee_is_nan (real (x, kind=default)) .or. ieee_is_nan (aimag (x)) &
          .or. ieee_is_nan (real (y, kind=default)) .or. ieee_is_nan (aimag (y))) then
       a = 0
```

```
else if (scale <= 0) then
       a = -1
    else
       dxy = abs (x - y) / scale
       if (dxy <= 0.0_default) then
          a = 1
       else
          a = log (dxy) / log (epsilon (scale))
          a = max (0.0_default, min (1.0_default, a))
           if (ieee_is_nan (a)) then
              a = 0
          end if
       end if
    end if
    if (ieee_is_nan (a)) then
       a = 0
    end if
  end function agreement_complex
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  elemental function agreement_real_complex (x, y, base) result (a)
    real(kind=default) :: a
    real(kind=default), intent(in) :: x
    complex(kind=default), intent(in) :: y
    real(kind=default), intent(in), optional :: base
    a = agreement_complex (cmplx (x, kind=default), y, base)
  end function agreement_real_complex
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  elemental function agreement_complex_real (x, y, base) result (a)
    real(kind=default) :: a
    complex(kind=default), intent(in) :: x
    real(kind=default), intent(in) :: y
    real(kind=default), intent(in), optional :: base
    a = agreement_complex (x, cmplx (y, kind=default), base)
  end function agreement_complex_real
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  elemental function agreement_integer_complex (x, y, base) result (a)
    real(kind=default) :: a
    integer, intent(in) :: x
    complex(kind=default), intent(in) :: y
    real(kind=default), intent(in), optional :: base
    a = agreement_complex (cmplx (x, kind=default), y, base)
  end function agreement_integer_complex
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
 elemental function agreement_complex_integer (x, y, base) result (a)
    real(kind=default) :: a
    complex(kind=default), intent(in) :: x
    integer, intent(in) :: y
    real(kind=default), intent(in), optional :: base
    a = agreement_complex (x, cmplx (y, kind=default), base)
  end function agreement_complex_integer
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
 elemental function agreement_integer_real (x, y, base) result (a)
    real(kind=default) :: a
    integer, intent(in) :: x
    real(kind=default), intent(in) :: y
    real(kind=default), intent(in), optional :: base
    a = agreement_real (real(x, kind=default), y, base)
  end function agreement_integer_real
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  elemental function agreement_real_integer (x, y, base) result (a)
    real(kind=default) :: a
    real(kind=default), intent(in) :: x
    integer, intent(in) :: y
```

```
real(kind=default), intent(in), optional :: base
    a = agreement_real (x, real (y, kind=default), base)
 end function agreement_real_integer
\langle Declaration \ of \ test \ support \ functions \rangle + \equiv
 public:: vanishes
  interface vanishes
     module procedure vanishes_real, vanishes_complex
  end interface
 private :: vanishes_real, vanishes_complex
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  elemental function vanishes_real (x, scale) result (a)
    real(kind=default) :: a
    real(kind=default), intent(in) :: x
    real(kind=default), intent(in), optional :: scale
    real(kind=default) :: scaled_x
    if (x == 0.0_default) then
       a = 1
       return
    else if (ieee_is_nan (x)) then
       a = 0
       return
    end if
    scaled_x = x
    if (present (scale)) then
       if (scale \neq 0) then
          scaled_x = x / abs (scale)
       else
          a = 0
          return
       end if
    else
    end if
    a = log (abs (scaled_x)) / log (epsilon (scaled_x))
    a = max (0.0_default, min (1.0_default, a))
    if (ieee_is_nan (a)) then
       a = 0
    end if
  end function vanishes_real
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
 elemental function vanishes_complex (x, scale) result (a)
    real(kind=default) :: a
    complex(kind=default), intent(in) :: x
    real(kind=default), intent(in), optional :: scale
    a = vanishes_real (abs (x), scale)
  end function vanishes_complex
\langle Declaration \ of \ test \ support \ functions \rangle + \equiv
 public :: expect
  interface expect
     module procedure expect_integer, expect_real, expect_complex, &
          expect_real_integer, expect_integer_real, &
          expect_complex_integer, expect_integer_complex, &
          expect_complex_real, expect_real_complex
  end interface
  private :: expect_integer, expect_real, expect_complex, &
       expect_real_integer, expect_integer_real, &
       expect_complex_integer, expect_integer_complex, &
       expect_complex_real, expect_real_complex
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  subroutine expect_integer (x, x0, msg, passed, quiet, buffer, unit)
    integer, intent(in) :: x, x0
    character(len=*), intent(in) :: msg
    logical, intent(inout), optional :: passed
    logical, intent(in), optional :: quiet
    character(len=*), intent(inout), optional :: buffer
```

```
integer, intent(in), optional :: unit
   logical :: failed, verbose
   character(len=*), parameter :: fmt = "(1X,A,': ',A)"
   character(len=*), parameter :: &
         fmt_verbose = "(1X,A,': ',A,' [expected ',I6,', got ',I6,']')"
   failed = .false.
   verbose = .true.
   if (present (quiet)) then
       verbose = .not.quiet
   end if
   if (x == x0) then
      if (verbose) then
          if (.not. (present (buffer) .or. present (unit))) then
             write (unit = *, fmt = fmt) msg, "passed"
          if (present (unit)) then
             write (unit = unit, fmt = fmt) msg, "passed"
         end if
          if (present (buffer)) then
             write (unit = buffer, fmt = fmt) msg, "passed"
       end if
   else
      if (.not. (present (buffer) .or. present (unit))) then
         write (unit = *, fmt = fmt_verbose) msg, "failed", x0, x
       end if
      if (present (unit)) then
         write (unit = unit, fmt = fmt_verbose) msg, "failed", x0, x
      if (present (buffer)) then
          write (unit = buffer, fmt = fmt_verbose) msg, "failed", x0, x
       end if
      failed = .true.
   end if
   if (present (passed)) then
      passed = passed .and. .not.failed
   end if
 end subroutine expect_integer
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
 subroutine expect_real (x, x0, msg, passed, threshold, quiet, abs_threshold)
   real(kind=default), intent(in) :: x, x0
   character(len=*), intent(in) :: msg
   logical, intent(inout), optional :: passed
   real(kind=default), intent(in), optional :: threshold
   real(kind=default), intent(in), optional :: abs_threshold
   logical, intent(in), optional :: quiet
   logical :: failed, verbose
   real(kind=default) :: agreement_threshold, abs_agreement_threshold
   character(len=*), parameter :: fmt = "(1X,A,': ',A,' at ',I4,'\%')"
   character(len=*), parameter :: fmt_verbose = "(1X,A,': ',A,' at ',I4,'%'," // &
         "' [expected ',E10.3,', got ',E10.3,']')"
   real(kind=default) :: a
   failed = .false.
   verbose = .true.
   if (present (quiet)) then
      verbose = .not.quiet
   end if
   if (x == x0) then
      if (verbose) then
         write (unit = *, fmt = fmt) msg, "passed", 100
      end if
   else
      if (x0 == 0) then
         a = vanishes (x)
       else
         a = agreement (x, x0)
```

```
end if
       if (present (threshold)) then
          agreement_threshold = threshold
       else
          agreement_threshold = THRESHOLD_DEFAULT
       end if
       if (present (abs_threshold)) then
          abs_agreement_threshold = abs_threshold
          abs_agreement_threshold = ABS_THRESHOLD_DEFAULT
       end if
       if (a >= agreement_threshold .or. &
           max(abs(x), abs(x0)) <= abs_agreement_threshold) then</pre>
          if (verbose) then
             if (a >= THRESHOLD_WARN) then
                write (unit = *, fmt = fmt) msg, "passed", int (a * 100)
                write (unit = *, fmt = fmt_verbose) msg, "passed", int (a * 100), x0, x
             end if
          end if
       else
          failed = .true.
          write (unit = *, fmt = fmt_verbose) msg, "failed", int (a * 100), x0, x
       end if
   end if
   if (present (passed)) then
      passed = passed .and. .not. failed
   end if
 end subroutine expect_real
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
 subroutine expect_complex (x, x0, msg, passed, threshold, quiet, abs_threshold)
   complex(kind=default), intent(in) :: x, x0
   character(len=*), intent(in) :: msg
   logical, intent(inout), optional :: passed
   real(kind=default), intent(in), optional :: threshold
   real(kind=default), intent(in), optional :: abs_threshold
   logical, intent(in), optional :: quiet
   logical :: failed, verbose
   \verb"real(kind=default) :: agreement\_threshold, abs\_agreement\_threshold
   character(len=*), parameter :: fmt = "(1X,A,': ',A,' at ',I4,'%')"
   character(len=*), parameter :: fmt_verbose = "(1X,A,': ',A,' at ',I4,'%'," // &
         "' [expected (',E10.3,',',E10.3,'), got (',E10.3,',',E10.3,')]')"
   character(len=*), parameter :: fmt_phase = "(1X,A,': ',A,' at ',I4,'%'," // &
         "' [modulus passed at ',I4,'%',', phases ',F5.3,' vs. ',F5.3,']')"
   real(kind=default) :: a, a_modulus
   failed = .false.
   verbose = .true.
   if (present (quiet)) then
      verbose = .not.quiet
   end if
   if (x == x0) then
       if (verbose) then
          write (unit = *, fmt = fmt) msg, "passed", 100
       end if
   else
       if (x0 == 0) then
          a = vanishes (x)
       else
          a = agreement (x, x0)
       end if
       if (present (threshold)) then
          agreement_threshold = threshold
       else
          agreement_threshold = THRESHOLD_DEFAULT
       if (present (abs_threshold)) then
```

```
abs_agreement_threshold = abs_threshold
       else
          abs_agreement_threshold = ABS_THRESHOLD_DEFAULT
       end if
       if (a >= agreement_threshold .or. &
           max(abs(x), abs(x0)) \le abs_agreement_threshold) then
          if (verbose) then
             if (a >= THRESHOLD_WARN) then
                 write (unit = *, fmt = fmt) msg, "passed", int (a * 100)
                 write (unit = *, fmt = fmt_verbose) msg, "passed", int (a * 100), x0, x
             end if
          end if
       else
          a_modulus = agreement (abs (x), abs (x0))
          if (a_modulus >= agreement_threshold) then
             write (unit = *, fmt = fmt_phase) msg, "failed", int (a * 100), &
                   int (a_modulus * 100), &
                   atan2 (real (x, kind=default), aimag (x)), &
                   atan2 (real (x0, kind=default), aimag (x0))
             write (unit = *, fmt = fmt_verbose) msg, "failed", int (a * 100), x0, x
          end if
          failed = .true.
       end if
    end if
    if (present (passed)) then
       passed = passed .and. .not.failed
    end if
  end subroutine expect_complex
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  subroutine expect_real_integer (x, x0, msg, passed, threshold, quiet)
    real(kind=default), intent(in) :: x
    integer, intent(in) :: x0
    character(len=*), intent(in) :: msg
    real(kind=default), intent(in), optional :: threshold
    logical, intent(inout), optional :: passed
    logical, intent(in), optional :: quiet
    call expect_real (x, real (x0, kind=default), msg, passed, threshold, quiet)
  end subroutine expect_real_integer
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  subroutine expect_integer_real (x, x0, msg, passed, threshold, quiet)
    integer, intent(in) :: x
    real(kind=default), intent(in) :: x0
    character(len=*), intent(in) :: msg
    real(kind=default), intent(in), optional :: threshold
    logical, intent(inout), optional :: passed
    logical, intent(in), optional :: quiet
    call expect_real (real (x, kind=default), x0, msg, passed, threshold, quiet)
  end subroutine expect_integer_real
\langle \mathit{Implementation of test support functions} \rangle + \equiv
  subroutine expect_complex_integer (x, x0, msg, passed, threshold, quiet)
    complex(kind=default), intent(in) :: x
    integer, intent(in) :: x0
    character(len=*), intent(in) :: msg
    logical, intent(inout), optional :: passed
    real(kind=default), intent(in), optional :: threshold
    logical, intent(in), optional :: quiet
    call expect_complex (x, cmplx (x0, kind=default), msg, passed, threshold, quiet)
  end subroutine expect_complex_integer
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  subroutine expect_integer_complex (x, x0, msg, passed, threshold, quiet)
    integer, intent(in) :: x
    complex(kind=default), intent(in) :: x0
```

```
character(len=*), intent(in) :: msg
    logical, intent(inout), optional :: passed
    real(kind=default), intent(in), optional :: threshold
    logical, intent(in), optional :: quiet
    call expect_complex (cmplx (x, kind=default), x0, msg, passed, threshold, quiet)
  end subroutine expect_integer_complex
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  subroutine expect_complex_real (x, x0, msg, passed, threshold, quiet)
    complex(kind=default), intent(in) :: x
    real(kind=default), intent(in) :: x0
    character(len=*), intent(in) :: msg
    logical, intent(inout), optional :: passed
    real(kind=default), intent(in), optional :: threshold
    logical, intent(in), optional :: quiet
    call expect_complex (x, cmplx (x0, kind=default), msg, passed, threshold, quiet)
  end subroutine expect_complex_real
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  subroutine expect_real_complex (x, x0, msg, passed, threshold, quiet)
    real(kind=default), intent(in) :: x
    complex(kind=default), intent(in) :: x0
    character(len=*), intent(in) :: msg
    logical, intent(inout), optional :: passed
    real(kind=default), intent(in), optional :: threshold
    logical, intent(in), optional :: quiet
    call expect_complex (cmplx (x, kind=default), x0, msg, passed, threshold, quiet)
  end subroutine expect_real_complex
\langle Declaration\ of\ test\ support\ functions \rangle + \equiv
 public :: expect_zero
  interface expect_zero
     module procedure expect_zero_integer, expect_zero_real, expect_zero_complex
  end interface
 private :: expect_zero_integer, expect_zero_real, expect_zero_complex
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  subroutine expect_zero_integer (x, msg, passed)
    integer, intent(in) :: x
    character(len=*), intent(in) :: msg
    logical, intent(inout), optional :: passed
    call expect_integer (x, 0, msg, passed)
  end subroutine expect_zero_integer
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  subroutine expect_zero_real (x, scale, msg, passed, threshold, quiet)
    real(kind=default), intent(in) :: x, scale
    character(len=*), intent(in) :: msg
    logical, intent(inout), optional :: passed
    real(kind=default), intent(in), optional :: threshold
    logical, intent(in), optional :: quiet
    logical :: failed, verbose
    real(kind=default) :: agreement_threshold
    character(len=*), parameter :: fmt = "(1X,A,': ',A,' at ',I4,'%')"
    character(len=*), parameter :: fmt_verbose = "(1X,A,': ',A,' at ',I4,'%'," // &
         "' [expected 0 (relative to ',E10.3,') got ',E10.3,']')"
    real(kind=default) :: a
    failed = .false.
    verbose = .true.
    if (present (quiet)) then
       verbose = .not.quiet
    end if
    if (x == 0) then
       if (verbose) then
          write (unit = *, fmt = fmt) msg, "passed", 100
       end if
    else
       a = vanishes (x, scale = scale)
       if (present (threshold)) then
```

```
agreement_threshold = threshold
       else
          agreement_threshold = THRESHOLD_DEFAULT
       end if
       if (a >= agreement_threshold) then
          if (verbose) then
              if (a >= THRESHOLD_WARN) then
                 write (unit = *, fmt = fmt) msg, "passed", int (a * 100)
                 write (unit = *, fmt = fmt_verbose) msg, "passed", int (a * 100), scale, x
              end if
          end if
       else
          failed = .true.
          write (unit = *, fmt = fmt_verbose) msg, "failed", int (a * 100), scale, x
       end if
    end if
    if (present (passed)) then
       passed = passed .and. .not.failed
    end if
  end subroutine expect_zero_real
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  subroutine expect_zero_complex (x, scale, msg, passed, threshold, quiet)
    complex(kind=default), intent(in) :: x
    real(kind=default), intent(in) :: scale
    character(len=*), intent(in) :: msg
    logical, intent(inout), optional :: passed
    real(kind=default), intent(in), optional :: threshold
    logical, intent(in), optional :: quiet
    call expect_zero_real (abs (x), scale, msg, passed, threshold, quiet)
  end subroutine expect_zero_complex
\langle Implementation \ of \ test \ support \ functions \rangle + \equiv
  subroutine print_matrix (a)
    complex(kind=default), dimension(:,:), intent(in) :: a
    integer :: row
    do row = 1, size (a, dim=1)
       write (unit = *, fmt = "(10(tr2, f5.2, '+', f5.2, 'I'))") a(row,:)
    end do
  end subroutine print_matrix
\langle Declaration \ of \ test \ support \ functions \rangle + \equiv
 public :: print_matrix
\langle \texttt{test\_omega95.f90} \rangle \equiv
  \langle Copyleft \rangle
 program test_omega95
    use kinds
    use omega95
    use omega_testtools
    implicit none
    real(kind=default) :: m, pabs, qabs, w
    real(kind=default), dimension(0:3) :: r
    complex(kind=default) :: c_one, c_nil
    type(momentum) :: p, q, p0
    type(vector) :: vp, vq, vtest, v0
    type(tensor) :: ttest
    type(spinor) :: test_psi, test_spinor1, test_spinor2
    \verb|type|(conjspinor|) :: test_psibar, test_conjspinor|, test_conjspinor||2|
    integer, dimension(8) :: date_time
    integer :: rsize, i
    logical :: passed
    call date_and_time (values = date_time)
    call random_seed (size = rsize)
    call random_seed (put = spread (product (date_time), dim = 1, ncopies = rsize))
    w = 1.4142
    c_one = 1.0_default
```

```
c_nil = 0.0_default
    m = 13
    pabs = 42
    qabs = 137
    call random_number (r)
    vtest%t = cmplx (10.0_default * r(0), kind=default)
    vtest\%x(1:3) = cmplx (10.0_default * r(1:3), kind=default)
    ttest = vtest.tprod.vtest
    call random_momentum (p, pabs, m)
    call random_momentum (q, qabs, m)
    call random_momentum (p0, 0.0_default, m)
    vp = p
    vq = q
    v0 = p0
    passed = .true.
    \langle Test \text{ omega95} \rangle
    if (.not. passed) then
      stop 1
    end if
  end program test_omega95
\langle Test \text{ omega95} \rangle \equiv
  print *, "*** Checking the equations of motion ***:"
   {\tt call \ expect \ (abs(f_vf(c_one,vp,u(m,p,+1))-m*u(m,p,+1)), \ 0, \ "|[p-m]u(+)|=0", \ passed) } 
   {\tt call \ expect \ (abs(f_vf(c_one,vp,u(m,p,-1))-m*u(m,p,-1)), \ 0, \ "|[p-m]u(-)|=0", \ passed) } 
   {\tt call \ expect \ (abs(f\_vf(c\_one,vp,v(m,p,+1))+m*v(m,p,+1)),\ 0,\ "|[p+m]v(+)|=0",\ passed) } 
   {\tt call \ expect \ (abs(f_vf(c_one,vp,v(m,p,-1))+m*v(m,p,-1)), \ 0, \ "|[p+m]v(-)|=0", \ passed) } 
   \texttt{call expect (abs(f_fv(c_one,ubar(m,p,+1),vp)-m*ubar(m,p,+1)), 0, "|ubar(+)[p-m]|=0", passed) } \\
   \texttt{call expect (abs(f\_fv(c\_one,ubar(m,p,-1),vp)-m*ubar(m,p,-1)), 0, "|ubar(-)[p-m]|=0", passed) } \\
   \texttt{call expect (abs(f\_fv(c\_one,vbar(m,p,+1),vp)+m*vbar(m,p,+1)), 0, "|vbar(+)[p+m]|=0", passed) } 
   \texttt{call expect (abs(f_fv(c_one,vbar(m,p,-1),vp)+m*vbar(m,p,-1)), 0, "|vbar(-)[p+m]|=0", passed) } \\
  print *, "*** Checking the equations of motion for negative mass***:"
   {\tt call \ expect \ (abs(f_vf(c_one,vp,u(-m,p,+1))+m*u(-m,p,+1)),\ 0,\ "|[p+m]u(+)|=0",\ passed) } 
   {\tt call \ expect \ (abs(f_vf(c_one,vp,u(-m,p,-1))+m*u(-m,p,-1)),\ 0,\ "|[p+m]u(-)|=0",\ passed) } 
   {\tt call \ expect \ (abs(f\_vf(c\_one,vp,v(-m,p,+1))-m*v(-m,p,+1)),\ 0,\ "|[p-m]v(+)|=0",\ passed) } 
   {\tt call \ expect \ (abs(f\_vf(c\_one,vp,v(-m,p,-1))-m*v(-m,p,-1)),\ 0,\ "|[p-m]v(-)|=0",\ passed) } 
   \texttt{call expect } (\texttt{abs}(\texttt{f_fv}(\texttt{c_one,ubar}(-\texttt{m,p,+1}),\texttt{vp}) + \texttt{m*ubar}(-\texttt{m,p,+1})), \texttt{ 0, "|ubar}(+)[\texttt{p+m}] | \texttt{=0", passed}) 
   \texttt{call expect (abs(f\_fv(c\_one,ubar(-m,p,-1),vp)+m*ubar(-m,p,-1)), 0, "|ubar(-)[p+m]|=0", passed) } 
   \texttt{call expect (abs(f\_fv(c\_one,vbar(-m,p,+1),vp)-m*vbar(-m,p,+1)), 0, "|vbar(+)[p-m]|=0", passed) } 
   \texttt{call expect (abs(f\_fv(c\_one,vbar(-m,p,-1),vp)-m*vbar(-m,p,-1)), 0, "|vbar(-)[p-m]|=0", passed) } \\
\langle Test \text{ omega95} \rangle + \equiv
  print *, "*** Spin Sums"
  test_psi%a = [one, two, three, four]
  test_spinor1 = f_vf (c_one, vp, test_psi) + m * test_psi
  test_spinor2 = u (m, p, +1) * (ubar (m, p, +1) * test_psi) + &
                   u (m, p, -1) * (ubar (m, p, -1) * test_psi)
  do i = 1, 4
    call expect (test_spinor1%a(i), test_spinor2%a(i), "(p+m)1=(sum u ubar)1", passed)
  test_spinor1 = f_vf (c_one, vp, test_psi) - m * test_psi
  test_spinor2 = v (m, p, +1) * (vbar (m, p, +1) * test_psi) + &
                   v (m, p, -1) * (vbar (m, p, -1) * test_psi)
  do i = 1, 4
    call expect (test_spinor1%a(i), test_spinor2%a(i), "(p-m)1=(sum v vbar)1", passed)
  test_psibar%a = [one, two, three, four]
  test_conjspinor1 = f_fv (c_one, test_psibar, vp) - m * test_psibar
  test_conjspinor2 = (test_psibar * v (m, p, +1)) * vbar (m, p, +1) + &
                        (\text{test\_psibar} * v (m, p, -1)) * vbar (m, p, -1)
  do i = 1.4
    call expect (test_conjspinor1%a(i), test_conjspinor2%a(i), "(p-m)1=(sum v vbar)1", passed)
  end do
\langle Test \text{ omega95} \rangle + \equiv
  print *, "*** Checking the normalization ***:"
  call expect (ubar(m,p,+1)*u(m,p,+1), +2*m, "ubar(+)*u(+)=+2m", passed)
  call expect (ubar(m,p,-1)*u(m,p,-1), +2*m, "ubar(-)*u(-)=+2m", passed)
```

```
call expect (vbar(m,p,+1)*v(m,p,+1), -2*m, "vbar(+)*v(+)=-2m", passed)
      call expect (vbar(m,p,-1)*v(m,p,-1), -2*m, "vbar(-)*v(-)=-2m", passed)
                                                                                                                                               0, "ubar(+)*v(+)=0 ", passed)
      call expect (ubar(m,p,+1)*v(m,p,+1),
                                                                                                                                                 0, "ubar(-)*v(-)=0 ", passed)
      call expect (ubar(m,p,-1)*v(m,p,-1),
                                                                                                                                                     0, "vbar(+)*u(+)=0 ", passed)
      call expect (vbar(m,p,+1)*u(m,p,+1),
                                                                                                                                                       0, "vbar(-)*u(-)=0 ", passed)
      call expect (vbar(m,p,-1)*u(m,p,-1),
      print *, "*** Checking the normalization for negative masses***:"
      \label{eq:call_expect} \mbox{call expect (ubar(-m,p,+1)*u(-m,p,+1), -2*m, "ubar(+)*u(+)=-2m", passed)}
      call expect (ubar(-m,p,-1)*u(-m,p,-1), -2*m, "ubar(-)*u(-)=-2m", passed)
       call expect (vbar(-m,p,+1)*v(-m,p,+1), +2*m, "vbar(+)*v(+)=+2m", passed)
       call expect (vbar(-m,p,-1)*v(-m,p,-1), +2*m, "vbar(-)*v(-)=+2m", passed)
                                                                                                                                                               0, "ubar(+)*v(+)=0 ", passed)
      call expect (ubar(-m,p,+1)*v(-m,p,+1),
                                                                                                                                                               0, "ubar(-)*v(-)=0 ", passed)
      call expect (ubar(-m,p,-1)*v(-m,p,-1),
                                                                                                                                                               0, "vbar(+)*u(+)=0 ", passed)
      call expect (vbar(-m,p,+1)*u(-m,p,+1),
                                                                                                                                                               0, "vbar(-)*u(-)=0 ", passed)
      call expect (vbar(-m,p,-1)*u(-m,p,-1),
\langle Test \text{ omega95} \rangle + \equiv
      print *, "*** Checking the currents ***:"
       \texttt{call expect (abs(v\_ff(c\_one,ubar(m,p,+1),u(m,p,+1))-2*vp), 0, "ubar(+).V.u(+)=2p", passed) } 
       \texttt{call expect (abs(v_ff(c_one,ubar(m,p,-1),u(m,p,-1))-2*vp), 0, "ubar(-).V.u(-)=2p", passed) } \\
       {\tt call \ expect \ (abs(v\_ff(c\_one,vbar(m,p,+1),v(m,p,+1))-2*vp),\ 0,\ "vbar(+).V.v(+)=2p",\ passed) } 
       {\tt call \ expect \ (abs(v\_ff(c\_one,vbar(m,p,-1),v(m,p,-1))-2*vp),\ 0,\ "vbar(-).V.v(-)=2p",\ passed) } 
      print *, "*** Checking the currents for negative masses***:"
      call expect (abs(v_ff(c_one,ubar(-m,p,+1),u(-m,p,+1))-2*vp), 0, "ubar(+).V.u(+)=2p", passed)
       {\tt call \ expect \ (abs(v_ff(c_one,ubar(-m,p,-1),u(-m,p,-1))-2*vp),\ 0,\ "ubar(-).V.u(-)=2p",\ passed) } 
       {\tt call \ expect \ (abs(v\_ff(c\_one,vbar(-m,p,+1),v(-m,p,+1))-2*vp),\ 0,\ "vbar(+).V.v(+)=2p",\ passed) } 
       {\tt call \ expect \ (abs(v_ff(c_one,vbar(-m,p,-1),v(-m,p,-1))-2*vp),\ 0,\ "vbar(-).V.v(-)=2p",\ passed) } 
\langle Test \text{ omega95} \rangle + \equiv
      print *, "*** Checking current conservation ***:"
       \texttt{call expect ((vp-vq)*v\_ff(c\_one,ubar(m,p,+1),u(m,q,+1)), 0, "d(ubar(+).V.u(+))=0", passed) } 
       \texttt{call expect ((vp-vq)*v\_ff(c\_one,ubar(m,p,-1),u(m,q,-1)), 0, "d(ubar(-).V.u(-))=0", passed) } 
       \texttt{call expect ((vp-vq)*v\_ff(c\_one,vbar(m,p,+1),v(m,q,+1)), 0, "d(vbar(+).V.v(+))=0", passed) } 
       \texttt{call expect ((vp-vq)*v\_ff(c\_one,vbar(m,p,-1),v(m,q,-1)), 0, "d(vbar(-).V.v(-))=0", passed) } 
      print *, "*** Checking current conservation for negative masses***:"
       call expect ((vp-vq)*v_ff(c_one,ubar(-m,p,+1),u(-m,q,+1)), 0, "d(ubar(+).V.u(+))=0", passed)
       {\tt call \ expect \ ((vp-vq)*v\_ff(c\_one,ubar(-m,p,-1),u(-m,q,-1)),\ 0,\ "d(ubar(-).V.u(-))=0",\ passed) } 
       {\tt call \ expect \ ((vp-vq)*v\_ff(c\_one,vbar(-m,p,-1),v(-m,q,-1)),\ 0,\ "d(vbar(-).V.v(-))=0",\ passed) } 
\langle Test \text{ omega95} \rangle + \equiv
      if (m == 0) then
                 print *, "*** Checking axial current conservation ***:"
                  {\tt call \ expect \ ((vp-vq)*a\_ff(c\_one,ubar(m,p,+1),u(m,q,+1)),\ 0,\ "d(ubar(+).A.u(+))=0",\ passed) } 
                  {\tt call \ expect \ ((vp-vq)*a\_ff(c\_one,ubar(m,p,-1),u(m,q,-1)),\ 0,\ "d(ubar(-).A.u(-))=0",\ passed) } 
                  \texttt{call expect ((vp-vq)*a\_ff(c\_one,vbar(m,p,+1),v(m,q,+1)), 0, "d(vbar(+).A.v(+))=0", passed) } 
                  {\tt call \ expect \ ((vp-vq)*a\_ff(c\_one,vbar(m,p,-1),v(m,q,-1)),\ 0,\ "d(vbar(-).A.v(-))=0",\ passed) } 
      end if
\langle Test \text{ omega95} \rangle + \equiv
      print *, "*** Checking implementation of the sigma vertex funktions ***:"
      call expect ((vp*tvam_ff(c_one,c_nil,ubar(m,p,+1),u(m,q,+1),q) - (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1))), 0, & (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1))), 0, & (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1))), 0, & (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1)), (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1))), 0, & (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1))), 0, (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1)))), 0, (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1)))), 0, (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1)))), 0, (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1))), 0, (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1)))), 0, (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1)*u(m,q,+1))), 0, (p*q-m**
                                                       p*[ubar(p,+).(Isigma*q).u(q,+)] - (p*q-m^2)*ubar(p,+).u(q,+) = 0", passed)
       \text{call expect } ((vp*tvam\_ff(c\_one,c\_nil,ubar(m,p,-1),u(m,q,-1),q) - (p*q-m**2)*(ubar(m,p,-1)*u(m,q,-1))), \ 0, \ \& \ (vp*tvam\_ff(c\_one,c\_nil,ubar(m,p,-1),u(m,q,-1),q) - (p*q-m**2)*(ubar(m,p,-1)*u(m,q,-1))), \ 0, \ \& \ (vp*tvam\_ff(c\_one,c\_nil,ubar(m,p,-1),u(m,q,-1),q)), \ (vp*tvam\_ff(c\_one,c\_nil,ubar(m,p,-1),u(m,q,-1),u(m,q,-1),q)), \ (vp*tvam\_ff(c\_one,c\_nil,ubar(m,p,-1),u(m,q,-1),u(m,q,-1),u(m,q,-1),u(m,q,-1),u(m,q,-1),u(m,q,-1),u(m,q,-1),u(m,q,-1),u(m,q,-1),u
                                                       p*[ubar(p,-).(Isigma*q).u(q,-)] - (p*q-m^2)*ubar(p,-).u(q,-) = 0", passed)
       \texttt{call expect ((vp*tvam\_ff(c\_one, c\_nil, vbar(m, p, +1), v(m, q, +1), q) - (p*q-m**2)*(vbar(m, p, +1)*v(m, q, +1))), \ 0, \ \& \ (\texttt{vbar}(m, p, +1) + \texttt{v}(m, q, +1))), \ 0, \ \& \ (\texttt{vbar}(m, p, +1) + \texttt{v}(m, q, +1))), \ 0, \ \& \ (\texttt{vbar}(m, p, +1) + \texttt{v}(m, q, +1))), \ 0, \ \& \ (\texttt{vbar}(m, p, +1) + \texttt{v}(m, q, +1))), \ 0, \ \& \ (\texttt{vbar}(m, p, +1) + \texttt{v}(m, q, +1))), \ 0, \ \& \ (\texttt{vbar}(m, p, +1) + \texttt{v}(m, q, +1))), \ 0, \ \& \ (\texttt{vbar}(m, p, +1) + \texttt{v}(m, q, +1))), \ 0, \ \& \ (\texttt{vbar}(m, p, +1) + \texttt{v}(m, q, +1))), \ 0, \ \& \ (\texttt{vbar}(m, p, +1) + \texttt{v}(m, q, +1))), \ (\texttt{vbar}(m, p, +1) + \texttt{vbar}(m, p, +1)))), \ (\texttt{vbar}(m, p, +1) + \texttt{vbar}(m, p, +1)))), \ (\texttt{vbar}(m, p, +1) + \texttt{vbar}(m, p, +1)))), \ (\texttt{vbar}(m, p, +1) + \texttt{vbar}(m, p, +1))))), \ (\texttt{vbar}(m, p, +1)
                                                       "p*[vbar(p,+).(Isigma*q).v(q,+)] - (p*q-m^2)*vbar(p,+).v(q,+) = 0", passed)
       \text{call expect } ((\text{vp*tvam\_ff}(c\_one,c\_nil,vbar(m,p,-1),v(m,q,-1),q}) - (p*q-m**2)*(\text{vbar}(m,p,-1)*v(m,q,-1))), \ 0, \ \& \ (p*q-m**2)*(\text{vbar}(m,p,-1)*v(m,q,-1)), \ (p*q-m**2)*(\text{vbar}(m,p,-1)*v(m,q,-1)), \ (p*q-m**2)*(\text{vbar}(m,p,-1)*v(m,q,-1))), \ (p*q-m**2)*(\text{vbar}(m,p,-1)*v(m,q,-1)))), \ (p*q-m**2)*(\text{vbar}(m,p,-1)*v(m,q,-1))), \ (p*q-m**2)*(\text{vbar}(m,p,-1)*v(m,q,-1)))), \ (p*q-m**2)*(\text{vbar}(m,p,-1)*v(m,q,-1)))), \ (p*q-m**2)*(\text{vbar}(m,p,-1)*v(m,q,-1)))), \ (p*q-m**2)*(\text{vbar}(m,p,-1)*v(m,q,-1))), \ (p*q-m**2)*(\text{vbar
                                                       p*[vbar(p,-).(Isigma*q).v(q,-)] - (p*q-m^2)*vbar(p,-).v(q,-) = 0", passed)
       \texttt{call expect ((ubar(m,p,+1)*f\_tvamf(c\_one,c\_nil,vp,u(m,q,+1),q) - (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1))), \ 0, \ \& \ (p*q-m**2)*(ubar(m,p,+1)*u(m,q,+1))), \ 0, \ (p*q-m**2)*(ubar(m
                                                       "ubar(p,+).[p*(Isigma*q).u(q,+)] - (p*q-m^2)*ubar(p,+).u(q,+) = 0", passed)
       call expect ((ubar(m,p,-1)*f_tvamf(c_one,c_nil,vp,u(m,q,-1),q) - (p*q-m**2)*(ubar(m,p,-1)*u(m,q,-1))), 0, &
                                                       "ubar(p,-).[p*(Isigma*q).u(q,-)] - (p*q-m^2)*ubar(p,-).u(q,-) = 0", passed)
       call expect ((vbar(m,p,+1)*f_tvamf(c_one,c_nil,vp,v(m,q,+1),q) - (p*q-m**2)*(vbar(m,p,+1)*v(m,q,+1))), 0, &
                                                       "vbar(p,+).[p*(Isigma*q).v(q,+)] - (p*q-m^2)*vbar(p,+).v(q,+) = 0", passed)
       call expect ((vbar(m,p,-1)*f_tvamf(c_one,c_nil,vp,v(m,q,-1),q) - (p*q-m**2)*(vbar(m,p,-1)*v(m,q,-1))), 0, &
                                                       "vbar(p,-).[p*(Isigma*q).v(q,-)] - (p*q-m^2)*vbar(p,-).v(q,-) = 0", passed)
       \text{call expect } ((f_ftvam(c_one, c_nil, ubar(m, p, +1), vp, q)*u(m, q, +1) - (p*q-m**2)*(ubar(m, p, +1)*u(m, q, +1))), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ 0, \ \& tuber(m, p, +1)*u(m, q, +1)), \ u(m, q, +1)*u(m, q, +1)), \ u(m, q, +1)*u(m, q, +1)), \ u(m, q, +1)*u(m, q, +1)*u(m, q, +1)), \ u(m, q, +1)*u(m, q, +1)), \ u(m, q, +1)*u(m, q, +1)), \ u(m, q, +1)*u(m, q, +1)*u(m, q, +1)), \ u(m, q, +1)*u(m, q, +1)*u(m, q, +1)*u(m, q, +1)), \ u(m, q, +1)*u(m, q, +1)*u(m,
```

```
[ubar(p,+).p*(Isigma*q)].u(q,+) - (p*q-m^2)*ubar(p,+).u(q,+) = 0", passed)
     call expect ((f_ftvam(c_one,c_nil,ubar(m,p,-1),vp,q)*u(m,q,-1) - (p*q-m**2)*(ubar(m,p,-1)*u(m,q,-1))), 0, & (p*q-m**2)*(ubar(m,p,-1)*u(m,q,-1))), 0, (p*q-m**2)*(ubar(m,p,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1))), 0, (p*q-m**2)*(ubar(m,p,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u(m,q,-1)*u
                                       [ubar(p,-).p*(Isigma*q)].u(q,-) - (p*q-m^2)*ubar(p,-).u(q,-) = 0", passed)
      \texttt{call expect ((f_ftvam(c_one,c_nil,vbar(m,p,+1),vp,q)*v(m,q,+1) - (p*q-m**2)*(vbar(m,p,+1)*v(m,q,+1))), 0, \& (p*q-m**2)*(vbar(m,p,+1)*v(m,q,+1))), 0, \& (p*q-m**2)*(vbar(m,p,+1)*v(m,q,+1))), 0, \& (p*q-m**2)*(vbar(m,p,+1)*v(m,q,+1)), 0, \&
                                       [vbar(p,+).p*(Isigma*q)].v(q,+) - (p*q-m^2)*vbar(p,+).v(q,+) = 0", passed)
      \texttt{call expect ((f\_ftvam(c\_one, c\_nil, vbar(m,p,-1), vp,q)*v(m,q,-1) - (p*q-m**2)*(vbar(m,p,-1)*v(m,q,-1))), 0, \& was a substitution of the property of the 
                                        "[vbar(p,-).p*(Isigma*q)].v(q,-) - (p*q-m^2)*vbar(p,-).v(q,-) = 0", passed)
     call expect ((vp*tvam_ff(c_nil,c_one,ubar(m,p,+1),u(m,q,+1),q) - (p*q+m**2)*p_ff(c_one,ubar(m,p,+1),u(m,q,+1))),
                                        "p*[ubar(p,+).(Isigma*q).g5.u(q,+)] - (p*q+m^2)*ubar(p,+).g5.u(q,+) = 0", passed)
     call expect ((vp*tvam_ff(c_nil,c_one,ubar(m,p,-1),u(m,q,-1),q) - (p*q+m**2)*p_ff(c_one,ubar(m,p,-1),u(m,q,-1))),
                                       "p*[ubar(p,-).(Isigma*q).g5.u(q,-)] - (p*q+m^2)*ubar(p,-).g5.u(q,-) = 0", passed)
     call expect ((vp*tvam_ff(c_nil,c_one,vbar(m,p,+1),v(m,q,+1),q) - (p*q+m**2)*p_ff(c_one,vbar(m,p,+1),v(m,q,+1))),
                                       "p*[vbar(p,+).(Isigma*q).g5.v(q,+)] - (p*q+m^2)*vbar(p,+).g5.v(q,+) = 0", passed)
     call expect ((vp*tvam_ff(c_nil,c_one,vbar(m,p,-1),v(m,q,-1),q) - (p*q+m**2)*p_ff(c_one,vbar(m,p,-1),v(m,q,-1))),
                                       p*[vbar(p,-).(Isigma*q).g5.v(q,-)] - (p*q+m^2)*vbar(p,-).g5.v(q,-) = 0", passed)
     call expect ((ubar(m,p,+1)*f_tvamf(c_nil,c_one,vp,u(m,q,+1),q) - (p*q+m**2)*p_ff(c_one,ubar(m,p,+1),u(m,q,+1))),
                                       "p*[ubar(p,+).(Isigma*q).g5.u(q,+)] - (p*q+m^2)*ubar(p,+).g5.u(q,+) = 0", passed)
      {\tt call \ expect \ ((ubar(m,p,-1)*f\_tvamf(c\_nil,c\_one,vp,u(m,q,-1),q) - (p*q+m**2)*p\_ff(c\_one,ubar(m,p,-1),u(m,q,-1))), } 
                                       p*[ubar(p,-).(Isigma*q).g5.u(q,-)] - (p*q+m^2)*ubar(p,-).g5.u(q,-) = 0", passed)
     call expect ((vbar(m,p,+1)*f_tvamf(c_nil,c_one,vp,v(m,q,+1),q) - (p*q+m**2)*p_ff(c_one,vbar(m,p,+1),v(m,q,+1))),
                                       p*[vbar(p,+).(Isigma*q).g5.v(q,+)] - (p*q+m^2)*vbar(p,+).g5.v(q,+) = 0", passed)
      {\tt call \ expect \ ((vbar(m,p,-1)*f\_tvamf(c\_nil,c\_one,vp,v(m,q,-1),q) - (p*q+m**2)*p\_ff(c\_one,vbar(m,p,-1),v(m,q,-1))), } 
                                       p*[vbar(p,-).(Isigma*q).g5.v(q,-)] - (p*q+m^2)*vbar(p,-).g5.v(q,-) = 0", passed)
     call expect ((f_ftvam(c_nil,c_one,ubar(m,p,+1),vp,q)*u(m,q,+1) - (p*q+m**2)*p_ff(c_one,ubar(m,p,+1),u(m,q,+1))),
                                       "p*[ubar(p,+).(Isigma*q).g5.u(q,+)] - (p*q+m^2)*ubar(p,+).g5.u(q,+) = 0", passed)
      \text{call expect } ((f_ftvam(c_nil,c_one,ubar(m,p,-1),vp,q)*u(m,q,-1) - (p*q+m**2)*p_ff(c_one,ubar(m,p,-1),u(m,q,-1))), \\
                                       "p*[ubar(p,-).(Isigma*q).g5.u(q,-)] - (p*q+m^2)*ubar(p,-).g5.u(q,-) = 0", passed)
      {\tt call \ expect \ ((f_ftvam(c_nil,c_one,vbar(m,p,+1),vp,q)*v(m,q,+1) - (p*q+m**2)*p_ff(c_one,vbar(m,p,+1),v(m,q,+1))), } 
                                       p*[vbar(p,+).(Isigma*q).g5.v(q,+)] - (p*q+m^2)*vbar(p,+).g5.v(q,+) = 0", passed)
      \text{call expect } ((f\_\texttt{ftvam}(c\_\texttt{nil}, c\_\texttt{one}, \texttt{vbar}(\texttt{m}, \texttt{p}, -1), \texttt{vp}, \texttt{q}) * \texttt{v}(\texttt{m}, \texttt{q}, -1) - (p*\texttt{q}+\texttt{m}**2) * p\_\texttt{ff}(c\_\texttt{one}, \texttt{vbar}(\texttt{m}, \texttt{p}, -1), \texttt{v}(\texttt{m}, \texttt{q}, -1))), \\
                                       p*[vbar(p,-).(Isigma*q).g5.v(q,-)] - (p*q+m^2)*vbar(p,-).g5.v(q,-) = 0", passed)
\langle Test \text{ omega95} \rangle + \equiv
     print *, "*** Checking polarisation vectors: ***"
     call expect (conjg(eps(m,p, 1))*eps(m,p, 1), -1, "e(1).e(1)=-1", passed)
     call expect (conjg(eps(m,p, 1))*eps(m,p,-1), 0, "e( 1).e(-1)= 0", passed)
     call expect (conjg(eps(m,p,-1))*eps(m,p, 1), 0, "e(-1).e(1)=0", passed)
    call expect (conjg(eps(m,p,-1))*eps(m,p,-1), -1, "e(-1).e(-1)=-1", passed)
                                                                                   p*eps(m,p, 1), 0, "
                                                                                                                                                 p.e( 1)= 0", passed)
    call expect (
                                                                                   p*eps(m,p,-1), 0, "
    call expect (
                                                                                                                                                 p.e(-1)=0", passed)
     if (m > 0) then
            call expect (conjg(eps(m,p, 1))*eps(m,p, 0), 0, "e( 1).e( 0)= 0", passed)
            call expect (conjg(eps(m,p, 0))*eps(m,p, 1), 0, "e( 0).e( 1)= 0", passed)
             call expect (conjg(eps(m,p, 0))*eps(m,p, 0), -1, "e( 0).e( 0)=-1", passed)
            call expect (conjg(eps(m,p, 0))*eps(m,p,-1), 0, "e( 0).e(-1)= 0", passed)
            call expect (conjg(eps(m,p,-1))*eps(m,p, 0), 0, "e(-1).e( 0)= 0", passed)
                                                                                         p*eps(m,p, 0), 0, " p.e(0)=0", passed)
             call expect (
     end if
\langle Test \text{ omega95} \rangle + \equiv
    print *, "*** Checking epsilon tensor: ***"
     call expect ( pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,p,0),eps(m,q,0)), &
                                          \cdot \text{ pseudo\_scalar(eps(m,q,1),eps(m,p,1),eps(m,p,0),eps(m,q,0)), "eps(1<->2)", passed)}
      {\tt call \ expect \ ( \ pseudo\_scalar(eps(m,p,1),eps(m,q,1),eps(m,p,0),eps(m,q,0)), \ \& \ } 
                                           pseudo_scalar(eps(m,p,0),eps(m,q,1),eps(m,p,1),eps(m,q,0)), "eps(1<->3)", passed)
      \texttt{call expect ( pseudo\_scalar(eps(m,p,1),eps(m,q,1),eps(m,p,0),eps(m,q,0)), \& } \\
                                            pseudo_scalar(eps(m,q,0),eps(m,q,1),eps(m,p,0),eps(m,p,1)), "eps(1<->4)", passed)
     call expect ( pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,p,0),eps(m,q,0)), &
                                           pseudo\_scalar(eps(m,p,1),eps(m,p,0),eps(m,q,1),eps(m,q,0)), \ "eps(2<->3)", \ passed)
     call expect ( pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,p,0),eps(m,q,0)), &
                                         - pseudo_scalar(eps(m,p,1),eps(m,q,0),eps(m,p,0),eps(m,q,1)), "eps(2<->4)", passed)
     call expect ( pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,p,0),eps(m,q,0)), &
                                          \label{eq:pseudo_scalar} $$ pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,q,0),eps(m,p,0)), "eps(3<->4)", passed) $$ $$ pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,q,0),eps(m,p,0)), "eps(3<->4)", passed) $$ $$ $$ pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,q,0),eps(m,p,0)), "eps(3<->4)", passed) $$ $$ $$ pseudo_scalar(eps(m,p,1),eps(m,q,1),eps(m,q,0),eps(m,p,0)), "eps(m,p,0)), "eps(m,p,0), "eps(m,p,0), "eps(m,p,0),eps(m,p,0),eps(m,p,0)), "eps(m,p,0),eps(m,p,0),eps(m,p,0)), "eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),eps(m,p,0),
      \texttt{call expect ( pseudo\_scalar(eps(m,p,1),eps(m,q,1),eps(m,p,0),eps(m,q,0)), \& } \\
                                            eps(m,p,1)*pseudo_vector(eps(m,q,1),eps(m,p,0),eps(m,q,0)), "eps'", passed)
```

```
\frac{1}{2}[x\wedge y]_{\mu\nu}^*[x\wedge y]^{\mu\nu} = \frac{1}{2}(x_\mu^*y_\nu^* - x_\nu^*y_\mu^*)(x^\mu y^\nu - x^\nu y^\mu) = (x^*x)(y^*y) - (x^*y)(y^*x)
                                                                                                         (Y.143)
\langle Test \text{ omega95} \rangle + \equiv
  print *, "*** Checking tensors: ***"
  call expect (conjg(p.wedge.q)*(p.wedge.q), (p*p)*(q*q)-(p*q)**2, &
        [p,q].[q,p]=p.p*q.q-p.q^2", passed)
  "[p,q].[q,p]=p.q^2-p.p*q.q", passed)
i.e.
                                     \frac{1}{2}[p \wedge \epsilon(p,i)]_{\mu\nu}^*[p \wedge \epsilon(p,j)]^{\mu\nu} = -p^2 \delta_{ij}
                                                                                                         (Y.144)
\langle Test \text{ omega95} \rangle + \equiv
  call expect (conjg(p.wedge.eps(m,p, 1))*(p.wedge.eps(m,p, 1)), -p*p, &
        "[p,e(1)].[p,e(1)]=-p.p", passed)
  call expect (conjg(p.wedge.eps(m,p, 1))*(p.wedge.eps(m,p,-1)),
        [p,e(1)].[p,e(-1)]=0, passed)
  call expect (conjg(p.wedge.eps(m,p,-1))*(p.wedge.eps(m,p, 1)),
        "[p,e(-1)].[p,e(1)]=0", passed)
  call expect (conjg(p.wedge.eps(m,p,-1))*(p.wedge.eps(m,p,-1)), -p*p, &
        "[p,e(-1)].[p,e(-1)]=-p.p", passed)
  if (m > 0) then
     call expect (conjg(p.wedge.eps(m,p, 1))*(p.wedge.eps(m,p, 0)),
           "[p,e(1)].[p,e(0)]=0", passed)
     call expect (conjg(p.wedge.eps(m,p, 0))*(p.wedge.eps(m,p, 1)),
           "[p,e( 0)].[p,e( 1)]=0", passed)
     call expect (conjg(p.wedge.eps(m,p, 0))*(p.wedge.eps(m,p, 0)), -p*p, &
           "[p,e(0)].[p,e(0)]=-p.p", passed)
     call expect (conjg(p.wedge.eps(m,p, 0))*(p.wedge.eps(m,p,-1)),
           "[p,e(1)].[p,e(-1)]=0", passed)
     \verb|call expect (conjg(p.wedge.eps(m,p,-1))*(p.wedge.eps(m,p, 0)), \\
                                                                              0, &
           "[p,e(-1)].[p,e(0)]=0", passed)
  end if
also
                                         [x \wedge y]_{\mu\nu}z^{\nu} = x_{\mu}(yz) - y_{\mu}(xz)
                                                                                                         (Y.145)
                                         z_{\mu}[x \wedge y]^{\mu\nu} = (zx)y^{\nu} - (zy)x^{\nu}
                                                                                                         (Y.146)
\langle Test \text{ omega95} \rangle + \equiv
  call expect (abs ((p.wedge.eps(m,p, 1))*p + (p*p)*eps(m,p, 1)), 0, &
        "[p,e(1)].p=-p.p*e(1)]", passed)
  call expect (abs ((p.wedge.eps(m,p, 0))*p + (p*p)*eps(m,p, 0)), 0, &
        "[p,e(0)].p=-p.p*e(0)]", passed)
  call expect (abs ((p.wedge.eps(m,p,-1))*p + (p*p)*eps(m,p,-1)), 0, &
        [p,e(-1)].p=-p.p*e(-1)], passed)
  call expect (abs (p*(p.wedge.eps(m,p, 1)) - (p*p)*eps(m,p, 1)), 0, &
        "p.[p,e(1)]=p.p*e(1)]", passed)
  call expect (abs (p*(p.wedge.eps(m,p, 0)) - (p*p)*eps(m,p, 0)), 0, &
        "p.[p,e( 0)]=p.p*e( 0)]", passed)
  call expect (abs (p*(p.wedge.eps(m,p,-1)) - (p*p)*eps(m,p,-1)), 0, &
        "p.[p,e(-1)]=p.p*e(-1)]", passed)
\langle Test \text{ omega95} \rangle + \equiv
  print *, "*** Checking polarisation tensors: ***"
  call expect (conjg(eps2(m,p, 2))*eps2(m,p, 2), 1, "e2( 2).e2( 2)=1", passed)
  call expect (conjg(eps2(m,p, 2))*eps2(m,p,-2), 0, "e2( 2).e2(-2)=0", passed)
  call expect (conjg(eps2(m,p,-2))*eps2(m,p, 2), 0, "e2(-2).e2( 2)=0", passed)
  call expect (conjg(eps2(m,p,-2))*eps2(m,p,-2), 1, "e2(-2).e2(-2)=1", passed)
  if (m > 0) then
     {\tt call \ expect \ (conjg(eps2(m,p,\ 2))*eps2(m,p,\ 1),\ 0,\ "e2(\ 2).e2(\ 1)=0",\ passed)}
     call expect (conjg(eps2(m,p, 2))*eps2(m,p, 0), 0, "e2( 2).e2( 0)=0", passed)
     call expect (conjg(eps2(m,p, 2))*eps2(m,p,-1), 0, "e2( 2).e2(-1)=0", passed)
     call expect (conjg(eps2(m,p, 1))*eps2(m,p, 2), 0, "e2( 1).e2( 2)=0", passed)
     call expect (conjg(eps2(m,p, 1))*eps2(m,p, 1), 1, "e2( 1).e2( 1)=1", passed)
     call expect (conjg(eps2(m,p, 1))*eps2(m,p, 0), 0, "e2(1).e2(0)=0", passed)
     call expect (conjg(eps2(m,p, 1))*eps2(m,p,-1), 0, "e2( 1).e2(-1)=0", passed)
```

```
call expect (conjg(eps2(m,p, 1))*eps2(m,p,-2), 0, "e2(1).e2(-2)=0", passed)
     call expect (conjg(eps2(m,p, 0))*eps2(m,p, 2), 0, "e2( 0).e2( 2)=0", passed)
     {\tt call \ expect \ (conjg(eps2(m,p,\ 0))*eps2(m,p,\ 1),\ 0,\ "e2(\ 0).e2(\ 1)=0",\ passed)}
     call expect (conjg(eps2(m,p, 0))*eps2(m,p, 0), 1, "e2( 0).e2( 0)=1", passed)
      {\tt call \ expect \ (conjg(eps2(m,p,\ 0))*eps2(m,p,-1),\ 0,\ "e2(\ 0).e2(-1)=0",\ passed) } 
     call expect (conjg(eps2(m,p, 0))*eps2(m,p,-2), 0, "e2( 0).e2(-2)=0", passed)
     \label{eq:call_expect} $$ $$ {\rm call\ expect\ (conjg(eps2(m,p,-1))*eps2(m,p,\ 2),\ 0,\ "e2(-1).e2(\ 2)=0",\ passed) $$ $$ $$ $$ $$
     call expect (conjg(eps2(m,p,-1))*eps2(m,p, 1), 0, "e2(-1).e2( 1)=0", passed)
     call expect (conjg(eps2(m,p,-1))*eps2(m,p, 0), 0, "e2(-1).e2( 0)=0", passed)
     call expect (conjg(eps2(m,p,-1))*eps2(m,p,-1), 1, "e2(-1).e2(-1)=1", passed)
     call expect (conjg(eps2(m,p,-1))*eps2(m,p,-2), 0, "e2(-1).e2(-2)=0", passed)
     call expect (conjg(eps2(m,p,-2))*eps2(m,p, 1), 0, "e2(-2).e2(1)=0", passed)
     call expect (conjg(eps2(m,p,-2))*eps2(m,p, 0), 0, "e2(-2).e2( 0)=0", passed)
     call expect (conjg(eps2(m,p,-2))*eps2(m,p,-1), 0, "e2(-2).e2(-1)=0", passed)
  end if
\langle \mathit{Test} \ \mathtt{omega95} \rangle + \equiv
 call expect (
                           abs(p*eps2(m,p, 2)), 0, "|p.e2(2)| =0", passed)
                             abs(eps2(m,p, 2)*p), 0, " | e2(2).p|=0", passed)
  call expect (
                           abs(p*eps2(m,p,-2)), 0, "|p.e2(-2)| =0", passed)
 call expect (
                             abs(eps2(m,p,-2)*p), 0, "
  call expect (
                                                         |e2(-2).p|=0", passed)
  if (m > 0) then
                              abs(p*eps2(m,p, 1)), 0, "|p.e2(1)| =0", passed)
     call expect (
                                abs(eps2(m,p, 1)*p), 0, "
     call expect (
                                                             |e2( 1).p|=0", passed)
                              abs(p*eps2(m,p, 0)), 0, "|p.e2(0)| =0", passed)
     call expect (
                                abs(eps2(m,p, 0)*p), 0, "
     call expect (
                                                             |e2( 0).p|=0", passed)
                              abs(p*eps2(m,p,-1)), 0, "|p.e2(-1)| =0", passed)
     call expect (
                                abs(eps2(m,p,-1)*p), 0, " | e2(-1).p|=0", passed)
     call expect (
  end if
\langle XXX \; Test \; \text{omega95} \rangle \equiv
 print *, " *** Checking the polarization tensors for massive gravitons:"
  call expect (abs(p * eps2(m,p,2)), 0, "p.e(+2)=0", passed)
 call expect (abs(p * eps2(m,p,1)), 0, "p.e(+1)=0", passed) call expect (abs(p * eps2(m,p,0)), 0, "p.e(0)=0", passed)
  call expect (abs(p * eps2(m,p,-1)), 0, "p.e(-1)=0", passed)
  call expect (abs(p * eps2(m,p,-2)), 0, "p.e(-2)=0", passed)
  call expect (abs(trace(eps2 (m,p,2))), 0, "Tr[e(+2)]=0", passed)
  call expect (abs(trace(eps2 (m,p,1))), 0, "Tr[e(+1)]=0", passed)
  call expect (abs(trace(eps2 (m,p,0))), 0, "Tr[e(0)]=0", passed)
  call expect (abs(trace(eps2 (m,p,-1))), 0, "Tr[e(-1)]=0", passed)
  call expect (abs(trace(eps2 (m,p,-2))), 0, "Tr[e(-2)]=0", passed)
  call expect (abs(eps2(m,p,2) * eps2(m,p,2)),
       "e(2).e(2)
                   = 1", passed)
  call expect (abs(eps2(m,p,2) * eps2(m,p,1)),
                   = 0", passed)
       "e(2).e(1)
  call expect (abs(eps2(m,p,2) * eps2(m,p,0)),
       "e(2).e(0)
                   = 0", passed)
  call expect (abs(eps2(m,p,2) * eps2(m,p,-1)),
       "e(2).e(-1) = 0", passed)
  call expect (abs(eps2(m,p,2) * eps2(m,p,-2)),
       "e(2).e(-2) = 0", passed)
  call expect (abs(eps2(m,p,1) * eps2(m,p,1)),
                   = 1", passed)
       "e(1).e(1)
  call expect (abs(eps2(m,p,1) * eps2(m,p,0)),
       "e(1).e(0)
                    = 0", passed)
  call expect (abs(eps2(m,p,1) * eps2(m,p,-1)), 0, &
       "e(1).e(-1) = 0", passed)
  call expect (abs(eps2(m,p,1) * eps2(m,p,-2)), 0, &
       "e(1).e(-2) = 0", passed)
  call expect (abs(eps2(m,p,0) * eps2(m,p,0)),
       "e(0).e(0)
                   = 1", passed)
  call expect (abs(eps2(m,p,0) * eps2(m,p,-1)), 0, &
       "e(0).e(-1) = 0", passed)
  call expect (abs(eps2(m,p,0) * eps2(m,p,-2)), 0, &
       "e(0).e(-2) = 0", passed)
  call expect (abs(eps2(m,p,-1) * eps2(m,p,-1)), 1, &
```

```
"e(-1).e(-1) = 1", passed)
  call expect (abs(eps2(m,p,-1) * eps2(m,p,-2)), 0, &
        "e(-1).e(-2) = 0", passed)
  call expect (abs(eps2(m,p,-2) * eps2(m,p,-2)), 1, &
        "e(-2).e(-2) = 1", passed)
\langle \mathit{Test} \ \mathtt{omega95} \rangle + \equiv
  print *, " *** Checking the graviton propagator:"
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
                 \label{eq:pr_tensor} \texttt{pr\_tensor}(\texttt{p},\texttt{m},\texttt{w},\texttt{eps2}(\texttt{m},\texttt{p},\texttt{-2})))), \ \texttt{0}, \ \texttt{"p.pr.e}(\texttt{-2})\texttt{"}, \ \texttt{passed})
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
                 pr_tensor(p,m,w,eps2(m,p,-1)))), 0, "p.pr.e(-1)", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
                 pr_tensor(p,m,w,eps2(m,p,0)))), 0, "p.pr.e(0)", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
                 pr_tensor(p,m,w,eps2(m,p,1)))), 0, "p.pr.e(1)", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
                 pr_tensor(p,m,w,eps2(m,p,2)))), 0, "p.pr.e(2)", passed)
  call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
                 pr_tensor(p,m,w,ttest))), 0, "p.pr.ttest", passed)
\langle \texttt{test\_omega95\_bispinors.f90} \rangle \equiv
  \langle Copyleft \rangle
  program test_omega95_bispinors
    use kinds
    use omega95_bispinors
    use omega_vspinor_polarizations
    use omega_testtools
    implicit none
    integer :: i, j
    real(kind=default) :: m, pabs, qabs, tabs, zabs, w
    real(kind=default), dimension(4) :: r
    complex(kind=default) :: c_nil, c_one, c_two
    \texttt{type}(\texttt{momentum}) \ :: \ \texttt{p, q, t, z, p\_0}
    type(vector) :: vp, vq, vt, vz
    type(vectorspinor) :: testv
    type(bispinor) :: vv
    logical :: passed
    call random_seed ()
    c_nil = 0.0_default
    c_one = 1.0_default
    c_two = 2.0_default
    w = 1.4142
    m = 13
    pabs = 42
    qabs = 137
    tabs = 84
    zabs = 3.1415
    p_0%t = m
    p_0%x = 0
    call random_momentum (p, pabs, m)
    call random_momentum (q, qabs, m)
    call random_momentum (t, tabs, m)
    call random_momentum (z, zabs, m)
    call random_number (r)
    do i = 1, 4
       testv%psi(1)%a(i) = (0.0_default, 0.0_default)
    end do
    do i = 2, 3
       do j = 1, 4
           testv%psi(i)%a(j) = cmplx (10.0_default * r(j), kind=default)
      end do
    testv%psi(4)%a(1) = (1.0_default, 0.0_default)
    testv\%psi(4)\%a(2) = (0.0_default, 2.0_default)
    testv\%psi(4)\%a(3) = (1.0_default, 0.0_default)
    testv\%psi(4)\%a(4) = (3.0_default, 0.0_default)
```

```
vp = p
    vq = q
    vt = t
    vz = z
    passed = .true.
    vv\%a(1) = (1.0_default, 0.0_default)
    vv\%a(2) = (0.0_default, 2.0_default)
    vv\%a(3) = (1.0_default, 0.0_default)
    vv\%a(4) = (3.0_default, 0.0_default)
    vv = pr_psi(p, m, w, .false., vv)
    ⟨ Test omega95_bispinors⟩
    if (.not. passed) then
      stop 1
    end if
  end program test_omega95_bispinors
\langle \mathit{Test} \ \mathtt{omega95\_bispinors} \rangle \equiv
 print *, "*** Checking the equations of motion ***:"
  {\tt call \ expect \ (abs(f_vf(c_one,vp,u(m,p,+1))-m*u(m,p,+1)),\ 0,\ "|[p-m]u(+)|=0",\ passed) } 
  {\tt call \ expect \ (abs(f\_vf(c\_one,vp,u(m,p,-1))-m*u(m,p,-1)),\ 0,\ "|[p-m]u(-)|=0",\ passed) } 
  {\tt call \ expect \ (abs(f_vf(c_one,vp,v(m,p,+1))+m*v(m,p,+1)),\ 0,\ "|[p+m]v(+)|=0",\ passed) } 
  {\tt call \ expect \ (abs(f_vf(c_one,vp,v(m,p,-1))+m*v(m,p,-1)), \ 0, \ "|[p+m]v(-)|=0", \ passed) } 
 print *, "*** Checking the equations of motion for negative masses***:"
  {\tt call \ expect \ (abs(f_vf(c_one,vp,u(-m,p,+1))+m*u(-m,p,+1)),\ 0,\ "|[p+m]u(+)|=0",\ passed) } 
  {\tt call \ expect \ (abs(f_vf(c_one,vp,u(-m,p,-1))+m*u(-m,p,-1)),\ 0,\ "|[p+m]u(-)|=0",\ passed) } 
  {\tt call \ expect \ (abs(f\_vf(c\_one,vp,v(-m,p,+1))-m*v(-m,p,+1)),\ 0,\ "|[p-m]v(+)|=0",\ passed) } 
  {\tt call \ expect \ (abs(f_vf(c_one,vp,v(-m,p,-1))-m*v(-m,p,-1)),\ 0,\ "|[p-m]v(-)|=0",\ passed) } 
\langle Test \text{ omega95\_bispinors} \rangle + \equiv
 print *, "*** Checking the normalization ***:"
  {\tt call \ expect \ (s\_ff(c\_one,v(m,p,+1),u(m,p,+1)), \ +2*m, \ "ubar(+)*u(+)=+2m", \ passed) } 
  {\tt call \ expect \ (s\_ff(c\_one,v(m,p,-1),u(m,p,-1)), \ +2*m, \ "ubar(-)*u(-)=+2m", \ passed) } 
  {\tt call \ expect \ (s\_ff(c\_one,u(m,p,+1),v(m,p,+1)), \ -2*m, \ "vbar(+)*v(+)=-2m", \ passed) } 
  {\tt call \ expect \ (s\_ff(c\_one,u(m,p,-1),v(m,p,-1)), \ -2*m, \ "vbar(-)*v(-)=-2m", \ passed) } 
                                                        0, "ubar(+)*v(+)=0 ", passed)
  call expect (s_ff(c_one,v(m,p,+1),v(m,p,+1)),
                                                         0, "ubar(-)*v(-)=0 ", passed)
 call expect (s_ff(c_one,v(m,p,-1),v(m,p,-1)),
                                                         0, "vbar(+)*u(+)=0 ", passed)
 call expect (s_ff(c_one,u(m,p,+1),u(m,p,+1)),
                                                         0, "vbar(-)*u(-)=0 ", passed)
 call expect (s_ff(c_one,u(m,p,-1),u(m,p,-1)),
 print *, "*** Checking the normalization for negative masses***:"
 call expect (s_ff(c_one, v(-m, p, +1), u(-m, p, +1)), -2*m, "ubar(+)*u(+)=-2m", passed)
  {\tt call \ expect \ (s_ff(c_one,v(-m,p,-1),u(-m,p,-1)), \ -2*m, \ "ubar(-)*u(-)=-2m", \ passed) } 
  {\tt call \ expect \ (s\_ff(c\_one,u(-m,p,+1),v(-m,p,+1)), \ +2*m, \ "vbar(+)*v(+)=+2m", \ passed) } 
  {\tt call \ expect \ (s\_ff(c\_one,u(-m,p,-1),v(-m,p,-1)), \ +2*m, \ "vbar(-)*v(-)=+2m", \ passed) } 
                                                           0, "ubar(+)*v(+)=0 ", passed)
 call expect (s_ff(c_one,v(-m,p,+1),v(-m,p,+1)),
                                                           0, "ubar(-)*v(-)=0 ", passed)
 call expect (s_ff(c_one,v(-m,p,-1),v(-m,p,-1)),
                                                           0, "vbar(+)*u(+)=0 ", passed)
  call expect (s_ff(c_one,u(-m,p,+1),u(-m,p,+1)),
 call expect (s_ff(c_one,u(-m,p,-1),u(-m,p,-1)),
                                                           0, "vbar(-)*u(-)=0 ", passed)
\langle Test \text{ omega95\_bispinors} \rangle + \equiv
 print *, "*** Checking the currents ***:"
  {\tt call \ expect \ (abs(v\_ff(c\_one,v(m,p,+1),u(m,p,+1))-2*vp),\ 0,\ "ubar(+).V.u(+)=2p",\ passed) } 
 call expect (abs(v_{ff}(c_{one},v(m,p,-1),u(m,p,-1))-2*vp), 0, "ubar(-).V.u(-)=2p", passed)
  {\tt call \ expect \ (abs(v_ff(c_one,u(m,p,+1),v(m,p,+1))-2*vp),\ 0,\ "vbar(+).V.v(+)=2p",\ passed) } 
  {\tt call \ expect \ (abs(v\_ff(c\_one,u(m,p,-1),v(m,p,-1))-2*vp),\ 0,\ "vbar(-).V.v(-)=2p",\ passed) } 
 print *, "*** Checking the currents for negative masses***:"
  {\tt call \ expect \ (abs(v_ff(c_one,v(-m,p,+1),u(-m,p,+1))-2*vp),\ 0,\ "ubar(+).V.u(+)=2p",\ passed) } 
  {\tt call \ expect \ (abs(v_ff(c_one,v(-m,p,-1),u(-m,p,-1))-2*vp),\ 0,\ "ubar(-).V.u(-)=2p",\ passed) } 
  call expect (abs(v_ff(c_one,u(-m,p,+1),v(-m,p,+1))-2*vp), 0, "vbar(+).V.v(+)=2p", passed)
 call expect (abs(v_{ff}(c_{ne},u(-m,p,-1),v(-m,p,-1))-2*vp), 0, "vbar(-).V.v(-)=2p", passed)
\langle Test \text{ omega95\_bispinors} \rangle + \equiv
 print *, "*** Checking current conservation ***:"
  {\tt call \ expect \ ((vp-vq)*v\_ff(c\_one,v(m,p,+1),u(m,q,+1)),\ 0,\ "d(ubar(+).V.u(+))=0",\ passed) } 
  {\tt call \ expect \ ((vp-vq)*v\_ff(c\_one,v(m,p,-1),u(m,q,-1)),\ 0,\ "d(ubar(-).V.u(-))=0",\ passed) } 
  {\tt call \ expect \ ((vp-vq)*v\_ff(c\_one,u(m,p,-1),v(m,q,-1)),\ 0,\ "d(vbar(-).V.v(-))=0",\ passed) } 
\langle Test \text{ omega95\_bispinors} \rangle + \equiv
 print *, "*** Checking current conservation for negative masses***:"
```

```
call expect ((vp-vq)*v_ff(c_one,v(-m,p,+1),u(-m,q,+1)), 0, "d(ubar(+).V.u(+))=0", passed)
          {\tt call \ expect \ ((vp-vq)*v\_ff(c\_one,v(-m,p,-1),u(-m,q,-1)),\ 0,\ "d(ubar(-).V.u(-))=0",\ passed) } 
          {\tt call \ expect \ ((vp-vq)*v\_ff(c\_one,u(-m,p,+1),v(-m,q,+1)),\ 0,\ "d(vbar(+).V.v(+))=0",\ passed) } 
         call expect ((vp-vq)*v_ff(c_one,u(-m,p,-1),v(-m,q,-1)), 0, "d(vbar(-).V.v(-))=0", passed)
\langle \mathit{Test} \ \mathtt{omega95\_bispinors} \rangle + \equiv
         if (m == 0) then
                       print *, "*** Checking axial current conservation ***:"
                       call expect ((vp-vq)*a_ff(c_one,v(m,p,+1),u(m,q,+1)), 0, "d(ubar(+).A.u(+))=0", passed)
                        {\tt call \ expect \ ((vp-vq)*a\_ff(c\_one,v(m,p,-1),u(m,q,-1)),\ 0,\ "d(ubar(-).A.u(-))=0",\ passed) } 
                       call expect ((vp-vq)*a_ff(c_one,u(m,p,+1),v(m,q,+1)), 0, "d(vbar(+).A.v(+))=0", passed)
                        {\tt call \ expect \ ((vp-vq)*a\_ff(c\_one,u(m,p,-1),v(m,q,-1)),\ 0,\ "d(vbar(-).A.v(-))=0",\ passed) } 
         end if
\langle Test \text{ omega95\_bispinors} \rangle + \equiv
        print *, "*** Checking implementation of the sigma vertex funktions ***:"
          \texttt{call expect ((vp*tvam\_ff(c\_one,c\_nil,v(m,p,+1),u(m,q,+1),q) - (p*q-m**2)*(v(m,p,+1)*u(m,q,+1))), 0, \& (a) = (a) + (b) + (
                                                                        "p*[ubar(p,+).(Isigma*q).u(q,+)] - (p*q-m^2)*ubar(p,+).u(q,+) = 0", passed)
          p*[ubar(p,-).(Isigma*q).u(q,-)] - (p*q-m^2)*ubar(p,-).u(q,-) = 0", passed)
          \text{call expect } ((\text{vp*tvam\_ff(c\_one,c\_nil}, \text{u(m,p,+1)}, \text{v(m,q,+1)}, \text{q}) - (\text{p*q-m**2})*(\text{u(m,p,+1)}*\text{v(m,q,+1)})), \text{ 0, \& call expect } (\text{vp*tvam\_ff(c\_one,c\_nil}, \text{u(m,p,+1)}, \text{v(m,q,+1)}, \text{q}) - (\text{p*q-m**2})*(\text{u(m,p,+1)}*\text{v(m,q,+1)})), \text{ 0, \& call expect } (\text{vp*tvam\_ff(c\_one,c\_nil}, \text{u(m,p,+1)}, \text{v(m,q,+1)}, \text{q})) - (\text{p*q-m**2})*(\text{u(m,p,+1)}*\text{v(m,q,+1)})), \text{ 0, \& call expect } (\text{vp*tvam\_ff(c\_one,c\_nil}, \text{u(m,p,+1)}, \text{v(m,q,+1)}, \text{q}))), \text{ 0, \& call expect } (\text{vp*tvam\_ff(c\_one,c\_nil}, \text{u(m,p,+1)}, \text{v(m,q,+1)}, \text{q}))), \text{ 0, \& call expect } (\text{vp*tvam\_ff(c\_one,c\_nil}, \text{u(m,p,+1)}, \text{v(m,q,+1)}, \text{q}))), \text{ 0, \& call expect } (\text{vp*tvam\_ff(c\_one,c\_nil}, \text{u(m,p,+1)}, \text{v(m,q,+1)}, \text{v(m,q,+1)})))), \text{ 0, \& call expect } (\text{vp*tvam\_ff(c\_one,c\_nil}, \text{u(m,p,+1)}, \text{v(m,q,+1)}, \text{v(m,q,+1)}, \text{v(m,q,+1)})))), \text{ 0, \& call expect } (\text{vp*tvam\_ff(c\_one,c\_nil}, \text{u(m,p,+1)}, \text{v(m,q,+1)}, \text{v(m,q,+1)}, \text{v(m,q,+1)}))))))
                                                                       p*[vbar(p,+).(Isigma*q).v(q,+)] - (p*q-m^2)*vbar(p,+).v(q,+) = 0", passed)
          call expect ((vp*tvam_ff(c_one,c_nil,u(m,p,-1),v(m,q,-1),q) - (p*q-m**2)*(u(m,p,-1)*v(m,q,-1))), 0, & (p*q-m**2)*(u(m,p,-1)*v(m,q,-1)), 0, & (p*q-m**2)*(u(m,p,-1)*v(m,q,-1)*v(m,q,-1)), 0, & (p*q-m**2)*(u(m,p,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1
                                                                       p*[vbar(p,-).(Isigma*q).v(q,-)] - (p*q-m^2)*vbar(p,-).v(q,-) = 0", passed)
           \text{call expect } ((v(\texttt{m},\texttt{p},+1)*f\_tvamf(\texttt{c\_one},\texttt{c\_nil},\texttt{vp},\texttt{u}(\texttt{m},\texttt{q},+1),\texttt{q}) - (p*q-m**2)*(v(\texttt{m},\texttt{p},+1)*u(\texttt{m},\texttt{q},+1))), \ 0, \ \& \ (v(\texttt{m},\texttt{p},+1)*v(\texttt{m},\texttt{q},+1))), \ 0, \ \& \ (v(\texttt{m},\texttt{p},+1)*v(\texttt{m},\texttt{q},+1))), \ 0, \ \& \ (v(\texttt{m},\texttt{p},+1)*v(\texttt{m},\texttt{q},+1))), \ (v(\texttt{m},\texttt{p},+1)*v(\texttt{m},\texttt{q},+1))), \ (v(\texttt{m},\texttt{p},+1)*v(\texttt{m},\texttt{q},+1))), \ (v(\texttt{m},\texttt{p},+1)*v(\texttt{m},\texttt{p},+1)*v(\texttt{m},\texttt{p},+1))), \ (v(\texttt{m},\texttt{p},+1)*v(\texttt{m},\texttt{p},+1))), \ (v(\texttt{m},\texttt{p},+1)*v(\texttt{m},+1)), \ (v(\texttt{m},\texttt{p},+1)*v(\texttt{m},+1))), \ (v(\texttt{m},\texttt{p},+1)*v(\texttt{m},+1))), \ (v(\texttt{m},\texttt{p},+1)*v(\texttt{m},+1)), \ (v(\texttt{m},\texttt{p},+1)*v(\texttt{m},+1))), \ (v(\texttt{m},\texttt{p},+1)*v(\texttt{m},+1)), \ (v(\texttt{m},\texttt{p},+1)), \ (v(\texttt{m},\texttt{p},+1)*v(\texttt{m},+1)), \ (v(\texttt{m},\texttt{p}
                                                                       "ubar(p,+).[p*(Isigma*q).u(q,+)] - (p*q-m^2)*ubar(p,+).u(q,+) = 0", passed)
         "ubar(p,-).[p*(Isigma*q).u(q,-)] - (p*q-m^2)*ubar(p,-).u(q,-) = 0", passed)
         call expect ((u(m,p,+1)*f_tvamf(c_one,c_nil,vp,v(m,q,+1),q) - (p*q-m**2)*(u(m,p,+1)*v(m,q,+1))), 0, &
                                                                       "vbar(p,+).[p*(Isigma*q).v(q,+)] - (p*q-m^2)*vbar(p,+).v(q,+) = 0", passed)
          call expect ((u(m,p,-1)*f_tvamf(c_one,c_nil,vp,v(m,q,-1),q) - (p*q-m**2)*(u(m,p,-1)*v(m,q,-1))), 0, & (p*q-m**2)*(u(m,p,-1)*v(m,q,-1))), 0, & (p*q-m**2)*(u(m,p,-1)*v(m,q,-1))), 0, & (p*q-m**2)*(u(m,p,-1)*v(m,q,-1)), (p*q-m**2)*(u(m,p,-1)*v(m,q,-1))), 0, & (p*q-m**2)*(u(m,p,-1)*v(m,q,-1)), (p*q-m**2)*(u(m,p,-1)*v(m,q,-1))), 0, & (p*q-m**2)*(u(m,p,-1)*v(m,q,-1)), (p*q-m**2)*(u(m,p,-1)*v(m,q,-1)), (p*q-m**2)*(u(m,p,-1)*v(m,q,-1))), (p*q-m**2)*(u(m,q,-1)*v(m,q,-1))), (p*q-m**2)*(u(m,q,-1)*v(m,q,-1)*v(m,q,-1))), (p*q-m**2)*(u(m,q,-1)*v(m,q,-1)*v(m,q,-1)*v(m,q,-1)), (p*q-m*
                                                                       "vbar(p,-).[p*(Isigma*q).v(q,-)] - (p*q-m^2)*vbar(p,-).v(q,-) = 0", passed)
          \text{call expect } ((vp*tvam\_ff(c\_nil,c\_one,v(m,p,+1),u(m,q,+1),q) - (p*q+m**2)*p\_ff(c\_one,v(m,p,+1),u(m,q,+1))), \ 0, \ \& \ (vp*tvam\_ff(c\_nil,c\_one,v(m,p,+1),u(m,q,+1),q)) - (p*q+m**2)*p\_ff(c\_one,v(m,p,+1),u(m,q,+1))), \ 0, \ \& \ (vp*tvam\_ff(c\_nil,c\_one,v(m,p,+1),u(m,q,+1),q)) - (p*q+m**2)*p\_ff(c\_one,v(m,p,+1),u(m,q,+1))), \ 0, \ \& \ (vp*tvam\_ff(c\_nil,c\_one,v(m,p,+1),u(m,q,+1),q)) - (p*q+m**2)*p\_ff(c\_one,v(m,p,+1),u(m,q,+1),q)) - (p*q+m**2)*p\_ff(c\_one,v(m,p,+1),u(m,q,+1),q) - (p*q+m**2)*p\_ff(c\_one,v(m,p,+1),u(m,q,+1),q) - (p*q+m**2)*p\_ff(c\_one,v(m,q,+1),q) - (p*q+m
                                                                       "p*[ubar(p,+).(Isigma*q).g5.u(q,+)] - (p*q+m^2)*ubar(p,+).g5.u(q,+) = 0", passed)
          \texttt{call expect ((vp*tvam\_ff(c\_nil,c\_one,v(m,p,-1),u(m,q,-1),q) - (p*q+m**2)*p\_ff(c\_one,v(m,p,-1),u(m,q,-1))), 0, \& (p*q+m**2)*p\_ff(c\_one,v(m,p,-1),u(m,q,-1))), 0, \& (p*q+m**2)*p\_ff(c\_one,v(m,p,-1),u(m,q,-1))), 0, \& (p*q+m**2)*p\_ff(c\_one,v(m,p,-1),u(m,q,-1))), 0, (p*q+m**2)*p\_ff(c\_one,v(m,p,-1),u(m,q,-1))), 0, \& (p*q+m**2)*p\_ff(c\_one,v(m,p,-1),u(m,q,-1))), 0, (p*q+m**2)*p\_ff(c\_one,v(m,q,-1),u(m,q,-1))), 0, (p*q+m**2)*p\_ff(c\_one,v(m,q,-1),u(m,q,-1))), 0, (p*q+m**2)*p\_ff(c\_one,v(m,q,-1),u(m,q,-1))), 0, (p*q+m**2)*p\_ff(c\_one,v(m,q,-1),u(m,q,-1))), 0, (p*q+m**2)*p\_ff(c\_one,v(m,q,-1),u(m,q,-1))), 0, (p*q+m**2)*p\_ff(c\_one,v(m,q,-1))), 0, (p*q+m**2)*p\_
                                                                        p*[ubar(p,-).(Isigma*q).g5.u(q,-)] - (p*q+m^2)*ubar(p,-).g5.u(q,-) = 0", passed)
         call expect ((vp*tvam_ff(c_nil,c_one,u(m,p,+1),v(m,q,+1),q) - (p*q+m**2)*p_ff(c_one,u(m,p,+1),v(m,q,+1))), 0, &
                                                                       p*[vbar(p,+).(Isigma*q).g5.v(q,+)] - (p*q+m^2)*vbar(p,+).g5.v(q,+) = 0", passed)
          call expect ((vp*tvam_ff(c_nil,c_one,u(m,p,-1),v(m,q,-1),q) - (p*q+m**2)*p_ff(c_one,u(m,p,-1),v(m,q,-1))), 0, &
                                                                       p*[vbar(p,-).(Isigma*q).g5.v(q,-)] - (p*q+m^2)*vbar(p,-).g5.v(q,-) = 0", passed)
         call expect ((v(m,p,+1)*f_tvamf(c_nil,c_one,vp,u(m,q,+1),q) - (p*q+m**2)*p_ff(c_one,v(m,p,+1),u(m,q,+1))), 0, & (p*q+m**2)*p_ff(c_one,v(m,q,+1),u(m,q,+1))), 0, & (p*q+m**2)*p_ff(c_one,v(m,q,+1),u(m,q,+1)))), 0, & (p*q+m**2)*p_ff(c_one,v(m,q,+1),u(m,q,+1)))), 0, & (p*q+m**2)*p_ff(c_one,v(m,q,+1)))), 0, & (p*q+m**2)*p_ff(c_one,v(m,q,+1))))
                                                                       p*[ubar(p,+).(Isigma*q).g5.u(q,+)] - (p*q+m^2)*ubar(p,+).g5.u(q,+) = 0", passed)
         p*[ubar(p,-).(Isigma*q).g5.u(q,-)] - (p*q+m^2)*ubar(p,-).g5.u(q,-) = 0", passed)
           \text{call expect } ((u(m,p,+1)*f\_tvamf(c\_nil,c\_one,vp,v(m,q,+1),q) - (p*q+m**2)*p\_ff(c\_one,u(m,p,+1),v(m,q,+1))), \ 0, \ \& \ (v(m,p,+1)*f\_tvamf(c\_nil,c\_one,vp,v(m,q,+1),q) - (p*q+m**2)*p\_ff(c\_one,u(m,p,+1),v(m,q,+1))), \ 0, \ \& \ (v(m,p,+1)*f\_tvamf(c\_one,vp,v(m,q,+1),q) - (p*q+m**2)*p\_ff(c\_one,u(m,p,+1),v(m,q,+1))), \ 0, \ \& \ (v(m,p,+1)*f\_tvamf(c\_one,vp,v(m,q,+1),q) - (p*q+m**2)*p\_ff(c\_one,u(m,p,+1),v(m,q,+1))), \ 0, \ \& \ (v(m,p,+1)*f\_tvamf(c\_one,vp,v(m,q,+1),q) - (p*q+m**2)*p\_ff(c\_one,u(m,p,+1),v(m,q,+1))), \ (v(m,p,+1)*f\_tvamf(c\_one,vp,v(m,q,+1),q) - (p*q+m**2)*p\_ff(c\_one,u(m,p,+1),v(m,q,+1))), \ (v(m,p,+1)*f\_tvamf(c\_one,vp,v(m,q,+1),q) - (p*q+m**2)*p\_ff(c\_one,vp,v(m,q,+1),v(m,q,+1))), \ (v(m,p,+1)*f\_tvamf(c\_one,vp,v(m,q,+1),q) - (p*q+m**2)*p\_ff(c\_one,vp,v(m,q,+1),q) - (p
                                                                       p*[vbar(p,+).(Isigma*q).g5.v(q,+)] - (p*q+m^2)*vbar(p,+).g5.v(q,+) = 0", passed)
         "p*[vbar(p,-).(Isigma*q).g5.v(q,-)] - (p*q+m^2)*vbar(p,-).g5.v(q,-) = 0", passed)
 \langle Test \text{ omega95\_bispinors} \rangle + \equiv
        print *, "*** Checking polarization vectors: ***"
         call expect (conjg(eps(m,p, 1))*eps(m,p, 1), -1, "e(1).e(1)=-1", passed)
         call expect (conjg(eps(m,p, 1))*eps(m,p,-1), 0, "e(1).e(-1)=0", passed)
         call expect (conjg(eps(m,p,-1))*eps(m,p, 1), 0, "e(-1).e( 1)= 0", passed)
        call expect (conjg(eps(m,p,-1))*eps(m,p,-1), -1, "e(-1).e(-1)=-1", passed)
                                                                                                                                                     p*eps(m,p, 1), 0, " p.e(1)=0", passed)
        call expect (
                                                                                                                                                    p*eps(m,p,-1), 0, "
        call expect (
                                                                                                                                                                                                                                                            p.e(-1)=0", passed)
         if (m > 0) then
                       call expect (conjg(eps(m,p, 1))*eps(m,p, 0), 0, "e(1).e(0)=0", passed)
                       call expect (conjg(eps(m,p, 0))*eps(m,p, 1), 0, "e( 0).e( 1)= 0", passed)
                       call expect (conjg(eps(m,p, 0))*eps(m,p, 0), -1, "e( 0).e( 0)=-1", passed)
                       call expect (conjg(eps(m,p, 0))*eps(m,p,-1), 0, "e( 0).e(-1)= 0", passed)
                       call expect (conjg(eps(m,p,-1))*eps(m,p, 0), 0, "e(-1).e( 0)= 0", passed)
                                                                                                                                                                   p*eps(m,p, 0), 0, p.e(0)=0, passed)
                       call expect (
         end if
\langle \mathit{Test} \ \mathtt{omega95\_bispinors} \rangle + \equiv
```

```
print *, "*** Checking polarization vectorspinors: ***"
 call expect (abs(p * ueps(m, p, 2)), 0, "p.ueps ( 2)= 0", passed)
 call expect (abs(p * ueps(m, p, 1)), 0, "p.ueps (1)= 0", passed)
 call expect (abs(p * ueps(m, p, -1)), 0, "p.ueps (-1)= 0", passed)
 call expect (abs(p * ueps(m, p, -2)), 0, "p.ueps (-2)= 0", passed)
 call expect (abs(p * veps(m, p, 2)), 0, "p.veps (2)= 0", passed)
 call expect (abs(p * veps(m, p, 1)), 0, "p.veps ( 1)= 0", passed)
 call expect (abs(p * veps(m, p, -1)), 0, "p.veps (-1)= 0", passed)
 call expect (abs(p * veps(m, p, -2)), 0, "p.veps (-2)= 0", passed)
 print *, "*** Checking polarization vectorspinors (neg. masses): ***"
 call expect (abs(p * ueps(-m, p, 2)), 0, "p.ueps (2)=0", passed)
 call expect (abs(p * ueps(-m, p, 1)), 0, "p.ueps (1)=0", passed)
 call expect (abs(p * ueps(-m, p, -1)), 0, "p.ueps (-1)= 0", passed)
 call expect (abs(p * ueps(-m, p, -2)), 0, "p.ueps (-2)= 0", passed)
 call expect (abs(p * veps(-m, p, 2)), 0, "p.veps (2)= 0", passed)
 call expect (abs(p * veps(-m, p, 1)), 0, "p.veps (1)=0", passed)
 call expect (abs(p * veps(-m, p, -1)), 0, "p.veps (-1)= 0", passed)
 call expect (abs(p * veps(-m, p, -2)), 0, "p.veps (-2)= 0", passed)
 print *, "*** in the rest frame ***"
 call expect (abs(p_0 * ueps(m, p_0, 2)), 0, "p0.ueps (2)= 0", passed)
 call expect (abs(p_0 * ueps(m, p_0, 1)), 0, "p0.ueps (1)=0", passed)
 call expect (abs(p_0 * ueps(m, p_0, -1)), 0, "p0.ueps (-1)= 0", passed)
 call expect (abs(p_0 * ueps(m, p_0, -2)), 0, "p0.ueps (-2)= 0", passed)
 call expect (abs(p_0 * veps(m, p_0, 2)), 0, "p0.veps (2)= 0", passed)
 call expect (abs(p_0 * veps(m, p_0, 1)), 0, "p0.veps (1)=0", passed)
 call expect (abs(p_0 * veps(m, p_0, -1)), 0, "p0.veps (-1)= 0", passed)
 call expect (abs(p_0 * veps(m, p_0, -2)), 0, "p0.veps (-2)= 0", passed)
 print *, "*** in the rest frame (neg. masses) ***"
 call expect (abs(p_0 * ueps(-m, p_0, 2)), 0, "p0.ueps (2)= 0", passed) call expect (abs(p_0 * ueps(-m, p_0, 1)), 0, "p0.ueps (1)= 0", passed)
 call expect (abs(p_0 * ueps(-m, p_0, -1)), 0, "p0.ueps (-1)= 0", passed)
 call expect (abs(p_0 * ueps(-m, p_0, -2)), 0, "p0.ueps (-2)= 0", passed) call expect (abs(p_0 * veps(-m, p_0, 2)), 0, "p0.veps (2)= 0", passed)
 call expect (abs(p_0 * veps(-m, p_0, 1)), 0, "p0.veps (1)= 0", passed)
 call expect (abs(p_0 * veps(-m, p_0, -1)), 0, "p0.veps (-1)= 0", passed)
 call expect (abs(p_0 * veps(-m, p_0, -2)), 0, "p0.veps (-2)= 0", passed)
\langle \mathit{Test} \ \mathtt{omega95\_bispinors} \rangle + \equiv
 print *, "*** Checking the irreducibility condition: ***"
 call expect (abs(f_potgr (c_one, c_one, ueps(m, p, 2))), 0, "g.ueps (2)", passed)
 call expect (abs(f_potgr (c_one, c_one, ueps(m, p, 1))), 0, "g.ueps (1)", passed)
 call expect (abs(f_potgr (c_one, c_one, ueps(m, p, -1))), 0, "g.ueps (-1)", passed)
 call expect (abs(f_potgr (c_one, c_one, ueps(m, p, -2))), 0, "g.ueps (-2)", passed)
 call expect (abs(f_potgr (c_one, c_one, veps(m, p, 2))), 0, "g.veps (2)", passed)
 call expect (abs(f_potgr (c_one, c_one, veps(m, p, 1))), 0, "g.veps (1)", passed)
 call expect (abs(f_potgr (c_one, c_one, veps(m, p, -1))), 0, "g.veps (-1)", passed)
 call expect (abs(f_potgr (c_one, c_one, veps(m, p, -2))), 0, "g.veps (-2)", passed)
 print *, "*** Checking the irreducibility condition (neg. masses): ***"
  {\tt call\ expect\ (abs(f\_potgr\ (c\_one,\ c\_one,\ ueps(-m,\ p,\ 2))),\ 0,\ "g.ueps\ (\ 2)",\ passed) } 
  {\tt call\ expect\ (abs(f\_potgr\ (c\_one,\ c\_one,\ ueps(-m,\ p,\ 1))),\ 0,\ "g.ueps\ (\ 1)",\ passed) } 
 call expect (abs(f_potgr (c_one, c_one, ueps(-m, p, -1))), 0, "g.ueps (-1)", passed)
 call expect (abs(f_potgr (c_one, c_one, ueps(-m, p, -2))), 0, "g.ueps (-2)", passed)
  {\tt call\ expect\ (abs(f\_potgr\ (c\_one,\ c\_one,\ veps(-m,\ p,\ 2))),\ 0,\ "g.veps\ (\ 2)",\ passed) } 
 call expect (abs(f_potgr (c_one, c_one, veps(-m, p, 1))), 0, "g.veps (1)", passed)
 call expect (abs(f_potgr (c_one, c_one, veps(-m, p, -1))), 0, "g.veps (-1)", passed)
 call expect (abs(f_potgr (c_one, c_one, veps(-m, p, -2))), 0, "g.veps (-2)", passed)
 print *, "*** in the rest frame ***"
  {\tt call\ expect\ (abs(f\_potgr\ (c\_one,\ c\_one,\ ueps(m,\ p\_0,\ 2))),\ 0,\ "g.ueps\ (\ 2)",\ passed) } 
 call expect (abs(f_potgr (c_one, c_one, ueps(m, p_0, 1))), 0, "g.ueps (1)", passed)
 call expect (abs(f_potgr (c_one, c_one, ueps(m, p_0, -1))), 0, "g.ueps (-1)", passed)
 call expect (abs(f_potgr (c_one, c_one, ueps(m, p_0, -2))), 0, "g.ueps (-2)", passed)
 call expect (abs(f_potgr (c_one, c_one, veps(m, p_0, 2))), 0, "g.veps (2)", passed)
  {\tt call\ expect\ (abs(f\_potgr\ (c\_one,\ c\_one,\ veps(m,\ p\_0,\ 1))),\ 0,\ "g.veps\ (\ 1)",\ passed) } 
 call expect (abs(f_potgr (c_one, c_one, veps(m, p_0, -1))), 0, "g.veps (-1)", passed)
 call expect (abs(f_potgr (c_one, c_one, veps(m, p_0, -2))), 0, "g.veps (-2)", passed)
 print *, "*** in the rest frame (neg. masses) ***"
 call expect (abs(f_potgr (c_one, c_one, ueps(m, p_0, 2))), 0, "g.ueps (2)", passed)
```

```
call expect (abs(f_potgr (c_one, c_one, ueps(m, p_0, 1))), 0, "g.ueps (1)", passed)
 call expect (abs(f_potgr (c_one, c_one, ueps(m, p_0, -1))), 0, "g.ueps (-1)", passed)
 call expect (abs(f_potgr (c_one, c_one, ueps(m, p_0, -2))), 0, "g.ueps (-2)", passed)
  {\tt call\ expect\ (abs(f\_potgr\ (c\_one,\ c\_one,\ veps(m,\ p\_0,\ 2))),\ 0,\ "g.veps\ (\ 2)",\ passed) } 
 call expect (abs(f_potgr (c_one, c_one, veps(m, p_0, 1))), 0, "g.veps (1)", passed)
 call expect (abs(f_potgr (c_one, c_one, veps(m, p_0, -1))), 0, "g.veps (-1)", passed)
 call expect (abs(f_potgr (c_one, c_one, veps(m, p_0, -2))), 0, "g.veps (-2)", passed)
\langle \mathit{Test} \ \mathsf{omega95\_bispinors} \rangle + \equiv
 print *, "*** Testing vectorspinor normalization ***"
 call expect (veps(m,p, 2)*ueps(m,p, 2), -2*m, "ueps( 2).ueps( 2)= -2m", passed)
 call expect (veps(m,p, 1)*ueps(m,p, 1), -2*m, "ueps( 1).ueps( 1)= -2m", passed)
 call expect (veps(m,p,-1)*ueps(m,p,-1), -2*m, "ueps(-1).ueps(-1)= -2m", passed)
 call expect (veps(m,p,-2)*ueps(m,p,-2), -2*m, "ueps(-2).ueps(-2)= -2m", passed)
 call expect (ueps(m,p, 2)*veps(m,p, 2), 2*m, "veps( 2).veps( 2)= +2m", passed)
                                            2*m, "veps( 1).veps( 1)= +2m", passed)
 call expect (ueps(m,p, 1)*veps(m,p, 1),
                                            2*m, "veps(-1).veps(-1)= +2m", passed)
 call expect (ueps(m,p,-1)*veps(m,p,-1),
                                            2*m, "veps(-2).veps(-2)= +2m", passed)
 call expect (ueps(m,p,-2)*veps(m,p,-2),
                                                                      0", passed)
                                              0, "ueps( 2).veps( 2)=
 call expect (ueps(m,p, 2)*ueps(m,p, 2),
 call expect (ueps(m,p, 1)*ueps(m,p, 1),
                                              0, "ueps( 1).veps( 1)=
                                                                       0", passed)
                                              0, "ueps(-1).veps(-1)=
 call expect (ueps(m,p,-1)*ueps(m,p,-1),
                                                                       0", passed)
 call expect (ueps(m,p,-2)*ueps(m,p,-2),
                                              0, "ueps(-2).veps(-2)=
                                                                        0", passed)
 call expect (veps(m,p, 2)*veps(m,p, 2),
                                              0, "veps( 2).ueps( 2)=
                                                                        0", passed)
 call expect (veps(m,p, 1)*veps(m,p, 1),
                                              0, "veps( 1).ueps( 1)=
                                                                        0", passed)
                                              0, "veps(-1).ueps(-1)=
 call expect (veps(m,p,-1)*veps(m,p,-1),
                                                                        0", passed)
                                              0, "veps(-2).ueps(-2)=
 call expect (veps(m,p,-2)*veps(m,p,-2),
                                                                        0", passed)
 print *, "*** Testing vectorspinor normalization (neg. masses) ***"
  {\tt call\ expect\ (veps(-m,p,\ 2)*ueps(-m,p,\ 2),\ +2*m,\ "ueps(\ 2).ueps(\ 2)=\ +2m",\ passed) } 
 call expect (veps(-m,p, 1)*ueps(-m,p, 1), +2*m, "ueps( 1).ueps( 1)= +2m", passed)
  {\tt call \ expect \ (veps(-m,p,-1)*ueps(-m,p,-1), \ +2*m, \ "ueps(-1).ueps(-1)= \ +2m", \ passed) } 
  {\tt call \ expect \ (veps(-m,p,-2)*ueps(-m,p,-2), \ +2*m, \ "ueps(-2).ueps(-2)= \ +2m", \ passed) } 
  {\tt call \ expect \ (ueps(-m,p,\ 2)*veps(-m,p,\ 2),\ -2*m,\ "veps(\ 2).veps(\ 2)=\ -2m",\ passed) } 
 call expect (ueps(-m,p, 1)*veps(-m,p, 1), -2*m, "veps( 1).veps( 1)= -2m", passed)
 call expect (ueps(-m,p,-1)*veps(-m,p,-1), -2*m, "veps(-1).veps(-1)= -2m", passed)
 call expect (ueps(-m,p,-2)*veps(-m,p,-2), -2*m, "veps(-2).veps(-2)= -2m", passed)
                                                0, "ueps( 2).veps( 2)=
                                                                         0", passed)
 call expect (ueps(-m,p, 2)*ueps(-m,p, 2),
                                                0, "ueps( 1).veps( 1)=
                                                                          0", passed)
 call expect (ueps(-m,p, 1)*ueps(-m,p, 1),
                                                0, "ueps(-1).veps(-1)=
                                                                          0", passed)
 call expect (ueps(-m,p,-1)*ueps(-m,p,-1),
                                                0, "ueps(-2).veps(-2)=
                                                                          0", passed)
 call expect (ueps(-m,p,-2)*ueps(-m,p,-2),
                                                0, "veps(2).ueps(2)=
                                                                          0", passed)
 call expect (veps(-m,p, 2)*veps(-m,p, 2),
                                                0, "veps( 1).ueps( 1)=
 call expect (veps(-m,p, 1)*veps(-m,p, 1),
                                                                          0", passed)
                                                0, "veps(-1).ueps(-1)=
 call expect (veps(-m,p,-1)*veps(-m,p,-1),
                                                                          0", passed)
 call expect (veps(-m,p,-2)*veps(-m,p,-2),
                                                0, "veps(-2).ueps(-2)=
                                                                          0", passed)
 print *, "*** in the rest frame ***"
 call expect (veps(m,p_0, 2)*ueps(m,p_0, 2), -2*m, "ueps(2).ueps(2)= -2m", passed)
 call expect (veps(m,p_0, 1)*ueps(m,p_0, 1), -2*m, "ueps( 1).ueps( 1)= -2m", passed)
 call expect (veps(m,p_0,-1)*ueps(m,p_0,-1), -2*m, "ueps(-1).ueps(-1) = -2m", passed)
  {\tt call \ expect \ (veps(m,p_0,-2)*ueps(m,p_0,-2), \ -2*m, \ "ueps(-2).ueps(-2)=-2m", \ passed) } 
 call expect (ueps(m,p_0, 2)*veps(m,p_0, 2), 2*m, "veps( 2).veps( 2)= +2m", passed)
 call expect (ueps(m,p_0, 1)*veps(m,p_0, 1),
                                                2*m, "veps( 1).veps( 1)= +2m", passed)
 call expect (ueps(m,p_0,-1)*veps(m,p_0,-1),
                                                2*m, "veps(-1).veps(-1)= +2m", passed)
 call expect (ueps(m,p_0,-2)*veps(m,p_0,-2),
                                                2*m, "veps(-2).veps(-2)= +2m", passed)
 call expect (ueps(m,p_0, 2)*ueps(m,p_0, 2),
                                                  0, "ueps( 2).veps( 2)=
                                                                           0", passed)
                                                  0, "ueps(1).veps(1)=
                                                                            0", passed)
 call expect (ueps(m,p_0, 1)*ueps(m,p_0, 1),
 call expect (ueps(m,p_0,-1)*ueps(m,p_0,-1),
                                                  0, "ueps(-1).veps(-1)=
                                                                            0", passed)
                                                  0, "ueps(-2).veps(-2)=
 call expect (ueps(m,p_0,-2)*ueps(m,p_0,-2),
                                                                            0", passed)
                                                  0, "veps(2).ueps(2)=
                                                                            0", passed)
 call expect (veps(m,p_0, 2)*veps(m,p_0, 2),
                                                  0, "veps(1).ueps(1)=
                                                                            0", passed)
 call expect (veps(m,p_0, 1)*veps(m,p_0, 1),
                                                  0, "veps(-1).ueps(-1)=
                                                                            0", passed)
 call expect (veps(m,p_0,-1)*veps(m,p_0,-1),
                                                  0, "veps(-2).ueps(-2)=
                                                                            0", passed)
 call expect (veps(m,p_0,-2)*veps(m,p_0,-2),
 print *, "*** in the rest frame (neg. masses) ***"
 call expect (veps(-m,p_0, 2)*ueps(-m,p_0, 2), +2*m, "ueps( 2).ueps( 2)= +2m", passed)
 call expect (veps(-m,p_0, 1)*ueps(-m,p_0, 1), +2*m, "ueps(1).ueps(1)= +2m", passed)
  {\tt call \ expect \ (veps(-m,p_0,-1)*ueps(-m,p_0,-1),\ +2*m,\ "ueps(-1).ueps(-1)= +2m",\ passed) } 
 call expect (veps(-m,p_0,-2)*ueps(-m,p_0,-2), +2*m, "ueps(-2).ueps(-2) = +2m", passed)
 call expect (ueps(-m,p_0, 2)*veps(-m,p_0, 2), -2*m, "veps( 2).veps( 2)= -2m", passed)
 call expect (ueps(-m,p_0, 1)*veps(-m,p_0, 1), -2*m, "veps( 1).veps( 1)= -2m", passed)
```

```
call expect (ueps(-m,p_0,-1)*veps(-m,p_0,-1), -2*m, "veps(-1).veps(-1) = -2m", passed)
 call expect (ueps(-m,p_0,-2)*veps(-m,p_0,-2), -2*m, "veps(-2).veps(-2) = -2m", passed)
                                                   0, "ueps( 2).veps( 2)=
                                                                             0", passed)
 call expect (ueps(-m,p_0, 2)*ueps(-m,p_0, 2),
                                                   0, "ueps( 1).veps( 1)=
                                                                             0", passed)
 call expect (ueps(-m,p_0, 1)*ueps(-m,p_0, 1),
 call expect (ueps(-m,p_0,-1)*ueps(-m,p_0,-1),
                                                   0, "ueps(-1).veps(-1)=
                                                                             0", passed)
                                                   0, "ueps(-2).veps(-2)=
                                                                             0", passed)
 call expect (ueps(-m,p_0,-2)*ueps(-m,p_0,-2),
                                                   0, "veps( 2).ueps( 2)=
                                                                             0", passed)
 call expect (veps(-m,p_0, 2)*veps(-m,p_0, 2),
 call expect (veps(-m,p_0, 1)*veps(-m,p_0, 1),
                                                   0, "veps(1).ueps(1)=
                                                                             0", passed)
                                                                             0", passed)
 call expect (veps(-m,p_0,-1)*veps(-m,p_0,-1),
                                                   0, "veps(-1).ueps(-1)=
 call expect (veps(-m,p_0,-2)*veps(-m,p_0,-2),
                                                   0, "veps(-2).ueps(-2)=
                                                                             0", passed)
\langle Test \text{ omega95\_bispinors} \rangle + \equiv
 print *, "*** Majorana properties of gravitino vertices: ***"
 call expect (abs(u (m,q,1) * f_sgr (c_one, c_one, ueps(m,p,2), t) + &
    ueps(m,p,2) * gr_sf(c_one,c_one,u(m,q,1),t)), 0, "f_sgr + gr_sf
                                                                               = 0", passed)
 !!! call expect (abs(u (m,q,-1) * f_sgr (c_one, c_one, ueps(m,p,2), t) + &
        \label{eq:ueps} \texttt{ueps(m,p,2)} \ * \ \texttt{gr\_sf(c\_one,c\_one,u(m,q,-1),t))}, \quad \texttt{0, "f\_sgr}
                                                                                     = 0", passed)
 !!!
 !!! call expect (abs(u (m,q,1) * f_sgr (c_one, c_one, ueps(m,p,1), t) + &
         ueps(m,p,1) * gr_sf(c_one,c_one,u(m,q,1),t)), 0, "f_sgr 
                                                                                    = 0", passed)
 !!! call expect (abs(u (m,q,-1) * f_sgr (c_one, c_one, ueps(m,p,1), t) + &
 !!!
        \label{eq:cone} ueps(\texttt{m},\texttt{p},\texttt{1}) \ * \ \text{gr\_sf(c\_one,c\_one,u(\texttt{m},\texttt{q},\texttt{-1}),\texttt{t})),} \quad \texttt{0, "f\_sgr}
                                                                                     = 0", passed)
                                                                        + gr_sf
 !!! call expect (abs(u (m,q,1) * f_sgr (c_one, c_one, ueps(m,p,-1), t) + &
        ueps(m,p,-1) * gr_sf(c_one,c_one,u(m,q,1),t)), 0, "f_sgr + gr_sf
                                                                                     = 0", passed)
 !!! call expect (abs(u (m,q,-1) * f_sgr (c_one, c_one, ueps(m,p,-1), t) + &
 !!!
        ueps(m,p,-1) * gr_sf(c_one,c_one,u(m,q,-1),t)), 0, "f_sgr
                                                                                     = 0", passed)
                                                                         + gr_sf
 !!! call expect (abs(u (m,q,1) * f_sgr (c_one, c_one, ueps(m,p,-2), t) + &
 !!!
        ueps(m,p,-2) * gr_sf(c_one,c_one,u(m,q,1),t)), 0, "f_sgr
                                                                                     = 0", passed)
 !!! call expect (abs(u (m,q,-1) * f_sgr (c_one, c_one, ueps(m,p,-2), t) + &
                                                                                     = 0", passed)
 !!!
        ueps(m,p,-2) * gr_sf(c_one,c_one,u(m,q,-1),t)), 0, "f_sgr
 call expect (abs(u (m,q,1) * f_slgr (c_one, c_one, ueps(m,p,2), t) + &
                                                                                = 0", passed, threshold = 0.5_defau
    ueps(m,p,2) * gr_slf(c_one,c_one,u(m,q,1),t)), 0, "f_slgr
                                                                    + gr_slf
 call expect (abs(u (m,q,1) * f_srgr (c_one, c_one, ueps(m,p,2), t) + &
    ueps(m,p,2) * gr_srf(c_one,c_one,u(m,q,1),t)), 0, "f_srgr
                                                                    + gr_srf
                                                                                = 0", passed, threshold = 0.5_defau
 call expect (abs(u (m,q,1) * f_slrgr (c_one, c_two, c_one, ueps(m,p,2), t) + &
                                                                         + gr_slrf = 0", passed, threshold = 0.
    ueps(m,p,2) * gr_slrf(c_one,c_two,c_one,u(m,q,1),t)), 0, "f_slrgr"
 call expect (abs(u (m,q,1) * f_pgr (c_one, c_one, ueps(m,p,2), t) + &
     ueps(m,p,2) * gr_pf(c_one,c_one,u(m,q,1),t)), 0, "f_pgr
                                                                               = 0", passed, threshold = 0.5_defaul
                                                                  + gr_pf
 call expect (abs(u (m,q,1) * f_vgr (c_one, vt, ueps(m,p,2), p+q) + &
    ueps(m,p,2) * gr_vf(c_one,vt,u(m,q,1),p+q)), 0, "f_vgr
                                                                 + gr_vf = 0", passed, threshold = 0.5_default)
 call expect (abs(u (m,q,1) * f_vlrgr (c_one, c_two, vt, ueps(m,p,2), p+q) + &
    passed, threshold = 0.5_default)
  !!! call expect (abs(u (m,q,-1) * f_vgr (c_one, vt, ueps(m,p,2), p+q) + &
        ueps(m,p,2) * gr_vf(c_one,vt,u(m,q,-1),p+q)), 0, "f_vgr
                                                                                    = 0", passed)
                                                                       + gr_vf
  !!! call expect (abs(u (m,q,1) * f_vgr (c_one, vt, ueps(m,p,1), p+q) + &
        ueps(m,p,1) * gr_vf(c_one,vt,u(m,q,1),p+q)), 0, "f_vgr
                                                                                   = 0", passed)
                                                                      + gr_vf
 !!! call expect (abs(u (m,q,-1) * f_vgr (c_one, vt, ueps(m,p,1), p+q) + &
 !!!
        ueps(m,p,1) * gr_vf(c_one,vt,u(m,q,-1),p+q)), 0, "f_vgr
                                                                                   = 0", passed)
                                                                       + gr_vf
 !!! call expect (abs(u (m,q,1) * f_vgr (c_one, vt, ueps(m,p,-1), p+q) + &
                                                                                    = 0", passed)
 !!!
        ueps(m,p,-1) * gr_vf(c_one,vt,u(m,q,1),p+q)), 0, "f_vgr
                                                                       + gr_vf
 !!! call expect (abs(u (m,q,-1) * f_vgr (c_one, vt, veps(m,p,-1), p+q) + &
 !!!
        veps(m,p,-1) * gr_vf(c_one,vt,u(m,q,-1),p+q)), 0, "f_vgr
                                                                                    = 0", passed)
                                                                        + gr_vf
 !!! call expect (abs(v (m,q,1) * f_vgr (c_one, vt, ueps(m,p,-2), p+q) + &
        ueps(m,p,-2) * gr_vf(c_one,vt,v(m,q,1),p+q)), 0, "f_vgr
                                                                       + gr_vf
                                                                                    = 0", passed)
 !!! call expect (abs(u (m,q,-1) * f_vgr (c_one, vt, ueps(m,p,-2), p+q) + &
                                                                                     = 0", passed)
        ueps(m,p,-2) * gr_vf(c_one,vt,u(m,q,-1),p+q)), 0, "f_vgr
                                                                        + gr_vf
 call expect (abs(s_grf (c_one, ueps(m,p,2), u(m,q,1),t) + &
     s_fgr(c_one,u(m,q,1),ueps(m,p,2),t)), 0, "s_grf
                                                                       = 0", passed)
 call expect (abs(sl_grf (c_one, ueps(m,p,2), u(m,q,1),t) + &
    sl_fgr(c_one,u(m,q,1),ueps(m,p,2),t)), 0, "sl_grf
                                                           + sl_fgr
                                                                        = 0", passed)
 call expect (abs(sr_grf (c_one, ueps(m,p,2), u(m,q,1),t) + &
    sr_fgr(c_one,u(m,q,1),ueps(m,p,2),t)), 0, "sr_grf
                                                                        = 0", passed)
                                                          + sr_fgr
 call expect (abs(slr_grf (c_one, c_two, ueps(m,p,2), u(m,q,1),t) + &
    slr_fgr(c_one,c_two,u(m,q,1),ueps(m,p,2),t)), 0, "slr_grf
                                                                  + slr_fgr = 0", passed)
 call expect (abs(p_grf (c_one, ueps(m,p,2), u(m,q,1),t) + &
    p_fgr(c_one,u(m,q,1),ueps(m,p,2),t)), 0, "p_grf
                                                           + p_fgr
                                                                       = 0", passed)
 call expect (abs(v_{grf} (c_one, ueps(m,p,2), u(m,q,1),t) + &
```

```
v_fgr(c_one,u(m,q,1),ueps(m,p,2),t)), 0, "v_grf
                                                                                                             + v_fgr
                                                                                                                                    = 0", passed)
   call expect (abs(vlr_grf (c_one, c_two, ueps(m,p,2), u(m,q,1),t) + &
               vlr_fgr(c_one, c_two, u(m,q,1), ueps(m,p,2),t)), \quad 0, \quad vlr_grf \quad + \ vlr_fgr \quad = 0", passed) 
   call expect (abs(u(m,p,1) * f_potgr (c_one,c_one,testv) - testv * gr_potf &
         (c_{one}, c_{one}, u (m,p,1))), 0, "f_{potgr} - gr_{potf} = 0", passed)
   call expect (abs (pot_fgr (c_one,u(m,p,1),testv) - pot_grf(c_one, &
         testv,u(m,p,1))), 0, "pot_fgr - pot_grf = 0", passed)
   call expect (abs(u(m,p,1) * f_s2gr (c_one,c_one,testv) - testv * gr_s2f &
         (c_one,c_one,c_one,u (m,p,1))), 0, "f_s2gr
                                                                                              - gr_s2f
                                                                                                                    = 0", passed)
   call expect (abs (s2_fgr (c_one,u(m,p,1),c_one,testv) - s2_grf(c_one, &
         testv,c_one,u(m,p,1))), 0, "s2_fgr
                                                                               -s2_grf = 0", passed)
   call expect (abs(u (m,q,1) * f_svgr (c_one, c_one, vt, ueps(m,p,2)) + &
         ueps(m,p,2) * gr_svf(c_one,c_one,vt,u(m,q,1)), 0, "f_svgr
                                                                                                                            + gr_svf
                                                                                                                                                       = 0", passed)
   call expect (abs(u (m,q,1) * f_slvgr (c_one, c_one, vt, ueps(m,p,2)) + &
         ueps(m,p,2) * gr_slvf(c_one,c_one,vt,u(m,q,1))), 0, "f_slvgr + gr_slvf = 0", passed)
   call expect (abs(u (m,q,1) * f_srvgr (c_one, c_one, vt, ueps(m,p,2)) + &
        ueps(m,p,2) * gr_srvf(c_one,c_one,vt,u(m,q,1))), 0, "f_srvgr + gr_srvf = 0", passed)
   call expect (abs(u (m,q,1) * f_slrvgr (c_one, c_two, c_one, vt, ueps(m,p,2)) + &
         ueps(m,p,2) * gr_slrvf(c_one,c_two,c_one,vt,u(m,q,1))), \quad 0, \\ "f_slrvgr + gr_slrvf = 0", passed) 
   call expect (abs (sv1_fgr (c_one,u(m,p,1),vt,ueps(m,q,2)) + sv1_grf(c_one, &
        ueps(m,q,2),vt,u(m,p,1))), 0, "sv1_fgr + sv1_grf = 0", passed)
   call expect (abs (sv2_fgr (c_one,u(m,p,1),c_one,ueps(m,q,2)) + sv2_grf(c_one, &
        ueps(m,q,2),c_one,u(m,p,1))), 0, "sv2_fgr + sv2_grf = 0", passed)
    \texttt{call expect (abs (slv1\_fgr (c\_one, u(m,p,1), vt, ueps(m,q,2)) + slv1\_grf(c\_one, \& cone, we cone, we cone, we cone, we cone and cone are also as a substitution of the cone and cone are also as a substitution of the cone and cone are also as a substitution of the cone and cone are also as a substitution of the cone and cone are also as a substitution of the cone are also as a substitution of the cone and cone are also as a substitution of the cone are also as a substitution of t
        ueps(m,q,2),vt,u(m,p,1)), 0, "slv1\_fgr + slv1\_grf = 0", passed)
   call expect (abs (srv2_fgr (c_one,u(m,p,1),c_one,ueps(m,q,2)) + srv2_grf(c_one, &
        ueps(m,q,2),c_one,u(m,p,1))), 0, "srv2_fgr + srv2_grf = 0", passed)
   call expect (abs (slrv1_fgr (c_one,c_two,u(m,p,1),vt,ueps(m,q,2)) + slrv1_grf(c_one,c_two, &
        ueps(m,q,2),vt,u(m,p,1))), 0, "slrv1_fgr + slrv1_grf = 0", passed)
   call expect (abs (slrv2_fgr (c_one,c_two,u(m,p,1),c_one,ueps(m,q,2)) + slrv2_grf(c_one, &
         c_{two,ueps(m,q,2),c_{one,u(m,p,1))}, 0, "slrv2_fgr + slrv2_grf = 0", passed)
   call expect (abs(u (m,q,1) * f_pvgr (c_one, c_one, vt, ueps(m,p,2)) + &
                                                                                                                                                       = 0", passed)
        ueps(m,p,2) * gr_pvf(c_one,c_one,vt,u(m,q,1))), 0, "f_pvgr + gr_pvf
   call expect (abs (pv1_fgr (c_one,u(m,p,1),vt,ueps(m,q,2)) + pv1_grf(c_one, &
        ueps(m,q,2),vt,u(m,p,1))), 0, "pv1_fgr + pv1_grf = 0", passed)
    \texttt{call expect (abs (pv2\_fgr (c\_one,u(m,p,1),c\_one,ueps(m,q,2)) + pv2\_grf(c\_one, \& one,ueps(m,q,2)) + pv2\_grf(c\_one, \& one,ueps(m,q,2)) } \\ 
        ueps(m,q,2),c_one,u(m,p,1))), 0, "pv2_fgr + pv2_grf = 0", passed)
   call expect (abs(u (m,q,1) * f_v2gr (c_one, vt, vz, ueps(m,p,2)) + &
        \label{eq:cone_vt_vz_u(m,q,1)} \verb"ueps(m,p,2) * \verb"gr_v2f(c_one_vt_vz_u(m,q,1))), 0, "f_v2gr + \verb"gr_v2f" + "gr_v2f" + "gr
                                                                                                                                                 = 0", passed)
   call expect (abs(u (m,q,1) * f_v2lrgr (c_one, c_two, vt, vz, ueps(m,p,2)) + &
        ueps(m,p,2) * gr_v2lrf(c_one,c_two,vt,vz,u(m,q,1))), 0, "f_v2lrgr + gr_v2lrf = 0", passed)
   call expect (abs (v2_fgr (c_one,u(m,p,1),vt,ueps(m,q,2)) + v2_grf(c_one, &
        ueps(m,q,2),vt,u(m,p,1))), 0, "v2_fgr + v2_grf
                                                                                                         = 0", passed)
   call expect (abs (v2lr_fgr (c_one,c_two,u(m,p,1),vt,ueps(m,q,2)) + v2lr_grf(c_one, c_two, &
        ueps(m,q,2),vt,u(m,p,1))), 0, "v2lr_fgr + v2lr_grf = 0", passed)
\langle \mathit{Test} \ \mathtt{omega95\_bispinors} \rangle + \equiv
   print *, "*** Testing the gravitino propagator: ***"
   print *, "Transversality:"
   call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
                           pr_grav(p,m,w,testv))), 0, "p.pr.test", passed)
   call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
                           pr_grav(p,m,w,ueps(m,p,2)))), 0, "p.pr.ueps ( 2)", passed)
   call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
                           pr_grav(p,m,w,ueps(m,p,1)))), 0, "p.pr.ueps ( 1)", passed)
   call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
                           pr_grav(p,m,w,ueps(m,p,-1)))), 0, "p.pr.ueps (-1)", passed)
   call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
                           pr_grav(p,m,w,ueps(m,p,-2)))), 0, "p.pr.ueps (-2)", passed)
   call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
                           pr_grav(p,m,w,veps(m,p,2)))), 0, "p.pr.veps ( 2)", passed)
   call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
                           \label{eq:pr_grav} \texttt{pr\_grav}(\texttt{p,m,w,veps}(\texttt{m,p,1}))))\,,\quad \texttt{0, "p.pr.veps (1)", passed)}
   call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
                           pr_grav(p,m,w,veps(m,p,-1)))), 0, "p.pr.veps (-1)", passed)
   call expect (abs(p * (cmplx (p*p - m**2, m*w, kind=default) * &
                           pr_grav(p,m,w,veps(m,p,-2)))), 0, "p.pr.veps (-2)", passed)
```

```
print *, "Irreducibility:"
 call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,testv)))), 0, "g.pr.test", passed)
 call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,ueps(m,p,2))))), 0, &
               "g.pr.ueps ( 2)", passed)
 call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,ueps(m,p,1)))), 0, &
               "g.pr.ueps ( 1)", passed)
  call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,ueps(m,p,-1)))), 0, &
               "g.pr.ueps (-1)", passed)
  call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,ueps(m,p,-2))))), 0, &
               "g.pr.ueps (-2)", passed)
 call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,veps(m,p,2))))), 0, &
               "g.pr.veps (2)", passed)
 call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,veps(m,p,1)))), 0, &
               "g.pr.veps (1)", passed)
  call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,veps(m,p,-1)))), 0, &
               "g.pr.veps (-1)", passed)
 call expect (abs(f_potgr (c_one, c_one, (cmplx (p*p - m**2, m*w, &
               kind=default) * pr_grav(p,m,w,veps(m,p,-2))))), 0, &
               "g.pr.veps (-2)", passed)
\langle omega\_bundle.f90 \rangle \equiv
  \langle omega\_vectors.f90 \rangle
  ⟨omega_spinors.f90⟩
  ⟨omega_bispinors.f90⟩
  ⟨omega_vectorspinors.f90⟩
  ⟨omega_polarizations.f90⟩
  ⟨omega_tensors.f90⟩
  ⟨omega_tensor_polarizations.f90⟩
  ⟨omega_couplings.f90⟩
  ⟨omega_spinor_couplings.f90⟩
  ⟨omega_bispinor_couplings.f90⟩
  ⟨omega_vspinor_polarizations.f90⟩
  ⟨omega_utils.f90⟩
  ⟨omega95.f90⟩
  ⟨omega95_bispinors.f90⟩
  ⟨omega_parameters.f90⟩
  ⟨omega_parameters_madgraph.f90⟩
\langle omega\_bundle\_whizard.f90 \rangle \equiv
  (omega_bundle.f90)
  ⟨omega_parameters_whizard.f90⟩
```

Y.33 O'Mega Virtual Machine

This module defines the O'Mega Virtual Machine (OVM) completely, whereby all environmental dependencies like masses, widths and couplings have to be given to the constructor vm%init at runtime.

Support for Majorana particles and vectorspinors is only partially, especially all fusions are missing. Maybe it would be easier to make an additional omegavm95_bispinors to avoid namespace issues. Non-type specific chunks could be reused

```
\langle (omegavm95.f90)\\
\langle \langle Copyleft \rangle
module omegavm95
use kinds, only: default
use constants
use iso_varying_string, string_t => varying_string
use, intrinsic :: iso_fortran_env, only : input_unit, output_unit, error_unit
use omega95
```

```
use omega95_bispinors, only: bispinor, vectorspinor, veps, pr_grav
    use omega95_bispinors, only: bi_u => u
    use omega95_bispinors, only: bi_v => v
    use omega95_bispinors, only: bi_pr_psi => pr_psi
    use omega_bispinors, only: operator (*), operator (+)
    use omega_color, only: ovm_color_sum, OCF => omega_color_factor
    implicit none
    private
    ⟨ Utilities Declarations⟩
     (OVM Data Declarations)
    \langle OVM\ Instructions \rangle
  contains
    ⟨OVM Procedure Implementations⟩
    ⟨ Utilities Procedure Implementations⟩
  end module omegavm95
This might not be the proper place but I don't know where to put it
\langle Utilities \ Declarations \rangle \equiv
  integer, parameter, public :: stdin = input_unit
  integer, parameter, public :: stdout = output_unit
  integer, parameter, public :: stderr = error_unit
  integer, parameter :: MIN_UNIT = 11, MAX_UNIT = 99
\langle OVM \ Procedure \ Implementations \rangle \equiv
  subroutine find_free_unit (u, iostat)
    integer, intent(out) :: u
    integer, intent(out), optional :: iostat
    logical :: exists, is_open
    integer :: i, status
    do i = MIN_UNIT, MAX_UNIT
       inquire (unit = i, exist = exists, opened = is_open, &
             iostat = status)
       if (status == 0) then
           if (exists .and. .not. is_open) then
              if (present (iostat)) then
                 iostat = 0
              end if
              return
           end if
       end if
    end do
    if (present (iostat)) then
       iostat = -1
    end if
    u = -1
  end subroutine find_free_unit
These abstract data types would ideally be the interface to communicate quantum numbers between O'Mega
and Whizard. This gives full flexibility to change the representation at any time
\langle Utilities\ Declarations \rangle + \equiv
  public :: color_t
  type color_t
  contains
    procedure :: write => color_write
  end type color_t
  public :: col_discrete
  type, extends(color_t) :: col_discrete
    integer :: i
  end type col_discrete
  public :: flavor_t
  type flavor_t
  contains
    procedure :: write => flavor_write
  end type flavor_t
```

```
public :: flv_discrete
 type, extends(flavor_t) :: flv_discrete
   integer :: i
 end type flv_discrete
 public :: helicity_t
 type :: helicity_t
 contains
   procedure :: write => helicity_write
 end type helicity_t
 public :: hel_discrete
 type, extends(helicity_t) :: hel_discrete
   integer :: i
 end type hel_discrete
 public :: hel_trigonometric
 type, extends(helicity_t) :: hel_trigonometric
   real :: theta
 end type hel_trigonometric
 public :: hel_exponential
 type, extends(helicity_t) :: hel_exponential
   real :: phi
 end type hel_exponential
 {\tt public} \ :: \ {\tt hel\_spherical}
 type, extends(helicity_t) :: hel_spherical
   real :: theta, phi
 end type hel_spherical
\langle \mathit{Utilities Procedure Implementations} \rangle \equiv
 subroutine color_write (color, fh)
   class(color_t), intent(in) :: color
   integer, intent(in) :: fh
   select type(color)
   type is (col_discrete)
      write(fh, *) 'color_discrete%i
                                             = ', color%i
   end select
  end subroutine color_write
 subroutine helicity_write (helicity, fh)
   class(helicity_t), intent(in) :: helicity
   integer, intent(in) :: fh
   select type(helicity)
   type is (hel_discrete)
      write(fh, *) 'helicity_discrete%i
                                                    = ', helicity%i
   type is (hel_trigonometric)
      write(fh, *) 'helicity_trigonometric%theta = ', helicity%theta
   type is (hel_exponential)
      write(fh, *) 'helicity_exponential%phi
                                                     = ', helicity%phi
   type is (hel_spherical)
       write(fh, *) 'helicity_spherical%phi
                                                    = ', helicity%phi
       write(fh, *) 'helicity_spherical%theta
                                                    = ', helicity%theta
   end select
 end subroutine helicity_write
 subroutine flavor_write (flavor, fh)
   class(flavor_t), intent(in) :: flavor
   integer, intent(in) :: fh
   select type(flavor)
   type is (flv_discrete)
       write(fh, *) 'flavor_discrete%i
                                                  = ', flavor%i
   end select
  end subroutine flavor_write
```

Y.33.1 Memory Layout

```
Some internal parameters
\langle OVM \ Data \ Declarations \rangle \equiv
  integer, parameter :: len_instructions = 8
  integer, parameter :: N_version_lines = 2
  ! Comment lines including the first header description line
  integer, parameter :: N_comments = 6
  ! Actual data lines plus intermediate description lines
  ! 'description \n 1 2 3 \n description <math>\n 3 2 1' would count as 3
  integer, parameter :: N_header_lines = 5
  real(default), parameter, public :: N_ = three
This is the basic type of a VM
\langle OVM \ Data \ Declarations \rangle + \equiv
  type :: basic_vm_t
     private
     logical :: verbose
     type(string_t) :: bytecode_file
     integer :: bytecode_fh, out_fh
     integer :: N_instructions, N_levels
     integer :: N_table_lines
     integer, dimension(:, :), allocatable :: instructions
     integer, dimension(:), allocatable :: levels
  end type
```

To allow for a lazy evaluation of amplitudes, we have to keep track whether a wave function has already been computed, to avoid multiple-computing that would arise when the bytecode has redundant fusions, which is necessary for flavor and color MC (and helicity MC when we use Weyl-van-der-Waerden-spinors)

```
\langle \mathit{OVM}\ \mathit{Data}\ \mathit{Declarations} \rangle + \equiv
  type :: vm_scalar
     logical :: c
     complex(kind=default) :: v
 end type
  type :: vm_spinor
     logical :: c
     type(spinor) :: v
  end type
  type :: vm_conjspinor
     logical :: c
     type(conjspinor) :: v
  end type
  type :: vm_bispinor
     logical :: c
     type(bispinor) :: v
  end type
  type :: vm_vector
     logical :: c
     type(vector) :: v
  end type
  type :: vm_tensor_2
     logical :: c
     type(tensor) :: v
  end type
  type :: vm_tensor_1
     logical :: c
     type(tensor2odd) :: v
  end type
```

```
type :: vm_vectorspinor
     logical :: c
     type(vectorspinor) :: v
  end type
We need a memory pool for all the intermediate results
\langle OVM \ Data \ Declarations \rangle + \equiv
  type, public, extends (basic_vm_t) :: vm_t
     private
     type(string_t) :: version
     type(string_t) :: model
     integer :: N_momenta, N_particles, N_prt_in, N_prt_out, N_amplitudes
     ! helicities = helicity combinations
     integer :: N_helicities, N_col_flows, N_col_indices, N_flavors, N_col_factors
     integer :: N_scalars, N_spinors, N_conjspinors, N_bispinors
     integer :: N_vectors, N_tensors_2, N_tensors_1, N_vectorspinors
     \verb|integer|: N_coupl_real|, N_coupl_real|2, N_coupl_cmplx|, N_coupl_cmplx|2|
     integer, dimension(:, :), allocatable :: table_flavor
     integer, dimension(:, :, :), allocatable :: table_color_flows
     integer, dimension(:, :), allocatable :: table_spin
     logical, dimension(:, :), allocatable :: table_ghost_flags
     type(OCF), dimension(:), allocatable :: table_color_factors
     logical, dimension(:, :), allocatable :: table_flv_col_is_allowed
     real(default), dimension(:), allocatable :: coupl_real
     real(default), dimension(:, :), allocatable :: coupl_real2
     {\tt complex(default),\ dimension(:),\ allocatable\ ::\ coupl\_cmplx}
     complex(default), dimension(:, :), allocatable :: coupl_cmplx2
     real(default), dimension(:), allocatable :: mass
     real(default), dimension(:), allocatable :: width
     type(momentum), dimension(:), allocatable :: momenta
     complex(default), dimension(:), allocatable :: amplitudes
     complex(default), dimension(:, :, :), allocatable :: table_amplitudes
     class(flavor_t), dimension(:), allocatable :: flavor
     class(color_t), dimension(:), allocatable :: color
     ! gfortran 4.7
     !class(helicity_t), dimension(:), pointer :: helicity => null()
     integer, dimension(:), allocatable :: helicity
     type(vm_scalar), dimension(:), allocatable :: scalars
     type(vm_spinor), dimension(:), allocatable :: spinors
     type(vm_conjspinor), dimension(:), allocatable :: conjspinors
     type(vm_bispinor), dimension(:), allocatable :: bispinors
     type(vm_vector), dimension(:), allocatable :: vectors
     type(vm_tensor_2), dimension(:), allocatable :: tensors_2
     type(vm_tensor_1), dimension(:), allocatable :: tensors_1
     type(vm_vectorspinor), dimension(:), allocatable :: vectorspinors
     logical, dimension(:), allocatable :: hel_is_allowed
     real(default), dimension(:), allocatable :: hel_max_abs
     real(default) :: hel_sum_abs = 0, hel_threshold = 1E10
     integer :: hel_count = 0, hel_cutoff = 100
     integer, dimension(:), allocatable :: hel_map
     integer :: hel_finite
     logical :: cms
     logical :: openmp
  contains
   \langle VM: TBP \rangle
  end type
```

```
\langle OVM \ Procedure \ Implementations \rangle + \equiv
  subroutine alloc_arrays (vm)
    type(vm_t), intent(inout) :: vm
    integer :: i
    allocate (vm%table_flavor(vm%N_particles, vm%N_flavors))
    allocate (vm%table_color_flows(vm%N_col_indices, vm%N_particles, &
                                    vm%N_col_flows))
    allocate (vm%table_spin(vm%N_particles, vm%N_helicities))
    allocate (vm%table_ghost_flags(vm%N_particles, vm%N_col_flows))
    allocate (vm%table_color_factors(vm%N_col_factors))
    allocate (vm%table_flv_col_is_allowed(vm%N_flavors, vm%N_col_flows))
    allocate (vm%momenta(vm%N_momenta))
    allocate (vm%amplitudes(vm%N_amplitudes))
    allocate (vm%table_amplitudes(vm%N_flavors, vm%N_col_flows, &
                                   vm%N_helicities))
    vm%table_amplitudes = zero
    allocate (vm%scalars(vm%N_scalars))
    allocate (vm%spinors(vm%N_spinors))
    allocate (vm%conjspinors(vm%N_conjspinors))
    allocate (vm%bispinors(vm%N_bispinors))
    allocate (vm%vectors(vm%N_vectors))
    allocate (vm%tensors_2(vm%N_tensors_2))
    allocate (vm%tensors_1(vm%N_tensors_1))
    allocate (vm%vectorspinors(vm%N_vectorspinors))
    allocate (vm%hel_is_allowed(vm%N_helicities))
    vm%hel_is_allowed = .True.
    allocate (vm%hel_max_abs(vm%N_helicities))
    vm\%hel_max_abs = 0
    allocate (vm%hel_map(vm%N_helicities))
    vm%hel_map = (/(i, i = 1, vm%N_helicities)/)
    vm%hel_finite = vm%N_helicities
  end subroutine alloc_arrays
                                     Y.33.2 Controlling the VM
These type-bound procedures steer the VM
\langle VM: TBP \rangle \equiv
    procedure :: init => vm_init
    procedure :: write => vm_write
    procedure :: reset => vm_reset
    procedure :: run => vm_run
    procedure :: final => vm_final
The init completely sets the environment for the OVM. Parameters can be changed with reset without
reloading the bytecode.
\langle OVM \ Procedure \ Implementations \rangle + \equiv
  subroutine vm_init (vm, bytecode_file, version, model, &
      coupl_real, coupl_real2, coupl_cmplx, coupl_cmplx2, &
      mass, width, verbose, out_fh, openmp)
    class(vm_t), intent(out) :: vm
    type(string_t), intent(in) :: bytecode_file
    type(string_t), intent(in) :: version
    type(string_t), intent(in) :: model
    real(default), dimension(:), optional, intent(in) :: coupl_real
    real(default), dimension(:, :), optional, intent(in) :: coupl_real2
    complex(default), dimension(:), optional, intent(in) :: coupl_cmplx
    complex(default), dimension(:, :), optional, intent(in) :: coupl_cmplx2
    real(default), dimension(:), optional, intent(in) :: mass
    real(default), dimension(:), optional, intent(in) :: width
    logical, optional, intent(in) :: verbose
    integer, optional, intent(in) :: out_fh
    logical, optional, intent(in) :: openmp
    vm%bytecode_file = bytecode_file
    vm%version = version
    vm%model = model
```

```
if (present (coupl_real)) then
       allocate (vm%coupl_real (size (coupl_real)), source=coupl_real)
   end if
   if (present (coupl_real2)) then
       allocate (vm%coupl_real2 (2, size (coupl_real2, 2)), source=coupl_real2)
   end if
   if (present (coupl_cmplx)) then
       allocate (vm%coupl_cmplx (size (coupl_cmplx)), source=coupl_cmplx)
   if (present (coupl_cmplx2)) then
       allocate (vm%coupl_cmplx2 (2, size (coupl_cmplx2, 2)), &
                 source=coupl_cmplx2)
   end if
   if (present (mass)) then
       allocate (vm/mass(size(mass)), source=mass)
   end if
   if (present (width)) then
       allocate (vm%width(size (width)), source=width)
   if (present (openmp)) then
       vm%openmp = openmp
       vm%openmp = .false.
   end if
   vm\%cms = .false.
   call basic_init (vm, verbose, out_fh)
 end subroutine vm_init
\langle OVM \ Procedure \ Implementations \rangle + \equiv
  subroutine vm_reset (vm, &
      coupl_real, coupl_real2, coupl_cmplx, coupl_cmplx2, &
      mass, width, verbose, out_fh)
   class(vm_t), intent(inout) :: vm
   real(default), dimension(:), optional, intent(in) :: coupl_real
   real(default), dimension(:, :), optional, intent(in) :: coupl_real2
   complex(default), dimension(:), optional, intent(in) :: coupl_cmplx
   complex(default), dimension(:, :), optional, intent(in) :: coupl_cmplx2
   real(default), dimension(:), optional, intent(in) :: mass
   real(default), dimension(:), optional, intent(in) :: width
   logical, optional, intent(in) :: verbose
   integer, optional, intent(in) :: out_fh
   if (present (coupl_real)) then
       vm%coupl_real = coupl_real
   if (present (coupl_real2)) then
       vm%coupl_real2 = coupl_real2
   end if
   if (present (coupl_cmplx)) then
       vm%coupl_cmplx = coupl_cmplx
   if (present (coupl_cmplx2)) then
       vm%coupl_cmplx2 = coupl_cmplx2
   if (present (mass)) then
       vm\%mass = mass
    end if
   if (present (width)) then
       vm%width = width
   end if
   if (present (verbose)) then
       vm%verbose = verbose
   if (present (out_fh)) then
       vm%out_fh = out_fh
   end if
```

end subroutine vm_reset

```
Mainly for debugging
\langle OVM\ Procedure\ Implementations \rangle + \equiv
  subroutine vm_write (vm)
    class(vm_t), intent(in) :: vm
    integer :: i, j, k
    call basic_write (vm)
    write(vm%out_fh, *) 'table_flavor
                                                   = ', vm%table_flavor
    write(vm%out_fh, *) 'table_color_flows
                                                   = ', vm%table_color_flows
    write(vm%out_fh, *) 'table_spin
                                                   = ', vm%table_spin
    write(vm%out_fh, *) 'table_ghost_flags
                                                   = ', vm%table_ghost_flags
    write(vm%out_fh, *) 'table_color_factors
    do i = 1, size(vm%table_color_factors)
       write(vm%out_fh, *) vm%table_color_factors(i)%i1, &
            vm%table_color_factors(i)%i2, &
            vm%table_color_factors(i)%factor
    end do
    write(vm%out_fh, *) 'table_flv_col_is_allowed = ', &
                      vm%table_flv_col_is_allowed
    do i = 1, vm%N_flavors
       do j = 1, vm%N_col_flows
          do k = 1, vm%N_helicities
             write(vm%out_fh, *) 'table_amplitudes(f,c,h), f, c, h = ', vm%table_amplitudes(i,j,k), i, j, k
          end do
       end do
    end do
    if (allocated(vm%coupl_real)) then
       write(vm%out_fh, *) 'coupl_real
                                                = ', vm%coupl_real
    if (allocated(vm%coupl_real2)) then
       write(vm%out_fh, *) 'coupl_real2
                                                = ', vm%coupl_real2
    if (allocated(vm%coupl_cmplx)) then
       write(vm%out_fh, *) 'coupl_cmplx
                                                = ', vm%coupl_cmplx
    if (allocated(vm%coupl_cmplx2)) then
       write(vm%out_fh, *) 'coupl_cmplx2
                                                = ', vm%coupl_cmplx2
    end if
    write(vm%out_fh, *) 'mass
                                             = ', vm%mass
                                             = ', vm%width
    write(vm%out_fh, *) 'width
    write(vm%out_fh, *) 'momenta
                                             = ', vm%momenta
    ! gfortran 4.7
    !do i = 1, size(vm%flavor)
       !call vm%flavor(i)%write (vm%out_fh)
    !end do
    !do i = 1, size(vm%color)
       !call vm%color(i)%write (vm%out_fh)
    !end do
    !do i = 1, size(vm%helicity)
       !call vm%helicity(i)%write (vm%out_fh)
    !end do
    write(vm%out_fh, *) 'helicity
                                             = ', vm%helicity
    write(vm%out_fh, *) 'amplitudes
                                         = ', vm%amplitudes
                                       = ', vm%scalars
    write(vm%out_fh, *) 'scalars
                                       = ', vm%spinors
    write(vm%out_fh, *) 'spinors
                                       = ', vm%conjspinors
    write(vm%out_fh, *) 'conjspinors
                                       = ', vm%bispinors
    write(vm%out_fh, *) 'bispinors
    write(vm%out_fh, *) 'vectors
                                       = ', vm%vectors
    write(vm%out_fh, *) 'tensors_2
                                       = ', vm%tensors_2
    write(vm%out_fh, *) 'tensors_1
                                      = ', vm%tensors_1
    !!! !!! Regression with ifort 16.0.0
    !!! write(vm%out_fh, *) 'vectorspinors = ', vm%vectorspinors
    write(vm%out_fh, *) 'N_momenta
                                       = ', vm%N_momenta
    write(vm%out_fh, *) 'N_particles
                                        = ', vm%N_particles
```

```
write(vm%out_fh, *) 'N_prt_in
                                      = ', vm%N_prt_in
                                     = ', vm%N_prt_out
 write(vm%out_fh, *) 'N_prt_out
 write(vm%out_fh, *) 'N_amplitudes = ', vm%N_amplitudes
 write(vm%out_fh, *) 'N_helicities = ', vm%N_helicities
                                     = ', vm%N_col_flows
 write(vm%out_fh, *) 'N_col_flows
 write(vm%out_fh, *) 'N_col_indices = ', vm%N_col_indices
                                      = ', vm%N_flavors
 write(vm%out_fh, *) 'N_flavors
                                     = ', vm%N_col_factors
 write(vm%out_fh, *) 'N_col_factors
                                      = ', vm%N_scalars
 write(vm%out_fh, *) 'N_scalars
                                     = ', vm%N_spinors
 write(vm%out_fh, *) 'N_spinors
 write(vm%out_fh, *) 'N_conjspinors = ', vm%N_conjspinors
                                     = ', vm%N_bispinors
 write(vm%out_fh, *) 'N_bispinors
                                     = ', vm%N_vectors
 write(vm%out_fh, *) 'N_vectors
                                      = ', vm%N_tensors_2
 write(vm%out_fh, *) 'N_tensors_2
                                    = ', vm%N_tensors_1
 write(vm%out_fh, *) 'N_tensors_1
 write(vm%out_fh, *) 'N_vectorspinors = ', vm%N_vectorspinors
 write(vm%out_fh, *) 'Overall size of VM: '
  ! GNU extension
  ! write(vm%out_fh, *) 'sizeof(wavefunctions) = ', &
     sizeof(vm%scalars) + sizeof(vm%spinors) + sizeof(vm%conjspinors) + &
     sizeof(vm%bispinors) + sizeof(vm%vectors) + sizeof(vm%tensors_2) + &
     sizeof(vm%tensors_1) + sizeof(vm%vectorspinors)
  ! write(vm%out_fh, *) 'sizeof(mometa) = ', sizeof(vm%momenta)
  ! write(vm%out_fh, *) 'sizeof(amplitudes) = ', sizeof(vm%amplitudes)
  ! write(vm%out_fh, *) 'sizeof(tables) = ', &
     sizeof(vm%table_amplitudes) + sizeof(vm%table_spin) + &
     sizeof(vm%table_flavor) + sizeof(vm%table_flv_col_is_allowed) + &
     sizeof(vm%table_color_flows) + sizeof(vm%table_color_factors) + &
     sizeof(vm%table_ghost_flags)
end subroutine vm_write
```

Most of this is redundant (Fortran will deallocate when we leave the scope) but when we change from allocatables to pointers, it is necessary to avoid leaks

```
\langle OVM \ Procedure \ Implementations \rangle + \equiv
 subroutine vm_final (vm)
   class(vm_t), intent(inout) :: vm
   deallocate (vm%table_flavor)
   deallocate (vm%table_color_flows)
   deallocate (vm%table_spin)
   deallocate (vm%table_ghost_flags)
   deallocate (vm%table_color_factors)
   deallocate (vm%table_flv_col_is_allowed)
   if (allocated (vm%coupl_real)) then
      deallocate (vm%coupl_real)
   if (allocated (vm%coupl_real2)) then
       deallocate (vm%coupl_real2)
   end if
   if (allocated (vm%coupl_cmplx)) then
       deallocate (vm%coupl_cmplx)
   if (allocated (vm%coupl_cmplx2)) then
       deallocate (vm%coupl_cmplx2)
   if (allocated (vm%mass)) then
       deallocate (vm%mass)
    end if
   if (allocated (vm%width)) then
       deallocate (vm%width)
   end if
   deallocate (vm%momenta)
   deallocate (vm%flavor)
   deallocate (vm%color)
   deallocate (vm%helicity)
   deallocate (vm%amplitudes)
   deallocate (vm%table_amplitudes)
```

```
deallocate (vm%scalars)
deallocate (vm%spinors)
deallocate (vm%conjspinors)
deallocate (vm%bispinors)
deallocate (vm%vectors)
deallocate (vm%tensors_2)
deallocate (vm%tensors_1)
deallocate (vm%vectorspinors)
end subroutine vm_final
```

Handing over the polymorph object helicity didn't work out as planned. A work-around is the use of pointers. flavor and color are not yet used but would have to be changed to pointers as well. At least this potentially avoids copying. Actually, neither the allocatable nor the pointer version works in gfortran 4.7 due to the broken select type. Back to Stone Age, i.e. integers.

```
\langle OVM \ Procedure \ Implementations \rangle + \equiv
 subroutine vm_run (vm, mom, flavor, color, helicity)
   class(vm_t), intent(inout) :: vm
   real(default), dimension(0:3, *), intent(in) :: mom
   class(flavor_t), dimension(:), optional, intent(in) :: flavor
   class(color_t), dimension(:), optional, intent(in) :: color
    ! gfortran 4.7
    !class(helicity_t), dimension(:), optional, target, intent(in) :: helicity
   integer, dimension(:), optional, intent(in) :: helicity
   integer :: i, h, hi
   do i = 1, vm%N_particles
      if (i <= vm%N_prt_in) then
        vm%momenta(i) = - mom(:, i)
                                              ! incoming, crossing symmetry
        vm%momenta(i) = mom(:, i)
                                              ! outgoing
      end if
   end do
   if (present (flavor)) then
       allocate(vm%flavor(size(flavor)), source=flavor)
   else
       if (.not. (allocated (vm%flavor))) then
          allocate(flv_discrete::vm%flavor(vm%N_particles))
       end if
   end if
   if (present (color)) then
       allocate(vm%color(size(color)), source=color)
   else
       if (.not. (allocated (vm%color))) then
          allocate(col_discrete::vm%color(vm%N_col_flows))
       end if
   end if
    ! gfortran 4.7
   if (present (helicity)) then
       !vm%helicity => helicity
       vm%helicity = helicity
       call vm_run_one_helicity (vm, 1)
      !if (.not. (associated (vm%helicity))) then
         !allocate(hel_discrete::vm%helicity(vm%N_particles))
      !end if
      if (.not. (allocated (vm%helicity))) then
         allocate(vm%helicity(vm%N_particles))
      if (vm%hel_finite == 0) return
      do hi = 1, vm%hel_finite
        h = vm%hel_map(hi)
         !<Work around [[gfortran 4.7 Bug 56731]] Implementation>>
         vm%helicity = vm%table_spin(:,h)
         call vm_run_one_helicity (vm, h)
      end do
   end if
```

end subroutine vm_run

This only removes the ICE but still leads to a segmentation fault in gfortran 4.7. I am running out of ideas how to make this compiler work with arrays of polymorph datatypes.

```
⟨Work around gfortran 4.7 Bug 56731 Declarations⟩≡
  integer :: hj
\langle Work \ around \ gfortran \ 4.7 \ Bug \ 56731 \ Implementation \rangle \equiv
  do hj = 1, size(vm%helicity)
     select type (hel => vm%helicity(hj))
     type is (hel_discrete)
        hel%i = vm%table_spin(hj,h)
     end select
  end do
\langle Original \ version \rangle \equiv
  select type (hel => vm%helicity)
  type is (hel_discrete)
     hel(:)%i = vm%table_spin(:,h)
  end select
\langle OVM \ Procedure \ Implementations \rangle + \equiv
  subroutine vm_run_one_helicity (vm, h)
    class(vm_t), intent(inout) :: vm
    integer, intent(in) :: h
    integer :: f, c, i
    vm%amplitudes = zero
    if (vm%N_levels > 0) then
       call null_all_wfs (vm)
       call iterate_instructions (vm)
    end if
    i = 1
    do c = 1, vm\%N_col_flows
       do f = 1, vm%N_flavors
           if (vm%table_flv_col_is_allowed(f,c)) then
              vm%table_amplitudes(f,c,h) = vm%amplitudes(i)
              i = i + 1
           end if
       end do
    end do
  end subroutine
\langle OVM \ Procedure \ Implementations \rangle + \equiv
  subroutine null_all_wfs (vm)
    type(vm_t), intent(inout) :: vm
    integer :: i, j
    vm\%scalars\%c = .False.
    vm%scalars%v = zero
    vm\%spinors\%c = .False.
    vm\%conjspinors\%c = .False.
    vm%bispinors%c = .False.
    vm%vectorspinors%c = .False.
    do i = 1, 4
       vm%spinors%v%a(i) = zero
       vm%conjspinors%v%a(i) = zero
       vm%bispinors%v%a(i) = zero
       do j = 1, 4
           vm%vectorspinors%v%psi(i)%a(j) = zero
       end do
    end do
    vm%vectors%c = .False.
    vm%vectors%v%t = zero
    vm\%tensors_1\%c = .False.
    vm\%tensors_2\%c = .False.
    do i = 1, 3
       vm%vectors%v%x(i) = zero
       vm%tensors_1%v%e(i) = zero
```

```
vm%tensors_1%v%b(i) = zero
do j = 1, 3
     vm%tensors_2%v%t(i,j) = zero
end do
end do
end subroutine
```

Y.33.3 Reading the bytecode

```
\langle \mathit{OVM}\ \mathit{Procedure}\ \mathit{Implementations} \rangle + \equiv
    subroutine load_header (vm, IO)
         type(vm_t), intent(inout) :: vm
         integer, intent(inout) :: IO
         integer, dimension(len_instructions) :: line
         read(vm%bytecode_fh, fmt = *, iostat = IO) line
         vm%N_momenta = line(1)
         vm%N_particles = line(2)
         vm%N_prt_in = line(3)
         vm%N_prt_out = line(4)
         vm%N_amplitudes = line(5)
         vm%N_helicities = line(6)
         vm\%N\_col\_flows = line(7)
         if (vm\%N_momenta == 0) then
                vm%N_col_indices = 2
         else
                vm%N_col_indices = line(8)
         end if
         read(vm%bytecode_fh, fmt = *, iostat = I0)
         read(vm%bytecode_fh, fmt = *, iostat = IO) line
         vm%N_flavors = line(1)
         vm%N_col_factors = line(2)
         vm%N_scalars = line(3)
         vm%N_spinors = line(4)
         vm%N_conjspinors = line(5)
         vm%N_bispinors = line(6)
         vm%N_vectors = line(7)
         vm%N_tensors_2 = line(8)
         read(vm%bytecode_fh, fmt = *, iostat = I0)
         read(vm%bytecode_fh, fmt = *, iostat = IO) line
         vm%N_tensors_1 = line(1)
         vm%N_vectorspinors = line(2)
         ! Add 1 for seperating label lines like 'Another table'
         \label{lines} $$vm\%N\_table\_lines = vm\%N\_helicities + 1 + vm\%N\_flavors + 1 + vm\%N\_col\_flows \& $$vm\%N\_table\_lines = vm\%N\_helicities + 1 + vm\%N\_flavors + 1 + vm\%N\_col\_flows & $$vm\%N\_table\_lines = vm\%N\_helicities + 1 + vm\%N\_flavors + 1 + vm\%N\_col\_flows & $$vm\%N\_table\_lines = vm\%N\_helicities + 1 + vm\%N\_flavors + 1 + vm\%N\_col\_flows & $$vm\%N\_table\_lines = vm\%N\_helicities + 1 + vm\%N\_flavors + 1 + vm\%N\_col\_flows & $$vm\%N\_table\_lines = vm\%N\_helicities + 1 + vm\%N\_flavors + 1 + vm\%N\_col\_flows & $$vm\%N\_table\_lines = vm\%N\_helicities + 1 + vm\%N\_table\_lines +
              + 1 + vm%N_col_flows + 1 + vm%N_col_factors + 1 + vm%N_col_flows
    end subroutine load_header
\langle OVM \ Procedure \ Implementations \rangle + \equiv
    subroutine read_tables (vm, IO)
         type(vm_t), intent(inout) :: vm
         integer, intent(inout) :: IO
         integer :: i
         integer, dimension(2) :: tmpcf
         integer, dimension(3) :: tmpfactor
         integer, dimension(vm%N_flavors) :: tmpF
         integer, dimension(vm%N_particles) :: tmpP
         real(default) :: factor
         do i = 1, vm%N_helicities
              read(vm%bytecode_fh, fmt = *, iostat = IO) vm%table_spin(:, i)
         end do
         read(vm%bytecode_fh, fmt = *, iostat = I0)
         do i = 1, vm%N_flavors
              read(vm%bytecode_fh, fmt = *, iostat = IO) vm%table_flavor(:, i)
         end do
```

```
read(vm%bytecode_fh, fmt = *, iostat = IO)
    do i = 1, vm%N_col_flows
      read(vm%bytecode_fh, fmt = *, iostat = I0) vm%table_color_flows(:, :, i)
    end do
    read(vm%bytecode_fh, fmt = *, iostat = I0)
    do i = 1, vm%N_col_flows
      read(vm%bytecode_fh, fmt = *, iostat = IO) tmpP
      vm%table_ghost_flags(:, i) = int_to_log(tmpP)
    end do
    read(vm%bytecode_fh, fmt = *, iostat = IO)
    do i = 1, vm%N_col_factors
      read(vm%bytecode_fh, fmt = '(219)', iostat = IO, advance='no') tmpcf
      factor = zero
        read(vm%bytecode_fh, fmt = '(3I9)', iostat = IO, advance='no', EOR=10) tmpfactor
        factor = factor + color_factor(tmpfactor(1), tmpfactor(2), tmpfactor(3))
      10 vm%table_color_factors(i) = OCF(tmpcf(1), tmpcf(2), factor)
    read(vm%bytecode_fh, fmt = *, iostat = IO)
    do i = 1, vm%N_col_flows
      read(vm%bytecode_fh, fmt = *, iostat = IO) tmpF
      vm%table_flv_col_is_allowed(:, i) = int_to_log(tmpF)
    end do
  end subroutine read_tables
This checking has proven useful more than once
\langle OVM\ Procedure\ Implementations \rangle + \equiv
  subroutine extended_version_check (vm, IO)
    type(vm_t), intent(in) :: vm
    integer, intent(inout) :: IO
    character(256) :: buffer
    read(vm%bytecode_fh, fmt = "(A)", iostat = IO) buffer
    if (vm%version /= buffer) then
      print *, "Warning: Bytecode has been generated with an older O'Mega version."
    else
      if (vm%verbose) then
         write (vm%out_fh, fmt = *) "Bytecode version fits."
      end if
    end if
  end subroutine extended_version_check
This chunk is copied verbatim from the basic_vm
\langle OVM \ Procedure \ Implementations \rangle + \equiv
  subroutine basic_init (vm, verbose, out_fh)
    type(vm_t), intent(inout) :: vm
    logical, optional, intent(in) :: verbose
    integer, optional, intent(in) :: out_fh
    if (present (verbose)) then
       vm%verbose = verbose
    else
       vm%verbose = .true.
    end if
    if (present (out_fh)) then
       vm%out_fh = out_fh
     else
       vm%out_fh = stdout
    end if
    call set_stream (vm)
    call alloc_and_count (vm)
    if (vm%N_levels > 0) then
```

```
call read_bytecode (vm)
     call sanity_check (vm)
 end if
 close (vm%bytecode_fh)
end subroutine basic_init
subroutine basic_write (vm)
 type(vm_t), intent(in) :: vm
 integer :: i
 write (vm%out_fh, *) '====> VM ', char(vm%version), ' <====='</pre>
 write (vm%out_fh, *) 'verbose
                                              ', vm%verbose
 write (vm%out_fh, *) 'bytecode_file
                                               ', char (vm%bytecode_file)
 write (vm%out_fh, *) 'N_instructions
                                              ', vm%N_instructions
                                         =
 write (vm%out_fh, *) 'N_levels
                                              ', vm%N_levels
                                         =
 write (vm%out_fh, *) 'instructions
 do i = 1, vm%N_instructions
     write (vm%out_fh, *) vm%instructions(:, i)
 end do
 write (vm%out_fh, *) 'levels
                                              ', vm%levels
end subroutine basic_write
subroutine alloc_and_count (vm)
 type(vm_t), intent(inout) :: vm
 integer, dimension(len_instructions) :: line
 character(256) :: buffer
 integer :: i, IO
 read(vm%bytecode_fh, fmt = "(A)", iostat = IO) buffer
 if (vm%model /= buffer) then
   print *, "Warning: Bytecode has been generated with an older O'Mega version."
 else
    if (vm%verbose) then
       write (vm%out_fh, fmt = *) "Using the model: "
       write (vm%out_fh, fmt = *) char(vm%model)
    end if
 end if
 call extended_version_check (vm, IO)
 if (vm%verbose) then
     write (vm%out_fh, fmt = *) "Trying to allocate."
 end if
 do i = 1, N_comments
   read(vm%bytecode_fh, fmt = *, iostat = IO)
 end do
 call load_header (vm, IO)
 call alloc_arrays (vm)
 if (vm%N_momenta /= 0) then
     do i = 1, vm%N_table_lines + 1
       read(vm%bytecode_fh, fmt = *, iostat = I0)
     end do
     vm\%N_instructions = 0
     vm\%N_levels = 0
     do
       read(vm%bytecode_fh, fmt = *, end = 42) line
       if (line(1) /= 0) then
         vm%N_instructions = vm%N_instructions + 1
         vm%N_levels = vm%N_levels + 1
       end if
     end do
     42 rewind(vm%bytecode_fh, iostat = IO)
     allocate (vm%instructions(len_instructions, vm%N_instructions))
     allocate (vm%levels(vm%N_levels))
     if (IO /= 0) then
      print *, "Error: vm.alloc : Couldn't load bytecode!"
       stop 1
     end if
 end if
```

```
end subroutine alloc_and_count
subroutine read_bytecode (vm)
 type(vm_t), intent(inout) :: vm
 integer, dimension(len_instructions) :: line
 integer :: i, j, IO
 ! Jump over version number, comments, header and first table description \ensuremath{\mathsf{I}}
 do i = 1, N_version_lines + N_comments + N_header_lines + 1
   read (vm%bytecode_fh, fmt = *, iostat = IO)
 end do
 call read_tables (vm, IO)
 read (vm%bytecode_fh, fmt = *, iostat = IO)
 i = 0; j = 0
 do
    read (vm%bytecode_fh, fmt = *, iostat = IO) line
    if (IO /= 0) exit
    if (line(1) == 0) then
      if (j <= vm\%N_levels) then
        j = j + 1
        vm\%levels(j) = i
                                          ! last index of a level is saved
        print *, 'Error: vm.read_bytecode: File has more levels than anticipated!'
      end if
    else
      if (i <= vm%N_instructions) then
                                          ! A valid instruction line
        i = i + 1
        vm%instructions(:, i) = line
        print *, 'Error: vm.read_bytecode: File is larger than anticipated!'
      end if
    end if
  end do
end subroutine read_bytecode
subroutine iterate_instructions (vm)
 type(vm_t), intent(inout) :: vm
 integer :: i, j
 if (vm%openmp) then
     !$omp parallel
     do j = 1, vm\%N_levels - 1
        !$omp do schedule (static)
        do i = vm\%levels (j) + 1, vm\%levels (j + 1)
           call decode (vm, i)
        end do
        !$omp end do
     end do
     !$omp end parallel
     do j = 1, vm\%N_levels - 1
        do i = vm%levels (j) + 1, vm%levels (j + 1)
           call decode (vm, i)
        end do
     end do
 end if
end subroutine iterate_instructions
subroutine set_stream (vm)
 type(vm_t), intent(inout) :: vm
 integer :: IO
 call find_free_unit (vm%bytecode_fh, IO)
 open (vm%bytecode_fh, file = char (vm%bytecode_file), form = 'formatted', &
    access = 'sequential', status = 'old', position = 'rewind', iostat = IO, &
    action = 'read')
 if (IO /= 0) then
```

```
print *, "Error: vm.set_stream: Bytecode file '", char(vm%bytecode_file), &
                "' not found!"
      stop 1
    end if
  end subroutine set_stream
  subroutine sanity_check (vm)
    type(vm_t), intent(in) :: vm
    if (vm%levels(1) /= 0) then
       print *, "Error: vm.vm_init: levels(1) != 0"
       stop 1
    end if
    if (vm%levels(vm%N_levels) /= vm%N_instructions) then
       print *, "Error: vm.vm_init: levels(N_levels) != N_instructions"
       stop 1
    end if
    if (vm%verbose) then
      write(vm%out_fh, *) "vm passed sanity check. Starting calculation."
  end subroutine sanity_check
                                      Y.33.4 Main Decode Function
This is the heart of the OVM
\langle OVM \ Procedure \ Implementations \rangle + \equiv
    ! pure & ! if no warnings
    subroutine decode (vm, instruction_index)
      type(vm_t), intent(inout) :: vm
       integer, intent(in) :: instruction_index
       integer, dimension(len_instructions) :: i, curr
       complex(default) :: braket
       integer :: tmp
      real(default) :: w
       i = vm%instructions (:, instruction_index)
       select case (i(1))
                           ! Jump over subinstructions
       case ( : -1)
       \langle \mathtt{case} s \ of \ \mathtt{decode} \rangle
       case (0)
        print *, 'Error: Levelbreak put in decode! Line:', &
                   instruction_index
        stop 1
       case default
        print *, "Error: Decode has case not catched! Line: ", &
                   instruction_index
         stop 1
       end select
    end subroutine decode
                                                    Momenta
The most trivial instruction
\langle OVM\ Instructions \rangle \equiv
  integer, parameter :: ovm_ADD_MOMENTA = 1
\langle \mathtt{case} s \ of \ \mathtt{decode} \rangle \equiv
  case (ovm_ADD_MOMENTA)
     vm%momenta(i(4)) = vm%momenta(i(5)) + vm%momenta(i(6))
     if (i(7) > 0) then
        vm%momenta(i(4)) = vm%momenta(i(4)) + vm%momenta(i(7))
     end if
```

Loading External states

```
\langle OVM\ Instructions \rangle + \equiv
  integer, parameter :: ovm_LOAD_SCALAR = 10
  integer, parameter :: ovm_LOAD_SPINOR_INC = 11
  integer, parameter :: ovm_LOAD_SPINOR_OUT = 12
  integer, parameter :: ovm_LOAD_CONJSPINOR_INC = 13
  integer, parameter :: ovm_LOAD_CONJSPINOR_OUT = 14
  integer, parameter :: ovm_LOAD_MAJORANA_INC = 15
  integer, parameter :: ovm_LOAD_MAJORANA_OUT = 16
  integer, parameter :: ovm_LOAD_VECTOR_INC = 17
  integer, parameter :: ovm_LOAD_VECTOR_OUT = 18
  integer, parameter :: ovm_LOAD_VECTORSPINOR_INC = 19
  integer, parameter :: ovm_LOAD_VECTORSPINOR_OUT = 20
  integer, parameter :: ovm_LOAD_TENSOR2_INC = 21
  integer, parameter :: ovm_LOAD_TENSOR2_OUT = 22
  integer, parameter :: ovm_LOAD_BRS_SCALAR = 30
  integer, parameter :: ovm_LOAD_BRS_SPINOR_INC = 31
  integer, parameter :: ovm_LOAD_BRS_SPINOR_OUT = 32
  integer, parameter :: ovm_LOAD_BRS_CONJSPINOR_INC = 33
  integer, parameter :: ovm_LOAD_BRS_CONJSPINOR_OUT = 34
  integer, parameter :: ovm_LOAD_BRS_VECTOR_INC = 37
  integer, parameter :: ovm_LOAD_BRS_VECTOR_OUT = 38
  integer, parameter :: ovm_LOAD_MAJORANA_GHOST_INC = 23
  integer, parameter :: ovm_LOAD_MAJORANA_GHOST_OUT = 24
  integer, parameter :: ovm_LOAD_BRS_MAJORANA_INC = 35
  integer, parameter :: ovm_LOAD_BRS_MAJORANA_OUT = 36
\langle case s \ of \ decode \rangle + \equiv
  case (ovm_LOAD_SCALAR)
    vm%scalars(i(4))%v = one
    vm\%scalars(i(4))\%c = .True.
  case (ovm_LOAD_SPINOR_INC)
     call load_spinor(vm%spinors(i(4)), - \langle p \rangle, \langle m \rangle, &
                        vm%helicity(i(5)), ovm_LOAD_SPINOR_INC)
  case (ovm_LOAD_SPINOR_OUT)
     call load_spinor(vm%spinors(i(4)), \langle p \rangle, \langle m \rangle, &
                        vm%helicity(i(5)), ovm_LOAD_SPINOR_OUT)
  case (ovm_LOAD_CONJSPINOR_INC)
     call load_conjspinor(vm\conjspinors(i(4)), - \langle p \rangle, &
       \langle m \rangle, vm%helicity(i(5)), ovm_LOAD_CONJSPINOR_INC)
  case (ovm_LOAD_CONJSPINOR_OUT)
     call load_conjspinor(vm%conjspinors(i(4)), \langle p \rangle, &
       \langle m \rangle, vm%helicity(i(5)), ovm_LOAD_CONJSPINOR_OUT)
  case (ovm_LOAD_MAJORANA_INC)
     call load_bispinor(vm%bispinors(i(4)), - \langle p \rangle, &
       \langle m \rangle, vm%helicity(i(5)), ovm_LOAD_MAJORANA_INC)
  case (ovm_LOAD_MAJORANA_OUT)
     call load_bispinor(vm%bispinors(i(4)), \langle p \rangle, \langle m \rangle, &
                        vm%helicity(i(5)), ovm_LOAD_MAJORANA_OUT)
  case (ovm_LOAD_VECTOR_INC)
     call load_vector(vm%vectors(i(4)), - \langle p \rangle, \langle m \rangle, &
                        vm%helicity(i(5)), ovm_LOAD_VECTOR_INC)
  case (ovm_LOAD_VECTOR_OUT)
     call load_vector(vm%vectors(i(4)), \langle p \rangle, \langle m \rangle, &
                        vm%helicity(i(5)), ovm_LOAD_VECTOR_OUT)
  case (ovm_LOAD_VECTORSPINOR_INC)
```

```
!select type (h => vm%helicity(i(5)))
  !type is (hel_discrete)
      !vm%vectorspinors(i(4))%v = veps(\langle m \rangle, - \langle p \rangle, &
                                           !h%i)
  !end select
  vm%vectorspinors(i(4))%v = veps(\langle m \rangle, - \langle p \rangle, &
                                        vm%helicity(i(5)))
  vm%vectorspinors(i(4))%c = .True.
case (ovm_LOAD_VECTORSPINOR_OUT)
  !select type (h => vm%helicity(i(5)))
  !type is (hel_discrete)
      !vm%vectorspinors(i(4))%v = veps(\langle m \rangle, \langle p \rangle, &
                                           !h%i)
  !end select
  vm%vectorspinors(i(4))%v = veps(\langle m \rangle, \langle p \rangle, &
                                        vm%helicity(i(5)))
  vm%vectorspinors(i(4))%c = .True.
case (ovm_LOAD_TENSOR2_INC)
  !select type (h => vm%helicity(i(5)))
  !type is (hel_discrete)
      !vm\%tensors_2(i(4))%v = eps2(\langle m \rangle, - \langle p \rangle, &
                                       !h%i)
  !end select
  vm\%tensors_2(i(4))\%c = .True.
case (ovm_LOAD_TENSOR2_OUT)
  !select type (h => vm%helicity(i(5)))
  !type is (hel_discrete)
      !vm%tensors_2(i(4))%v = eps2(\langle m \rangle, \langle p \rangle, h%i)
  !end select
  vm\%tensors_2(i(4))\%c = .True.
case (ovm_LOAD_BRS_SCALAR)
  vm%scalars(i(4))%v = (0, -1) * (\langle p \rangle * \langle p \rangle - &
                                        \langle m \rangle **2)
  vm\%scalars(i(4))\%c = .True.
case (ovm_LOAD_BRS_SPINOR_INC)
  print *, 'not implemented'
  stop 1
case (ovm_LOAD_BRS_SPINOR_OUT)
  print *, 'not implemented'
  stop 1
case (ovm_LOAD_BRS_CONJSPINOR_INC)
  print *, 'not implemented'
  stop 1
case (ovm_LOAD_BRS_CONJSPINOR_OUT)
  print *, 'not implemented'
  stop 1
case (ovm_LOAD_BRS_VECTOR_INC)
  print *, 'not implemented'
case (ovm_LOAD_BRS_VECTOR_OUT)
  print *, 'not implemented'
  stop 1
case (ovm_LOAD_MAJORANA_GHOST_INC)
  print *, 'not implemented'
case (ovm_LOAD_MAJORANA_GHOST_OUT)
  print *, 'not implemented'
case (ovm_LOAD_BRS_MAJORANA_INC)
  print *, 'not implemented'
  stop 1
```

```
case (ovm_LOAD_BRS_MAJORANA_OUT)
  print *, 'not implemented'
  stop 1
```

 $\langle OVM\ Instructions \rangle + \equiv$

Brakets and Fusions

NB: during, execution, the type of the coupling constant is implicit in the instruction

```
integer, parameter :: ovm_CALC_BRAKET = 2
integer, parameter :: ovm_FUSE_V_FF = -1
integer, parameter :: ovm_FUSE_F_VF = -2
integer, parameter :: ovm_FUSE_F_FV = -3
integer, parameter :: ovm_FUSE_VA_FF = -4
integer, parameter :: ovm_FUSE_F_VAF = -5
integer, parameter :: ovm_FUSE_F_FVA = -6
integer, parameter :: ovm_FUSE_VA2_FF = -7
integer, parameter :: ovm_FUSE_F_VA2F = -8
integer, parameter :: ovm_FUSE_F_FVA2 = -9
integer, parameter :: ovm_FUSE_A_FF = -10
integer, parameter :: ovm_FUSE_F_AF = -11
integer, parameter :: ovm_FUSE_F_FA = -12
integer, parameter :: ovm_FUSE_VL_FF = -13
integer, parameter :: ovm_FUSE_F_VLF = -14
integer, parameter :: ovm_FUSE_F_FVL = -15
integer, parameter :: ovm_FUSE_VR_FF = -16
integer, parameter :: ovm_FUSE_F_VRF = -17
integer, parameter :: ovm_FUSE_F_FVR = -18
integer, parameter :: ovm_FUSE_VLR_FF = -19
integer, parameter :: ovm_FUSE_F_VLRF = -20
integer, parameter :: ovm_FUSE_F_FVLR = -21
integer, parameter :: ovm_FUSE_SP_FF = -22
integer, parameter :: ovm_FUSE_F_SPF = -23
integer, parameter :: ovm_FUSE_F_FSP = -24
integer, parameter :: ovm_FUSE_S_FF = -25
integer, parameter :: ovm_FUSE_F_SF = -26
integer, parameter :: ovm_FUSE_F_FS = -27
integer, parameter :: ovm_FUSE_P_FF = -28
integer, parameter :: ovm_FUSE_F_PF = -29
integer, parameter :: ovm_FUSE_F_FP = -30
integer, parameter :: ovm_FUSE_SL_FF = -31
integer, parameter :: ovm_FUSE_F_SLF = -32
integer, parameter :: ovm_FUSE_F_FSL = -33
integer, parameter :: ovm_FUSE_SR_FF = -34
integer, parameter :: ovm_FUSE_F_SRF = -35
integer, parameter :: ovm_FUSE_F_FSR = -36
integer, parameter :: ovm_FUSE_SLR_FF = -37
integer, parameter :: ovm_FUSE_F_SLRF = -38
integer, parameter :: ovm_FUSE_F_FSLR = -39
integer, parameter :: ovm_FUSE_G_GG = -40
integer, parameter :: ovm_FUSE_V_SS = -41
integer, parameter :: ovm_FUSE_S_VV = -42
integer, parameter :: ovm_FUSE_S_VS = -43
integer, parameter :: ovm_FUSE_V_SV = -44
integer, parameter :: ovm_FUSE_S_SS = -45
integer, parameter :: ovm_FUSE_S_SVV = -46
integer, parameter :: ovm_FUSE_V_SSV = -47
integer, parameter :: ovm_FUSE_S_SSS = -48
integer, parameter :: ovm_FUSE_V_VVV = -49
integer, parameter :: ovm_FUSE_S_G2 = -50
integer, parameter :: ovm_FUSE_G_SG = -51
integer, parameter :: ovm_FUSE_G_GS = -52
integer, parameter :: ovm_FUSE_S_G2_SKEW = -53
```

```
integer, parameter :: ovm_FUSE_G_SG_SKEW = -54
  integer, parameter :: ovm_FUSE_G_GS_SKEW = -55
Shorthands
\langle p \rangle \equiv
  vm%momenta(i(5))
\langle m \rangle \equiv
  vm%mass(i(2))
\langle p1 \rangle \equiv
  vm%momenta(curr(6))
\langle p2 \rangle \equiv
  vm%momenta(curr(8))
\langle v1 \rangle \equiv
  vm%vectors(curr(5))%v
\langle v2\rangle \equiv
  vm%vectors(curr(7))%v
\langle s1 \rangle \equiv
  vm%scalars(curr(5))%v
\langle s2 \rangle \equiv
  vm%scalars(curr(7))%v
  sgn_coupl_cmplx(vm, curr(2))
\langle c1 \rangle \equiv
  sgn_coupl_cmplx2(vm, curr(2), 1)
\langle c2 \rangle \equiv
  sgn_coupl_cmplx2(vm, curr(2), 2)
\langle check \ for \ matching \ color \ and \ flavor \ amplitude \ of \ braket \ (old) \rangle \equiv
  if ((i(4) == o\%cols(1)) .or. (i(4) == o\%cols(2)) .or. &
     ((mode%col_MC .eq. FULL_SUM) .or. (mode%col_MC .eq. DIAG_COL))) then
Just a stub for now. Will be reimplemented with the polymorph type color similar to the select type(helicity)
when we need it.
\langle check \ for \ matching \ color \ and \ flavor \ amplitude \rangle \equiv
\langle case s \ of \ decode \rangle + \equiv
  case (ovm_CALC_BRAKET)
      \langle check \ for \ matching \ color \ and \ flavor \ amplitude \rangle
      tmp = instruction_index + 1
      do
        if (tmp > vm%N_instructions) exit
        curr = vm%instructions(:, tmp)
        if (curr(1) >= 0) exit
                                                                  ! End of fusions
        select case (curr(1))
        case (ovm_FUSE_V_FF, ovm_FUSE_VL_FF, ovm_FUSE_VR_FF)
           braket = vm%vectors(curr(4))%v * vec_ff(vm, curr)
        case (ovm_FUSE_F_VF, ovm_FUSE_F_VLF, ovm_FUSE_F_VRF)
           braket = vm%conjspinors(curr(4))%v * ferm_vf(vm, curr)
        case (ovm_FUSE_F_FV, ovm_FUSE_F_FVL, ovm_FUSE_F_FVR)
           braket = ferm_fv(vm, curr) * vm%spinors(curr(4))%v
        case (ovm_FUSE_VA_FF)
           braket = vm%vectors(curr(4))%v * vec_ff2(vm, curr)
        case (ovm_FUSE_F_VAF)
           braket = vm%conjspinors(curr(4))%v * ferm_vf2(vm, curr)
        case (ovm_FUSE_F_FVA)
           braket = ferm_fv2(vm, curr) * vm%spinors(curr(4))%v
        case (ovm_FUSE_S_FF, ovm_FUSE_SP_FF)
```

```
braket = vm%scalars(curr(4))%v * scal_ff(vm, curr)
case (ovm_FUSE_F_SF, ovm_FUSE_F_SPF)
  braket = vm%conjspinors(curr(4))%v * ferm_sf(vm, curr)
case (ovm_FUSE_F_FS, ovm_FUSE_F_FSP)
  braket = ferm_fs(vm, curr) * vm%spinors(curr(4))%v
case (ovm_FUSE_G_GG)
  braket = vm%vectors(curr(4))%v * &
     g_g(\langle c \rangle, \&
            \langle v1 \rangle, \langle p1 \rangle, &
            \langle v2 \rangle, \langle p2 \rangle)
case (ovm_FUSE_S_VV)
  braket = vm%scalars(curr(4))%v * \langle c \rangle * &
               (\langle v1 \rangle * vm%vectors(curr(6))%v)
case (ovm_FUSE_V_SS)
  braket = vm%vectors(curr(4))%v * &
               v_s(\langle c \rangle, \langle s1 \rangle, \langle p1 \rangle, \&
                               \langle s2 \rangle, \langle p2 \rangle)
case (ovm_FUSE_S_G2, ovm_FUSE_S_G2_SKEW)
  braket = vm%scalars(curr(4))%v * scal_g2(vm, curr)
case (ovm_FUSE_G_SG, ovm_FUSE_G_GS, ovm_FUSE_G_SG_SKEW, ovm_FUSE_G_GS_SKEW)
  braket = vm%vectors(curr(4))%v * gauge_sg(vm, curr)
case (ovm_FUSE_S_VS)
  braket = vm%scalars(curr(4))%v * &
     s_vs(\langle c \rangle, \&
            \langle v1 \rangle, \langle p1 \rangle, &
            \langle s2 \rangle, \langle p2 \rangle)
case (ovm_FUSE_V_SV)
  braket = (vm%vectors(curr(4))%v * vm%vectors(curr(6))%v) * &
               (\langle c \rangle * \langle s1 \rangle)
case (ovm_FUSE_S_SS)
  braket = vm%scalars(curr(4))%v * &
     \langle c \rangle * &
     (\langle s1 \rangle * vm\%scalars(curr(6))\%v)
case (ovm_FUSE_S_SSS)
  braket = vm%scalars(curr(4))%v * &
     \langle c \rangle * &
     (\langle s1 \rangle * vm\%scalars(curr(6))\%v * &
      \langle s2 \rangle)
case (ovm_FUSE_S_SVV)
  braket = vm%scalars(curr(4))%v * &
     \langle c \rangle * \&
     \langle s1 \rangle * (vm%vectors(curr(6))%v * &
                                       \langle v2\rangle)
case (ovm_FUSE_V_SSV)
  braket = vm%vectors(curr(4))%v * &
     (\langle c \rangle * \langle s1 \rangle * \&
      vm%scalars(curr(6))%v) * \langle v2 \rangle
case (ovm_FUSE_V_VVV)
  braket = \langle c \rangle * &
     (\langle v1 \rangle * vm%vectors(curr(6))%v) * &
     (vm%vectors(curr(4))%v * \langle v2 \rangle)
```

```
case default
         print *, 'Braket', curr(1), 'not implemented'
         stop 1
       end select
       vm%amplitudes(i(4)) = vm%amplitudes(i(4)) + curr(3) * braket
       tmp = tmp + 1
     end do
     vm%amplitudes(i(4)) = vm%amplitudes(i(4)) * i(2)
     if (i(5) > 1) then
       vm%amplitudes(i(4)) = vm%amplitudes(i(4)) * &
                                                                 ! Symmetry factor
                              (one / sqrt(real(i(5), kind=default)))
    end if
                                                Propagators
\langle OVM\ Instructions \rangle + \equiv
  integer, parameter :: ovm_PROPAGATE_SCALAR = 51
  integer, parameter :: ovm_PROPAGATE_COL_SCALAR = 52
 integer, parameter :: ovm_PROPAGATE_GHOST = 53
 integer, parameter :: ovm_PROPAGATE_SPINOR = 54
 integer, parameter :: ovm_PROPAGATE_CONJSPINOR = 55
 integer, parameter :: ovm_PROPAGATE_MAJORANA = 56
 integer, parameter :: ovm_PROPAGATE_COL_MAJORANA = 57
 integer, parameter :: ovm_PROPAGATE_UNITARITY = 58
 integer, parameter :: ovm_PROPAGATE_COL_UNITARITY = 59
 integer, parameter :: ovm_PROPAGATE_FEYNMAN = 60
 integer, parameter :: ovm_PROPAGATE_COL_FEYNMAN = 61
 integer, parameter :: ovm_PROPAGATE_VECTORSPINOR = 62
  integer, parameter :: ovm_PROPAGATE_TENSOR2 = 63
 integer, parameter :: ovm_PROPAGATE_NONE = 64
\langle check \ for \ matching \ color \ and \ flavor \ amplitude \ of \ propagator \ (old) \rangle \equiv
 if ((mode%col_MC .eq. FULL_SUM) .or. (mode%col_MC .eq. DIAG_COL)) then
   select case(i(1))
   case (ovm_PROPAGATE_PSI)
     go = .not. vm%spinors%c(i(4))
   case (ovm_PROPAGATE_PSIBAR)
     go = .not. vm%conjspinors%c(i(4))
   case (ovm_PROPAGATE_UNITARITY, ovm_PROPAGATE_FEYNMAN, &
     ovm_PROPAGATE_COL_FEYNMAN)
      go = .not. vm%vectors%c(i(4))
   end select
 else
   go = (i(8) == o\%cols(1)) .or. (i(8) == o\%cols(2))
 end if
 if (go) then
\langle case s \ of \ decode \rangle + \equiv
  ⟨check for matching color and flavor amplitude⟩
 case (ovm_PROPAGATE_SCALAR : ovm_PROPAGATE_NONE)
    tmp = instruction_index + 1
    do
       curr = vm%instructions(:,tmp)
       if (curr(1) >= 0) exit
                                                        ! End of fusions
       select case (curr(1))
       case (ovm_FUSE_V_FF, ovm_FUSE_VL_FF, ovm_FUSE_VR_FF)
         vm%vectors(curr(4))%v = vm%vectors(curr(4))%v + curr(3) * &
                                  vec_ff(vm, curr)
       case (ovm_FUSE_F_VF, ovm_FUSE_F_VLF, ovm_FUSE_F_VRF)
         vm%spinors(curr(4))%v = vm%spinors(curr(4))%v + curr(3) * &
                                  ferm_vf(vm, curr)
       case (ovm_FUSE_F_FV, ovm_FUSE_F_FVL, ovm_FUSE_F_FVR)
```

```
vm%conjspinors(curr(4))%v = vm%conjspinors(curr(4))%v + curr(3) * &
                                   ferm_fv(vm, curr)
case (ovm_FUSE_VA_FF)
  vm%vectors(curr(4))%v = vm%vectors(curr(4))%v + curr(3) * &
                               vec_ff2(vm, curr)
case (ovm_FUSE_F_VAF)
  vm%spinors(curr(4))%v = vm%spinors(curr(4))%v + curr(3) * &
                               ferm_vf2(vm, curr)
case (ovm_FUSE_F_FVA)
  vm%conjspinors(curr(4))%v = vm%conjspinors(curr(4))%v + curr(3) * &
                                   ferm_fv2(vm, curr)
case (ovm_FUSE_S_FF, ovm_FUSE_SP_FF)
  vm%scalars(curr(4))%v = vm%scalars(curr(4))%v + curr(3) * &
                               scal_ff(vm, curr)
case (ovm_FUSE_F_SF, ovm_FUSE_F_SPF)
  vm%spinors(curr(4))%v = vm%spinors(curr(4))%v + curr(3) * &
                               ferm_sf(vm, curr)
case (ovm_FUSE_F_FS, ovm_FUSE_F_FSP)
  vm%conjspinors(curr(4))%v = vm%conjspinors(curr(4))%v + curr(3) * &
                                   ferm_fs(vm, curr)
case (ovm_FUSE_G_GG)
  vm\\/vectors(curr(4))\\/v = vm\\/vectors(curr(4))\\/v + curr(3) * &
    g_g(\langle c \rangle, \langle v1 \rangle, \&
           \langle p1 \rangle, \langle v2 \rangle, &
           \langle p2 \rangle)
case (ovm_FUSE_S_VV)
  vm%scalars(curr(4))%v = vm%scalars(curr(4))%v + curr(3) * &
    \langle c \rangle * &
    (\langle v1 \rangle * vm%vectors(curr(6))%v)
case (ovm_FUSE_V_SS)
  vm%vectors(curr(4))%v = vm%vectors(curr(4))%v + curr(3) * &
             v_s(\langle c \rangle, \langle s1 \rangle, \langle p1 \rangle, &
                           \langle s2 \rangle, \langle p2 \rangle)
case (ovm_FUSE_S_G2, ovm_FUSE_S_G2_SKEW)
  vm%scalars(curr(4))%v = vm%scalars(curr(4))%v + &
                               scal_g2(vm, curr) * curr(3)
case (ovm_FUSE_G_SG, ovm_FUSE_G_GS, ovm_FUSE_G_SG_SKEW, ovm_FUSE_G_GS_SKEW)
  vm%vectors(curr(4))%v = vm%vectors(curr(4))%v + &
                               gauge_sg(vm, curr) * curr(3)
case (ovm_FUSE_S_VS)
  vm%scalars(curr(4))%v = vm%scalars(curr(4))%v + &
    s_vs(\langle c \rangle, \&
           \langle v1 \rangle, \langle p1 \rangle, &
           \langle s2 \rangle, \langle p2 \rangle) * curr(3)
case (ovm_FUSE_V_SV)
  vm%vectors(curr(4))%v = vm%vectors(curr(4))%v + &
    vm%vectors(curr(6))%v * &
    (\langle c \rangle * \langle s1 \rangle * curr(3))
case (ovm_FUSE_S_SS)
  vm%scalars(curr(4))%v = vm%scalars(curr(4))%v + &
    \langle c \rangle * \&
```

```
(\langle s1 \rangle * vm\%scalars(curr(6))\%v) * curr(3)
                      case (ovm_FUSE_S_SSS)
                            vm%scalars(curr(4))%v = vm%scalars(curr(4))%v + &
                                   \langle c \rangle * &
                                   (\langle s1 \rangle * vm\%scalars(curr(6))\%v * &
                                      \langle s2 \rangle) * curr(3)
                      case (ovm_FUSE_S_SVV)
                            vm%scalars(curr(4))%v = vm%scalars(curr(4))%v + &
                                   \langle c \rangle * \&
                                   \langle s1 \rangle * (vm%vectors(curr(6))%v * &
                                                                                                                   \langle v2\rangle) * curr(3)
                      case (ovm_FUSE_V_SSV)
                            vm\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\vectors(curr(4))\v
                                   (\langle c \rangle * \langle s1 \rangle * \&
                                      vm%scalars(curr(6))%v) * \langle v2 \rangle * curr(3)
                      case (ovm_FUSE_V_VVV)
                            vm%vectors(curr(4))%v = vm%vectors(curr(4))%v + &
                                   (\langle c \rangle * (\langle v1 \rangle * \&
                                      vm%vectors(curr(6))%v)) * curr(3) * \langle v2 \rangle
                      case default
                            print *, 'Fusion', curr(1), 'not implemented'
                            stop 1
                      end select
                      tmp = tmp + 1
                end do
               select case (i(3))
               case (0)
                     w = zero
               case (1)
                     w = vm\%width(i(2))
                     vm%cms = .false.
               case (2)
                      w = wd_t(\langle p \rangle, vm\%width(i(2)))
               case (3)
                     w = vm\%width(i(2))
                      vm\%cms = .true.
               case (4)
                      \label{eq:wd_run} \texttt{w} = \texttt{wd\_run}(\langle p \rangle, \ \langle m \rangle, \ \texttt{vm\%width(i(2))})
               case default
                        print *, 'not implemented'
               end select
               select case (i(1))
                \langle propagator \; \mathtt{case} s \; in \; \mathtt{decode} \rangle
               end select
\langle propagator \ cases \ in \ decode \rangle \equiv
     case (ovm_PROPAGATE_SCALAR)
            vm%scalars(i(4))%v = pr_phi(\langle p \rangle, \langle m \rangle, &
                            w, vm%scalars(i(4))%v)
            vm%scalars(i(4))%c = .True.
```

```
case (ovm_PROPAGATE_COL_SCALAR)
  vm%scalars(i(4))%v = - one / N_ * pr_phi(\langle p \rangle, &
        \langle m \rangle, w, vm%scalars(i(4))%v)
  vm\%scalars(i(4))\%c = .True.
case (ovm_PROPAGATE_GHOST)
  vm%scalars(i(4))%v = imago * pr_phi(\langle p \rangle, \langle m \rangle, &
        w, vm%scalars(i(4))%v)
  vm%scalars(i(4))%c = .True.
case (ovm_PROPAGATE_SPINOR)
  vm\%spinors(i(4))\%v = pr_psi(\langle p \rangle, \langle m \rangle, \&
        w, vm%cms, vm%spinors(i(4))%v)
  vm\%spinors(i(4))\%c = .True.
case (ovm_PROPAGATE_CONJSPINOR)
  vm%conjspinors(i(4))%v = pr_psibar(\langle p \rangle, \langle m \rangle, &
        w, vm%cms, vm%conjspinors(i(4))%v)
  vm%conjspinors(i(4))%c = .True.
case (ovm_PROPAGATE_MAJORANA)
  vm%bispinors(i(4))%v = bi_pr_psi(\langle p \rangle, \langle m \rangle, &
        w, vm%cms, vm%bispinors(i(4))%v)
  vm\%bispinors(i(4))\%c = .True.
case (ovm_PROPAGATE_COL_MAJORANA)
  vm%bispinors(i(4))%v = (- one / N_) * &
        bi\_pr\_psi(\langle p \rangle, \langle m \rangle, \&
        w, vm%cms, vm%bispinors(i(4))%v)
  vm\%bispinors(i(4))\%c = .True.
case (ovm_PROPAGATE_UNITARITY)
  vm%vectors(i(4))%v = pr_unitarity(\langle p \rangle, \langle m \rangle, &
        w, vm%cms, vm%vectors(i(4))%v)
  vm\\vectors(i(4))\\c = .True.
case (ovm_PROPAGATE_COL_UNITARITY)
  vm%vectors(i(4))%v = - one / N_ * pr_unitarity(\langle p \rangle, &
        \langle m \rangle, w, vm%cms, vm%vectors(i(4))%v)
  vm%vectors(i(4))%c = .True.
case (ovm_PROPAGATE_FEYNMAN)
  vm%vectors(i(4))%v = pr_feynman(\langle p \rangle, vm%vectors(i(4))%v)
  vm\\vectors(i(4))\\c = .True.
case (ovm_PROPAGATE_COL_FEYNMAN)
  vm\%vectors(i(4))\%v = - one / N_ * &
        pr_feynman(\langle p \rangle, vm%vectors(i(4))%v)
  vm\\vectors(i(4))\\c = .True.
case (ovm_PROPAGATE_VECTORSPINOR)
  vm%vectorspinors(i(4))%v = pr_grav(\langle p \rangle, \langle m \rangle, &
        w, vm%vectorspinors(i(4))%v)
  vm%vectorspinors(i(4))%c = .True.
case (ovm_PROPAGATE_TENSOR2)
  vm\%tensors_2(i(4))\%v = pr_tensor(\langle p \rangle, \langle m \rangle, \&
        w, vm%tensors_2(i(4))%v)
  vm\%tensors_2(i(4))\%c = .True.
case (ovm_PROPAGATE_NONE)
! This will not work with color MC. Appropriate type%c has to be set to
! .True.
```

Y.33.5 Helper functions

Factoring out these parts helps a lot to keep sane but might hurt the performance of the VM noticably. In that case, we have to copy & paste to avoid the additional function calls. Note that with preprocessor macros, we could maintain this factorized form (and factor out even more since types don't have to match), in case we would decide to allow this

```
\langle load\ outer\ wave\ function \rangle \equiv
  !select type (h)
  !type is (hel_trigonometric)
     !wf%v = (cos (h%theta) * load_wf (m, p, + 1) + &
              !sin (h%theta) * load_wf (m, p, - 1)) * sqrt2
  !type is (hel_exponential)
     !wf%v = exp (+ imago * h%phi) * load_wf (m, p, + 1) + &
             !exp (-imago * h\%phi) * load_wf (m, p, -1)
  !type is (hel_spherical)
     !wf%v = (exp (+ imago * h%phi) * cos (h%theta) * load_wf (m, p, + 1) + &
              ! exp (-imago * h\%phi) * sin (h\%theta) * load_wf (m, p, -1)) * &
             !sqrt2
  !type is(hel_discrete)
       !wf%v = load_wf (m, p, h%i)
  !end select
  wf%v = load_wf (m, p, h)
  wf%c = .True.
Caveat: Helicity MC not tested with Majorana particles but should be fine
\langle check \ for \ matching \ color \ and \ flavor \ amplitude \ of \ wf \ (old) \rangle \equiv
  if ((mode%col_MC .eq. FULL_SUM) .or. (mode%col_MC .eq. DIAG_COL)) then
    go = .not. vm%spinors%c(i(4))
  else
    go = (i(8) == o\%cols(1)) .or. (i(8) == o\%cols(2))
  end if
  if (go) ..
\langle OVM \ Procedure \ Implementations \rangle + \equiv
  subroutine load_bispinor(wf, p, m, h, opcode)
    type(vm_bispinor), intent(out) :: wf
    type(momentum), intent(in) :: p
    real(default), intent(in) :: m
    !class(helicity_t), intent(in) :: h
    integer, intent(in) :: h
    integer, intent(in) :: opcode
    procedure(bi_u), pointer :: load_wf
    ⟨check for matching color and flavor amplitude⟩
    select case (opcode)
    case (ovm_LOAD_MAJORANA_INC)
       load_wf => bi_u
    case (ovm_LOAD_MAJORANA_OUT)
       load_wf => bi_v
    case default
       load_wf => null()
    end select
    \langle load\ outer\ wave\ function \rangle
  end subroutine load_bispinor
  subroutine load_spinor(wf, p, m, h, opcode)
    type(vm_spinor), intent(out) :: wf
    type(momentum), intent(in) :: p
    real(default), intent(in) :: m
    !class(helicity_t), intent(in) :: h
    integer, intent(in) :: h
    integer, intent(in) :: opcode
    procedure(u), pointer :: load_wf
    \langle check\ for\ matching\ color\ and\ flavor\ amplitude \rangle
    select case (opcode)
    case (ovm_LOAD_SPINOR_INC)
       load_wf => u
    case (ovm_LOAD_SPINOR_OUT)
```

```
load_wf => v
    case default
       load_wf => null()
    end select
    \langle load\ outer\ wave\ function \rangle
  end subroutine load_spinor
  subroutine load_conjspinor(wf, p, m, h, opcode)
    type(vm_conjspinor), intent(out) :: wf
    type(momentum), intent(in) :: p
    real(default), intent(in) :: m
    !class(helicity_t), intent(in) :: h
    integer, intent(in) :: h
    integer, intent(in) :: opcode
    procedure(ubar), pointer :: load_wf
    \langle check\ for\ matching\ color\ and\ flavor\ amplitude \rangle
    select case (opcode)
    case (ovm_LOAD_CONJSPINOR_INC)
       load_wf => vbar
    case (ovm_LOAD_CONJSPINOR_OUT)
       load_wf => ubar
    case default
       load_wf => null()
    end select
    \langle load\ outer\ wave\ function \rangle
  end subroutine load_conjspinor
  subroutine load_vector(wf, p, m, h, opcode)
    type(vm_vector), intent(out) :: wf
    type(momentum), intent(in) :: p
    real(default), intent(in) :: m
    !class(helicity_t), intent(in) :: h
    integer, intent(in) :: h
    integer, intent(in) :: opcode
    procedure(eps), pointer :: load_wf
    ⟨check for matching color and flavor amplitude⟩
    load_wf => eps
    \langle load\ outer\ wave\ function \rangle
    if (opcode == ovm_LOAD_VECTOR_OUT) then
       wf%v = conjg(wf%v)
    end if
  end subroutine load_vector
\langle OVM \ Procedure \ Implementations \rangle + \equiv
  function ferm_vf(vm, curr) result (x)
    type(spinor) :: x
    class(vm_t), intent(in) :: vm
    integer, dimension(:), intent(in) :: curr
    procedure(f_vf), pointer :: load_wf
    select case (curr(1))
    case (ovm_FUSE_F_VF)
       load_wf => f_vf
    case (ovm_FUSE_F_VLF)
       load_wf => f_vlf
    case (ovm_FUSE_F_VRF)
       load_wf => f_vrf
    case default
       load_wf => null()
    \quad \hbox{end select} \quad
    x = load_wf(\langle c \rangle, \langle v1 \rangle, vm%spinors(curr(6))%v)
  end function ferm_vf
  function ferm_vf2(vm, curr) result (x)
    type(spinor) :: x
    class(vm_t), intent(in) :: vm
    integer, dimension(:), intent(in) :: curr
```

```
procedure(f_vaf), pointer :: load_wf
  select case (curr(1))
  case (ovm_FUSE_F_VAF)
     load_wf => f_vaf
  case default
     load_wf => null()
  end select
  x = f_vaf(\langle c1 \rangle, \langle c2 \rangle, \langle v1 \rangle, vm%spinors(curr(6))%v)
end function ferm_vf2
function ferm_sf(vm, curr) result (x)
  type(spinor) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  select case (curr(1))
  case (ovm_FUSE_F_SF)
     x = f_sf(\langle c \rangle, \langle s1 \rangle, vm%spinors(curr(6))%v)
  case (ovm_FUSE_F_SPF)
     x = f_spf(\langle c1 \rangle, \langle c2 \rangle, \langle s1 \rangle, vm%spinors(curr(6))%v)
  case default
  end select
end function ferm_sf
function ferm_fv(vm, curr) result (x)
  type(conjspinor) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  procedure(f_fv), pointer :: load_wf
  select case (curr(1))
  case (ovm_FUSE_F_FV)
     load_wf => f_fv
  case (ovm_FUSE_F_FVL)
     load_wf => f_fvl
  case (ovm_FUSE_F_FVR)
     load_wf => f_fvr
  case default
     load_wf => null()
  end select
  x = load_wf(\langle c \rangle, vm\%conjspinors(curr(5))\%v, vm\%vectors(curr(6))\%v)
end function ferm_fv
function ferm_fv2(vm, curr) result (x)
  type(conjspinor) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  procedure(f_fva), pointer :: load_wf
  select case (curr(1))
  case (ovm_FUSE_F_FVA)
     load_wf => f_fva
  case default
     load_wf => null()
  end select
  x = f_fva(\langle c1 \rangle, \langle c2 \rangle, &
              vm%conjspinors(curr(5))%v, vm%vectors(curr(6))%v)
end function ferm_fv2
function ferm_fs(vm, curr) result (x)
  type(conjspinor) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  procedure(f_fs), pointer :: load_wf
  select case (curr(1))
  case (ovm_FUSE_F_FS)
     x = f_fs(\langle c \rangle, vm\%conjspinors(curr(5))\%v, vm\%scalars(curr(6))\%v)
  case (ovm_FUSE_F_FSP)
     x = f_f sp(\langle c1 \rangle, \langle c2 \rangle, \&
```

```
vm%conjspinors(curr(5))%v, vm%scalars(curr(6))%v)
  case default
     x\%a = zero
  end select
end function ferm_fs
function vec_ff(vm, curr) result (x)
  type(vector) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  procedure(v_ff), pointer :: load_wf
  select case (curr(1))
  case (ovm_FUSE_V_FF)
     load_wf => v_ff
  case (ovm_FUSE_VL_FF)
     load_wf => vl_ff
  case (ovm_FUSE_VR_FF)
     load_wf => vr_ff
  case default
     load_wf => null()
  end select
  x = load_wf(\langle c \rangle, vm\%conjspinors(curr(5))\%v, vm\%spinors(curr(6))\%v)
end function vec_ff
function vec_ff2(vm, curr) result (x)
  type(vector) :: x
  {\tt class(vm\_t),\ intent(in)\ ::\ vm}
  integer, dimension(:), intent(in) :: curr
  procedure(va_ff), pointer :: load_wf
  select case (curr(1))
  case (ovm_FUSE_VA_FF)
     load_wf => va_ff
  case default
     load_wf => null()
  end select
  x = load_wf(\langle c1 \rangle, \langle c2 \rangle, \&
                vm%conjspinors(curr(5))%v, vm%spinors(curr(6))%v)
end function vec_ff2
function scal_ff(vm, curr) result (x)
  complex(default) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  select case (curr(1))
  case (ovm_FUSE_S_FF)
     x = s_ff(\langle c \rangle, \&
           vm%conjspinors(curr(5))%v, vm%spinors(curr(6))%v)
  case (ovm_FUSE_SP_FF)
     x = sp_ff(\langle c1 \rangle, \langle c2 \rangle, \&
           vm%conjspinors(curr(5))%v, vm%spinors(curr(6))%v)
  case default
     x = zero
  end select
end function scal_ff
function scal_g2(vm, curr) result (x)
  complex(default) :: x
  class(vm_t), intent(in) :: vm
  integer, dimension(:), intent(in) :: curr
  select case (curr(1))
  case (ovm_FUSE_S_G2)
     x = \langle c \rangle * ((\langle p1 \rangle * \langle v2 \rangle) * &
                     (\langle p2 \rangle * \langle v1 \rangle) - &
                     (\langle p1 \rangle * \langle p2 \rangle) * &
                     (\langle v2 \rangle * \langle v1 \rangle))
  case (ovm_FUSE_S_G2_SKEW)
```

```
x = - phi_vv(\langle c \rangle, \langle p1 \rangle, \langle p2 \rangle, &
                            \langle v1 \rangle, \langle v2 \rangle)
     case default
         x = zero
     end select
   end function scal_g2
  pure function gauge_sg(vm, curr) result (x)
     type(vector) :: x
     class(vm_t), intent(in) :: vm
     integer, dimension(:), intent(in) :: curr
     select case (curr(1))
     case (ovm_FUSE_G_SG)
         x = \langle c \rangle * \langle s1 \rangle * ( &
                 -((\langle p1 \rangle + \langle p2 \rangle) * &
                     \langle v2 \rangle) * \langle p2 \rangle - &
                 (-(\langle p1 \rangle + \langle p2 \rangle) * &
                     \langle p2 \rangle) * \langle v2 \rangle)
     case (ovm_FUSE_G_GS)
         x = \langle c \rangle * \langle s1 \rangle * ( &
                 -((\langle p1 \rangle + \langle p2 \rangle) * &
                     \langle v2 \rangle) * \langle p2 \rangle - &
                 (-(\langle p1 \rangle + \langle p2 \rangle) * &
                     \langle p2 \rangle) * \langle v2 \rangle)
     case (ovm_FUSE_G_SG_SKEW)
         x = -v_{phiv}(\langle c \rangle, \langle s1 \rangle, \langle p1 \rangle, &
                           \langle p2 \rangle, \langle v2 \rangle)
     case (ovm_FUSE_G_GS_SKEW)
         x = -v_{phiv}(\langle c \rangle, \langle s2 \rangle, \langle p1 \rangle, &
                            \langle p2 \rangle, \langle v1 \rangle)
     case default
         x = [zero, zero, zero, zero]
     end select
   end function gauge_sg
Some really tiny ones that hopefully get inlined by the compiler
\langle OVM\ Procedure\ Implementations \rangle + \equiv
  elemental function sgn_coupl_cmplx(vm, j) result (s)
     class(vm_t), intent(in) :: vm
     integer, intent(in) :: j
     complex(default) :: s
     s = isign(1, j) * vm%coupl_cmplx(abs(j))
   end function sgn_coupl_cmplx
   elemental function sgn_coupl_cmplx2(vm, j, i) result (s)
     class(vm_t), intent(in) :: vm
     integer, intent(in) :: j, i
     complex(default) :: s
     if (i == 1) then
         s = isign(1, j) * vm%coupl_cmplx2(i, abs(j))
       else
         s = isign(1, j) * vm%coupl_cmplx2(i, abs(j))
     end if
   end function sgn_coupl_cmplx2
   elemental function int_to_log(i) result(yorn)
     integer, intent(in) :: i
     logical :: yorn
     if (i \neq 0) then
        yorn = .true.
     else
        yorn = .false.
     end if
   end function
   elemental function color_factor(num, den, pwr) result (cf)
```

```
integer, intent(in) :: num, den, pwr
real(kind=default) :: cf
if (pwr == 0) then
   cf = (one * num) / den
else
   cf = (one * num) / den * (N_**pwr)
end if
end function color_factor
```

Y.33.6 O'Mega Interface

We want to keep the interface close to the native Fortran code but of course one has to hand over the vm additionally

```
\langle VM: TBP \rangle + \equiv
    procedure :: number_particles_in => vm_number_particles_in
    procedure :: number_particles_out => vm_number_particles_out
    procedure :: number_color_indices => vm_number_color_indices
    procedure :: reset_helicity_selection => vm_reset_helicity_selection
    procedure :: new_event => vm_new_event
    procedure :: color_sum => vm_color_sum
    procedure :: spin_states => vm_spin_states
    procedure :: number_spin_states => vm_number_spin_states
    procedure :: number_color_flows => vm_number_color_flows
    procedure :: flavor_states => vm_flavor_states
    procedure :: number_flavor_states => vm_number_flavor_states
    procedure :: color_flows => vm_color_flows
    procedure :: color_factors => vm_color_factors
    procedure :: number_color_factors => vm_number_color_factors
    procedure :: is_allowed => vm_is_allowed
    procedure :: get_amplitude => vm_get_amplitude
\langle OVM \ Procedure \ Implementations \rangle + \equiv
  elemental function vm_number_particles_in (vm) result (n)
    class(vm_t), intent(in) :: vm
    integer :: n
    n = vm%N_prt_in
  end function vm_number_particles_in
  elemental function vm_number_particles_out (vm) result (n)
    class(vm_t), intent(in) :: vm
    integer :: n
    n = vm%N_prt_out
  end function vm_number_particles_out
  elemental function vm_number_spin_states (vm) result (n)
    class(vm_t), intent(in) :: vm
    integer :: n
    n = vm\%N_helicities
  end function vm_number_spin_states
  pure subroutine vm_spin_states (vm, a)
    class(vm_t), intent(in) :: vm
    integer, dimension(:,:), intent(out) :: a
    a = vm%table_spin
  end subroutine vm_spin_states
  elemental function vm_number_flavor_states (vm) result (n)
    class(vm_t), intent(in) :: vm
    integer :: n
    n = vm%N_flavors
  end function vm_number_flavor_states
 pure subroutine vm_flavor_states (vm, a)
    class(vm_t), intent(in) :: vm
    integer, dimension(:,:), intent(out) :: a
```

```
a = vm%table_flavor
end subroutine vm_flavor_states
elemental function vm_number_color_indices (vm) result (n)
  class(vm_t), intent(in) :: vm
  integer :: n
  n = vm%N_col_indices
end function vm_number_color_indices
elemental function vm_number_color_flows (vm) result (n)
  class(vm_t), intent(in) :: vm
  integer :: n
  n = vm%N_col_flows
end function vm_number_color_flows
pure subroutine vm_color_flows (vm, a, g)
  class(vm_t), intent(in) :: vm
  integer, dimension(:,:,:), intent(out) :: a
  logical, dimension(:,:), intent(out) :: g
  a = vm%table_color_flows
  g = vm%table_ghost_flags
end subroutine vm_color_flows
elemental function vm_number_color_factors (vm) result (n)
  class(vm_t), intent(in) :: vm
  integer :: n
  n = vm%N_col_factors
end function vm_number_color_factors
pure subroutine vm_color_factors (vm, cf)
  class(vm_t), intent(in) :: vm
  type(OCF), dimension(:), intent(out) :: cf
  cf = vm%table_color_factors
end subroutine vm_color_factors
! pure & ! pure unless OpenMp
function vm_color_sum (vm, flv, hel) result (amp2)
  class(vm_t), intent(in) :: vm
  integer, intent(in) :: flv, hel
  real(default) :: amp2
  amp2 = ovm_color_sum (flv, hel, vm%table_amplitudes, vm%table_color_factors)
end function vm_color_sum
subroutine vm_new_event (vm, p)
  class(vm_t), intent(inout) :: vm
  real(default), dimension(0:3,*), intent(in) :: p
  logical :: mask_dirty
  integer :: hel
  call vm%run (p)
  if ((vm/hel_threshold .gt. 0) .and. (vm/hel_count .le. vm/hel_cutoff)) then
     call omega_update_helicity_selection (vm%hel_count, vm%table_amplitudes, &
       vm%hel_max_abs, vm%hel_sum_abs, vm%hel_is_allowed, vm%hel_threshold, &
       vm%hel_cutoff, mask_dirty)
     if (mask_dirty) then
        vm%hel_finite = 0
        do hel = 1, vm%N_helicities
           if (vm%hel_is_allowed(hel)) then
              vm%hel_finite = vm%hel_finite + 1
              vm%hel_map(vm%hel_finite) = hel
           end if
        end do
    end if
  end if
end subroutine vm_new_event
pure subroutine vm_reset_helicity_selection (vm, threshold, cutoff)
```

```
class(vm_t), intent(inout) :: vm
   real(kind=default), intent(in) :: threshold
   integer, intent(in) :: cutoff
   integer :: i
   vm%hel_is_allowed = .True.
   vm\%hel_max_abs = 0
   vm\%hel_sum_abs = 0
   vm%hel_count = 0
   vm%hel_threshold = threshold
   vm%hel_cutoff = cutoff
   vm%hel_map = (/(i, i = 1, vm%N_helicities)/)
   vm%hel_finite = vm%N_helicities
 end subroutine vm_reset_helicity_selection
 pure function vm_is_allowed (vm, flv, hel, col) result (yorn)
   class(vm_t), intent(in) :: vm
   logical :: yorn
   integer, intent(in) :: flv, hel, col
   yorn = vm%table_flv_col_is_allowed(flv,col) .and. vm%hel_is_allowed(hel)
 end function vm_is_allowed
 pure function vm_get_amplitude (vm, flv, hel, col) result (amp_result)
   class(vm_t), intent(in) :: vm
   complex(kind=default) :: amp_result
   integer, intent(in) :: flv, hel, col
   amp_result = vm%table_amplitudes(flv, col, hel)
 end function vm_get_amplitude
\langle Copyleft \rangle \equiv
 ! omegalib.nw --
    Copyright (C) 1999-2022 by
        Wolfgang Kilian <kilian@physik.uni-siegen.de>
        Thorsten Ohl <ohl@physik.uni-wuerzburg.de>
        Juergen Reuter <juergen.reuter@desy.de>
        with contributions from
        Fabian Bach <fabian.bach@t-online.de>
        Bijan Chokoufe Nejad <br/> <br/>bijan.chokoufe@desy.de>
        Christian Speckner <cnspeckn@googlemail.com>
    WHIZARD is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2, or (at your option)
    any later version.
   WHIZARD is distributed in the hope that it will be useful, but
    WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
    GNU General Public License for more details.
   You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

—Z— Index

This index has been generated automatically and might not be 100%ly accurate. In particular, hyperlinks have been observed to be off by one page.

```
22, 23, 17, used: 675, 675, 676, 677, 680, 680,
    704, 694, 696, 23, 24, 25, 25, 26, 26, 26,
    27, 27, 38, 45, 50, 291, ??, 342, ??, ??,
    ??, ??, ??, ??, ??, ??, 301
22, 23, 17, used: ??, ??, ??, ??, 645, 645,
    641, 626, 627, 630, 675, 676, 676, 678,
    678, 678, 694, 695, 696, 696, 696, 697, 23,
    24, 24, 25, 25, 25, 26, 26, 26, 27, 27, 38,
    46, 46, 49, 50, 51, 53, 249, 249, 590, 291,
    ??, ??, ??, ??, ??, ??, ??, ??, ??, 335,
    342, 279, ??, ??, ??, ??, ??, ??, ??, 598,
    603, 604, ??, ??, ??, ??, ??, ??, ??, ??,
    ??, ??, ??, ??, ??, ??, 301
```

- /, **22**, **23**, **17**, used: 675, 675, 676, 676, 696, 18, 20, 20, 21, 23, 25, 26, 342, 60, ??, ??, ??, ??, ??

- <<, **596**, used: 599, 599
- <=, **22**, **23**, **17**, used: ??, 445, 642, 642, 642, 625, 625, 625, 626, 626, 626, 626, 676, 677, 690, 690, 704, 707, 694, 694, 695, 696, 696, 23, 23, 44, 45, 50, 51, 584, 289, 290, 315, 338, 340, 341, 350, 60, ??, ??, ??, 598, ??, ??, ??, ??, ??, ??, ??, ??, ??
- <>, 22, 23, 17, used: ??, 649, 640, 641, 641, 628, 628, 628, 630, 630, 632, 660, 662, 732, 733, 678, 678, 678, 684, 686, 687, 704, 707, 708, 696, 8, 10, 11, 13, 23, 34, 45, 50, 51, 52, 53, ??, ??, 589, 591, 312, 315, 315, 352, 354, 280, 280, 280,

```
282, ??, ??, ??, ??
22, 23, 17, used: ??, ??, ??, ??, ??, ??, ??,
    646, 647, 647, 647, 647, 647, 648, 648,
    649, 650, 650, 650, 651, 651, 651, 651,
    652, 652, 446, 641, 642, 642, 626, 628,
     628, 628, 629, 629, 629, 630, 631, 631,
    633, 636, 637, 637, 638, 638, 638, 638,
    638, 638, 639, 639, 622, 659, 661, 676,
    676, 676, 677, 678, 679, 679, 680, 682,
    683, 684, 684, 685, 685, 686, 686, 702,
    702, 703, 703, 704, 708, 708, 696, 9, 10,
    12, 23, 24, 24, 25, 25, 26, 26, 26, 45, 45,
    45, 46, 46, 46, 46, 47, 47, 47, 47, 47,
    47, 47, 47, 48, 48, 48, 48, 50, 50, 50, 51,
    51, 51, 51, 52, 52, 52, 52, 52, 52, 52, 52,
    52, 53, 53, 585, 249, 249, ??, ??, ??, ??,
    217, 217, 218, 218, 290, 308, 308, 315,
    319, 319, 323, 324, 336, 338, 340, 342,
    342, 342, 346, 350, 352, 352, 355, 64, 64,
    65, 66, 281, ??, ??, ??, ??, ??, ??, ??, ??,
```

=>, **304**, used: 304, 305, 305, 305, 305, 305, 305, 306, 306

??, ??, ??, ??, ??, 609

=>!, **305**, used: 305, 305

=>!!!, **304**, used:

=>>, **306**, used: **306**

>, **22**, **23**, **17**, used: ??, 645, 646, 445, 445, 640, 641, 641, 626, 626, 636, 636, 616, 616, 732, 675, 675, 676, 677, 677, 679, 680, 681, 690, 690, 690, 694, 14, 18, 23, 25, 44, 45, 45, 48, 48, 49, 50, 52, 52, 52, 52, 53, 53, 293, 321, 322, 322, 323, 341, 342, 342, 342, 342, 352, 352, 353, 355, 279, 279, 279, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, ??, 607, 608, 608, 608, 610, 610, 610

>:, ??, ??, used:

```
>:::, ??, ??, used: 737, 737, 737, 737, 447, 643,
                                                  abs, 44, 45, 50, 715, 42, used: 587, ??, ??, ??,
                                                          643, 643, 631, 632, 632, 633, 633, 634,
                                                          634, 638, 638, 639, 639, 639, 681, 683,
       683, 687, 688, 688, 688, 689, 689, 697,
                                                         697, 697, 697, 697, 218, 219, 219, 305,
                                                         305, 305, 305, 305, 305, 305, 305, 306,
       306, 306, 313, 313, 314, 328
                                                         22, 23, 17, used: 645, 645, 648, 649, 445,
                                                         ??, ??
       640, 641, 630, 637, 733, 675, 690, 694,
                                                  Abs, 337, 203, used: 337, 339
       695, 697, 9, 9, 10, 10, 12, 14, 23, 24, 46,
                                                  Acos, 337, 203, used: 337, 339
       47, 51, 51, 289, 293, ??, ??, ??, 598, ??,
                                                  Acosh, 337, used: 337
       Acsc, 337, used: 337
       actions (field), 615, used: 615, 615, ??, ??, ??,
       ??, ??, ??
                                                  ac\_lhs, 217, used: 217, 217
>>, 596, used:
                                                  ac\_lhs\_all, 217, used: 218
? >, 304, used: 305, 305, 305
                                                  ac\_rhs, 217, used: 217, 217
?allow_denominator (label), 365, used:
                                                  ac\_rhs\_all, 217, used: 218
?arg\_specs (label), ??, ??, used:
                                                  AD, ??, used:
?cmp (label), ??, 628, 630, 630, 636, 680, 681,
                                                  add, 645, 648, 650, 667, 668, 668, 669, 658,
       681, ??, 624, 625, 625, 635, 674, 675,
                                                          658, 658, 660, 661, 25, 35, 36, 39, 44,
       used:
                                                          46, 50, 249, 249, 249, 249, 249, 249,
?env (label), ??, ??, used:
                                                          214, 216, 291, 330, 346, 644, 667, 667,
?epsilon (label), ??, ??, used:
                                                          656, 656, 657, 715, 716, 717, 718, 718,
?exit\_code (label), ??, ??, used:
                                                          42, 248, 214, 285, 329, used: ??, ??,
                                                          736, 650, 636, 668, 668, 669, 671, 671,
?f (label), 23, 26, 17, used:
?fortran_module (label), 378, used:
                                                         672, 658, 659, 661, 661, 615, 684, 708,
?majorana (label), 378, used:
                                                         708, 24, 24, 25, 25, 36, 36, 36, 37, 37, 37,
?mode (label), ??, ??, used: ??
?msg (label), ??, ??, ??, ??, ??, ??, ??, ??,
                                                         37, 39, 249, 249, 249, 249, ??, ??, ??, ??,
                                                         ??, ??, ??, 590, 590, 590, 590, 592, 216,
                                                         216, 217, 291, ??, 308, 308, 309, 321, 321,
       used: ??, ??, ??, ??
?name (label), 378, used:
                                                         322, ??, 336, 336, 343, 343, 345, 346, 354,
?offset (label), 627, 696, 623, 693, used:
                                                         355, 60, 281, 282, 282, ??, ??, ??, ??, ??,
?only (label), 378, used:
                                                         ??, 603, 301, 332
?parameter_module (label), 378, used:
?pp\_diff (label), ??, ??, used:
                                                  add', 45, used: 45, 46
?predicate (label), 627, 623, used:
                                                  addto_ketset, ??, used: ??
?prefix (label), ??, ??, used:
                                                  addto\_ketset\_map, ??, used: ??
?printer (label), ??, ??, used:
                                                  add\_complex\_array\_tag, ??, used: ??
                                                  add\_degree, 24, used: 24
?ratio (label), 697, used:
?set\_verbose (label), ??, ??, used:
                                                  add\_factor, 355, used: 355
?skip (label), ??, ??, used:
                                                  add_{-}flavors, 590, used: 590
                                                  add_fusion2, ??, used: ??
?stride (label), 626, 627, 623, 623, used:
?suffix (label), ??, ??, used:
                                                  add_fusion3, ??, used: ??
                                                  add_fusionn, ??, used: ??
?truncate (label), 7, 9, 9, 10, 10, 12, 13, 14, 14,
       14, 6, used: 13, 14, 14
                                                  add\_kind, 309, used: 310
                                                  add_node, 33, 37, 30, used: 37, 38, ??, ??
?use_stderr (label), ??, ??, used:
?verbose (label), ??, ??, ??, ??, used:
                                                  add_non_empty, 659, 661, used: 659, 661
?width (label), 444, 445, 446, 446, 446, 444,
                                                  add_offspring, 33, 37, ??, 30, used: ??, ??, ??
       444, used:
                                                  add_offspring_unsafe, 33, 37, 31, used: 37, 38
                                                  add_permute3, ??, used: ??
@?, ??, ??, used:
                                                  add_permute4, ??, used: ??
A, 316, 318, 322, 323, 357, ??, ??, 180, used:
                                                  add_permuten, ??, used: ??
       318, 357, 358, 358, ??, ??, ??, ??, ??, ??,
       add\_stem, 309, used: 310
       add\_stem\_kind, 308, used: 309
       ??
                                                  add\_symbol\_kind, 308, used: 309
                                                  add\_to\_list, ??, used: ??
A (module), 615, ??, used: 615, 615, 615, ??, ??,
       add\_tree, 590, used: 590
       add\_unique, 708, used: 708
       add\_vertex3, ??, used: ??
       ??, ??, ??, ??, ??, ??, ??, ??
                                                  add\_vertex4, ??, used: ??
```

```
add_vertexn, ??, used: ??
                                                      amplitudes_to_channel, 246, used: 603, 603
add_vertices, ??, used: ??
                                                      amplitudes_to_feynmf, 600, used: 605
adjoint, 214, 216, 214, used: 218
                                                      amplitudes_to_feynmf_color_only, 600, used:
AH, ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??,
        amplitudes_to_feynmf_sans_color, 600, used:
        ??, ??, ??, ??, ??
Al, ??, used:
                                                      amplitude_sans_color, ??, ??, 123, ??, used: 604
AL, ??, used:
                                                      amplitude_sans_color (type), ??, ??, 123, ??,
                                                              used: ??, 123, 124, ??, ??
Algebra (module), 719, 715, used: 249, 214, 215,
        337, 343, 344, 344, 351, 65, ??, ??, ??,
                                                      amplitude_to_dot, ??, ??, 124, ??, used: 604
        analyze_tree, 592, used:
        ??, ??, ??, ??, ??, ??, 184, 248, 248,
                                                      And, 55, 61, 55, used: 56, 61
                                                      AND, ??, ??, used: ??, 57
        213, 333, 333, 365, 422, 378
Alias, 292, 293, 285, 286, used: ??, 300, 300
                                                      AND (camlyacc token), 57, used: 57, 58
ALIAS, ??, ??, used: ??, 297
                                                      and\_cascades\_wf, 61, used: 61
                                                      angle (type), ??, used: ??
ALIAS (camlyacc token), 298, used: 300, 300
Alias\_Anti, 292, 285, used:
                                                      anomalous\_bbA, ??, ??, ??, ??, used: ??, ??,
alist_of_list, 627, 623, used: 633
                                                               ??, ??, ??
                                                      anomalous\_bbWW, ??, ??, ??, ??, ??, used: ??,
all, 684, 685, 686, 683, used:
All, 246, used: 601
                                                               ??, ??, ??, ??
                                                      anomalous\_bbZ, ??, ??, ??, ??, used: ??, ??,
allowed, 278, ??, ??, 124, ??, used: 280, 280, ??
                                                               ??, ??, ??
allows_fusion, 587, used: 587
allows_momentum_fusion, 587, used: 587
                                                      anomalous_couplings, 327, used: 328
all_bosons, ??, used: ??
                                                      anomalous_couplings_broken, 328, used: 328
all_compatible, 60, used: 64, 64
                                                      anomalous\_couplings\_index\_mismatch, 327,
all_feynman, ??, used: ??, ??
                                                               used: 328
all_leptons, ??, used: ??
                                                      anomalous\_couplings\_ok, 328, used: 328
all_quarks, ??, used: ??
                                                      anomalous_dim6_gauge_higgs, ??, used: ??
Alpha_QED, ??, ??, ??, ??, ??, ??, ??, ??,
                                                      anomalous_dim6_gauge_higgs4, ??, used: ??
        ??, ??, ??, ??, ??, ??, ??, ??, used:
                                                      anomalous_dim6_higgs, ??, used: ??
                                                      anomalous_dim6_higgs4, ??, used: ??
        ??, ??, ??, ??, ??, ??
Alpha_WWWW0, ??, ??, ??, ??, ??, used: ??,
                                                      anomalous_dim6_quartic_gauge, ??, used: ??
        ??, ??, ??, ??
                                                      anomalous_dim6_triple_gauge, ??, used: ??
                                                      anomalous_gauge_higgs, ??, ??, ??, ??, ??, ??,
Alpha_WWWW2, ??, ??, ??, ??, ??, used: ??,
        ??, ??, ??, ??
                                                              used: ??, ??, ??
Alpha_{-}ZZWW0, ??, ??, ??, ??, used: ??, ??,
                                                                               ??, ??, ??, ??, ??, ??,
                                                      anomalous_gauge_higgs4,
                                                               used: ??, ??, ??
        ??, ??, ??
Alpha_ZZWW1, ??, ??, ??, ??, used: ??, ??,
                                                      anomalous\_higgs, ??, ??, ??, ??, ??, used:
                                                               ??, ??, ??
        ??, ??, ??
Alpha_{-}ZZZZ, ??, ??, ??, ??, used: ??, ??,
                                                      anomalous\_higgs4,
                                                                         ??, ??, ??, ??, ??, used:
        ??, ??, ??
                                                               ??, ??, ??
AltH (module), ??, ??, used: ??
                                                      anomalous_quartic_gauge, ??, ??, ??, ??, ??,
                                                               used: ??, ??, ??, ??, ??
alt_fuse2, ??, used:
alt_fuse3, ??, used:
                                                      anomalous\_tbW, ??, ??, ??, ??, used: ??,
                                                               ??, ??, ??, ??
amplitude, ??, 213, 213, used: ??, ??, ??
amplitude (type), 589, 590, ??, ??, ??, ??, ??,
                                                      anomalous_tbWA, ??, ??, ??, ??, ??, used: ??,
        ??, 589, 123, 125, ??, ??, used: 589,
                                                               ??, ??, ??, ??
        590, ??, ??, ??, ??, ??, ??, ??, 589,
                                                      anomalous\_tbWZ, ??, ??, ??, ??, ??, used: ??,
        589, 123, 123, 123, 124, 124, 124, 124, 124,
                                                               ??, ??, ??, ??
                                                      anomalous\_top\_DuDd, ??, ??, ??, ??, ??, used:
        124, 125, 125, 125, 125, 126, ??, ??, ??,
        ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                               ??, ??, ??, ??, ??
                                                      anomalous\_top\_qBuB, ??, ??, ??, ??, ??, used:
Amplitude (module), ??, used: ??, ??
                                                               ??, ??, ??, ??, ??
Amplitude (sig), ??, used: ??
amplitudes, ??, ??, ??, ??, 123, 125, ??, ??,
                                                      anomalous\_top\_qBuB\_ff, ??, ??, ??, ??, ??,
        used: ??, 603, 606
                                                              used: ??, ??, ??, ??, ??
amplitudes (type), ??, ??, 246, 125, ??, used:
                                                      anomalous\_top\_qBuB\_tt, ??, ??, ??, ??, ??,
                                                               used: ??, ??, ??, ??, ??
        ??, 246, 247, 125, 125, 125, 125, 125, 125,
        125, 125, 125, 125, 125, 125, ??, ??, ??,
                                                      anomalous\_top\_qGuG, ??, ??, ??, ??, ??, used:
                                                               ??, ??, ??, ??, ??
        ??, ??, ??, ??, ??, ??, ??, ??
                                                      amplitudes' (type), ??, used:
amplitudes_by_flavor, 597, used: 600, 600, 600
```

```
anomalous\_top\_qGuG\_tt, ??, ??, ??, ??, ??,
                                                        ANTI (camlyacc token), 298, used: 300, 300
        used: ??, ??, ??, ??, ??
                                                        anticommute, 218, used: 219
anomalous_top_quqd1, ??, ??, used: ??, ??
                                                        AntiFermion, ??, ??, used: ??, ??, ??, ??
anomalous_top_quqd1_ff, ??, ??, used: ??, ??
                                                        AntiMuon, ??, used: ??, ??, ??, ??
anomalous_top_quqd1_tq, ??, ??, used: ??, ??
                                                        AntiTau, ??, used: ??, ??, ??, ??
anomalous_top_quqd8, ??, ??, used: ??, ??
                                                        any (type), 278, 278, 277, used: 278, 278, 277
anomalous_top_quqd8_ff, ??, ??, used: ??, ??
                                                        Any, 278, 278, 277, used: 279, 279
anomalous_top_quqd8_tq, ??, ??, used: ??, ??
                                                        Any_flavor, 55, 61, 55, used: 56, 61
anomalous\_top\_qW, ??, ??, ??, ??, ??, used:
                                                        Application, 329, 338, 329, used: 330, 336, 338,
        ??, ??, ??, ??, ??
anomalous\_top\_qW\_ff, ??, ??, ??, ??, ??, used: ??, ??, ??, ??, ??
                                                        apply, 735, 736, 292, 330, 735, 285, 329, used:
                                                                 736, 736, 736, 737, ??, 324, ??, 301, 332
anomalous\_top\_qW\_tq, ??, ??, ??, ??, ??, used:
                                                        apply1, 339, used: 339
        ??, ??, ??, ??, ??
                                                        apply_invalid_lengths, 688, used: 688
anomalous_triple_gauge, ??, ??, ??, ??, ??,
                                                        apply\_mismatch, 737, used: 737
        used: ??, ??, ??, ??, ??
                                                        apply_ok, 736, 688, used: 737, 688
anomalous\_ttA, ??, ??, ??, ??, used: ??, ??,
                                                        apply\_ok2, 736, used: 737
                                                        apply_shadowed, 736, used: 737
        ??, ??, ??
anomalous\_ttG, ??, ??, ??, ??, used: ??, ??,
                                                        apply_shadowed2, 737, used: 737
        ??, ??, ??
                                                        apply_with_fallback, 735, 736, 735, used: 737
                                                        apply\_with\_fallback\_ok, 737, used: 737
anomalous\_ttGG, ??, ??, ??, ??, ??, used: ??,
        ??, ??, ??, ??
                                                        arg (camlyacc non-terminal), 301, used: 300,
anomalous_ttH, ??, ??, ??, used: ??, ??, ??
                                                                300.301
anomalous\_ttWW, ??, ??, ??, ??, ??, used: ??,
                                                        args (camlyacc non-terminal), 332, used: 332,
        ??, ??, ??, ??
                                                                332
anomalous\_ttZ, ??, ??, ??, ??, used: ??, ??,
                                                        arguments, 593, used: 593, 593
        ??, ??, ??
                                                        Arguments, 246, used: 601
anomaly_higgs, ??, ??, ??, ??, ??, ??, ??,
                                                        arg_list (camlyacc non-terminal), 301, used: 301,
        used: ??, ??, ??, ??, ??, ??, ??
Anomtop (module), ??, used: ??
                                                        ariadne_sort, 630, 625, used: 685, 283
Anom_Dim6_AAWW_DW, ??, used: ??
                                                        ariadne\_unsort, 630, 625, used:
Anom\_Dim6\_AAWW\_W, ??, used: ??
                                                        arity, 7, 8, 10, 11, 13, 5, used:
Anom\_Dim6\_AHHZ\_D, ??, used: ??
                                                        arity (field), 315, used: 315, 324
Anom\_Dim6\_AHHZ\_DP, ??, used: ??
                                                        array, 684, 685, 686, 683, used:
Anom\_Dim6\_AHHZ\_PB, ??, used: ??
                                                        array_list, ??, used: ??
Anom\_Dim6\_AHWW\_DPB, ??, used: ??
                                                        array_of_tableau, 695, used: 695, 696
Anom_Dim6_AHWW_DPW, ??, used: ??
Anom_Dim6_AHWW_DW, ??, used: ??
                                                        ascii_a, 290, used: 290
                                                        ascii_A, 290, used: 290
                                                        ascii_z, 290, used: 290
Anom\_Dim6\_AZWW\_DWDPW, ??, used: ??
Anom_Dim6_AZWW_W, ??, used: ??
                                                        ascii_Z, 290, used: 290
Anom_Dim6_H_4_P_2, ??, used: ??
                                                        Asec, 337, used: 337
Anom\_Dim6\_H4\_v2, ??, used: ??
                                                        Asin, 337, 203, used: 337, 339
Anom_Dim6_HHAA, ??, used: ??
                                                        Asinh, 337, used: 337
Anom_Dim6_HHWW_DPW, ??, used: ??
Anom_Dim6_HHWW_DW, ??, used: ??
                                                        assert_bool, ??, ??, used: ??, 446, 682, 683, 218,
                                                                 218, 218, 219, 219, 219, 327, 327, 328, 328
Anom\_Dim6\_HHZZ\_D, ??, used: ??
                                                        assert_command, ??, ??, used:
Anom\_Dim6\_HHZZ\_DP, ??, used: ??
                                                        assert_equal, ??, ??, ??, ??, ??, ??, used: ??,
Anom\_Dim6\_HHZZ\_PB, ??, used: ??
                                                                 ??, ??, ??, ??, 736, 736, 736, 737, 737,
Anom_Dim6_HHZZ_T, ??, used: ??
                                                                 737, 642, 642, 643, 643, 643, 643, 643,
Anom\_Dim6\_HWWZ\_DDPW, ??, used: ??
                                                                 643, 643, 643, 643, 643, 643, 631, 631,
Anom\_Dim6\_HWWZ\_DPB, ??, used: ??
                                                                 631, 632, 632, 632, 633, 633, 633, 634,
Anom_Dim6_HWWZ_DPW, ??, used: ??
Anom_Dim6_HWWZ_DW, ??, used: ??
                                                                 637, 637, 638, 638, 638, 638, 638, 638,
                                                                 638, 638, 638, 639, 639, 639, 639, 681,
Anom\_Dim6\_WWWW\_DWDPW, ??, used: ??
                                                                 681, 681, 682, 682, 687, 688, 688, 688,
Anom_Dim6_WWWW_W, ??, used: ??
                                                                 688, 688, 697, 697, 697, 304, 305, 306, 314
Anom_Dim6_WWZZ_DWDPW, ??, used: ??
                                                        assert\_equal\_int\_list, 633, used: 633
Anom_Dim6_WWZZ_W, ??, used: ??
                                                        assert_equal_perms, 681, used: 681, 681
anom_ferm_ass, ??, ??, ??, ??, ??, used: ??, ??,
                                                        assert_failure, ??, ??, used: ??, ??, ??, ??, ??
        ??, ??, ??, ??, ??, ??, ??
                                                        assert_power, 632, used: 632, 632, 632, 632, 632
```

Anti, 307, 307, used: 310 ANTI, ??, ??, used: ??, 297

```
assert_raises, ??, ??, used: 737, 632, 633, 634,
                                                   Aux_-Vector, 178, used: ??, ??, ??, ??, ??
       638, 639, 682, 687, 687, 688, 304, 313,
                                                   Aux_-Vector_-DScalar, 180, used:
       313, 314, 327
                                                   Aux_-Vector_-Vector, 180, used:
assert_string, ??, ??, used:
                                                   axial, 316, 316, 320, 320, 321, 322, used: 322
Atan, 337, 203, used: 337, 339
                                                   axial\_map, 322, used: 322
Atan2, 203, used:
                                                   A_{-}\theta, ??, used:
Atanh, 337, used: 337
                                                   a\_name (field), 359, used: ??, 362
atom, 717, 717, 718, 718, used:
                                                   A_P, ??, used:
atom (field), 364, used:
                                                   a\_value (field), 359, used: ??, 362
                                                   B, ??, used: ??, ??, ??, ??
atom (type), 343, 344, 333, used: 343, 344, 344,
                                                   B1, ??, used: ??, ??, ??
       345, 333, 333, 335, 335
       203, used: 339, ??, ??, ??, ??, ??, ??, ??,
                                                   B2, ??, used: ??, ??, ??
Atom,
       bag (type), 278, used: 278
       ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                   bal, 645, used: 646, 646, 647, 647, 648
Atom (sig), 343, 334, used: 344, 350, 356
                                                   Bar, 288, 284, used: 288
atoms, 717, 718, 719, used:
                                                   bare_scripted_token (camlyacc non-terminal),
Atpsi, ??, ??, used:
                                                           302, used: 301
attach_to_fst_sorted, 677, used: 677
                                                   bare_token (camlyacc non-terminal), 302, used:
attach_to_fst_unsorted, 677, used: 677, 678
                                                           301
                                                           ??, ??, ??, ??, ??, ??, ??, ??, ??,
attr (field), 292, 293, 295, 295, 285, 286, 287,
                                                   baryon,
                                                           287, used: 293, 293, 295, 295, ??, 309,
                                                           used: ??, ??, ??, ??, ??, ??, ??, ??,
       309, 299, 300, 300, 300
attr (type), 292, 293, 294, 295, 285, 286, 287,
                                                           ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
       287, used: 292, 293, 295, 295, 285, 286,
                                                         667, 668, 668, 669, 667, 667, used: 669,
                                                   base,
       287, 287
                                                           282
                                                   base (type), 667, 667, 668, 668, 669, 280, 282,
attrib (type), 359, used: 359
attribs (field), 359, used: ??, 362
                                                           ??, 666, 667, 667, used: 667, 667, 668,
attribute (camlyacc non-terminal), 362, used: 362
                                                           668, 668, 668, 669, 666, 666, 667, 667,
attributes (camlyacc non-terminal), 362, used:
                                                           667, 667, 667, 667, 667
       362, 362
                                                         671, 672, 670, used: 672, 66, 282
                                                   basis,
                                                   Bbar, ??, used: ??, ??, ??, ??
attr_to_string, 292, 293, 295, 295, used: 293,
       293, 295, 295
                                                   BBB, 180, used: ??, ??, ??, ??, ??, ??, ??
AU, ??, used:
                                                   BBWW, ??, ??, ??, ??, used: ??, ??, ??, ??,
augment_basis, 672, used: 672
                                                           ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                   bcdi_of_flavor, ??, used: ??, ??
augment_basis_overlapping, 672, used: 672
auto, 735, 736, 735, used: 737
                                                   Be, ??, used:
auto_ok, 737, used: 737
                                                   begin_step, 617, 616, used: ??
Aux\_Col\_Scalar, 178, used:
                                                   below_to_channel, ??, used: ??
Aux\_Col\_Tensor\_1, 178, used:
                                                   Binary (module), 8, 18, ??, 606, 6, 16, 124, ??,
                                                           596, used: 13, 19, 20, 28, 28, ??, ??, 606,
Aux\_Col\_Vector, 178, used:
Aux\_ConjSpinor, 178, used:
                                                           606, ??, 16, 18
Aux_DScalar_DScalar, 180, used:
                                                   Binary (sig), 8, 6, used: 6
Aux_Gauge_Gauge, 180, used: ??, ??, ??, ??,
                                                   Binary_Majorana (module), ??, 606, 124, ??,
       596, used: 606
        ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                   binomial, 675, 673, used:
                                                   binomial', 675, used: 675, 675
Aux\_Majorana, 178, used:
Aux_Scalar, 178, used: ??, ??, ??, ??, ??
                                                   bits, 49, used: 50, 50, 50, 51, 51, 51, 51, 52, 52,
Aux\_Scalar\_Scalar, 180, used:
Aux\_Scalar\_Vector, 180, used:
                                                   Bits (module), 49, 43, used: 53, 54, 54, 43
Aux\_Spinor, 178, used:
                                                   bits0, 49, used: 49
Aux_Tensor_1, 178, used: ??, ??, ??, ??, ??
                                                   Bits W (module), 53, 43, used: 54, 54
Aux\_top, ??, ??, ??, ??, used: ??, ??, ??,
                                                   blank\_line, 445, used: 445
                                                   boson (type), ??, 180, used: ??, 180
       Boson, ??, ??, ??, used: ??, ??, ??, ??, ??, ??,
       ??, ??, ??, ??, ??, ??, ??
       boson2 (type), 180, used: 182
       Bound (module), 606, used: 606
       ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                   Bound (sig), 15, 7, used: 20, 27, ??, ??, 606, 7,
aux_top_flavors, ??, ??, ??, ??, ??, used: ??,
                                                           16, 18, 124, 124, ??, ??
       ??, ??, ??, ??, ??
                                                   bounds\_check2, 321, used: 321, 322
```

```
bounds\_check3, 322, used: 322, 322
                                                   Cache (module), 620, 620, used:
                                                   Cache_Ignore, ??, used: ??
bounds\_check4, 322, used: 322
bra, ??, ??, ??, ??, 123, ??, used: ??
                                                   cache\_mode (type), ??, used:
                                                   cache_option, ??, used:
bra (field), 364, used:
bracket, ??, ??, used: ??
                                                   Cache_Overwrite, ??, used:
bracket_tmpfile, ??, ??, used: ??, 446
                                                   Cache\_Use, ??, used:
braket (type), ??, ??, ??, 123, ??, used: ??,
                                                   call\_function, 593, used: 593
        ??, ??, ??, ??, ??, 123, 124, ??, ??
                                                   call_subroutine, 593, used: 593
brakets, ??, ??, ??, ??, 124, ??, used: ??, 603
                                                   canonicalize, 645, 650, 652, 702, 644, 699,
brakets (field), ??, ??, used: ??, ??, ??, ??, ??,
                                                          used: 702, 598
       ??, ??, ??, ??, ??, ??
                                                   cascade (camlyacc non-terminal), 58, used: 58
Branch, 584, used: 584, 585, 585, 585
                                                   Cascade (module), 60, 59, used: ??, ??, ??, 597,
bra_to_channel, ??, used: ??
                                                          597, 124, ??
Broken\_Gauge (sig), 212, used:
                                                   cascades (camlyacc non-terminal), 58, used: 58,
BRS, 178, used:
                                                          58
BSM_anom (module), ??, ??, used: ??
                                                   Cascade_lexer (module), ??, ??, 57, used: 61
BSM\_bsm (module), ??, ??, used: ??, ??, ??,
                                                   Cascade_parser (module), ??, ??, 57, used: ??,
       ??, ??, ??, ??, ??, ??
                                                          61, ??, 57
BSM\_flags (sig), ??, ??, used: ??, ??, ??, ??.
                                                   Cascade_syntax (module), 55, 55, used: ??, ??,
       ??, ??, ??, ??, ??, ??, ??
                                                          60, ??, 57, 58
BSM_ungauged (module), ??, ??, used: ??
                                                          ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                                   cave ats,
buff_printf, ??, used: ??, ??
                                                          build_forest, 38, used: 38, 39
                                                          ??, ??, ??, ??, ??, ??, ??, ??, ??,
builtin (type), 337, used: 338
                                                          607, 211, used: ??, ??, ??, ??, ??
                                                   Cbar, ??, used: ??, ??, ??, ??
builtin\_of\_string, 337, used: 340
builtin_to_string, 337, used: 338, 339, 339
                                                   cc, 214, 215, 215, 220, 220, 221, 214, used:
bundle (field), ??, used: ??, ??, ??, ??, ??, ??
                                                          215, 219, 219, 219, 219
Bundle (module), 667, 666, used: 281, 283, ??
                                                   cc\_gamma, 219, used: 219
bundle_to_strings, 281, used:
                                                   cc\_inv, 219, used: 219, 219, 219
by\_color, 280, used: 280, 282, 282, 283, 283
                                                   cc_{-}is_{-}i_{-}gamma_{-}\theta, 215, 220, 220,
c (type), ??, ??, ??, ??, 65, ??, ??, ??, ??.
                                                          221, used: 219
       cc\_times, 364, used:
       cells_tableau, 695, 693, used: 696
       ??, 608, 716, ??, ??, ??, used: ??, ??,
                                                   CF (module), 597, used: 597, 601, 603, 603, 603,
       ??, ??, ??, 717, 717, 717, 717, 717, ??,
                                                          604, 604, 604, 605
                                                   cf_arbitrary, ??, ??, ??, ??, used: ??, ??
       ??, ??
                                                   Ch (module), ??, ??, ??, ??, ??, ??, ??, ??,
C, 316, 320, 322, 323, 348, 349, 357, ??, 334,
       364, used: 350, 352, 357, 358, 358, ??,
                                                          ??, ??, ??
                                                          C \text{ (module)}, ??, ??, 597, 718, 718, 448, used:
                                                          608, 210, used: ??, ??, ??, 278, ??, ??,
       ??, ??, ??, ??, 603, 606, 718, 718,
                                                          ??, 210, ??
       718, 718, 718
                                                   channel, 617, 616, used: ??
C1, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                   channel (field), 616, used: 617, 617
                                                  channel (type), 616, used: 616
Channel, 616, ??, used: 617, ??
       C1c, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                   Chaotic (module), 35, 32, used:
                                                   char (type), ??, ??, ??, used:
        ??, ??, ??
    ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                   char (camllex regexpr), 57, 297, 331, used:
       CHAR, ??, ??, used: ??, 297
       ??, ??
                                                   CHAR (camlyacc token), 298, used: 302, 302,
     ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
C2c,
                                                          ??, ??, ??
                                                   charge,
C3, ??, used: ??, ??, ??
                                                          ??, ??, ??, ??, ??, ??, ??, ??, used:
C3c, ??, used: ??, ??, ??
                                                          C4, ??, used: ??, ??, ??
                                                          ??, ??, ??, ??, ??, ??, ??, ??
C4c, ??, used: ??, ??, ??
                                                   Charge, 292, 285, used: ??, 300
CA (module), ??, used: ??, ??, ??, ??, ??, ??,
                                                   CHARGE, ??, ??, used: ??, 297
       CHARGE (camlyacc token), 298, used: 300, 300
       Charged, 292, 307, 307, 285, used: ??, 308, 310,
       ??
                                                   CHARGED, ??, ??, used: ??, 297
```

```
CHARGED (camlyacc token), 298, used: 299
                                                            36, ??, ??, ??, ??, ??, ??, ??, ??, 16,
charged_chargino_currents, ??, ??, ??, used: ??,
                                                            16, 18, 18, 30, 30, 30, 31, 31, 31, 31, 32,
                                                            32, 33, 33, 126, 126, ??, ??
charged_currents, ??, ??, ??, ??, ??, ??, ??,
                                                    children_to_channel, 712, used: 711
                                                    children_to_string, 711, used: 711
        ??, ??, ??, ??, ??, ??, used: ??, ??,
                                                    Chiral (module), 221, 317, 320, 325, 214, used:
        325, 325, 365
charged\_currents', ??, ??, ??, ??, ??, ??, used:
                                                    Chiral_R (module), 219, used: 220, 221
       ??, ??, ??, ??, ??, ??, ??
                                                    choose, 645, 647, 651, 679, 644, 674, used: 8,
charged_currents", ??, ??, ??, ??, ??, used: ??,
                                                            8, 14, 15, 60, ??, ??
        ??, ??, ??, ??
                                                    choose', 678, used: 678, 679
charged\_currents\_ckm, ??, ??, ??, ??, ??, used:
                                                    choose2, 8, ??, ??, used: 9, 10, ??, ??
        ??, ??, ??, ??, ??
                                                    choose3, 8, used: 10
charged_currents_triv, ??, ??, ??, ??, ??, used: ??, ??, ??, ??, ??
                                                    choose_opt, 645, 647, 651, 644, used:
                                                    chopn, 626, 623, used:
                                                    chopn', 626, used: 626, 626
charged_heavy_currents, ??, used: ??
charged_quark_currents, ??, ??, ??, used: ??,
                                                    chopn'', 626, used: 626, 626
        ??, ??
                                                    CKetSet (module), ??, used: ??, ??, ??
charged_slepton_currents, ??, ??, ??, used: ??,
                                                    CKM_{-}12, ??, used:
       ??, ??
                                                    CKM_{-}13, ??, used:
                                                    CKM_23, ??, used:
charged_squark_currents, ??, ??, ??, used: ??,
        ??. ??
                                                    ckm_present, ??, ??, ??, ??, ??, ??, ??, ??,
charged_squark_currents', ??, ??, ??, used: ??,
                                                            ??, ??
                                                            ??, ??, ??, ??, ??, ??, ??, ??, used:
charged_up_currents, ??, used: ??
charged_Z, ??, ??, used: ??, ??, ??
                                                            charges, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                                            ??, ??, ??, ??, ??, ??, ??, ??
       ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                                    classes\_to\_string, 342, 342, 333, used:
       608, 210, used: ??, ??, ??, 278, ??, ??,
                                                    classify, 629, 26, 624, used: 637, 676, 678, 678,
                                                            14, 15, 26, 312, 319, 319, 324
Charges (module), 249, 248, used: ??, ??, ??,
                                                    classify_dirac_indices1, 354, used: 354
       classify_indices, 343, 343, 348, 355, 358, 334,
       334, 364, used: 348, 348, 355
       ??, ??, 608, 210, ??, ??, ??, ??, ??, ??,
                                                    classify_indices', 348, used: 348
       ??, ??, ??, ??, ??, ??, ??
                                                    classify_indices1, 354, 358, used: 355, 358
                                                    classify_nodes, ??, used: ??
charge\_conjugate, 365, used:
                                                    classify\_vector\_indices1, 354, used: 354
charge_conjugation, 219, used: 219
Chargino, ??, ??, ??, used: ??, ??, ??, ??, ??,
                                                    class_size, 25, used: 25, 25, 26
       class\_size\_3, 25, used: 25
       class\_size\_n, 25, used: 25
                                                    clist, ??, ??, used: ??, ??
       ??, ??, ??, ??, ??, ??, ??, ??, ??
charlist, ??, used: ??
                                                    clone, 629, ??, 624, used: 629, 664, 695, 14, ??,
check, 44, used: 44, 45
check_charges, ??, ??, 124, ??, used: 603
                                                    clone3,
                                                            ??, used: ??
                                                    clone4, ??, used: ??
check\_indices, 348, used: 348
chg, ??, ??, used: ??, ??, ??, ??
                                                    clonen, ??, used: ??
     180, used: ??, ??, ??, ??, ??, ??, ??,
                                                    cloop, ??, ??, used: ??, ??, ??, ??
                                                    cloop_kk, ??, used: ??, ??, ??, ??, ??, ??, ??,
        ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                            ??, ??, ??
                                                    cloop\_kk2, ??, used: ??
        180, used: ??, ??, ??, ??, ??, ??, ??,
Chibar,
        \begin{array}{ll} close\_channel, & \textbf{617}, \text{ used: } \textbf{618} \\ CM \text{ (module)}, & ??, ??, \textbf{597}, \text{ used: } ??, ??, ??, ??, \end{array}
       ??, ??, ??, ??, ??, ??, ??
chiggs (type), ??, used: ??
                                                            ??, ??, ??, ??, ??, ??, ??, ??, 598,
       ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                            599, 599
CHiggs,
       ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                    CMap (module), ??, used: ??, ??
children, ??, ??, ??, ??, 123, ??, used: ??, ??
                                                    cmdline, 615, 615, used: 601
children (type), 18, 19, 21, 22, 22, 27, 28, 33,
                                                    cmp\_float, ??, ??, used:
       33, 34, 34, 35, 36, ??, ??, ??, ??, ??,
                                                    coarsest_partition, 66, 282, used: 67, 282
       ??, ??, ??, ??, 16, 30, 30, 32, 122, 126,
                                                    coarsest_partition', 66, used: 66, 66
                                                    coeff (field), 365, used:
       ??, ??, used: 18, 33, 33, 34, 34, 34, 35,
```

```
Coeff, 349, 349, 335, used: 350, 350
                                                      color_flows, ??, ??, 125, ??, used:
collect23, 22, used: 22
                                                      color_flows (field), ??, used: ??, ??
                                                      color\_rep (type), 322, 323, used:
collect3, 22, used: 22
                                                      color_reps (field), 315, used: 323, 324
collect34, 22, used: 22
                                                      col_currents, ??, ??, used: ??, ??, ??
collect4, 22, used: 22
collect_vertices, ??, used: ??
                                                      col_lqino_currents, ??, used: ??
COLON, ??, ??, ??, used: ??, ??, 57
                                                      col_lq_currents, ??, used: ??
COLON (camlyacc token), 57, 361, used: 57, 58,
                                                      col_sfermion_currents, ??, ??, ??, used: ??, ??,
        363, 363, 364
       Combinatorics (module), 675, 673, used: 685,
color.
                                                               685, 685, 685, 685, 686, 686, 686, 686,
        ??, ??, ??, ??, ??, ??, ??, ??, 608,
                                                               686, 696, 696, 8, 8, 14, 15, 20, 21, 23, 27,
        210, used: ??, ??, ??, 324, 280, ??, 599,
                                                               28, ??, ??, ??, 325, 60, ??, ??, ??, ??, ??
        599, ??, ??
                                                      combine_decays, 587, used: 587, 587
color (field), 311, 324, used: 311, 311, 311, 312,
                                                      combine\_or\_add\_index, 354, used: 355
        312, 324, 324, 326, 326, 326, 326, 327,
                                                      combine_or_add_indices, 355, used: 355
        327, 327, 327
                                                      COMMA, ??, ??, ??, ??, ??, ??, ??, used:
color (label), ??, ??, 211, used:
                                                               ??, ??, ??, ??, 57, 297, 331
       292, 294, 295, 306, 307, 285, 287, 287,
                                                      COMMA (camlyacc token), 57, 298, 331, 361,
        used: ??, 309, 300, 300, 301
                                                               used: 57, 58, 302, 303, 332, 362, 363, 363,
Color (module), 73, 323, 358, 68, 335, used:
                                                              363, 363, 363, 363, 363
                                                      common, 631, 625, used:
        315, 324, 324, 324, 326, 326, 326, 326,
        327, 327, 356, 358, 358, 65, 280, ??, ??.
                                                      common\_stem, 307, 308, used:
        common_stems (field), 308, used: 308, 308, 309,
        ??, ??, ??, ??, ??, ??, ??, 608, 184,
                                                      compare, ??, ??, ??, ??, 736, 645, 650, 651, ??,
        210, 211, 213, 213, 335, 125, 125, ??, ??
                                                               ??, ??, 628, 636, 668, 670, 658, 658,
                                                               660, 615, 7, 8, 10, 11, 13, 22, 23, 24,
Color (sig), 322, used: 323
COLOR, ??, ??, used: ??, 297
                                                               33, 33, 34, 34, 35, 35, 35, 44, 45, 50,
                                                               ??, ??, ??, ??, ?89, 590, 592, 291,
COLOR (camlyacc token), 298, used: 300, 300,
                                                               304, 321, 321, 322, 324, 336, 354, 65,
        300.301
                                                               66, 281, 282, 282, ??, ??, ??, ??, ??,
colored_only, 600, used: 600
Colorize (module), 250, 250, used: ??, ??, ??,
                                                               ??, ??, ??, ??, ??, ??, ??, 597, ??,
        ??, 597, 124, 126, ??, ??
                                                               Colorized (sig), 213, used: 250
                                                               Colorized\_Gauge (sig), 213, used: 250
                                                               ??, ??, ??, ??, 608, ??, ??, 644, 624,
colorize_amplitude, ??, used: ?? colorize_amplitudes, ??, used: ??
                                                               635, 670, 656, 656, 717, 5, 17, 29, 30,
                                                               32, 42, ??, 285, used: ??, ??, ??, ??, ??,
colorize_braket, ??, used: ??
                                                               736, 651, 641, 642, ??, ??, 625, 635, 636,
colorize_braket1, ??, used: ??
                                                               637, 637, 638, 638, 638, 638, 711, 711,
colorize_coupling, ??, used: ??, ??
                                                               615, 675, 678, 678, 681, 684, 684, 684,
colorize_dag, ??, used: ??
                                                               684, 685, 686, 702, 702, 703, 708, 708,
colorize_external, ??, used: ??
colorize_fusion, ??, used: ??
colorize_nodes, ??, used: ??
                                                               708, 694, 13, 23, 24, 34, 35, 37, 38, 39, 45,
                                                               45, 50, 586, 587, ??, ??, ??, ??, 589, 590,
                                                               590, 591, 592, 288, 293, 303, 304, 309,
colorize_sterile_nodes, ??, used: ??
                                                               309, 315, 324, 336, 342, 346, 348, 354,
colorize\_wf, ??, used: ??
                                                               358, 56, 61, 65, 66, 280, 280, 281, 282,
colors, ??, used:
                                                               282, 282, 283, ??, ??, ??, ??, ??, ??, ??,
                                                               ??, ??, ??, ??, ??, ??, ??, ??, 597,
colors (type), ??, used:
                                                               598, 602, ??, ??, ??, ??, ??, ??, ??, ??,
Color_Atom (module), 356, 335, used: 356, 356,
                                                               356, 335
                                                              ??, ??, ??, ??, ??, ??, ??, 608
Color\_Atom (sig), 355, 335, used: 335
Color\_Atom' (module), 356, used: 358
                                                      compare', 649, used: 649, 650
color_current, ??, used: ??
                                                      compare_base, 667, 281, 283, ??, 666, used: 668
color_currents, ??, ??, ??, ??, ??, ??, ??,
                                                      compare_caseless, 642, 640, used: ??
        ??, ??, ??, ??, ??, ??, ??, used: ??,
                                                      compare_elt, 667, 280, 282, ??, 666, used: 668,
        ??, ??, ??, ??
                                                      compare_lists_by_size, 631, used: 632
color_factors, ??, ??, 125, ??, used:
                                                      complement, 631, 625, used: 634
color_factors (field), ??, used: ??, ??, ??
                                                      complement\_index\_sets, 630, used: 630
color\_flow, ??, used: ??, ??
                                                      complete_fusion_tower, ??, used: ??
```

```
Complex, 338, 208, used: 340, ??, ??, ??, ??, ??
                                                conjugate_diagram_new, 694, used:
                                                conjugate_sans_color, 213, 213, used:
Complex (module), 316, 320, used: 316, 316,
       320, 320, 320, 320, 320, 321, 321, 321,
                                                conjugate_tableau, 695, 693, used: 695, 695
       322, 322, 325
                                                conjugate_transpose, 364, used:
Complex_Array, 208, used: ??
                                                ConjWeyl, 294, 286, used:
Complex_Mass, 180, used: ??, ??, ??, ??, ??,
                                                          ??, ??, ??, used: ??, ??, ??, ??, ??,
                                                conj\_char,
                                                       ??, ??, ??
complex\_rational (type), 215, used: 215, 215,
       215, 219, 220, 220
                                                conj_chiggs, ??, used: ??
compose, 645, 648, 652, 684, 685, 686, 644,
                                                conj_iso, ??, used: ??, ??, ??
       683, used: 648, 652, 688, 688
                                                conj_symbol, ??, ??, ??, used: ??, ??, ??, ??
                                                cons, 645, 647, 650, 711, 701, 644, 711, 699,
compose_inv, 684, 685, 686, 683, used:
                                                       used: 647, 648, 650, 38, 38
compose_inverse_ok, 688, used: 689
                                                      717, used:
compose\_ok, 688, used: 689
                                                const,
compress, 636, 340, 346, 635, used: 636, 340,
                                                constant (type), ??, ??, ??, ??, 60, 60, ??, ??,
                                                       346
                                                       compress2, 636, 635, used:
                                                       ??, ??, ??, ??, 607, 210, 59, 122.
compressed (type), 635, 635, used: 635, 635
compressed2 (type), 636, 635, used: 635, 635
                                                       ??, used: ??, ??, ??, ??, ??, ??, 60,
concat, 646, used: ??, ??, 646, 648, 631, 671,
                                                       60, 60, 61, 63, 65, ??, ??, ??, ??, ??, ??,
                                                       ??, ??, ??, 597, ??, ??, ??, ??, ??, ??,
       671, 619, 621, 621, 622, 622, 711, 711,
       685, 687, 703, 704, 14, 15, 45, 50, 587,
                                                       588, 591, 591, 591, 593, 290, 291, 292,
                                                       293, 293, 295, 295, 304, 310, 312, 312,
                                                       ??, 608, 210, 210, 210, 211, 211, ??, ??,
       312, 313, 319, 325, 338, 342, 342, 344,
                                                       ??, ??, 59, 59, 250, 250, 122, 124, 126, ??,
       344, 344, 56, 56, 61, 62, 62, 62, 63, 63, 66,
                                                       ??, ??
       67, 281, 281, 282, ??, ??, ??, 598, 599,
                                                Constant, 180, used: ??, ??, ??, ??, ??, ??, ??,
                                                       599, 600, 601, 602, 603, 603, 604, 611
concat\_tokens, 291, used: 290
                                                       ??, ??, ??
Config (module), 614, used:
                                                Constant (module), ??, 61, ??, used: 61, 61
Conflicting_Space (exn), 307, 309, used:
                                                Constant (sig), ??, ??, used: ??, ??
                                                constant_symbol, ??, ??, ??, ??, ??, ??, ??,
conj, 214, 217, 716, 214, used: 215, 216, 217
Conj, 337, 203, used: 337, 339, 340
                                                       ConjDirac, 294, 286, used:
                                                       ??, 611, 211, used: ??, ??, ??, ??, 63,
conjspinor (type), 316, 317, used: 316, 318
ConjSpinor, 316, 317, 178, used: 317, 326, 355,
                                                       ??, ??
       constant_symbol (label), ??, ??, 211, used:
       constraints, ??, ??, ??, ??, ??, 124, 125, ??,
       ??, ??, ??, 607
                                                       ??, used: ??
conjspinor_contraction_ok, 319, used: 319
                                                constraints (field), ??, ??, ??, used: ??, ??, ??,
                                                       ??, ??, ??, ??
conjspinor_field, 326, used: 326, 326, 326, 326,
       327, 327
                                                cons\_and\_wf, 61, used: 61
conjspinor_indices, 319, used: 319
                                                context, 326, 327, used: 326, 326, 326, 327,
conjspinor_ok, 318, used: 318
conjugate, ??, ??, ??, ??, ??, 65, ??, ??, ??,
                                                       327, 327
                                                context (type), 315, used: 315, 316, 322
       context\_of\_flavors, 324, used: 324
       context\_of\_flavor\_names, 324, used: 324, 326,
       327, 327
       context\_of\_vertex, 324, used: 324, 324
       ??, ??, ??, ??, ??, ??, 608, 608, 210,
                                                Continuation\_Lines (exn), 444, used:
       ??, 364, 122, ??, used: ??, ??, ??, ??,
                                                contract4 (type), 182, used: 182, 183
       ??, ??, ??, ??, 61, 64, 64, 65, 65, 278,
                                                contraction (type), 365, used: 365, 365
       283, 283, ??, ??, ??, ??, ??, ??, ??, ??,
                                                contraction\_ok, 319, 324, used: 320
                                                contract_pair, 343, 345, 350, 356, 334, used:
       ??, 599, 599, 599, 599, ??, ??, ??, ??, ??,
       345, 346
       contract_pairs, 343, 346, 334, used: 346
       ??, ??, ??, 608
                                                contract_pairs1, 346, used: 346, 346
conjugate (label), ??, ??, 211, used:
                                                copy_matrix, 731, 731, used: 733, 734, 734
                                                Cos, 337, ??, 203, used: 337, 339
conjugated (field), 364, used:
conjugate_diagram, 695, 692, used: 696, 696
                                                Cos2al, ??, used:
```

```
Cos2am2b, ??, used:
                                            CRing (sig), 715, used: 715
Cos2be, ??, used:
                                            cross, ??, used: ??
Cos4be, ??, used:
                                            crossing, 278, 283, 278, used:
Cosamb, ??, used:
                                            Crossing_Bundle (module), 283, used:
Cosapb, ??, used:
                                            Crossing_Projection (module), 282, used: 283
Cosh, 337, 203, used: 337, 339
                                            CS (module), 60, used: 62
Cospsi, ??, ??, used:
                                            Csc, 337, used: 337
Costhw, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                            CSet (module), 330, used: 330, 330
      ??, ??, ??, ??, ??, ??, used: ??, ??,
                                            csign (type), ??, used: ??, ??
      CSp, 353, used: 353, 354, 354
      ??, ??, ??, ??
                                            CT (module), 597, used: 604
Cot, 203, used:
                                            current_cline (field), 444, used: 444, 445, 445,
Count (module), 23, 17, used:
Count (sig), 23, 17, used: 17
                                            Custom, 180, used: ??, ??, ??, ??, ??, ??, ??,
count_diagrams, ??, ??, 124, ??, used: 603
                                                   count\_fusions, ??, ??, 124, ??, used: 603
                                                   ??, ??, ??
count_indices, 311, 312, used: 312
                                            CVSp, 353, used: 353, 354, 354
count\_maps, 590, used: 590
                                            CWFBundle (module), ??, used: ??, ??, ??, ??,
count\_non\_zero, 53, used: 54
count\_non\_zero', 53, used: 53, 53
                                            CWFMap (module), ??, used: ??, ??, ??, ??, ??
count_permutations, 681, used: 681
                                            cycle, 626, 623, used: 633
count_processes, ??, used: ??
                                            Cycle (exn), 33, 37, 30, used:
count_propagators, ??, ??, 124, ??, used: 603
                                            cycle', 626, used: 626, 626
count_trees, 33, 38, 32, used: ??
                                            cycle_until, 626, 623, used: 633
coupling, ??, ??, ??, ??, ??, 122, ??, used:
                                            cyclic, 684, 685, 686, 683, used:
                                            C_{-12_{-34}}, 182, used: ??, ??, ??, ??, ??, ??, ??,
      ??, ??, 603
                                                   coupling (field), ??, ??, used: ??, ??, ??
coupling (type), ??, ??, ??, ??, ??, ??, ??,
                                                   ??, ??, ??, ??, ??, ??, 609
      122, 126, ??, ??, used: ??, ??, ??, ??,
                                            C_{-13}_{-42}, 182, used: ??, ??, ??, ??, ??, ??, ??,
      ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                                   ??, ??, ??, ??, ??, ??, 609
      123, 124, 126, ??, ??, ??
                                            C_{-}14_{-}23, 182, used: ??, ??, ??, ??, ??, ??, ??,
coupling (camlyacc non-terminal), 363, used: 363
Coupling (module), 178, used: ??, ??, ??, 315,
                                                   339, 339, 350, 355, 60, 63, 65, ??, ??, ??,
                                                   ??, ??, ??, ??, ??, ??, 609
      ??, ??, ??, ??, 597, ??, ??, ??, ??, ??,
                                            C_{-}quqd1L_{-}bt, ??, ??, used: ??, ??
                                            C_{-quqd1}L_{-tb}, ??, ??, used: ??, ??
      C_{-quqd1}R_{-b}t, ??, ??, used: ??, ?? C_{-quqd1}R_{-t}b, ??, ??, used: ??, ??
      C_{-}quqd8L_{-}bt, ??, ??, used: ??, ??
      ??, ??, ??, ??, 607, 210, 210, 210,
                                            C_{-quqd8L_{-}tb}, ??, ??, used: ??, ??
      210, 210, 210, 211, ??, 332, 335, 365, 365,
                                            C_{-quqd8R_{-}bt}, ??, ??, used: ??, ??
      365, 365, 422, 422, 422, 422, 378, 378, 59,
                                            C_{quqd8R_{tb}}, ??, ??, used: ??, ??
      122, ??
                                            d (field), 44, used: 45, 45, 45, 45, 45, 45, 46, 46,
couplings (field), 61, 378, used: 61, 63, 65
                                                   46, 47, 47, 47, 48, 48, 48, 49, 54
                                            D, 322, 323, 355, 356, ??, ??, ??, ??, ??, ??,
couplings (camlyacc non-terminal), 363, used:
      363, 363
                                                   ??, ??, ??, ??, 335, used: 323, 356, 356,
Coupling_Dictionary, 359, used: ??, 363
coupling_orders (field), 378, used:
                                                   coupling_tag, ??, ??, ??, ??, ??, ??, 122, ??,
                                                   used: ??, ??
                                                   coupling_tag (field), ??, ??, used: ??, ??, ??
                                                   coupling_tag_raw, ??, used: ??
coupling_to_string, ??, ??, ??, ??, 126, ??, used:
                                                   ??
                                                   Cpp (module), 448, used:
                                                   create, 645, 617, 615, 615, used: ??, ??, 645,
      447, 617, 617, 617, ??, ??, ??, ??, ??, ??,
                                                   ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                   created (field), 616, used: 617, 618
create_class, 24, used: 26
```

```
??, ??, ??, ??, ??, ??, ??, ??
                                              Delta, ??, used:
       denominator (field), 365, 378, 378, used:
       dependencies, 33, 38, ??, ??, ??, ??, 31, 123,
                                                     ??, used: ??, ??, ??, ??, ??, ??, ??
       dependencies (field), ??, ??, used: ??, ??, ??, ??
       derive, 717, used:
                                              derived (field), 208, used: ??, ??, ??, ??, ??, ??,
D (module), 668, ??, ??, used: 669, 669, ??, ??,
                                                     ??, ??, ??, ??, ??, ??, ??, 609
       Derived, 293, 307, 307, 286, used: ??, 310, 300
       ??, ??, ??, ??, ??, ??
                                              DERIVED, ??, ??, used: ??, 297
D' (module), ??, used: ??
                                              DERIVED (camlyacc token), 298, used: 300
dag (field), ??, used: ??, ??
                                              derived_arrays (field), 208, used: ??, ??, ??,
DAG (module), 33, 29, used: ??, ??, ??, ??
                                                     ??, ??, ??, ??, ??, ??, ??, ??, ??, 609
Dagger, 288, 284, used: 288
daq\_sans\_color, 598, used:
                                              derived_parameters, ??, ??, ??, ??, ??, ??, ??,
dag\_to\_dot, ??, used: ??, ??
                                                     ??, ??, ??, ??, ??, ??, ??, ??, used:
data (field), 702, 703, used: 702, 702, 703, 703
                                                     ??, ??, ??, ??, ??
date. 614. used:
                                              derived_parameter_arrays, ??, ??, ??, ??, ??,
Dbar, ??, used: ??, ??, ??, ??
                                                     ??,??,??,??,??,??,??,??,??,??,??,??
debug, 600, used: 601
decay (type), 584, 584, 278, 278, 583, 277,
                                                     ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                     ??, ??, ??, ??, ??, ??, ??, ??
       used: 584, 584, 586, 586, 278, 278, 583,
       583, 583, 583, 277, 277
                                              derive_inner, 718, used:
                                              derive_inner', 718, used:
decay (camlyacc non-terminal), 364, used: 363
                                              derive_outer, 718, used:
Decay, 278, 278, 277, used: 279, 279, 279
Decay (module), 44, 48, 52, 43, used: ??
                                              descend, 659, 659, 660, 661, 661, 662,
decays (field), 586, 378, used: 586, 586, 587, 588
                                                     662, used: 659, 659, 659, 660, 661, 661,
decays (camlyacc non-terminal), 363, used: 363,
                                                     662, 662
                                              description, 23, 60, 63, 59, used: ??, ??
       363
Decay_Dictionary, 359, used: ??, 363
                                              description (field), 63, used: 63, 63, 67
decay_of_momenta, 584, 586, 583, used:
                                              DH. ??. used:
declaration (type), 295, 296, 287, 287, 359,
                                              diag, ??, ??, used: ??, ??, ??, ??
       used: 295, 296, 287, 288, 359, 359
                                              diagnostic (type), 246, used: 246
declaration (camlyacc non-terminal), 299, 362,
                                              diagram, 599, used: 599
       used: 299, 362
                                              diagram (type), 694, 692, used: 692, 692, 692,
                                                     692, 692, 693, 693
Declaration (module), 311, used: 313, 313, 314
Declaration (sig), 311, used: 311
                                              diagrams, 23, 26, 596, 606, 606, 17, 596, used:
declarations (camlyacc non-terminal), 299, 362,
       used: 299, 299, 362, 362
                                              diagrams (field), 705, 700, used: 705, 706, 599,
declare, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                     600,600
       diagrams_per_keystone, 23, 27, 17, used: 27
       ??, ??, ??, ??, ??, ??
                                              diagrams_via_keystones, 23, 27, 17, used:
declare\_argument, 593, used: 593, 593
                                              diagram\_class (type), 24, used:
decode, 606, used: 606
                                              diagram_columns, 694, used: 696, 696
decreasing, 694, used:
                                              diagram_of_tableau, 695, 693, used: 695, 695,
default, 61, used: 61, 61, 62
Default (module), 687, 54, 684, 43, used: 324,
                                              diagram_rows, 694, used: 696, 696
       596, 596, 606, 43, 422, 596
                                              diagram\_sans\_color, 599, used: 600
                                              dictionary, ??, ??, 125, ??, used:
Default W (module), 54, 43, used: 596
default_function, 593, used: 593
                                              dictionary (field), ??, used: ??, ??
default_group, 293, 286, used: 309
                                              dictionary (camlyacc non-terminal), 363, used:
default_subroutine, 593, used: 593
default\_UFO\_dir, 614, used:
                                                   ??, ??, ??, ??, ??, ??, ??, used: ??
default_width, 444, ??, ??, ??, ??, ??, ??, ??.
                                                   278, 282, 717, 277, used: ??, 630, 671,
                                              diff,
       282, ??
       ??, ??, ??, used: 444, 445, 446, 446, 446,
                                              Diff,
                                                    291, 203, 285, used: 291, 339, ??, ??, ??,
       ??, ??, ??
```

```
Difference, 329, 338, 329, used: 330, 336, 340
                                                       Dim6\_Scalar2\_Vector2\_D, 182, used: ??
DIFF\_ELEMENT (sig), ??, ??, used: ??, ??,
                                                       Dim6\_Scalar2\_Vector2\_DP, 182, used: ??
        ??, ??
                                                       Dim6\_Scalar2\_Vector2\_PB, 182, used: ??
                                                       Dim6\_Scalar\_Vector\_Vector\_D, 180, used: ??
      289, 284, used: ??, 302
digit,
                                                       Dim6\_Scalar\_Vector\_Vector\_DP, 180, used: ??
digit (camllex regexpr), 57, 297, 331, used: 57,
                                                       Dim6\_Vector4\_DW, 182, used: ??
        297, 331, 331
                                                       Dim6\_Vector4\_W, 182, used: ??
Digit, 288, 284, used: 289
                                                       Dim6\_Vector\_Vector\_Vector\_T, 180, used:
DIGIT, ??, ??, used: ??, 297
DIGIT (camlyacc token), 298, used: 301, 302,
                                                       Dim6\_vev3, ??, used: ??
                                                       Dim6\_WWZ\_D, 180, used: ??
                                                       Dim6\_WWZ\_DPWDW, 180, used: ??
digits, 616, used: 617, 617
digits (field), 616, used: 617, 617, 617
                                                       Dim6\_WWZ\_DW, 180, used: ??
                                                       Dim7\_Tensor\_2\_Vector\_Vector\_T, 180, used:
dim, 44, 45, 49, 41, used: 50, 50, 50, 52, 53, 53,
                                                       dim8_gauge_higgs4, ??, ??, used: ??, ??
        587, 588, 60, ??
dim0, 49, used: 49, 50, 50, 51, 51, 51
                                                       dim8\_gauge\_higgs4\_m, ??, ??, used: ??, ??
Dim4\_Vector\_Vector\_Vector\_L, 180, used: ??,
                                                       Dim8\_Scalar2\_Vector2\_1, 182, used: ??, ??
        ??, ??, ??, ??
                                                       Dim8\_Scalar2\_Vector2\_2, 182, used: ??, ??
Dim4_Vector_Vector_Vector_L5, 180, used: ??,
                                                       Dim8\_Scalar2\_Vector2\_m\_0, 182, used: ??, ??
        ??, ??, ??, ??
                                                       Dim8\_Scalar2\_Vector2\_m\_1, 182, used: ??, ??
                                                       Dim8\_Scalar2\_Vector2\_m\_7, 182, used: ??, ??
Dim4\_Vector\_Vector\_Vector\_T, 180, used: ??,
        ??, ??, ??, ??
                                                       Dim8_Scalar4, 182, used: ??, ??
Dim4_-Vector_-Vector_-Vector_-T5, 180, used: ??,
                                                       Dim8 - Vector4 - m - \theta, 182, used: ??, ??
        ??, ??, ??, ??
                                                       Dim8\_Vector4\_m\_1, 182, used: ??, ??
Dim5\_Scalar\_Gauge2, 180, used: ??, ??, ??, ??,
                                                       Dim8\_Vector4\_m\_7, 182, used: ??, ??
        ??, ??, ??, ??, ??, ??, ??, 609
                                                       Dim8\_Vector4\_t\_0, 182, used: ??, ??
Dim5\_Scalar\_Gauge2\_Skew, 180, used: ??, ??,
                                                       Dim8\_Vector4\_t\_1, 182, used: ??, ??
        ??, ??, ??, ??
                                                       Dim8\_Vector4\_t\_2, 182, used: ??, ??
Dim5\_Scalar\_Scalar2, 180, used: ??, ??, ??, ??,
                                                       dirac (field), 365, used:
                                                       dirac (type), 348, 349, 334, 364, used: 349, 349,
Dim5\_Scalar\_Vector\_Vector\_T, 180, used:
                                                               335, 364
Dim5\_Scalar\_Vector\_Vector\_TU, 180, used: ??
                                                              294, 349, 349, 286, 335, used: 350, 350,
                                                       Dirac,
Dim5\_Scalar\_Vector\_Vector\_U, 180, used: ??,
                                                               352
        ??, ??
                                                       Dirac (module), 214, 221, 317, 321, 213, 214,
Dim5\_Tensor\_2\_Vector\_Vector\_1, 180, used:
                                                               used: 325, 365
Dim5\_Tensor\_2\_Vector\_Vector\_2, 180, used:
                                                       Dirac \text{ (sig)}, \quad 316, 320, \text{ used: } 317, 321
Dirac\_Matrices (sig), 316, 320, used: 317, 317,
        ??, used: ??, ??, ??, ??, ??, ??
                                                               320, 321
Dim6\_AHHZ\_D, 182, used: ??
                                                       Dirac_R (module), 220, used: 221
Dim6\_AHHZ\_DP, 182, used: ??
                                                       dirac\_string (type), 364, used: 364, 364,
Dim6\_AHHZ\_PB, 182, used: ??
                                                               364, 364, 364, 365, 365, 365
Dim6_AHWW_DPB, 182, used: ??
                                                       dirac\_string\_to\_matrix, 365, used:
Dim6\_AHWW\_DPW, 182, used: ??
                                                       dirac_string_to_string, 365, used:
Dim6\_AHWW\_DW, 182, used: ??
                                                       dirac\_to\_string, 351, used: 351
Dim6\_AWW\_DP, 180, used: ??
                                                       disambiguate_fusions, ??, used: ??
Dim6\_AWW\_DW, 180, used: ??
                                                       disambiguate_indices, 343, 348, 355, 358, 334,
Dim6\_Cphi, ??, used: ??
                                                               used: 348, 348
Dim6\_Gauge\_Gauge\_Gauge, 180, used: ??, ??,
                                                       disambiguate_indices1, 348, 355, used: 348, 355
        ??, ??, ??
                                                       Discrete (module), 35, 32, used:
Dim6\_Gauge\_Gauge\_5, 180, used: ??,
                                                       disjoint, 671, used: 672
        ??, ??, ??, ??
                                                       distinct, 315, used: 318
                                                       distinct2, 315, used: 318, 323
Dim6\_Gauge\_Gauge\_i, 180, used: ??
                                                       distinct3, 315, used: 323
Dim6\_GGG, 180, used:
Dim6_H_4_P_2, 182, used: ??
                                                       distribute_degrees, 26, used: 26
Dim6\_HAZ\_D, 180, used: ??
                                                       distribute\_degrees', 25, used: 26, 26
Dim6\_HAZ\_DP, 180, used: ??
                                                       distribute_degrees", 26, used: 25, 26
Dim6\_HHH, 180, used: ??
                                                       div, 292, 715, 716, 285, used: ??, 346, 301
                                                       DIV, ??, ??, ??, ??, ??, used: ??, ??, ??,
Dim6\_HHZZ\_T, 182, used: ??
Dim6\_HWWZ\_DDPW, 182, used: ??
                                                               297, 331
Dim6\_HWWZ\_DPB, 182, used: ??
                                                       DIV (camlyacc token), 298, 331, 361, used:
Dim6\_HWWZ\_DPW, 182, used: ??
                                                               299, 301, 302, 303, 332, 332, 362
Dim6\_HWWZ\_DW, 182, used: ??
                                                       divide, 330, 329, used: ??, 332
```

```
division, 708, used: 709
                                                    D_Alpha_ZZWW0_T, ??, ??, ??, ??, used:
    ??, ??, ??, ??, used: ??, ??, ??, ??, ??,
                                                            ??, ??, ??, ??, ??
                                                    D_Alpha_ZZWW1_S, ??, ??, ??, ??, ??, used:
       ??, ??, ??, ??, ??, ??, ??, ??, ??
      ??, used: ??, ??, ??
                                                           ??, ??, ??, ??, ??
Dodd,
                                                    D_Alpha_ZZWW1_T, ??, ??, ??, ??, vsed:
domain (type), 735, 736, 735, used: 735, 735,
                                                           ??, ??, ??, ??, ??
       736, 735, 735, 735, 735, 735
                                                    D_Alpha_ZZWW1_U, ??, ??, ??, ??, ??, used:
DOT, ??, ??, ??, used: ??
                                                           ??, ??, ??, ??, ??
DOT (camlyacc token), 331, 361, used: 362
    ??, ??, ??, ??, used: ??, ??, ??, ??, ??,
                                                    D_Alpha_ZZZZ_S, ??, ??, ??, ??, !?, used: ??,
        ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                           ??, ??, ??, ??
drb, 49, used: 50, 50, 50, 50, 53, 54
                                                    D_{-}Alpha_{-}ZZZZ_{-}T, ??, ??, ??, ??, used: ??,
                                                           ??, ??, ??, ??
drb\theta, 49, used: 50, 51, 51, 51
DScalar2 \_ Vector2, 182, used:
                                                    D_FM0_HHWW0_S, ??, ??, used: ??, ??
                                                                         ??, ??, used: ??, ??
                                                    D_FM0_HHWW0_T
DScalar2\_Vector2\_K\_Matrix\_ms, 182, used:
                                                    D_FM0_HHWW0_U, ??, ??, used: ??, ??
        ??, ??
DScalar2\_Vector2\_m\_0\_K\_Matrix\_cf, 182,
                                                    D_FM0_HHWW1_S, ??, ??, used: ??, ??
                                                    D_FM0_HHWW1_T, ??, ??, used: ??, ??
       used: ??, ??
DScalar2\_Vector2\_m\_1\_K\_Matrix\_cf,
                                    182,
                                                    D_FM0_HHWW1_U, ??, ??, used: ??, ??
                                                    D_FM0_HHZZ0_S, ??, ??, used: ??, ??
       used: ??, ??
                                                    D_FM0_HHZZ0_T, ??, ??, used: ??, ??
DScalar2\_Vector2\_m\_7\_K\_Matrix\_cf, 182,
                                                    D_FM0_HHZZ0_U, ??, ??, used: ??, ??
       used: ??, ??
                                                    D_FM0_HHZZ1_S, ??, ??, used: ??, ??
DScalar4, 182, used:
DScalar4\_K\_Matrix\_ms, 182, used: ??, ??
                                                    D_FM0_HHZZ1_T, ??, ??, used: ??, ??
dummy, 617, 616, 365, used: ??
                                                    D_FM0_HHZZ1_U, ??, ??, used: ??, ??
                                                    D_FM0_WWWW0_S, ??, ??, used: ??, ??
Dummy \text{ (module)}, 448, used:
                                                    D_FM0_WWWW0_T, ??, ??, used: ??, ??
dummy3, 65, used: 65
                                                    D_FM0_WWWW0_U, ??, ??, used: ??, ??
dummy4, 65, used: 65
dummyn, 65, used: 65
                                                                         ??, ??, used: ??, ??
                                                    D_FM0_WWWW2_S,
dump, 710, 307, 310, 701, used:
                                                    D_FM0_WWWW2_T,
                                                                          ??, ??, used: ??, ??
                                                    D_FM0_WWWW2_U, ??, ??, used: ??, ??
dump2, 218, used: 218, 218
dump2\_all, 218, used: 218
                                                    D_{FM0} = ZZWW0 = S, ??, ??, used: ??, ??
dump_file, 304, used: 314
                                                    D_{FM0} = ZZWW0 = T, ??, ??, used: ??, ??
Duplicate (exn), 44, 44, 50, 584, 585, 41, 583,
                                                    D_FM0_ZZWW0_U, ??, ??, used: ??, ??
                                                                        ??. ??. used: ??. ??
       used:
                                                    D_FM0_ZZWW1_S,
                                                    D_FM0_ZZWW1_T, ??, ??, used: ??, ??
Dyn (module), 668, 667, used: 668, 281
                                                    D_FM0_ZZWW1_U, ??, ??, used: ??, ??
Dyn (sig), 667, 667, used: 667
                                                    D_FM0_ZZZZ_S, ??, ??, used: ??, ??
D_Alpha_HHHH_S, ??, ??, used: ??, ??
                                                   D_FM0_ZZZZ_T, ??, ??, used: ??, ??
D_Alpha_HHHH_T, ??, ??, used: ??, ??
D_Alpha_HHWW0_S, ??, ??, used: ??, ??
                                                    D_FM0_ZZZZ_U, ??, ??, used: ??, ??
D_Alpha_HHWW0_T, ??, ??, used: ??, ??
                                                    D_FM1_HHWW0_S, ??, ??, used: ??, ??
D_-Alpha_-HHWW1_-S, ??, ??, used: ??, ??
                                                                         ??, ??, used: ??, ??
                                                    D_FM1_HHWW0_T,
                                                    D_FM1_HHWW0_U, ??, ??, used: ??, ??
D_Alpha_HHWW1_T, ??, ??, used: ??, ??
D_Alpha_HHWW1_U, ??, ??, used: ??, ??
                                                    D_FM1_HHWW1_S, ??, ??, used: ??, ??
D_Alpha_HHZZO_S, ??, ??, used: ??, ??
                                                    D_FM1_HHWW1_T, ??, ??, used: ??, ??
D\_Alpha\_HHZZ0\_T, ??, ??, used: ??, ?? D\_Alpha\_HHZZ1\_S, ??, ??, used: ??, ??
                                                    D_FM1_HHWW1_U, ??, ??, used: ??, ??
                                                    D_FM1_HHZZ0_S, ??, ??, used: ??, ??
D_Alpha_HHZZ1_T, ??, ??, used: ??, ??
                                                    D_FM1_HHZZ0_T, ??, ??, used: ??, ??
D_Alpha_HHZZ1_U, ??, ??, used: ??, ??
                                                    D_FM1_HHZZ0_U, ??, ??, used: ??, ??
D_-Alpha_-WWWW0_-S, ??, ??, ??, ??, used:
                                                    D_FM1_HHZZ1_S, ??, ??, used: ??, ??
       ??, ??, ??, ??, ??
                                                    D_FM1_HHZZ1_T, ??, ??, used: ??, ??
D_-Alpha_-WWWW0_-T, ??, ??, ??, ??, used:
                                                    D_FM1_HHZZ1_U, ??, ??, used: ??, ??
       ??, ??, ??, ??, ??
                                                                          ??, ??, used: ??, ??
                                                    D_FM1_WWWW0_S,
                                                                          ??, ??, used: ??, ??
D_Alpha_WWWW0_U, ??, ??, ??, ??, used:
                                                    D_FM1_WWWW0_T,
       ??, ??, ??, ??, ??
                                                    D_FM1_WWWW0_U
                                                                         ??, ??, used: ??, ??
                                                                          ??, ??, used: ??, ??
D_Alpha_WWWW2_S, ??, ??, ??, ??, ??, used:
                                                    D_FM1_WWWW2_S,
       ??, ??, ??, ??, ??
                                                    D_FM1_WWWW2_T,
                                                                         ??, ??, used: ??, ??
                                                    D_FM1_WWWW2_U, ??, ??, used: ??, ??
D_Alpha_WWWW2_T, ??, ??, ??, ??, used:
                                                    D_FM1_ZZWW0_S, ??, ??, used: ??, ??
        ??, ??, ??, ??, ??
D_Alpha_ZZWW0_S, ??, ??, ??, ??, ??, used:
                                                    D_{FM1}_{ZZWW0_{T}}, ??, ??, used: ??, ??
                                                    D_FM1_ZZWW0_U, ??, ??, used: ??, ??
       ??, ??, ??, ??, ??
```

 $D_FM1_ZZWW1_S$, ??, ??, used: ??, ??

```
D_FM1_ZZWW1_T, ??, ??, used: ??, ??
D_FM1_ZZWW1_U, ??, ??, used: ??, ??
D_FM1_ZZZZ_S, ??, ??, used: ??, ??
                 ??, ??, used: ??, ??
D_FM1_ZZZZ_T,
D_FM1_ZZZZ_U, ??, ??, used: ??, ??
D_FM7_HHWW0_S, ??, ??, used: ??, ??
                    ??, ??, used: ??, ??
D_FM7_HHWW0_T,
D_FM7_HHWW0_U, ??, ??, used: ??, ??
D_FM7_HHWW1_S, ??, ??, used: ??, ??
D_FM7_HHWW1_T,
                    ??, ??, used: ??, ??
D_FM7_HHWW1_U, ??, ??, used: ??, ??
D_FM7_HHZZ0_S, ??, ??, used: ??, ??
                   ??, ??, used: ??, ??
D_FM7_HHZZ0_T
                  ??, ??, used: ??, ??
D_FM7_HHZZ0_U,
D_FM7_HHZZ1_S, ??, ??, used: ??, ??
D_FM7_HHZZ1_T, ??, ??, used: ??, ??
D_FM7_HHZZ1_U, ??, ??, used: ??, ??
D_FM7_WWWW0_S,
                     ??, ??, used: ??, ??
D_FM7_WWWW0_T,
                     ??, ??, used: ??, ??
D_FM7_WWWW0_U, ??, ??, used: ??, ??
D_FM7_WWWW2_S, ??, ??, used: ??, ??
D_{-}FM7_{-}WWWW2_{-}T
                     ??, ??, used: ??, ??
D_FM7_WWWW2_U, ??, ??, used: ??, ??
D_FM7_ZZWW0_S, ??, ??, used: ??, ??
D_FM7_ZZWW0_T, ??, ??, used: ??, ??
D_FM7_ZZWW0_U, ??, ??, used: ??, ??
D_FM7_ZZWW1_S,
                   ??, ??, used: ??, ??
                    ??, ??, used: ??, ??
D_FM7_ZZWW1_T,
D_FM7_ZZWW1_U, ??, ??, used: ??, ??
D_FM7_ZZZZ_S, ??, ??, used: ??, ??
                 ??, ??, used: ??, ??
D_{-}FM\gamma_{-}ZZZZ_{-}T,
D_{FM7}ZZZZU_{U}
                 ??, ??, used: ??, ??
D_FT0_AAAA_S,
                 ??, ??, used: ??, ??
D_FT0_AAAA_T,
                  ??, ??, used: ??, ??
                 ??, ??, used: ??, ??
D_FT0_AAAA_U,
                 ??, ??, used: ??, ??
D_FT0_AAAZ_S,
D_FT0_AAAZ_T
                 ??, ??, used: ??, ??
D_FT0_AAAZ_U, ??, ??, used: ??, ??
D_FT0_AAWW0_S, ??, ??, used: ??, ??
D_FT0_AAWW0_T, ??, ??, used: ??, ??
D_FT0_AAWW0_U, ??, ??, used: ??, ??
D_FT0_AAWW1_S, ??, ??, used: ??, ??
                    ??, ??, used: ??, ??
D_FT0_AAWW1_T,
D_FT0_AAWW1_U, ??, ??, used: ??, ??
D_FT0_AAZZ_S, ??, ??, used: ??, ??
D_FT0_AAZZ_T, ??, ??, used: ??, ??
D_FT0_AAZZ_U, ??, ??, used: ??, ??
D_FT0_AZWW0_S, ??, ??, used: ??, ??
D_{-}FT0_{-}AZWW0_{-}T,
                   ??, ??, used: ??, ??
D_FT0_AZWW0_U, ??, ??, used: ??, ??
                   ??, ??, used: ??, ??
D_FT0_AZWW1_S,
                   ??, ??, used: ??, ??
D_FT0_AZWW1_T,
D_FT0_AZWW1_U, ??, ??, used: ??, ??
D_FT0_AZZZ_S, ??, ??, used: ??, ??
D_FT0_AZZZ_T, ??, ??, used: ??, ??
D_FT0_AZZZ_U, ??, ??, used: ??, ??
D_FTO_WWWWO_S, ??, ??, used: ??, ??
                     ??, ??, used: ??, ??
D_{-}FT\theta_{-}WWWW\theta_{-}T,
                     ??, ??, used: ??, ??
D_FT0_WWWW0_U,
                    ??, ??, used: ??, ??
D_FT0_WWWW2_S,
D_FT0_WWWW2_T, ??, ??, used: ??, ??
```

```
D_{FT0}WWWW2_{U}, ??, ??, used: ??, ??
                   ??, ??, used: ??, ??
D_FT0_ZZWW0_S,
                   ??, ??, used: ??, ??
D_{-}FT0_{-}ZZWW0_{-}T,
                    ??, ??, used: ??, ??
D_FT0_ZZWW0_U,
                   ??, ??, used: ??, ??
D_FT0_ZZWW1_S,
D_FT0_ZZWW1_T,
                   ??, ??, used: ??, ??
D_FT0_ZZWW1_U,
                   ??, ??, used: ??, ??
D_FT0_ZZZZ_S, ??, ??, used: ??, ??
D_FT0_ZZZZ_T, ??, ??, used: ??, ??
D_FT0_ZZZZ_U, ??, ??, used: ??, ??
D_FT1_AAAA_S,
                 ??, ??, used: ??, ??
D_{F}T1_{A}AAA_{T}
                 ??, ??, used: ??, ??
D_{FT1}AAAA_{U}, ??, ??, used: ??, ??
D_FT1_AAAZ_S, ??, ??, used: ??, ??
D_FT1_AAAZ_T,
                 ??, ??, used: ??, ??
D_FT1_AAAZ_U, ??, ??, used: ??, ??
D_FT1_AAWW0_S, ??, ??, used: ??, ??
D_{FT1}AAWW0_{T}, ??, ??, used: ??, ??
D_FT1_AAWW0_U, ??, ??, used: ??, ??
                   ??, ??, used: ??, ??
D_FT1_AAWW1_S,
                    ??, ??, used: ??, ??
D_FT1_AAWW1_T,
D_FT1_AAWW1_U, ??, ??, used: ??, ??
D_FT1_AAZZ_S, ??, ??, used: ??, ??
D_FT1_AAZZ_T, ??, ??, used: ??, ??
D_FT1_AAZZ_U, ??, ??, used: ??, ??
D_FT1_AZWW0_S, ??, ??, used: ??, ??
                   ??, ??, used: ??, ??
D_FT1_AZWW0_T,
D_FT1_AZWW0_U, ??, ??, used: ??, ??
D_FT1_AZWW1_S,
                   ??, ??, used: ??, ??
D_{-}FT1_{-}AZWW1_{-}T,
                   ??, ??, used: ??, ??
D_{FT1}AZWW1_{U}, ??, ??, used: ??, ??
D_FT1_AZZZ_S, ??, ??, used: ??, ??
D_FT1_AZZZ_T, ??, ??, used: ??, ??
D_FT1_AZZZ_U, ??, ??, used: ??, ??
D_FT1_WWWW0_S, ??, ??, used: ??, ??
                     ??, ??, used: ??, ??
D_FT1_WWWW0_T,
D_FT1_WWWW0_U,
                     ??, ??, used: ??, ??
D_FT1_WWWW2_S, ??, ??, used: ??, ??
D_FT1_WWWW2_T, ??, ??, used: ??, ??
D_FT1_WWWW2_U, ??, ??, used: ??, ??
D_FT1_ZZWW0_S, ??, ??, used: ??, ??
D_FT1_ZZWW0_T, ??, ??, used: ??, ??
D_FT1_ZZWW0_U, ??, ??, used: ??, ??
D_FT1_ZZWW1_S, ??, ??, used: ??, ??
D_FT1_ZZWW1_T
                   ??, ??, used: ??, ??
D_FT1_ZZWW1_U, ??, ??, used: ??, ??
D_FT1_ZZZZ_S, ??, ??, used: ??, ??
D_FT1_ZZZZ_T,
                ??, ??, used: ??, ??
                 ??, ??, used: ??, ??
D_FT1_ZZZZ_U,
D_{F}T2_{A}AAA_{S},
                 ??, ??, used: ??, ??
                 ??, ??, used: ??, ??
D_FT2_AAAA_T,
                 ??, ??, used: ??, ??
D_FT2_AAAA_U,
D_FT2_AAAZ_S, ??, ??, used: ??, ??
D_FT2_AAAZ_T, ??, ??, used: ??, ??
D_FT2_AAAZ_U, ??, ??, used: ??, ??
D_FT2_AAWW0_S, ??, ??, used: ??, ??
D_FT2_AAWW0_T, ??, ??, used: ??, ??
D_FT2_AAWW0_U, ??, ??, used: ??, ??
                   ??, ??, used: ??, ??
D_FT2_AAWW1_S,
D_FT2_AAWW1_T
                   ??, ??, used: ??, ??
D_FT2_AAWW1_U, ??, ??, used: ??, ??
```

```
D_FT2_AAZZ_S, ??, ??, used: ??, ??
                                                   D_FTrsi_ZZWW1_T, ??, ??, used: ??, ??
D_FT2_AAZZ_T, ??, ??, used: ??, ??
                                                   D_FTrsi_ZZWW1_U, ??, ??, used: ??, ??
D_FT2_AAZZ_U, ??, ??, used: ??, ??
                                                   D_FTrsi_ZZZZ_S, ??, ??, used: ??, ??
D_FT2_AZWW0_S,
                    ??, ??, used: ??, ??
                                                   D_FTrsi_ZZZZ_T, ??, ??, used: ??, ??
                                                   D_FTrsi_ZZZZ_U, ??, ??, used: ??, ??
                    ??, ??, used: ??, ??
D_FT2_AZWW0_T,
                                                   D_K1_L, ??, used: ??, ??, ??
D_FT2_AZWW0_U, ??, ??, used: ??, ??
                   ??, ??, used: ??, ??
                                                   D_K1_R, ??, used: ??, ??, ??
D_FT2_AZWW1_S,
D_FT2_AZWW1_T, ??, ??, used: ??, ??
                                                   D_{K2}L, ??, used: ??, ??, ??
D_FT2_AZWW1_U, ??, ??, used: ??, ??
                                                   D_{K2}R, ??, used: ??, ??, ??
D_FT2_AZZZ_S, ??, ??, used: ??, ??
                                                   e (type), ??, ??, ??, ??, used: ??, ??, ??, ??, ??,
D_FT2_AZZZ_T, ??, ??, used: ??, ??
                                                           ??. ??
D_FT2_AZZZ_U, ??, ??, used: ??, ??
                                                   E, 316, 318, 322, 323, ??, ??, ??, ??, ??, ??,
D_FT2_WWWW0_S,
                     ??, ??, used: ??, ??
                                                           D_FT2_WWWW0_T,
                      ??, ??, used: ??, ??
                                                           ??, ??, ??, used: 318, 323, ??, ??, ??, ??,
D_FT2_WWWW0_U,
                     ??, ??, used: ??, ??
D_FT2_WWWW2_S,
                     ??, ??, used: ??, ??
                                                   E (module), ??, 298, used: ??, 301, 301
D_FT2_WWWW2_T, ??, ??, used: ??, ??
                                                   E6, 293, used: 293
D_FT2_WWWW2_U, ??, ??, used: ??, ??
                                                   E7,
                                                        293, used: 293
D_FT2_ZZWW0_S, ??, ??, used: ??, ??
                                                        293, used: 293
                                                   E8,
D_FT2_ZZWW0_T, ??, ??, used: ??, ??
                                                   edge (type), 33, 33, 34, 34, 35, 36, ??, 30, 30,
D_FT2_ZZWW0_U, ??, ??, used: ??, ??
                                                           32, used: 33, 33, 34, 34, 34, 35, 36, ??,
D_FT2_ZZWW1_S, ??, ??, used: ??, ??
                                                           ??, 30, 30, 30, 31, 31, 31, 31, 31, 32, 32,
                                                           32, 32, 33, 33
D_FT2_ZZWW1_T, ??, ??, used: ??, ??
D_FT2_ZZWW1_U, ??, ??, used: ??, ??
                                                   Edges (module), ??, used: ??
D_FT2_ZZZZ_S, ??, ??, used: ??, ??
                                                   edges\_feynmf, 704, used: 704
D_FT2_ZZZZ_T, ??, ??, used: ??, ??
                                                   edges_feynmf', 704, used: 704, 704
D_FT2_ZZZZ_U, ??, ??, used: ??, ??
                                                   EEnd, ??, ??, used: ??
D_FTrsi_AAAA_S, ??, ??, used: ??, ??
                                                   Eidelta, ??, used:
D_FTrsi_AAAA_T, ??, ??, used: ??, ??
                                                   electromagnetic_currents, ??, ??, ??, ??, ??, ??,
D_FTrsi_AAAA_U, ??, ??, used: ??, ??
                                                           D_FTrsi_AAAZ_S, ??, ??, used: ??, ??
                                                           used: ??, ??, ??, ??, ??, ??, ??, ??,
D_FTrsi_AAAZ_T, ??, ??, used: ??, ??
                                                           ??, ??, ??, ??, ??, ??, ??, ??
D_FTrsi_AAAZ_U, ??, ??, used: ??, ??
                                                   electromagnetic_currents', ??, used: ??
D_FTrsi_AAWW0_S, ??, ??, used: ??, ??
                                                   electromagnetic_currents_2, ??, ??, ??, used: ??,
D_FTrsi_AAWW0_T, ??, ??, used: ??, ??
                                                           ??, ??
D_FTrsi_AAWW0_U, ??, ??, used: ??, ??
                                                   electromagnetic_currents_3, ??, ??, ??, used: ??,
                    ??, ??, used: ??, ??
D_FTrsi_AAWW1_S,
                                                           ??, ??
D_FTrsi_AAWW1_T,
                     ??, ??, used: ??, ??
                                                   electromagnetic_sfermion_currents, ??, ??, ??,
D_FTrsi_AAWW1_U, ??, ??, used: ??, ??
                                                           used: ??, ??, ??
D_FTrsi_AAZZ_S, ??, ??, used: ??, ??
                                                   Electron, ??, used: ??, ??, ??, ??
D_FTrsi_AAZZ_T, ??, ??, used: ??, ??
                                                   electron1, 305, used: 305
D_FTrsi_AAZZ_U, ??, ??, used: ??, ??
                                                   electron2, 305, used: 305
D_FTrsi_AZWW0_S, ??, ??, used: ??, ??
                                                   elements, 645, 650, 651, 644, used: 630, 668,
D_FTrsi_AZWW0_T, ??, ??, used: ??, ??
                                                           668, 671, 671, 671, 36, 38, 591, 308, 310,
D_FTrsi_AZWW0_U, ??, ??, used: ??, ??
                                                           345, 281, 282, 282, ??, ??, ??, ??, 603
D_FTrsi_AZWW1_S,
                     ??, ??, used: ??, ??
                                                   elements', 650, used: 650, 650
D_FTrsi_AZWW1_T, ??, ??, used: ??, ??
                                                   eliminate_common_fusions, ??, used: ??
D_FTrsi_AZWW1_U, ??, ??, used: ??, ??
                                                   eliminate_common_fusions1, ??, used: ??
D_FTrsi_AZZZ_S, ??, ??, used: ??, ??
                                                   elt (type), 667, 667, 668, 668, 669, 671, 671,
D_FTrsi_AZZZ_T, ??, ??, used: ??, ??
                                                           280, 282, ??, 666, 667, 667, 670, used:
D_FTrsi_AZZZ_U, ??, ??, used: ??, ??
                                                           667, 667, 668, 668, 668, 668, 668, 669,
                      ??, ??, used: ??, ??
                                                           671, 280, ??, 666, 666, 667, 667, 667, 667,
D_FTrsi_WWWW0_S,
                      ??, ??, used: ??, ??
                                                           667, 667, 667, 670, 670
D_FTrsi_WWWW0_T,
D_FTrsi_WWWW0_U, ??, ??, used: ??, ??
                                                   Elt_Base (sig), 667, 666, used: 668, 668, 666,
D_FTrsi_WWWW2_S, ??, ??, used: ??, ??
                                                           667
D_FTrsi_WWWW2_T, ??, ??, used: ??, ??
                                                   embedding, 636, 635, used:
D_FTrsi_WWWW2_U, ??, ??, used: ??, ??
                                                   embedding (field), 635, used: 636, 636, 636
D_FTrsi_ZZWW0_S, ??, ??, used: ??, ??
                                                   embedding1, 636, 635, used:
D_FTrsi_ZZWW0_T, ??, ??, used: ??, ??
                                                   embedding1 (field), 636, used: 636, 636, 636
D_FTrsi_ZZWW0_U, ??, ??, used: ??, ??
                                                   embedding2, 636, 635, used:
D_FTrsi_ZZWW1_S, ??, ??, used: ??, ??
                                                   embedding2 (field), 636, used: 636, 636, 636
```

```
embed_in_decay, 585, used: 585, 586, 586
                                                     eval_offspring, 38, used: 39
embed_in_decays, 586, used: 586, 586
                                                     even, 684, 685, 686, 683, used:
       645, 645, 650, 671, 671, 658, 658, 658,
                                                     ew\_order, ??, used: ??, ??
empty,
        660, 661, 615, 33, 35, 36, 37, 39, 586,
                                                     ew\_order (type), ??, used:
        295, 296, 305, 308, ??, ??, 644, 670,
                                                     EW\_order, ??, used: ??, ??
        656, 656, 657, 615, 30, 287, 288, 125,
                                                     example, 303, 304, 313, 327, 358, 303, 335,
        ??, used: ??, ??, 736, 650, 630, 636, 668,
                                                             used:
        668, 671, 671, 671, 672, 672, 658, 658, 659,
                                                     exceptional, 293, used: 294
                                                     exchange_momenta, ??, used: ??
        660, 661, 661, 661, 662, 615, 615, 684,
        708, 708, 24, 36, 36, 36, 37, 37, 37, 37, 37,
                                                     exchange_momenta_list, ??, used: ??
        38, 39, 39, 39, 587, ??, ??, ??, ??, 590,
                                                     exclusion (camlyacc non-terminal), 58, used: 58
        590, 591, 592, 306, 308, 309, 310, 321,
                                                     exclusions (type), ??, ??, ??, ??, ??, ??, 122,
                                                             125, ??, ??, used: ??, ??, ??, ??, ??,
        321, 322, 330, 330, 336, 343, 345, 345,
        345, 355, 355, 66, 282, 282, ??, ??, ??,
                                                             122, 123, 125, 125, ??, ??, ??, ??
        exists, 621, 621, 620, used: 630, 26, 66
        ??, ??, 603, 603, 603, ??, ??, ??, ??, 607
                                                     Exp, 337, 203, used: 337, 339
Empty, 645, used: 645, 645, 646, 647, 647, 648,
                                                     expand_decays, 278, 280, 277, used: 602
        649, 649, 649
                                                     expand_home, 619, 619, used:
Empty\_List, 359, used: ??, 362
                                                     expand_includes, 296, 288, used: 303, 304
em_lqino_currents, ??, used: ??
                                                     expand_scatterings, 278, 280, 277, used: 602
em\_lq\_currents, ??, used: ??
                                                     export, 658, 660, 660, 662, 656, 657, used:
em_up_type_currents, ??, used: ??
                                                     export', 660, 662, used: 660, 660, 662, 662
END, ??, ??, ??, ??, ??, ??, ??, used: ??,
                                                     expr, 305, used: 306
        ??, ??, ??, 57, 297, 331
                                                     expr (type), 329, 203, 329, used: 329, ??, 336,
END (camlyacc token), 57, 298, 332, 362, used:
                                                             343, 343, 203, 208, 210, 211, 329, 329,
        58, 299, 299, 299, 299, 301, 301, 301,
                                                             329, ??, 332, 334, 334, 332
        301, 332, 362
                                                     expr (camlyacc non-terminal), 301, 332, used:
end_step, 618, 616, used: ??
                                                             299, 301, 301, 332, 332, 332
enumerate, 627, 623, used: 630, ??, ??
                                                     Expr (module), 291, 336, 285, 332, used: 292,
env\_arg\theta (camllex regexpr), 297, used: 297
                                                             292, 293, 293, 295, 296, 296, ??, 348, 348,
env\_arg1 (camllex regexpr), 297, used: 297
                                                             285, 286, 287, 287, 332, 298
EPowSet (module), 671, used: 671, 671, 671,
                                                     expr_arg (camlyacc non-terminal), 299, used:
        671, 671, 671, 672, 672, 672
                                                             301
eps4\_g4\_g44\_decl, 423, used:
                                                     ext (field), 24, used: 24, 24, 25, 25, 25
eps4 \_g4 \_g44 \_init, 423, used:
                                                     extend, 615, 615, used: 615, ??
                                                     externals, ??, ??, ??, ??, 124, ??, used: 590, ??,
Epsilon, 348, 349, 355, 356, 335, 335, used:
        349, 349, 352, 356, 356
                                                             597, 598, 599, 605, 606
EpsilonBar, 355, 356, 335, used: 356, 356
                                                     externals (field), ??, ??, used: ??, ??
equal, 658, 658, 660, ??, ??, ??, ??, 656, 656,
                                                     external\_edges, 707, used: 708
                                                     external_edges_from, 707, used: 707, 707
EQUAL, ??, ??, ??, used: ??, ??, 297
                                                     external_flavors, ??, ??, ??, ??, ??, ??, ??, ??,
EQUAL (camlyacc token), 298, 361, used: 362,
                                                             ??, 607, 210, used: ??, ??, ??, 601, ??,
equal_or_dump2, 218, used: 218, 218, 219, 219,
                                                             equal\_or\_dump\_lines, 446, used: 446
EResult, ??, ??, used: ??
                                                             ??, ??, ??, 607
error_in_file, 303, 336, used: 304
                                                     external_flavors', ??, used: ??
                                                     external_flavors", ??, used: ??
error_in_string, 303, 335, used: 303, 336
                                                     external\_wfs, ??, ??, used: ??, ??
ESet (module), 671, used: 671, 671, 671,
                                                     ExtMSSM (module), ??, ??, used: ??
        671, 671, 671, 671
EStart, ??, ??, used: ??
                                                     extMSSM_flags (sig), ??, ??, used: ??, ??, ??,
    ??, ??, ??, ??, used: ??, ??, ??, ??, ??, ??,
Eta.
        ext_edges (field), 707, used: 708, 708, 710
        ??, ??, ??
                                                     ext_incidence, 708, used: 709
eta_higgs_gauge, ??, used: ??
                                                     ext_layout (type), 709, 700, used: 709, 700, 700
                                                     ext_nodes (field), 707, used: 708, 709, 709, 710,
eval, 33, 38, 584, 586, 717, 32, 583, used: 38,
                                                             710, 710, 710
                                                     f (type), ??, ??, ??, ??, 65, ??, ??, ??, ??,
eval_decay, 584, 585, 583, used: 585, 586
                                                             eval\_memoized, 33, 39, 32, used: 39, ??
eval\_memoized', 39, used: 39
```

```
??, 608, ??, ??, ??, used: ??, ??, ??, ??,
                                      factory (type), 343, used:
      ??, ??, ??, ??, ??, ??, ??, ??
                                      factor_add_color_index, 311, used: 312
  315, 317, 322, 323, 323, 355, 356, 357,
                                      factor_add_flavor_index, 311, used: 312
                                      factor\_add\_index, 312, used: 312
      335, used: 317, 323, 326, 326, 326, 356,
                                      factor\_add\_kind, 312, used: 312
      356, 357, 358, 358
F (module), ??, 589, ??, 307, ??, ??, 597, ??.
                                      factor_add_lorentz_index, 311, used: 312
      factor\_add\_other\_index, 311, used: 312
      factor_add_particle, 311, used: 312
      ??, ??, ??, ??, 608, 298, used: ??, 589,
                                      factor\_add\_prefix, 311, used: 312
      590, 590, 590, 590, 590, ??, 307, ??, ??,
                                      factor_of_token, 312, used: 312
      ??, ??, ??, ??, ??, ??, 597, 597, 598,
                                      factor\_stem, 311, used: 312
     598, 598, 598, 599, 599, 599, 603, 603,
                                      factor_to_string, 312, used: 313
      604, 604, 605, 606, 606, ??, ??, ??, ??,
                                       Fake\_Graded\_Map \text{ (module)}, \quad 39, \text{ used: } 40
      Fake\_Grading \text{ (module)}, \quad 35, \text{ used: } 40
      False, 55, 61, 55, used: 56, 56, 61, 61
                                      609, 299
                                            ??, ??, ??, ??, ??, ??, ??, ??, used:
F12, 184, used: ??
F123, 184, used: ??
                                            F124, 184, used: ??
                                             ??, ??, ??, ??, ??, ??, ??, ??, ??
F13, 184, used: ??
                                          702, 699, used: 702
                                      fan,
                                       FBF, 180, used: ??, ??, ??, ??, ??, ??, ??.
F132, 184, used: ??
F134, 184, used: ??
                                             F142, 184, used: ??
                                             F143, 184, used: ??
F2 (module), ??, used: ??, ??
                                            F2' (module), ??, used: ??, ??
                                            F21, 184, used: ??
                                             F213, 184, used: ??
                                            F214, 184, used: ??
                                            F23, 184, used: ??, ??
                                             F231, 184, used: ??
                                            F234, 184, used: ??, ??
F241, 184, used: ??
                                             F243, 184, used: ??
F3 (module), ??, used: ??, ??
                                             F3' (module), ??, used: ??, ??
                                             F31, 184, used: ??
                                             F312, 184, used: ??
F314, 184, used: ??
                                            F32, 184, used: ??
F321, 184, used: ??
                                            F324, 184, used: ??
F341, 184, used: ??
                                            F342, 184, used: ??
F4, 293, used: 293
                                            F412, 184, used: ??
                                            ??, ??, ??, 608
F413, 184, used: ??
F421, 184, used: ??
                                      fermion, ??, ??, ??, ??, ??, ??, ??, ??, ??,
F423, 184, used: ??
                                             F431, 184, used: ??
                                             ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                             608, 210, used: ??, ??, ??, ??, ??, ??, ??
F432, 184, used: ??
factor, 341, 342, 333, used: 342, ??
                                      fermion (label), ??, ??, 211, used:
                                      fermion (type), ??, 180, used: ??, 180, 182
factor (type), 311, used: 311, ??, ??, 125, ??
factorial, 675, 22, 23, 673, 17, used: 676, 676,
                                       Fermion, ??, ??, ??, used: ??, ??, ??, ??, ??,
                                             ??, ??, ??, ??, ??, ??, ??, ??
     681, 696, 696, 23, 26, 26
factorial', 675, used: 675, 675
                                      fermionbar (type), 180, used: 180, 182
factorize, 629, 624, used: 678, 678, ??, 346
                                      fermion\_lines, 365, used:
factorized_keystones, 678, 674, used: 20, 21
                                      fermion_lines (type), 178, used: 184, 365, 365,
factorized_partitions, 678, 674, used:
                                             365, 422, 422, 378
factorize_brakets, ??, used: ??
                                      fermion\_lines\_to\_string, 365, used:
factory, 343, used: 343, 343
                                      ferm\_of\_sff, ??, used:
```

```
feynmf (type), 703, 700, used: 705, 700, 700
                                                finalize1, 585, used: 585, 587
feynmf_footer, 706, used: 706, 707
                                                finalize_decay, 585, used: 586
feynmf_header, 706, used: 706, 707
                                                find, 645, 647, 650, 621, 621, 658, 658, 659,
feynmf_levels, 706, used: 706, 706, 707
                                                        660, 661, 35, 36, 39, 644, 620, 656,
feynmf_levels (type), 705, 700, used: 705, 700,
                                                        656, 657, used: 736, 736, 736, 647, 650,
       700, 700
                                                        668, 668, 621, 658, 659, 659, 661, 661,
feynmf_levels_plain, 706, 700, used:
                                                        661, 684, 708, 24, 36, 36, 37, 37, 37, 38,
feynmf_levels_wrapped, 707, 700, used:
                                                        38, 38, 38, 39, ??, ??, ??, ??, 590, 590,
feynmf\_set, 705, 599, used: 705, 600, 600
                                                        308, 308, 308, 308, 308, 309, 321, 322,
                                                       322, 336, 343, 354, ??, ??, ??, ??, ??, ??,
feynmf_set (type), 705, 700, used: 705, 705,
                                                        ??, ??, ??, ??
       700, 700
feynmf_sets, 705, used: 706, 706
                                                find1, 659, 661, used: 659, 659, 659, 660, 661,
feynmf_sets (type), 705, 700, used: 700, 700
feynmf_sets_plain, 706, 700, used:
                                                        661, 662, 662
                                                find_all, 637, 635, used: 637, 638, 638, 639, 639
feynmf_sets_wrapped, 706, 700, used: 600, 600,
                                                find_first, 637, 621, 635, used: 637, 638, 638,
                                                       638, 638, 621
feynmf_set_sans_color, 600, used: 600, 600
                                                find_opt, 645, 647, 651, 644, used: 647, 651,
feynmf\_set\_sans\_color\_empty, 600, used: 600
feynmf_style, 598, used: 599, 599
                                                first\_anonymous, 343, used: 343
fiber, 667, 668, 668, 669, 667, 667, used:
                                                first_char, 290, used: 291
fiber (type), 667, 668, 668, 669, 667, 667, used:
                                                first_match, 701, 583, used: 701, 584
       667, 668, 669, 667, 667, 667, 667
                                                first\_match', 583, used: 583, 583
                                                first_pair, 584, used: 587
Fiber (module), 668, used: 668, 668, 668, 668
fibered_dag (type), ??, used:
                                                first_pair', 584, used: 584, 584
fibers, 667, 668, 668, 669, 667, 667, used: 669
                                                flatmap, 627, 624, used: 627, 679, 701, 701, 707,
fiber_to_string, 281, used: 281
                                                        707, 707, 707, 15, 20, 21, 27, 28, ??, 591,
312, 319, 319, 319, 324, 348, 60, 280, 280,
       ??, ??, ??, ??, ??, 212, used: ??, ??, ??,
                                                        ??, ??, ??, ??, ??, ??, ??, 598, 601,
                                                        603, ??, ??, ??, ??, ??, ??, ??, ??, ??,
field (type), 313, 315, ??, ??, ??, ??, ??, ??,
                                                       ??, ??, ??, ??, ??, ??, ??, ??, ??, 212,
                                                        used: 315, 316, 316, 317, 318, 322, 323,
       323, 324, 212
                                                       Field (module), 315, used: 315, 317, 317, 318,
                                                        323, 324, 326, 326, 326, 326, 326, 326,
                                                        326, 327, 327, 327, 327, 327
                                                        Field (sig), 315, used: 315
fields (field), 324, 61, used: 324, 324, 325, 325,
                                                        326, 326, 326, 326, 327, 327, 327, 327, 61,
                                                       63, 65
                                                        Fields (module), 65, used: 65
file, 617, ??, ??, 616, ??, ??, used: 303, 304, ??
                                                       ??, ??, ??, ??, ??, 607
file (type), 306, 307, used: 307
                                                        289, used: 695, 290, 290, 325, ??, ??, ??,
                                                flatten,
                                                        file (camlyacc non-terminal), 299, 362, used:
                                                        299, 362
File, 616, ??, used: 617, ??
                                                        File (module), 295, 287, used: 303, 304, 304,
                                                        304, 306, 307, 303
                                                        Files (sig), 378, used:
                                                        ??, ??, ??, ??, ??, ??, ??, ??, ??
File_Tree (module), 295, 287, used: ??, ??, 288,
                                                flatten_keystones, ??, used: ??
       ??, 298, 299
                                                flatten_list, 290, used: 289
fill_table, ??, used: ??
                                                flatten_scripted, 290, used: 289
fill_tables, ??, used: ??
                                                flavor, ??, ??, ??, ??, 122, ??, used: 590, 590,
filtermap, 630, 624, used: 631, 631, 631, 632
                                                        ??, ??, ??, ??, 598, 599, 599, 606
filter_couplings, ??, used: ??, ??, ??
                                                flavor (field), 311, ??, ??, used: 311, 311, 311,
                                                       312, 312, ??, ??, ??, ??, ??, ??, ??, ??,
filter_couplings_triples, ??, used: ??, ??
filter_keystone, ??, used: ??
                                                        ??, ??, ??, ??, ??, ??, ??
filter_offspring, ??, used: ??
                                                flavor (type), ??, ??, ??, 60, 60, 278, 278,
                                                        ??, ??, ??, ??, ??, ??, ??, ??, 596,
filter_sign, 679, used: 679, 679, 680, 680
filter_vertices, ??, used: ??
                                                        597, ??, ??, ??, ??, ??, ??, ??, ??, ??,
finalize, 587, used: 587
```

```
??, ??, ??, ??, ??, ??, 607, 208, 59,
                                                           63, 281, ??, ??, 598, 601, 603, 604, ??,
        277, 122, 125, ??, ??, 596, used: ??, ??,
                                                           ??, ??
                                                   flavor\_to\_string (label), ??, ??, 211, used:
        ??, ??, ??, 589, 590, 590, 60, 60, 60, 60,
                                                   flavor_to_TeX, ??, ??, ??, ??, ??, ??, ??.
        61, 61, 61, 63, 65, 278, 278, 278, 278, 281,
                                                           282, ??, ??, ??, ??, ??, ??, ??, ??, ??,
        ??, ??, ??, ??, 596, 597, ??, ??, ??,
                                                           ??, 610, 211, used: ??, ??, ??, 598, 599,
        599, 600, ??, ??
        flavor\_to\_TeX (label), ??, ??, 211, used:
        flip\_incoming, 587, used: 587
        ??, ??, ??, ??, ??, ??, ??, 608, 210,
                                                   flip_momenta, ??, used: ??
       210, 210, 210, 210, 210, 210, 210, 210,
                                                   flip_s_channel_in, 44, 48, 52, 43, used: 587, 590
        210, 210, 210, 210, 211, 211, 211, 212,
                                                   float, 330, 329, used:
        212, 213, 213, 213, 213, ??, ??, ??, 59, 59,
                                                   Float, 329, 203, 329, 359, used: 330, ??, 339,
        59, 59, 59, 250, 250, 277, 277, 278, 278,
                                                           362
                                                   FLOAT, ??, ??, ??, used: ??, ??, 331
        122, 123, 124, 125, 126, 126, ??, ??, ??,
        ??, ??, ??, 596, 596, 596, 596
                                                   FLOAT (camlyacc token), 331, 361, used: 332,
Flavor,
       294, 295, 306, 307, 287, 287, used: ??,
        309, 309, 300, 301
                                                   flow,
                                                         213, 213, used: ??
Flavor (sig), ??, ??, used: ??, ??
                                                   flush, 446, 444, used: 617, 617, 319, 602, 603,
FLAVOR, ??, ??, used: ??, 297
                                                           603, 604
                                                   FM0_HHWW, ??, ??, used: ??, ??
FM0_HHZZ, ??, ??, used: ??, ??
FLAVOR (camlyacc token), 298, used: 300, 301
flavors, ??, ??, ??, ??, 281, ??, ??, ??, ??, ??,
        FM0_-WWWW0, ??, ??, used: ??, ??
                                                   FM0_WWWW2, ??, ??, used: ??, ??
        FM0_ZZWW0, ??, ??, used: ??, ??
        ??, ??, 607, 210, 125, ??, used: ??, ??,
       ??, 282, ??, ??, ??, ??
                                                   FM0_ZZWW1, ??, ??, used: ??, ??
flavors (field), 61, ??, used: 61, 61, 64, ??, ??
                                                   FM0_ZZZZ, ??, ??, used: ??, ??
flavors (label), ??, ??, 211, used: flavors (type), ??, used:
                                                   FM1_HHWW, ??, ??, used: ??, ??
                                                   FM1_HHZZ, ??, ??, used: ??, ??
                                                   FM1_WWWW0, ??, ??, used: ??, ??
flavors_to_string, 591, 62, used: 591, 591, 592,
                                                   FM1_WWWW2, ??, ??, used: ??, ??
        62, 63
flavor_keystone, ??, used: ??
                                                   FM1\_ZZWW0, ??, ??, used: ??, ??
flavor_keystones, ??, used: ??
                                                   FM1_ZZWW1, ??, ??, used: ??, ??
flavor\_of\_b, ??, used: ??, ??
                                                   FM1\_ZZZZ, ??, ??, used: ??, ??
flavor_of_f, ??, used: ??, ??
                                                   FM7\_HHWW, ??, ??, used: ??, ?? FM7\_HHZZ, ??, ??, used: ??, ??
flavor_of_string, ??, ??, ??, ??, ??, ??, ??, ??,
        FM7_WWWW0, ??, ??, used: ??, ??
                                                   FM7_WWWW2, ??, ??, used: ??, ??
        ??, 609, 210, used: ??, ??, ??, 324, 325,
                                                   FM7_ZZWW0, ??, ??, used: ??, ??
        325, 61, 279, ??, ??
                                                   FM7_ZZWW1, ??, ??, used: ??, ??
                                                   FM7_ZZZZ, ??, ??, used: ??, ??
flavor_of_string (label), ??, ??, 211, used:
flavor_sans_color, ??, ??, 213, 213, 122, ??,
                                                   FMap \text{ (module)}, ??, used: ??, ??
                                                   Fn (module), ??, used: ??, ??
Fn' (module), ??, used: ??, ??
        used: ??, ??, ??, ??, 598, 598
flavor_sans_color (type), ??, ??, 213, 213, 122, 
??, used: ??, ??, ??, 213, 213, 213, 213,
                                                   fold, 645, 649, 652, 658, 658, 659, 660, 662,
        250, 250, 122, 123, 124, 124, 126, ??, ??,
                                                           664, 702, 33, 33, 34, 35, 35, 35, 36, 37,
        ??, ??, ??
                                                           39, ??, 644, 656, 656, 657, 663, 699,
Flavor_Set (module), 590, used: 590, 590, 591
                                                           30, 31, 32, used: 649, 650, 652, 668, 668,
flavor_sums, 281, used: 282
                                                           672, 672, 659, 662, 664, 664, 702, 702,
flavor_symbol, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                                           703, 14, 14, 15, 25, 35, 36, 36, 36, 37, 37,
        37, 37, 37, 38, 38, 38, 38, 38, 38, 39, 39,
        591, 280, 280, ??, ??, ??, ??, ??, ??, ??,
                                                           ??, ??, 603
        610, 211, used: ??, ??, ??, ??, ??, ??,
        fold (type), 665, 663, used: 663
        ??, ??, ??, ??, ??, ??, ??
                                                   fold', 659, 662, used: 659, 662
flavor_symbol (label), ??, ??, 211, used:
                                                   fold2, 664, 663, used: 664, 664, 664, 9, 9, 10, 12,
flavor_to_string, ??, ??, ??, ??, ??, ??, ??, ??,
                                                           38, ??, ??, ??, ??
        fold2 (type), 665, 663, used: 663
        fold2\_rev, 663, used: 664
        ??, 610, 210, used: ??, ??, ??, 591, 62,
                                                   fold3, 664, 663, used: 664, 664, 10, 10, 12
```

```
fold3_rev, 664, used: 664
                                                        FORTRAN, ??, ??, used: ??, 297
fold_left, 7, 9, 10, 12, 14, 5, used: ??, ??, ??,
                                                        FORTRAN (camlyacc token), 298, used: 300,
        736, 627, 627, 627, 627, 627, 630, 630,
        631, 671, 619, 615, 663, 664, 664, 664,
                                                        Fortran77 (module), 448, used:
                                                        Fortran_Anti, 292, 285, used: ??, 300
        676, 676, 677, 677, 677, 677, 678,
        678, 678, 678, 679, 680, 684, 684, 708,
                                                        fortran\_digit, 642, used: 643
        708, 709, 695, 14, 22, 22, 24, 24, 24, 26,
                                                        fortran\_digit\_alpha, 643, used: 643
        26, 26, 26, 26, 27, 27, 38, 39, 50, 249, 249,
                                                        fortran_empty, 642, used: 643
        249, 249, ??, ??, ??, ??, 590, 590, 591,
                                                        Fortran_Majorana (module), 448, used: ??, ??,
        309, 309, 310, 312, 312, 321, 321, 322,
                                                                ??, 611, ??, ??, ??, ??
        330, 330, 336, 345, 345, 345, 355, 60, 65,
        281, ??, ??, ??, ??, ??, ??, ??, ??,
                                                        fortran_questionmark, 643, used: 643
        ??, ??, ??, ??, ??, ??, ??, ??, 603
                                                        Fortran\_Target (sig), 378, used: 378
fold_left2, 627, 624, used: fold_lefti, ??, used: ??, ??
                                                        fortran_underscore, 643, used: 643
                                                        fortran_underscore_alpha, 643, used: 643
                                                        fortran_valid, 643, used: 643
fold_left_internal, 7, 9, 10, 12, 14, 5, used: ??
fold_nodes, 33, 37, 31, used: ??, ??, ??
                                                        for_all, 7, 9, 10, 12, 13, 33, 34, 35, 35, ??, 5,
fold_rev, 664, used: 664, 664
                                                                30, 32, used: 628, 631, 682, 683, 695, 13,
fold_right, 7, 9, 10, 12, 14, 5, used: 627, 630,
                                                                26, 35, 35, 37, 249, 249, 318, 319, 319,
        668, 671, 671, 680, 690, 704, 695, 9, 9, 10,
                                                                320, 324, 324, 324, 346, 60, 64, 64, 64, 65,
                                                                278, ??, ??, ??
        11, 13, 14, 14, 15, 35, 586, 587, 590, 590,
        592, 217, 296, 321, 322, 336, 355, 355,
                                                        for_all_pairs, 694, used: 694, 694, 696
        358, 61, 62, 281, 281, 281, 281, 282, 282,
                                                        four_gluon, ??, 609, used: ??, 609
        ??, ??, ??, ??, ??
                                                        fprintf, 446, 444, used: ??, ??, ??, ??, ??, ??,
                                                                ??, ??, ??, ??, 446, 446, 447, 447,
fold_right2, 627, 624, used: ??
                                                                447, 617, 618, 618, 711, 712, 703, 704,
fold_right_internal, 7, 9, 10, 12, 14, 5, used: ??
forest, 33, 38, ??, ??, 32, 124, ??, used: 598,
                                                                704, 704, 704, 705, 706, 706, 706, 592,
        599, 605, 606
                                                                593, 593, 593, 593, 593, 593, 593,
Forest (module), 34, 30, used: ??, ??
                                                                593, 594, 594, 594, 595, 217, 310, ??, ??,
                                                                ??, 604
Forest (sig), 33, 29, used: 34, 34, 35, 40, 30, 32,
                                                        Fraction, 359, used: ??, 362
        33
                                                        free, 342, 342, 333, used: 348
forest', ??, used: ??
Forest_Grader (sig), 34, 33, used: 33
                                                        fresh_summation, 342, 343, 333, used: 343, 352
                                                        FS0_HHWW, ??, ??, used: ??, ??
forest_memoized, 33, 39, 32, used: ??
forest_sans_color, 598, used: 600, 600, 600
                                                        FS0_HHZZ, ??, ??, used: ??, ??
                                                        FS1_HHWW, ??, ??, used: ??, ??
formatter (field), 444, used: 444, 444, 445, 445,
                                                        FS1_HHZZ, ??, ??, used: ??, ??
        446, 446, 446, 446
formatter (type), 444, 444, used: ??, ??, ??, ??,
                                                        FSet (module), 281, used: 281, 281, 281, 281
        444, 214, ??, ??, ??, ??, ??, ??, 444, 444,
                                                        fspec\_of\_gen, ??, used: ??
                                                        fspec\_of\_iso, ??, used: ??
        444, 444, 444, 444, 417, 214, 422,
        422, 423, 378
                                                        fspec\_of\_kk2, ??, used: ??
                                                        fspec\_of\_kkmode, ??, used: ??
formatter_of_buffer, 446, 444, used: ??, ??,
                                                        FS_H4, ??, ??, used: ??, ??
        446, 447
                                                        fs_higgs4, ??, ??, used: ??, ??
formatter_of_formatter, 444, used: 445, 445
                                                        FT0_AAAA, ??, ??, used: ??, ??
formatter_of_out_channel, 445, 444, used: 445,
                                                        FT0\_AAAZ, ??, ??, used: ??, ??
        447, 447, 447
                                                        FT0\_AAWW0, ??, ??, used: ??, ??
format_and_compare, 446, used: 447, 447, 447,
        448
                                                        FT0\_AAWW1, ??, ??, used: ??, ??
                                                        FT0\_AAZZ, ??, ??, used: ??, ??
format_flavor, 592, used: 592
Format_Fortran (module), 444, 444, used: 422,
                                                        FT0\_AZWW0, ??, ??, used: ??, ??
                                                        FT0_AZWW1, ??, ??, used: ??, ??
        422, 423, 378
                                                        FT0_AZZZ, ??, ??, used: ??, ??
format_generation, ??, used: ??, ??
                                                        FT0_WWWW0, ??, ??, used: ??, ??
FT0_WWWW2, ??, ??, used: ??, ??
format\_maps, 591, used:
format\_mismatch, 312, used: 312
                                                        FT0_ZZWW0, ??, ??, used: ??, ??
format_p, 598, used: 598, 598, 600, 600, 600
                                                        FT0_ZZWW1, ??, ??, used: ??, ??
Fortran, 292, 293, 285, 286, used: ??, 300, 300
                                                        FT0\_ZZZZ, ??, ??, used: ??, ??
Fortran (module), 572, ??, 423, 378, 572, ??,
        448, used: 606, ??, 607, ??, ??, ??, ??,
                                                        FT1\_AAAA, ??, ??, used: ??, ??
                                                        FT1_AAAZ, ??, ??, used: ??, ??
        FT1_AAWW0, ??, ??, used: ??, ??
        ??, ??
                                                        FT1\_AAWW1, ??, ??, used: ??, ??
```

```
FT1\_AAZZ, ??, ??, used: ??, ??
                                      Fusion (module), 126, 122, used: 589, 596, 597,
FT1\_AZWW0, ??, ??, used: ??, ??
                                           603, 606, 606, 247, 589
FT1\_AZWW1, ??, ??, used: ??, ??
                                           ??, ??, ??, ??, ??, ??, 124, 125, ??, ??,
                                     fusions,
FT1_AZZZ, ??, ??, used: ??, ??
                                           used: ??, ??, ??, 603
FT1_WWWW0, ??, ??, used: ??, ??
                                     fusions (field), ??, ??, ??, used: ??, ??, ??, ??,
FT1_WWWW2, ??, ??, used: ??, ??
                                           ??, ??, ??
FT1\_ZZWW0, ??, ??, used: ??, ?? FT1\_ZZWW1, ??, ??, used: ??, ??
                                      Fusions (module), ??, 65, ??, used: ??, 65, 66,
                                            FT1_ZZZZ, ??, ??, used: ??, ??
                                           FT2\_AAAA, ??, ??, used: ??, ??
                                           ??, ??, ??, ??, ??, 608
                                      Fusions (sig), ??, ??, used: ??, ??
FT2\_AAAZ, ??, ??, used: ??, ??
                                     fusion\_dag, ??, ??, used:
FT2\_AAWW0, ??, ??, used: ??, ??
FT2_AAWW1, ??, ??, used: ??, ??
                                     fusion_dag (field), ??, ??, used: ??, ??, ??, ??,
FT2_AAZZ, ??, ??, used: ??, ??
                                           ??, ??, ??, ??, ??, ??
FT2\_AZWW0, ??, ??, used: ??, ??
                                     fusion\_name, 422, used:
FT2\_AZWW1, ??, ??, used: ??, ??
                                     fusion_tower, ??, used: ??, ??
FT2\_AZZZ, ??, ??, used: ??, ??
                                     fusion_tower (field), ??, ??, used: ??, ??, ??, ??
FT2\_WWWW0, ??, ??, used: ??, ??
                                     fusion\_tower', ??, used: ??, ??
FT2_WWWW2, ??, ??, used: ??, ??
                                     Fusion_vintage (module), ??, ??, used: 606
                                     f_{-aux\_top} (type), ??, ??, ??, ??, used: ??,
FT2_ZZWW0, ??, ??, used: ??, ??
FT2_ZZWW1, ??, ??, used: ??, ??
                                           ??, ??, ??, ??
FT2\_ZZZZ, ??, ??, used: ??, ??
                                      g (type), ??, ??, used: ??, ??
                                      G, 316, 318, ??, ??, ??, ??, ??, ??, ??, ??,
Fudged, 180, used: ??, ??, ??, ??, ??, ??,
                                           ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                           ??, ??, ??, ??, 607, used: 318, ??, ??,
Function, 291, 285, used: 292
functions, 330, 337, 329, 332, used: 330, 337
                                           fuse, 701, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                           ??, ??, ??, ??, ??, 609, 210, 699, ??,
                                           422, 378, 126, ??, used: ??, ??, ??, ??,
                                           66, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                           ??. ??. ??. ??. ??. ??. ??. ??. 609
                                           9, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                           fuse 2,
     ??, 609, 210, ??, used: 9, 13, ??, ??, ??,
     ??, ??, ??, ??, ??, ??, 609
                                           fuse2 (type), 184, used: 184
fuse3, 10, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                           ??, 609, 210, ??, used: 11, 13, ??, ??,
                                           ??, ??, ??, ??, 607, 608, 609, 609,
     609, 609, 609, 609
     G \text{ (module)}, 34, 35, 35, 35, 36, ??, 32, used:
     ??, ??, ??, ??, ??, ??, 609
                                           34, 36, 36, 36, 36, 36, 36, 39, 39, 32, 33
fuse3 (type), 184, used: 184
                                      G2, 293, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                           ??, ??, ??, ??, ??, ??, used: 293, ??,
fusen, ??, used: ??
                                           fusen (type), 184, used: 184, 422, 378
fuse\_c\_wf, ??, used: ??, ??
                                           ??, ??, ??, ??, ??
                                      G22, ??, used: ??
fuse_n, 14, 15, used: 15
                                      G3, ??, 607, used: ??, ??, 608, 609
fuse\_rhs, ??, used: ??, ??
                                      G4, ??, 607, used: ??, ??, 609, 609
fuse_trees, ??, used: ??
fusion (type), ??, ??, ??, ??, ??, ??, 123, 125,
                                      G5\_AWW, ??, ??, ??, ??, used: ??, ??, ??,
     ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                           ??, ??
     ??, ??, ??, 123, 124, 125, 126, ??, ??, ??,
                                      G5_ZWW, ??, ??, ??, ??, used: ??, ??, ??,
                                            ??, ??
     ??
```

```
??, ??, ??, ??, ??, ??, ??, ??, used:
                                             ??, ??, ??, ??, ??, ??, 212, 212, 212,
      212, 213, 213
                                       Gauge_Gauge_Gauge, 180, used: ??, ??, ??.
      ??, 609
      Gauge\_Gauge\_i, 180, used:
      gauge_higgs, ??, ??, ??, ??, ??, ??, ??, ??,
      ??, ??, ??, ??, ??, ??, ??, ??, used:
      ??, ??, ??, ??, ??, ??, ??
      gauge_higgs4, ??, ??, ??, ??, ??, ??, ??, ??,
      ??, ??, ??, ??, ??, ??, ??, ??, used:
      ??, ??, ??, ??, ??, ??, ??
      gauge\_higgs4\_GaGaCC, ??, ??, used: ??, ??
      gauge\_higgs4\_GaWPC, ??, ??, used: ??, ??
      gauge_higgs4_GaWSC, ??, ??, used: ??, ??
      gauge_higgs4_WWCC, ??, ??, used: ??, ??
      gauge_higgs4_WWPP, ??, ??, used: ??, ??
     ??, ??, ??, ??, ??
                                       gauge_higgs4_WWSS, ??, ??, used: ??, ??
gamma, 214, 215, 214, used: 217
                                       gauge\_higgs4\_ZGaCC, ??, ??, used: ??, ??
Gamma, 348, 349, 334, 364, used: 350, 352
                                       gauge_higgs4_ZWPC, ??, ??, used: ??, ??
                                       gauge_higgs4_ZWSC, ??, ??, used: ??, ??
gamma0, 214, 215, 215, 219, 220, 220, 213,
                                       gauge_higgs4_ZZCC, ??, ??, used: ??, ??
      used: 215, 218, 218, 219, 219
                                       gauge\_higgs4\_ZZPP, ??, ??, used: ??, ??
gamma1, 214, 215, 215, 219, 220, 221, 213,
                                       gauge_higgs4_ZZSS, ??, ??, used: ??, ??
      used: 215, 218, 218, 219, 220
                                       gauge_higgs_GaCC, ??, ??, used: ??, ??
gamma2, 214, 215, 215, 220, 220, 221, 213,
      used: 215, 218, 218, 219, 219, 220
                                       gauge\_higgs\_GlGlP,
                                                     ??, used: ??
gamma3, 214, 215, 215, 220, 220, 221, 213,
                                       gauge_higgs_GlGlS, ??, used: ??
                                       gauge_higgs_gold, ??, used: ??
      used: 215, 218, 218, 219, 220
       214, 215, 215, 220, 220, 221, 213,
                                       gauge_higgs_gold4, ??, used: ??
     used: 215, 218, 218, 219
                                       qauge\_higgs\_WPC,
                                                     ??, ??, used: ??, ??
                                       gauge\_higgs\_WSC, ??, ??, used: ??, ??
Gamma5, 348, 349, 334, 364, used: 350, 352
                                       gauge_higgs_WWS, ??, ??, used: ??, ??
gamma5\_def, 218, used: 219
                                                    ??, ??, used: ??, ??
gammas (field), 364, used:
                                       gauge_higgs_ZCC,
gauge (type), ??, ??, ??, ??, ??, ??, ??, ??,
                                       gauge\_higgs\_ZSP,
                                                    ??, ??, used: ??, ??
      gauge_higgs_ZZS, ??, ??, used: ??, ??
      ??, ??, ??, used: ??, ??, ??
                                       gauge\_sfermion4,
                                                    ??, ??, ??, used: ??, ??, ??
      607, 210, used: ??, ??, ??, ??, ??, 210,
                                       gauge_sfermion4',
                                       gauge_sfermion4", ??, ??, vsed: ??, ??, ??
      211, 211, ??, ??, ??
                                       gauge_squark4, ??, ??, ??, used: ??, ??, ??
??, ??, ??, ??, ??, 212, 246, used: 601,
                                       gauge_squark4', ??, ??, ??, used: ??, ??, ??
                                       gauge_squark4", ??, ??, ??, used: ??, ??, ?? gauge_symbol, ??, ??, ??, ??, ??, ??, ??, ??, ??,
      ??, ??, ??, ??, ??
Gauge (module), 250, used:
                                             Gauge (sig), 212, used: ??, 212, 212, 213, 250,
                                             ??, ??, ??, ??, ??, ??, ??, ??
                                             ??, 611, 211, used: ??, ??, ??, ??
gauge4, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                       gauge_symbol (label), ??, ??, 211, used:
      Gauss, 55, 61, 55, used: 56, 61
      ??, 609, used: ??, ??, ??, ??, ??, ??, ??,
                                       GAUSS, ??, ??, used: ??, 57
      GAUSS (camlyacc token), 57, used: 58
      ??, ??, ??, ??, ??, ??, 609
                                       Gauss_not, 55, 61, 55, used: 56
gauge_boson, ??, ??, ??, ??, ??, ??, ??, ??,
                                       GBBG, 182, used: ??, ??, ??, ??, ??, ??, ??
      ??, ??, ??, ??, ??, ??, 212, used: ??,
                                       GBG, 180, used: ??, ??, ??, ??, ??, ??, ??
      gen (type), ??, ??, used: ??, ??, ??
      ??, ??, ??, ??
                                       Gen\theta, ??, used: ??
                                            ??, used: ??
gauge_boson (type), ??, ??, ??, ??, ??, ??, ??,
                                       Gen1,
      ??, ??, ??, ??, ??, ??, ??, ??, ??, <mark>212</mark>,
                                            ??, used: ??
                                       Gen2,
      used: ??, ??, ??, ??, ??, ??, ??, ??,
```

```
gluon\_gauge\_squark'', ??, ??, ??, used: ??, ??,
generation, ??, ??, ??, ??, ??, ??, ??, ??, ??,
       ??, ??, ??, ??, ??, used: ??, ??, ??,
       gluon_w_squark, ??, ??, used: ??, ??, ??
                                             gluon_w_squark', ??, ??, ??, used: ??, ??, ??
       ??, ??
                                             gluon_w_squark", ??, ??, ??, used: ??, ??, ??
generation (type), ??, used: ??, ??
generation', ??, ??, ??, ??, ??, ??, ??, ??,
                                             Gl_K1, ??, used: ??, ??, ??, ??, ??
       ??, ??, ??, ??, ??, ??, used: ??, ??,
                                             Gl_K2, ??, used: ??, ??, ??, ??, ??
       goldstone, ??, ??, ??, ??, ??, ??, ??, ??, ??,
       ??, ??, ??
                                                    ??, ??, ??, ??, ??, ??, ??, ??, ??,
generations, ??, 607, used: ??, ??, ??, ??, 607,
       607, 607, 607
                                                    608, 210, 212, used: ??, ??, ??, ??, ??,
generations_pairs, 607, used: 609
                                                    ??, ??
generations_quadruples, 607, used: 609, 609
                                             goldstone (label), ??, ??, 211, used:
                                             goldstone (type), 212, used: 212, 212
generations_triples, 607, used: 608, 609, 609
get, 315, 315, used: ??, ??, 640, 641, 641, 641,
                                             Goldstone, 212, used:
                                             goldstone4, ??, used: ??
       642, 642, 626, 630, 636, 636, 636, 636,
       637, 637, 731, 731, 731, 731, 732, 732,
                                             goldstone_charg_neutr, ??, used: ??
       732, 733, 733, 681, 686, 686, 687, 708,
                                             qoldstone_neutr, ??, used: ??
                                             goldstone_neutr', ??, used: ??
       708, 709, 709, 710, 710, 710, 695, 695,
                                             goldstone_neutr", ??, used: ??
       695, 696, 696, 9, 10, 14, 15, 587, 215, 216,
      216, 216, 216, 216, 216, 217, 217, 217,
                                             goldstone_sfermion, ??, used: ??
                                             goldstone_sfermion',
goldstone_sfermion'',
       217, 217, 218, 218, 290, 290, 315, 317,
                                                              ??. used: ??
      317, 318, 323, ??, ??, ??, ??, ??, 352,
                                                               ??, used: ??
                                             goldstone_sfermion''', ??, used: ??
      352, 279, 279, 279, ??, ??, 597, 601
get_kk2, ??, used: ??, ??, ??, ??, ??
                                             goldstone_squark, ??, used: ??
GF (module), ??, used: ??
                                             goldstone_squark', ??, used: ??
GG, ??, ??, ??, used:
                                             goldstone\_squark\_a, ??, used: ??
                                             ghh, ??, used: ??
Ghost, 353, used: 353, 354
??, ??, ??, ??, ??, ??, ??, ??, used:
                                                    ??, ??, ??, ??, ??, ??
       graded (type), 7, 9, 10, 13, 14, 6, used: 7, 6
       Graded (module), 39, 33, used: ??
       Graded (sig), 39, 33, used: 33
       Graded_Forest (sig), 34, 32, used: 34, 36, 39, 33,
       Graded\_Map \text{ (module)}, \quad \mathbf{36}, \text{ used: } \mathbf{39}
       Graded_Map (sig), 35, used: 36, 36, 39
      Graded\_Map\_Maker (sig), 36, used: 36
       Graded_Ord (sig), 34, 32, used: 34, 34, 36, 36,
      39, 32, 32
       graded_sym_power, 7, 9, 11, 13, 15, 6, used:
       graded_sym_power_fold, 7, 9, 11, 13, 15, 6,
       used: 9, 11, 13, 15, ??
       ??, ??
                                             Grader (sig), 34, 32, used: 34, 35, 32, 33
global_verbose, ??, used: ??, ??, ??, ??
                                             Grade_Forest (module), 35, 33, used: 40
Gluino, ??, ??, ??, used: ??, ??, ??, ??, ??, ??,
                                             graph (type), 707, 700, used: 709, 700, 700
       graph_of_tree, 708, 700, used:
       ??
                                             Grav, ??, ??, 180, used: ??, ??, ??, ??, ??, ??,
          ??, used: ??
                                                    gluon2\_lq2,
gluon2\_lq2', ??, used: ??
                                                    ??, ??, ??
                                             Gravbar,
gluon2\_phi, 609, used: 609
                                                     180, used: ??, ??, ??, ??, ??, ??, ??,
gluon2_squark2, ??, ??, ??, used: ??, ??, ??
                                                    ??, ??, ??, ??, ??
gluon2_squark2', ??, ??, used: ??, ??
                                             gravitino, ??, ??, ??, ??, ??, ??, ??, used:
qluon3_phi, 609, used: 609
                                                    ??, ??, ??
gluon4, ??, used:
                                             gravitino_coup, ??, used: ??
gluon_gauge_squark, ??, ??, ??, used: ??, ??, ??
                                             gravitino_gauge, ??, used: ??
gluon_gauge_squark', ??, ??, ??, used: ??, ??,
                                             gravitino_gaugino_3, ??, used: ??
                                             Graviton_Scalar_Scalar, 180, used: ??, ??
                                             Graviton_Spinor_Spinor, 180, used: ??, ??
```

```
Graviton_Vector_Vector, 180, used: ??, ??
                                                      G_{-}CICIH2, ??, used: ??
gravity_currents, ??, ??, used: ??, ??
                                                      G_{-}CICIP, ??, ??, used: ??, ??
gravity_gauge, ??, ??, used: ??, ??
                                                      G_{-}CICIS, ??, ??, used: ??, ??
gravity_higgs, ??, ??, used: ??, ??
                                                      G_{-}CPC, ??, ??, used: ??, ??
GravTest (module), ??, ??, used: ??
                                                      G_{-}CSC, ??, ??, used: ??, ??
                                                      G_{-}CWN, ??, ??, used: ??, ??, ??
Grino, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                      G_{-}CZC, ??, ??, qued: ??, ??, ??
        ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                      G_{-}Ebb, ??, ??, ??, used: ??, ??, ??
group (type), 293, 286, used: 294, 286, 286, 286
group_of_string, 294, 286, used: 309
                                                      G_{-}EGaGa, ??, ??, ??, used: ??, ??, ??
                                                      G\_EGaZ, ??, ??, used: ??, ??, ??
group_rep_of_tokens, 309, used: 309, 309
                                                      G_{-}EGlGl, ??, ??, used: ??, ??, ??
group_to_string, 294, 286, used: 294, 307
Groves (module), ??, ??, used: ??
                                                      G_{-}EtaAA, ??, used: ??
G_{-}EtaAZ, ??, used: ??
                                                      G\_EtaGG', ??, used: ??
        ??, ??, ??, ??, ??, ??, ??, used: ??,
                                                      G\_EtaWW, ??, used: ?? G\_EtaZZ, ??, used: ??
        G_-Etht, ??, ??, used: ??, ??, ??
                                                      G_-Ethth, ??, ??, used:
        ??, ??, ??, ??, ??, ??, ??, ??
G\_AASFSF, ??, used: ??
                                                      G_{-}Ett, ??, ??, used: ??, ??, ??
G_{-}aaww, ??, used: ??, ??
                                                      G_{-}FDDSHH, ??, used: ??
G_{-}AAWW, ??, ??, ??, ??, ??, ??, ??, ??,
                                                      G\_FDDSHH\_CF, ??, used: ??
        ??, ??, ??, ??, ??, ??, ??, used: ??,
                                                      G_{-}FDDSWW, ??, used: ??
        G_FDDSWW_CF, ??, used: ??
        ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                      G_{-}FDDSZZ, ??, used: ??
G\_AGOSFSF, ??, used: ??, ?? G\_AGSNSL, ??, used: ??
                                                      G\_FDDSZZ\_CF, ??, used: ??
                                                      G\_FFHH, ??, used: ??
G\_AGSUSD, ??, used: ??
                                                      G\_FFHH\_CF, ??, used: ??
G\_AHpHm, ??, used: ??
                                                      G\_FFWW, ??, used: ??
G\_AHPsip, ??, ??, used: ??, ?? G\_AHTHT, ??, ??, used: ??, ??
                                                      G\_FFWW\_CF, ??, used: ??
                                                      G_{-}FFZZ, ??, used: ??
G_-AHTHTH, ??, ??, used: ??, ??
                                                      G_FFZZ_CF, ??, used: ??
                                                      G_{-}FHH, ??, ??, used: ??, ??
G\_AHTT, ??, ??, used: ??, ??
G\_ASFSF, ??, used: ??
                                                      G_FHH_CF, ??, ??, used: ??, ??
G_{-}aww, ??, used: ??, ??
                                                      G\_FSHH, ??, used: ??
G_{-}AZWW, ??, ??, ??, ??, ??, ??, ??, ??,
                                                      G\_FSWW, ??, used: ??
                                                      G_{-}FSZZ, ??, used: ?? G_{-}FVHH, ??, used: ??
        ??, ??, ??, ??, ??, ??, ??, used: ??,
        G_FVHH_CF, ??, used: ??
        ??, ??, ??, ??, ??, ??, ??
                                                      G_{-}FVWW, ??, used: ??
G_{-}a_{-}lep, ??, used: ??, ??
g_a = quark, ??, used: ??
                                                      G_FVWW_CF, ??, used: ??
G_{-}a_{-}quark, ??, used: ??, ??
                                                      G_{-}FVZZ, ??, used: ??
G_{-}CAC, ??, used: ??
                                                      G_FVZZ_CF, ??, used: ??
G_FWW, ??, ??, ??, used: ??, ??, ??, ??, ??
                                                      G_{-}FWW_{-}CF, ??, ??, used: ??, ?? G_{-}FWW_{-}T, ??, ??, ??, used: ??, ??, ??
        ??, ??, ??, ??, ??, ??, ??, ??, used:
        G_{-}FZZ, ??, ??, ??, used: ??, ??, ??, ??, ??
        G\_FZZ\_CF, ??, ??, used: ??, ?? G\_FZZ\_T, ??, ??, ??, used: ??, ??, ??
        ??, ??, ??
G_{-}CCQ, ??, ??, ??, ??, ??, ??, ??, ??, ??,
        ??, used: ??, ??, ??, ??, ??, ??, ??
                                                      G_{-}G0G0SFSF, ??, used: ??, ??
G_{-}CCtop, ??, ??, used: ??, ??
                                                      G_{-}GG4, ??, used: ??
G_-CC_-heavy, ??, ??, used: ??, ??
                                                      G_{-}GGSFSF, ??, used: ??, ??
                                                      G\_GGSNSL, ??, used: ?? G\_GGSUSD, ??, used: ??
G_{-}CC_{-}W, ??, ??, used: ??, ??
G_{-}CC_{-}WH, ??, ??, used: ??, ??
G\_CGC, ??, used: ??
                                                      G_-GH, ??, used: ??, ??
G_{-}CGN, ??, used: ??
                                                      G_-GH_4, ??, used: ??
G_{-}CH1C, ??, used: ??
                                                      G_{-}GH_{4}_{-}GaGaCC, ??, ??, used: ??, ??
G_-CH2C, ??, used: ??
                                                      G_-GH_4_-GaWPC, ??, ??, used: ??, ??
G_-CHN, ??, used: ??
                                                      G_-GH_4_-GaWSC, ??, ??, used: ??, ??
                                                      G_{-}GH4_{-}WWCC, ??, ??, used: ??, ?? G_{-}GH4_{-}WWPP, ??, ??, used: ??, ?? G_{-}GH4_{-}WWSS, ??, ??, used: ??, ??
G_{-}CICIA, ??, used: ??
G_{-}CICIG, ??, used: ??
G\_CICIH1, ??, used: ??
```

```
G_-GH_4_-ZGaCC, ??, ??, used: ??, ??
                                                          G_-Gr_4_Z_-H_2, ??, used: ??
G_-GH_4_-ZWPC, ??, ??, used: ??, ??
                                                          G_-Gr_4_Z_-H_3, ??, used: ??
G_-GH_4_-ZWSC, ??, ??, used: ??, ??
                                                          G_{-}Grav, ??, ??, ??, used: ??, ??, ??, ??, ??,
G_-GH_4_-ZZCC, ??, ??, used: ??, ??
                                                                  ??, ??, ??, ??
G_-GH_4_-ZZPP, ??, ??, used: ??, ??
                                                          G_{-}GravGl, ??, used: ??
                                                         G\_Grav\_D, ??, used: ??
G_-GH_4_-ZZSS, ??, ??, used: ??, ??
                                                          G\_Grav\_Dc, ??, used: ??
G_{-}GHGo, ??, used: ??
G_{-}GHGo4, ??, used: ??
                                                          G\_Grav\_L, ??, used: ??
G_-GH_-GaCC, ??, ??, used: ??, ??
                                                          G\_Grav\_Lc, ??, used: ??
G_-GH_-WPC, ??, ??, used: ??, ??
                                                          G_{-}Grav_{-}N, ??, used: ??
G_-GH_-WSC, ??, ??, used: ??, ??
                                                          G_-Grav_-U, ??, used: ??
                                                          G_-Grav_-Uc, ??, used: ??
G_-GH_-WWS, ??, ??, used: ??, ??
G_{-}GH_{-}ZCC, ??, ??, used: ??, ?? G_{-}GH_{-}ZSP, ??, ??, used: ??, ??
                                                          G_-Gr_-A_-Neu, ??, used: ??
                                                          G_-Gr_-Ch, ??, used: ??
G_-GH_-ZZS, ??, ??, used: ??, ??
                                                          G\_Gr\_H1\_Neu, ??, used: ??
                                                          G\_Gr\_H2\_Neu, ??, used: ??
G_{-}GLGLA, ??, used: ??
G_{-}GLGLH, ??, used: ??
                                                          G_-Gr_-H3_-Neu, ??, used: ??
G_{-}GLGLHH, ??, used: ??
                                                          G_-Gr_-H_-Ch, ??, used: ??
G_-GlGlLQLQ, ??, used: ??
                                                          G_-Gr_-Z_-Neu, ??, used: ??
G_{-}GlGlSQSQ, ??, ??, ??, used: ??, ??, ??
                                                          G_{-}GSUSD, ??, used: ??
G_{-}GlPSQSQ, ??, ??, ??, used: ??, ??, ??
                                                         G_-h111, ??, used: ?? G_-h1111, ??, used: ??
G_{-}GLUGLUA0, ??, used: ?? G_{-}GLUGLUH0, ??, used: ??
                                                          G_h1112, ??, used: ??
G_{-}GlWSUSD, ??, ??, ??, used: ??, ??, ?? G_{-}GlZSFSF, ??, ??, ??, used: ??, ??, ??
                                                          G_h1113, ??, used: ??
                                                          G_h112, ??, used: ??
G\_GoSFSF, ??, used: ??
                                                          G_h1122, ??, used: ??
G\_GoSNSL, ??, used: ??
                                                          G_h1123, ??, used: ??
                                                          G_h113, ??, used: ??
G_-Gr_4A_-Sd, ??, used: ??
G_-Gr_4A_-Sdc, ??, used: ?? G_-Gr_4A_-Sl, ??, used: ??
                                                          G_h1133, ??, used: ??
                                                          G_h1222, ??, used: ??
                                                          G_h1223, ??, used: ??
G_-Gr_4A_-Slc, ??, used: ??
G_-Gr_4A_-Su, ??, used: ??
                                                          G_h123, ??, used: ??
G_-Gr4A_-Suc, ??, used: ??
                                                          G_h1233, ??, used: ??
G_Gr_4Gl_Sd, ??, used: ??
                                                          G_h1333, ??, used: ??
G_-Gr_4Gl_-Sdc, ??, used: ??
                                                          G_h1bb, ??, used: ??, ??
G_-Gr_4Gl_-Su, ??, used: ??
                                                          G_h1bd, ??, used: ??
                                                          G\_h1bs, ??, used: ??
G_-Gr_4Gl_-Suc, ??, used: ??
                                                          G_h1cc, ??, used: ??, ??
G_Gr_4W_Sd, ??, used: ??
                                                          G_-h1ct, ??, used: ??, ??
G_-Gr_4W_-Sdc, ??, used: ??
G_-Gr_4W_-Sl, ??, used: ??
                                                          G_-h1cu, ??, used: ??, ??
G_-Gr_4W_-Slc, ??, used:
                                                          G_-h1db, ??, used: ??, ??
G_GGr_4W_Sn, ??, used: ??
                                                          G_h1dd, ??, used: ??, ??
G_{-}Gr_{4}W_{-}Snc, ??, used: ??
                                                          G_h1ds, ??, used: ??, ??
G_-Gr_4W_-Su, ??, used: ??
                                                          G_h1e1e1, ??, used: ??, ??
G\_Gr4W\_Suc, ??, used: ??
                                                          G_h1e1e2, ??, used: ??, ??
G_-Gr_4Z_-Sd, ??, used: ??
                                                          G_h1e1e3, ??, used: ??, ??
                                                          G_h1e2e1, ??, used: ??, ??
G_-Gr4Z_-Sdc, ??, used: ??
G_-Gr4Z_-Sl, ??, used: ??
                                                          G_h1e2e2, ??, used: ??, ??
G_-Gr_4Z_-Slc, ??, used: ??
                                                          G_h1e2e3, ??, used: ??, ??
G_-Gr_4Z_-Sn, ??, used: ??
                                                          G_h1e3e1,
                                                                      ??, used: ??, ??
                                                                      ??, used: ??, ??
G_-Gr_4Z_-Snc, ??, used: ??
                                                          G_h1e3e2,
                                                          G_h1e3e3, ??, used: ??, ??
G_Gr_4Z_Su, ??, used: ??
                                                         G\_H1GSNSL, ??, used: ?? G\_H1GSUSD, ??, used: ??
G_Gr_4Z_Suc, ??, used: ??
G_Gr_4_A_Ch, ??, used: ??
G_Gr_4_H_A, ??, used: ??
                                                          G_H1H1SFSF, ??, used: ??, ??
                                                          G_H1H2SFSF, ??, used: ??, ??
G_-Gr_4_H_Z, ??, used: ??
G_Gr_4_Neu, ??, used: ??
                                                          G_h1HpAWm, ??, used: ??
G_-Gr_4_W_H, ??, used: ??
                                                          G_h1HpAWmC, ??, used: ??
G_-Gr_4_-W_-Hc, ??, used: ??
                                                          G_-h1HpHm, ??, used: ??
G_-Gr_4_-Z_-Ch, ??, used: ??
                                                          G_-h1HpZWm, ??, used: ??
G_Gr_4_Z_H1, ??, used: ??
                                                          G_h1HpZWmC, ??, used: ??
```

```
G_h1sb, ??, used: ??, ??
                                                      G_{-}h332,
                                                               ??, used: ??
G_h1sd, ??, used: ??, ??
                                                      G_h333, ??, used: ??
G_{-}H1SFSF, ??, used: ??, ??
                                                      G_h3333, ??, used: ??
G_-h1ss, ??, used: ??, ??
                                                      G_{-}h3bb, ??, used: ??, ??
G_h1tc, ??, used: ??, ??
                                                      G_h3bd, ??, used: ??
G_h1tt, ??, used: ??, ??
                                                      G_h3bs, ??, used: ??
G_h1tu, ??, used: ??, ??
                                                      G_h3cc, ??, used: ??, ??
G_h1uc, ??, used: ??, ??
                                                      G_h3ct, ??, used: ??, ??
G_h1ut, ??, used: ??, ??
                                                      G_h3cu, ??, used: ??, ??
G_-h1uu, ??, used: ??, ??
                                                      G_h3db, ??, used: ??, ??
G_h1WpWm, ??, used: ??
                                                      G_h3dd, ??, used: ??, ??
G_h1ZZ, ??, used: ??
                                                      G_h3ds, ??, used: ??, ??
G_{-}h221, ??, used: ??
                                                      G_h3e1e1, ??, used: ??, ??
G_h222, ??, used: ??
                                                                 ??, used: ??, ??
                                                      G_h3e1e2,
                                                      G_h3e1e3, ??, used: ??, ??
G_h2222, ??, used: ??
G_h2223, ??, used: ??
                                                      G_h3e2e1, ??, used: ??, ??
G_h223, ??, used: ??
                                                      G_h3e2e2, ??, used: ??, ??
                                                      G_h3e2e3, ??, used: ??, ??
G_h2233, ??, used: ??
G_h2333, ??, used: ??
                                                      G_h3e3e1, ??, used: ??, ??
G_h 2bb, ??, used: ??, ??
                                                      G_h3e3e2, ??, used: ??, ??
G_h 2bd, ??, used: ??
                                                      G_h3e3e3, ??, used: ??, ??
                                                      G_-h3HpAWm, ??, used: ?? G_-h3HpAWmC, ??, used: ??
G_h 2bs, ??, used: ??
G_h2cc, ??, used: ??, ??
                                                      G_h3HpHm, ??, used: ??
G_h2ct, ??, used: ??, ??
G_h2cu, ??, used: ??, ??
                                                      G_h3HpZWm, ??, used: ??
G_h2db, ??, used: ??, ??
                                                      G_h3HpZWmC, ??, used: ??
                                                      G_h3sb, ??, used: ??
G_h2dd, ??, used: ??, ??
G_{-}h2ds, ??, used: ??, ??
                                                      G_{-}h3sd, ??, used: ??, ??
                                                      G_h3ss, ??, used: ??, ??
G_h2e1e1, ??, used: ??, ??
G_h2e1e2, ??, used: ??, ??
                                                      G_{-}h3tc, ??, used: ??, ??
G_h2e1e3, ??, used: ??, ??
                                                      G_-h3tt, ??, used: ??, ??
G_h2e2e1, ??, used: ??, ??
                                                      G_-h3tu, ??, used: ??, ??
G_h2e2e2, ??, used: ??, ??
                                                      G_h3uc, ??, used: ??, ??
G_h2e2e3, ??, used: ??, ??
                                                      G_h3ut, ??, used: ??, ??
                                                      G_h3uu, ??, used: ??, ??
G_h2e3e1, ??, used: ??, ??
G_h2e3e2, ??, used: ??, ??
                                                      G_h3WpWm, ??, used: ??
G_h2e3e3, ??, used: ??, ??
                                                      G_h 3ZZ, ??, used: ??
                                                      G_{-}H3_{-}SCC, ??, ??, used: ??, ?? G_{-}H3_{-}SPP, ??, ??, used: ??, ??
G_-H2GSNSL, ??, used: ?? G_-H2GSUSD, ??, used: ??
G_H2H2SFSF, ??, used: ??, ??
                                                      G_{-}H3_{-}SSS, ??, ??, used: ??, ??
G_h2HpAWm, ??, used: ??
                                                      G_h2HpAWmC, ??, used: ??
                                                              ??, ??, ??, used: ??, ??, ??, ??, ??, ??,
G_h2HpHm, ??, used: ??
                                                              ??, ??, ??, ??, ??, ??, ??, ??
G_h2HpZWm, ??, used: ??
                                                      G_H_{4-1}, ??, used: ??
                                                      G_H_{4-2}, ??, used: ??
G_h2HpZWmC, ??, used: ??
G_h 2sb, ??, used: ??, ??
                                                      G_H_{4-3}, ??, used: ??
                                                      G_{-}H_{4-4}, ??, used: ??
G_h 2sd, ??, used: ??, ??
G_{-}H2SFSF, ??, used: ??, ??
                                                      G_-H_{4-5}, ??, used: ??
                                                      G_{-}HAHAH, ??, ??, used: ??, ??
G_h2ss, ??, used: ??, ??
G_{-}h2tc, ??, used: ??, ??
                                                      G\_HAHZ, ??, ??, used: ??, ??
G_-h2tt, ??, used: ??, ??
                                                      G_{-}HASLSN, ??, used: ??
G_h2tu, ??, used: ??, ??
                                                      G\_HASUSD, ??, used: ??
                                                      G\_hbb, ??, used:
G_h 2uc, ??, used: ??, ??
G_h2ut, ??, used: ??, ??
                                                      G_h2uu, ??, used: ??, ??
                                                              ??, ??, ??, ??, used: ??, ??, ??, ??,
                                                              ??, ??, ??, ??, ??, ??, ??, ??, ??
G_h2WpWm, ??, used: ??
G_h2ZZ, ??, used: ??
                                                      G\_hbc,
                                                              ??, used: ??
??, used: ??, ??
                                                      G\_hbt,
        ??, ??, ??, ??, used: ??, ??, ??, ??, ??,
                                                              ??, used: ??
                                                      G_-hbu,
        ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                              ??, used: ??
                                                      G_{-}hcb,
G_h331, ??, used: ??
                                                      G\_hcc, ??, used:
```

```
G_{-}HHZZ, ??, ??, ??, ??, ??, ??, ??, ??, ??,
        ??, ??, ??, ??, used: ??, ??, ??, ??,
                                                              ??, ??, ??, ??, used: ??, ??, ??, ??, ??,
        ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                              G-hcd,
       ??, used: ??
                                                              ??
G_{-}hcs, ??, used: ??, ??
                                                      G_-HHZZH, ??, used: ??
G\_Hcs, ??, used:
                                                      G_{-}Hmm, ??, ??, ??, ??, used: ??, ??, ??,
G\_hdc, ??, used: ??
                                                              ??, ??
G_-hdt, ??, used: ??
                                                      G\_hmumu, ??, used:
G_-hdu, ??, used: ??, ??
                                                      G\_Hmumu, ??, used:
                                                      G\_Hmunu, ??, used:
G\_he1n1, ??, used: ??
                                                      G_-hn1e1, ??, used: ??
G\_he1n2, ??, used: ??
G\_he1n3, ??, used: ??
                                                      G_hn1e2, ??, used: ??
          ??, used: ??
                                                     G\_hn1e3,
                                                                ??, used: ??
G_{-}he2n1,
          ??, used: ??
                                                                ??, used: ??
G_he2n2,
                                                      G_hn2e1,
                                                      G_hn2e2, ??, used: ??
G\_he2n3,
          ??, used: ??
                                                      G_hn2e3, ??, used: ??
G_he3n1, ??, used: ??
G_he3n2, ??, used: ??
                                                      G_hn3e1, ??, used: ??
G_he3n3, ??, used: ??
                                                      G_hn3e2, ??, used: ??
G\_heavy\_HVV, ??, used: ??
                                                      G_hn3e3, ??, used: ??
G_heavy_HWW, ??, used: ??
                                                      G_{-}HpHm11, ??, used: ??
G\_heavy\_HZZ, ??, used:
                                                      G_{-}HpHm12, ??, used: ??
                                                      G_{-}HpHm13, ??, used: ??
G_{-}Hee, ??, used: ??
G_{-}HGaGa, ??, ??, ??, ??, ??, ??, ??, ??,
                                                      G\_HpHm22,
                                                                   ??, used: ??
        ??, ??, used: ??, ??, ??, ??, ??, ??, ??
                                                      G_{-}HpHm23, ??, used: ??
G_{-}HGaGa6, ??, used: ??
                                                      G_{-}HpHm33, ??, used: ??
G_{-}HGaGa_{-}anom, ??, ??, ??, used: ??, ??, ??
                                                      G_{-}HpHmAA, ??, used: ??
G_{-}HGaZ, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                                      G\_HpHmAZ, ??, used: ??
                                                      G_{-}HpHmHpHm, ??, used: ??
        ??, used: ??, ??, ??, ??, ??, ??, ??
G_-HGaZ6_-D, ??, used: ??
                                                      G_{-}HpHmWpWm, ??, used: ??
                                                      G\_HpHmZZ, ??, used: ??
G_-HGaZ6_-DP, ??, used: ??
G\_HGaZ6\_PB, ??, used: ??
                                                      G\_HPsi0AHAH, ??, ??, used: ??, ??
                                                     G\_HPsi0WHW, ??, ??, used: ??, ?? G\_HPsi0WW, ??, ??, used: ??, ??
G_{-}HGaZ_{-}anom, ??, ??, used: ??, ??, ??
G_{-}HGaZ_{-}u, ??, ??, used: ??, ??, ??
G_{-}Hgg, ??, ??, ??, ??, ??, ??, ??, ??, used:
                                                      G_{-}HPsi0ZAH, ??, ??, used: ??, ??
        ??, ??, ??, ??, ??, ??, ??
                                                      G_{-}HPsi0ZHAH, ??, ??, used: ??, ??
G_{-}HGo3, ??, used: ?? G_{-}HGo4, ??, used: ??
                                                      G_{-}HPsi0ZHZ, ??, ??, used: ??, ??
                                                      G_{-}HPsi0ZHZH, ??, ??, used: ??, ??
G\_HGSFSF, ??, used: ??, ??
                                                      G\_HPsi0ZZ, ??, ??, used: ??, ??
                                                      G_{-}HPsippWHW, ??, ??, used: ??, ??
G\_HGSNSL, ??, used: ??
G_{-}HGSUSD, ??, used: ??
                                                      G_{-}HPsippWHWH, ??, ??, used: ??, ??
                                                      G_{-}HPsippWW, ??, ??, used: ??, ??
G_{-}HH1SLSN, ??, used: ??
G_-HH1SUSD, ??, used: ??
                                                      G_{-}HPsipWA, ??, ??, used: ??, ??
G_{-}HH2SLSN, ??, used: ??
                                                      G_{-}HPsipWAH, ??, ??, used: ??, ??
G\_HH2SUSD, ??, used: ??
                                                      G_{-}HPsipWHA, ??, ??, used: ??, ??
G_{-}HHAA, ??, ??, used: ??, ??
                                                     G_HPsipWHAH, ??, ??, used: ??, ??
G_{-}HHAHZ, ??, ??, used: ??, ??
                                                      G_{-}HPsipWHZ, ??, ??, used: ??, ??
G\_HHS, ??, used: ??
                                                      G_{-}HPsipWHZH, ??, ??, used: ??, ??
G_{-}HHSFSF, ??, used: ??, ??
                                                      G_{-}HPsipWZ, ??, ??, used: ??, ??
G_-HHtht, ??, ??, used: ??, ??
                                                      G_{-}HPsipWZH, ??, ??, used: ??, ??
                                                      G_-Hqhq, ??, used: ??
G_{-}HHthth, ??, ??, ??, used: ??, ??, ??
G_{-}HHtt, ??, ??, used: ??, ??
                                                      G_{-}hsc, ??, used: ??, ??
G_-HHWHW, ??, ??, used: ??, ??
                                                      G\_HSF31, ??, used:
G_-hhWpWm, ??, used: ??
                                                     G_{-}HSF32, ??, used:
G_{-}HHWW, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                                      G\_HSF41, ??, used:
        ??, ??, ??, ??, used: ??, ??, ??, ??,
                                                      G\_HSF42, ??, used:
        G_{-}HSNSL, ??, ??, used: ??, ??, ??
                                                      G\_Hss, ??, used: ??
        ??, ??
G_{-}HHZHAH, ??, ??, used: ??, ??
                                                      G\_HSS, ??, used: ??
G_-HHZHZ, ??, ??, used: ??, ??
                                                      G\_hst, ??, used: ??
                                                      G_-hsu, ??, used: ??
G_-hhZZ, ??, used: ??
                                                      G_{-}HSUSD, ??, ??, used: ??, ??, ??
```

```
G_{-}HSWW, ??, used: ??
                                                   G_-NC_-h_-down, ??, ??, ??, ??, used: ??, ??,
G\_HSZZ, ??, used: ??
                                                           ??, ??, ??, ??, ??, ??, ??
G_{-}Htaunu, ??, used:
                                                   G_-NC_-h_-lepton, ??, ??, ??, ??, ??, used: ??,
G_-htautau, ??, used:
                                                           ??, ??, ??, ??, ??, ??, ??, ??
G_{-}Htautau, ??, ??, ??, ??, ??, ??, ??, ??,
                                                   G_NC_h_neutrino, ??, ??, ??, ??, used: ??,
       ??, ??, ??, ??, ??, used: ??, ??, ??,
                                                           ??, ??, ??, ??, ??, ??, ??, ??
                                                   \begin{array}{ll} G_{-}NC_{-}h_{-}top, & ??, \text{ used: } ?? \\ G_{-}NC_{-}h_{-}up, & ??, ??, ??, ??, ??, \text{ used: } ??, ??, \end{array}
       ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
G_-htb, ??, used: ??, ??
G\_Htb, ??, used:
                                                           ??, ??, ??, ??, ??, ??, ??
G_-htd, ??, used: ??
                                                   G_NC_lepton, ??, ??, ??, ??, ??, ??, ??, ??, ??.
                                                           G_{-}Htht, ??, ??, used: ??, ??, ??
G_-Hthth, ??, ??, used: ??, ??
                                                           used: ??, ??, ??, ??, ??, ??, ??, ??,
G\_hts, ??, used: ??
                                                           G_-htt, ??, used:
                                                           ??
       ??, ??, ??, ??, used: ??, ??, ??, ??, ??,
                                                   G_{-}NC_{-}neutrino, ??, ??, ??, ??, ??, ??, ??, ??,
       ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                           G_-hub, ??, used: ??
                                                           ??, used: ??, ??, ??, ??, ??, ??, ??,
G_-hud, ??, used: ??, ??
                                                           G\_hus, ??, used: ??
                                                           G_-HWHW, ??, ??, used: ??, ??
                                                           ??, ??
G_{-}HWHWH, ??, ??, used: ??, ??
                                                   G\_NC\_top, ??, used: ??
G_{-}NC_{-}up, ??, ??, ??, ??, ??, ??, ??, ??, ??,
       ??, ??, ??, ??, used: ??, ??, ??, ??, ??,
                                                           ??, ??, ??, ??, ??, ??, ??, ??, ??,
       ??, ??, ??, ??, ??, ??, ??, ??
                                                           used: ??, ??, ??, ??, ??, ??, ??, ??,
G_-HWW_-6_-D, ??, used: ??
                                                           G_-HWW_-6_-DP, ??, used: ??
                                                           G_-HWW_-anom, ??, ??, used: ??, ??, ??
                                                           ??
G_{-}HWW_{-}u, ??, ??, ??, used: ??, ??, ?? G_{-}HZHAH, ??, ??, used: ??, ??
                                                   G_-NC_-X, ??, used: ??
                                                   G\_NC\_X\_t, ??, used: ??
G_{-}HZHZ, ??, ??, used: ??, ??
                                                   G_-NC_-Y, ??, used: ??
G_-NC_-Y_-t, ??, used: ??
                                                   G\_NGC, ??, used: ??
       ??, ??, ??, ??, used: ??, ??, ??, ??, ??,
       ??, ??, ??, ??, ??, ??, ??, ??
                                                   G_NHC, ??, ??, ??, used: ??, ??, ??
                                                   G_-NLQC, ??, used: ??
G_HZZ6_D, ??, used: ??
                                                   G_-NNA, ??, used: ?? G_-NWC, ??, ??, ??, used: ??, ??, ??
G_HZZ6_DP, ??, used: ??
G\_HZZ6\_PB, ??, used: ??
G_HZZ6_V3, ??, used: ??
                                                   G_{-}NZN, ??, ??, used: ??, ??, ??
G_HZZ_anom, ??, ??, ??, used: ??, ??, ??
                                                   g_{-}over_{-}2_{-}costh, ??, ??, ??, ??, ??, ??, ??, ??,
G_{-}HZZ_{-}u, ??, ??, used: ??, ??, ??
                                                           ??, ??, ??, ??, ??, ??, ??, used: ??,
G_{-}LQ_{-}EC_{-}UC, ??, used: ??, ??, ??, ??, ??, ??,
                                                           ??, ??, ??, ??, ??, ??
                                                           ??, ??, ??, ??
                                                   G\_PGlLQLQ, ??, used: ??
G_LQ_GG, ??, used: ??
                                                   G_PNWW, ??, ??, ??, used: ??, ??, ?? G_PNZZ, ??, ??, ??, used: ??, ??, ?? G_PPA, ??, used: ??
G_LQ_NEU, ??, used: ??, ??
G_{-}LQ_{-}P, ??, used: ?? G_{-}LQ_{-}S, ??, used: ??
G_{LQ\_SSD}, ??, used: ??
                                                   G\_PPH, ??, used: ??
G_{-}LQ_{-}SSU, ??, used: ??
                                                   G\_PPHH, ??, used: ??
G_-NCHt, ??, used: ??
                                                   G\_PPLQLQ, ??, used: ??
                                                   G_{-}PPSFSF, ??, ??, ??, used: ??, ??, ??
G_NCH_D, ??, used: ??
G_-NCH_-N, ??, used: ??
                                                   G_{-}PPWW, ??, ??, used: ??, ??, ??
G_NCH_U, ??, used: ??
                                                   G_{-}Psi00AH, ??, ??, used: ??, ??
                                                   G_{-}Psi00ZH, ??, ??, used: ??, ??
G_{-}NC_{-}down, ??, ??, ??, ??, ??, ??, ??, ??,
       G_Psi00ZHAH, ??, ??, used: ??, ??
       used: ??, ??, ??, ??, ??, ??, ??, ??,
                                                   G_{-}Psi01AH, ??, ??, used: ??, ??
       G_{-}Psi01Z, ??, ??, used: ??, ??
       G_{-}Psi01ZH, ??, ??, used: ??, ??
                                                   G_{-}Psi0bb, ??, ??, used: ??, ??
       ??
                                                   G_-Psi\theta cc, ??, ??, used: ??, ??
G_-NC_-H, ??, used: ??
\begin{array}{ll} G\_NC\_heavy, & ??, ??, \text{ used: } ??, ?? \\ G\_NC\_h\_bot, & ??, \text{ used: } ?? \end{array}
                                                   G_-Psi0ppWHW, ??, ??, used: ??, ??
                                                   G_{-}Psi0ppWHWH, ??, ??, used: ??, ??
```

```
G_-Psi0ppWW, ??, ??, used: ??, ??
G_{-}Psi\partial pWA, ??, ??, used: ??, ??
G_-Psi0pWAH, ??, ??, used: ??, ??
G_-Psi0pWHA, ??, ??, used: ??, ??
G_-Psi0pWHAH, ??, ??, used: ??, ??
G_-Psi\partial p\,WHZ, ??, ??, used: ??, ?? G_-Psi\partial p\,WHZH, ??, ??, used: ??, ??
G_-Psi\theta pWZ, ??, ??, used: ??, ??
G_{-}Psi0pWZH, ??, ??, used: ??, ??
G_{-}Psi0tautau, ??, ??, used: ??, ??
G_-Psi\theta tt, ??, ??, used: ??, ??
G_{-}Psi0tth, ??, ??, used: ??, ??
G_{-}Psi\theta W, ??, ??, used: ??, ??
G_-Psi0WH, ??, ??, used: ??, ??
G_{-}Psi1bb, ??, ??, used: ??, ??
G_{-}Psi1cc, ??, ??, used: ??, ??
G_{-}Psi1HAH, ??, ??, used: ??, ??
G_Psi1HZ, ??, ??, used: ??, ??
G_{-}Psi1HZH, ??, ??, used: ??, ??
G_{-}Psi1tautau, ??, ??, used: ??, ??
G_{-}Psi1tt, ??, ??, used: ??, ??
G_{-}Psi1tth, ??, ??, used: ??, ?? G_{-}Psi1W, ??, ??, used: ??, ??
G_Psi1WH, ??, ??, used: ??, ??
G_{-}PsiAA, ??, used: ??
G_{-}PsiAHAH, ??, ??, used: ??, ??
G_{-}PsiAHW, ??, ??, used: ??, ??
G_{-}PsiAHWH, ??, ??, used: ??, ??
G_{-}PsiAZ, ??, used: ??
G_{-}PsiccAAH, ??, ??, used: ??, ??
G_{-}PsiccAZ, ??, ??, used: ??, ??
G_{-}PsiccAZH, ??, ??, used: ??, ??
G_PsiccZAH, ??, ??, used: ??, ??
G_{-}PsiccZZ, ??, ??, used: ??, ??
G_{-}PsiccZZH, ??, ??, used: ??, ??
G\_PsiGG, ??, used: ??
G_{-}PsiHW, ??, ??, used: ??, ??
G_{-}PsiHWH, ??, ??, used: ??, ??
G_-Psipbth, ??, ??, used: ??, ??
G_{-}Psipl3, ??, ??, used: ??, ??
G_{-}PsippAAH, ??, ??, used: ??, ??
G_PsippAZ, ??, ??, used: ??, ??
G_PsiPPW, ??, ??, used: ??, ??
G_PsippWA, ??, ??, used: ??, ??
G_{-}PsippWAH, ??, ??, used: ??, ?? G_{-}PsiPPWH, ??, ??, used: ??, ??
G_{-}PsippWHA, ??, ??, used: ??, ??
G_{-}PsippWHAH, ??, ??, used: ??, ??
G_{-}PsippWHW, ??, ??, used: ??, ??
G_{-}PsippWHWH, ??, ??, used: ??, ??
G_{-}PsippWHZ, ??, ??, used: ??, ??
G_{-}PsippWHZH, ??, ??, used: ??, ??
G_-PsippWW, ??, ??, used: ??, ?? G_-PsippWZ, ??, ??, used: ??, ??
G_{-}PsippWZH, ??, ??, used: ??, ??
G_PsippZAH, ??, ??, used: ??, ??
G_{-}PsippZHZH, ??, ??, used: ??, ??
G_{-}PsippZZ, ??, ??, used: ??, ??
G_{-}Psipq2, ??, ??, used: ??, ?? G_{-}Psipq3, ??, ??, used: ??, ??
G_PsiWHW, ??, ??, used: ??, ??
```

 $G_{-}PsiWW$, ??, ??, ??, used: ??, ??, ?? $G_{-}PsiZAH$, ??, ??, used: ??, ?? $G_{-}PsiZHAH$, ??, ??, used: ??, ?? $G_{-}PsiZHW$, ??, ??, used: ??, ?? $G_{-}PsiZHWH$, ??, ??, used: ??, ?? $G_{-}PsiZHZ$, ??, ??, used: ??, ?? $G_{-}PsiZHZH$, ??, ??, used: ??, ?? $G_{-}PsiZW$, ??, ??, used: ??, ?? G_PsiZWH , ??, ??, used: ??, ?? $G_{-}PsiZZ$, ??, ??, used: ??, ??, ?? $G_{-}PSNHH$, ??, ??, used: ??, ?? $G_{-}PSNWW$, ??, ??, used: ??, ?? $G_{-}PSNZZ$, ??, ??, used: ??, ?? $G_{-}PWW$, ??, ??, ??, used: ??, ??, ?? $G_{-}PWZ$, ??, ??, ??, used: ??, ??, ?? $G_{-}PZWW$, ??, ??, used: ??, ??, ?? $G_{-}s$, ??, used: ?? $G_{-}s2$, ??, used: ?? $G_{-}saa$, **607**, used: **609** G-saaa, **607**, used: $G_{-}SAA_{-}T$, ??, ??, used: ??, ?? $G_{-}SAZ_{-}T$, ??, ??, used: ??, ?? G_Sbb , ??, used: ?? G_Scc , ??, used: ?? G_SD4 , ??, used: ?? G_SD4_2 , ??, used: ?? G_SF4 , ??, used: G_SF4_3 , ??, used: G_SF4_4 , ??, used: G_SFSFP , ??, ??, used: ??, ?? $G_{-}SFSFS$, ??, ??, used: ??, ??, ??, ?? G_SGaGa , ??, used: ?? G_SGaZ , ??, used: ?? G_Sgg , ??, used: ?? G_SHH , ??, ??, used: ??, ?? G_SL2SQ2 , ??, used: ?? G_SL4 , ??, used: ?? G_SL4_2 , ??, used: ?? G_SLR_DbtR , ??, ??, ??, ??, used: ??, ??, ??, ??, ?? $G_{-}SLSNW$, ??, ??, used: ??, ??, ?? $G_{-}SL_{-}DbtL$, ??, ??, ??, ??, used: ??, ??, ??, ??, ?? $G_{-}SL_{-}DttL$, ??, ??, ??, ??, used: ??, ??, ??, ??, ?? $G_{-}SL_{-}DttR$, ??, ??, ??, ??, used: ??, ??, ??, ??, ?? G_Smm , ??, used: ?? G_SN2SL2_1 , ??, used: ?? $G_-SN2SL2_-2$, ??, used: ?? $G_{-}SN2SQ2$, ??, used: ?? G_SN_4 , ??, used: ?? $G_{-}SP_{-}ttH$, ??, ??, ??, used: ??, ??, ?? $G_{-}SR_{-}DttR$, ??, ??, ??, ??, used: ??, ??, ??, ??, ?? G_SS , ??, ??, ??, used: ??, ??, ?? $G_{-}SSWW$, ??, ??, ??, ??, used: ??, ??, ??, ?? $G_{-}SSZZ$, ??, ??, ??, used: ??, ??, ??, ?? $G_{-}Stautau$, ??, used: ?? G_strong , ??, ??, used: G_Stt , ??, used: ??

```
G_{-}SU2SD2, ??, used: ??
G_{-}SU_{4}, ??, used: ??
G_{-}SU_{4-}2, ??, used: ??
G_{-}SUSDSNSL, ??, used: ??
G\_SWS, ??, used: ??
G_{-}SWW, ??, ??, ??, used: ??, ??, ??
G_SWW_T, ??, ??, ??, used: ??, ??, ??
G_{-}SZZ, ??, ??, ??, used: ??, ??, ??
G_{-}SZZ_{-}T, ??, ??, used: ??, ??, ??
G\_S\_Sqrt, ??, used: ??
G_{-}TLR_{-}btW, ??, ??, ??, ??, used: ??, ??,
        ??, ??, ??
G_{-}TLR_{-}btWA, ??, ??, ??, ??, used: ??, ??,
        ??, ??, ??
G_{-}TLR_{-}btWZ, ??, ??, ??, ??, used: ??, ??,
        ??, ??, ??
G_{-}TNWW, ??, ??, ??, used: ??, ??, ??, ??
G_TNWW_CF, ??, ??, used: ??, ??
G_{-}TNZZ, ??, ??, ??, used: ??, ??, ??, ??, ??
G\_TNZZ\_CF, ??, ??, used: ??, ??
G_{-}TRL_{-}tbW, ??, ??, ??, ??, used: ??, ??,
        ??, ??, ??
G_{-}TRL_{-}tbWA, ??, ??, ??, ??, used: ??, ??,
        ??, ??, ??
G_{-}TRL_{-}tbWZ, ??, ??, ??, ??, used: ??, ??,
        ??, ??, ??
G_{-}TSNWW, ??, ??, used: ??, ??, ??, ??
G_{-}TSNWW_{-}CF, ??, ??, used: ??, ??
G_{-}TSNZZ, ??, ??, used: ??, ??, ??, ??
G_TSNZZ_CF, ??, ??, used: ??, ??
G_{-}TVA_{-}bbA, ??, ??, ??, ??, used: ??, ??,
        ??, ??, ??
G_{-}TVA_{-}bbWW, ??, ??, ??, ??, used: ??,
        ??, ??, ??, ??
G_{-}TVA_{-}bbZ, ??, ??, ??, ??, used: ??, ??,
        ??, ??, ??
G_{-}TVA_{-}tcA, ??, used: ??
G_{-}TVA_{-}tcG, ??, used: ??
G_{-}TVA_{-}tcGG, ??, used: ??
G_{-}TVA_{-}tcZ, ??, used: ??
G_{-}TVA_{-}ttA, ??, ??, ??, ??, used: ??, ??,
        ??, ??, ??
G_{-}TVA_{-}ttG, ??, ??, ??, ??, used: ??, ??,
        ??, ??, ??
G_{-}TVA_{-}ttGG, ??, ??, ??, ??, used: ??, ??,
        ??, ??, ??
G_{-}TVA_{-}ttWW, ??, ??, ??, ??, used: ??, ??,
        ??, ??, ??
G_{-}TVA_{-}ttZ, ??, ??, ??, ??, used: ??, ??,
        ??, ??, ??
G_{-}TVA_{-}tuA, ??, used: ??
G_{-}TVA_{-}tuG, ??, used: ??
G_TVA_tuGG, ??, used: ??
G_{-}TVA_{-}tuZ, ??, used: ??
G_{-}TWW, ??, ??, ??, used: ??, ??, ??, ??, ??
G_TWW_CF, ??, ??, used: ??, ??
G_{-}TWZ, ??, ??, ??, used: ??, ??, ??, ??, ??
G_TWZ_CF, ??, ??, used: ??, ??
G_{-}VLR_{-}btW, ??, ??, ??, ??, used: ??, ??,
        ??, ??, ??
```

```
G_{-}VLR_{-}qBuB, ??, ??, ??, ??, used: ??, ??,
                   ??, ??, ??
G_{-}VLR_{-}qBuB_{-}d, ??, ??, ??, ??, ??, used: ??,
                   ??, ??, ??, ??
G_{VLR_{q}BuB_{e}}, ??, ??, ??, ??, used: ??,
                   ??, ??, ??, ??
G_{-}VLR_{-}qBuB_{-}u, ??, ??, ??, ??, ??, used: ??,
                   ??, ??, ??, ??
G_{-}VLR_{-}qGuG, ??, ??, ??, ??, used: ??, ??,
                   ??, ??, ??
G_{-}VLR_{-}tbW, ??, ??, ??, ??, used: ??, ??,
                   ??, ??, ??
\begin{array}{ll} G_-VLR\_tcZ, & \ref{eq:constraints} \text{ $q$}. & \ref{eq:constraints}, \text{ used: } \ref{eq:co
                   ??, ??, ??
G_-VLR_-tuZ, ??, used: ??
G_-VL_-qBuB_-n, ??, ??, ??, ??, used: ??, ??,
                   ??, ??, ??
G_{-}VL_{-}qW, ??, ??, ??, ??, used: ??, ??, ??,
                   ??, ??
G_{-}VL_{-}qW_{-}d, ??, ??, ??, ??, used: ??, ??,
                   ??, ??, ??
G_{-}VL_{-}qW_{-}u, ??, ??, ??, ??, used: ??, ??,
                   ??, ??, ??
G_{-weak}, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                   ??, ??, ??, ??, ??, ??, used: ??, ??,
                   ??, ??, ??, ??
G_-WH3W, ??, ??, used: ??, ??
G_-WH_4, ??, ??, used: ??, ??
G_-WHWAAH, ??, ??, used: ??, ??
G_-WHWAZ, ??, ??, used: ??, ??
G_{-}WHWAZH, ??, ??, used: ??, ??
G_WHWHAAH, ??, ??, used: ??, ??
G_-WHWHAZH, ??, ??, used: ??, ??
G_-WHWHWW, ??, ??, used: ??, ??
G_-WHWHZAH, ??, ??, used: ??, ??
G_-WHWHZHAH, ??, ??, used: ??, ??
G_-WHWHZZH, ??, ??, used: ??, ??
G_-WHWWW, ??, ??, used: ??, ??
G_-WHWZAH, ??, ??, used: ??, ??
G_WHWZHAH, ??, ??, used: ??, ??
G_-WHWZHZH, ??, ??, used: ??, ??
G_-WHWZZ, ??, ??, used: ??, ??
G_-WHWZZH, ??, ??, used: ??, ??
G_-WmHph1, ??, used: ??
G_-WmHph2, ??, used: ??
G_-WmHph3, ??, used: ??
G_-WpHmh1, ??, used: ??
G_-WpHmh2, ??, used: ??
G_-WpHmh3, ??, used: ??
G_{-}WPSLSN, ??, ??, ??, used: ??, ??, ??
G_{-}WPSUSD, ??, ??, used: ??, ??, ??
G_-WSQ, ??, ??, used: ??, ??
G_-WWAAH, ??, ??, used: ??, ??
G_-WWAZH, ??, ??, used: ??, ??
G_{-}WWSFSF, ??, ??, ??, used: ??, ??, ??, ??,
                  ??
g_-wwww, ??, used: ??
G_-wwww, ??, used: ??, ??
```

```
G_{-}WWWW, ??, ??, ??, ??, ??, ??, ??, ??,
                                              g_z = lep, ??, used: ??
       G_{-}z_{-}lep, ??, used: ??, ??
                                              g_z = quark, ??, used: ??
       used: ??. ??. ??. ??. ??. ??. ??. ??.
                                              G_{-}z_{-}quark, ??, used: ??, ??
       ??, ??
                                              g_-wwz, ??, used: ??
                                                     ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
G_-wwz, ??, used: ??, ??
                                                     g_-wwza, ??, used: ??
                                                     G_-wwza, ??, used: ??, ??
G_-WWZAH, ??, ??, used: ??, ??
                                                     G_-WWZHAH, ??, ??, used: ??, ??
                                                     g_-wwzz, ??, used: ??
                                                     G_-wwzz, ??, used: ??, ??
                                                     G_{-}WWZZH, ??, ??, used: ??, ?? G_{-}WZSLSN, ??, ??, ??, used: ??, ??, ??
                                                     G_-WZSUSD, ??, ??, ??, used: ??, ??, ??
                                                     g_w_{lep}, ??, used: ??
                                                     G_-w_-lep, ??, used: ??, ??
                                                     g_w_quark, ??, used: ??
G_w_quark, ??, used: ??, ??, ??
                                                     ??, ??, ??, ??, ??
G_-YUK, ??, used: ??, ??
                                              H1, ??, used:
G_-YUK_-1, ??, used: ?? G_-YUK_-2, ??, used: ??
                                              H1dm, ??, used: ??, ??, ??
                                              H1dp, ??, used: ??, ??, ??
G_{-}YUK_{-}3, ??, used: ??
                                              H1um, ??, used: ??, ??, ??
G_-YUK_-4, ??, used: ??
                                              H1up, ??, used: ??, ??, ??
                                              H2, ??, used:
G_{-}YUK_{-}C, ??, ??, ??, used: ??, ??, ??
G_-YUK_-DCU, ??, ??, used: ??, ??
                                              H2 (module), ??, used: ??, ??, ??, ??
G_-YUK_-FFP, ??, ??, used: ??, ??
                                              H2dm, ??, used: ??, ??, ??
G_YUK_FFS, ??, ??, used: ??, ??
                                              H2dp, ??, used: ??, ??, ??
                                              H2um, ??, used: ??, ??, ??
G_{-}YUK_{-}G, ??, ??, ??, used: ??, ??, ??
                                              H2up, ??, used: ??, ??, ??
G_-YUK_-LCN, ??, ??, used: ??, ??
G_-YUK_-LQ_-P, ??, used: ??
                                              H3, ??, used:
G_{-}YUK_{-}LQ_{-}S, ??, used: ??
                                              H3 (module), ??, used: ??, ??, ??, ??
G_{-}YUK_{-}N, ??, ??, ??, used: ??, ??, ??
                                              H_4, ??, used:
                                              H5, ??, used:
G_{-}YUK_{-}Q, ??, ??, ??, used: ??, ??, ??
                                              H6, ??, used:
G_YUK_UCD, ??, ??, used: ??, ??
G_{-}Z, ??, ??, used:
                                              HA, ??, used: ??, ??, ??, ??, ??, ??, ??, ??,
G_{-}ZEH, ??, used: ??
                                                     ??, ??, ??, ??
G_{-}ZGlLQLQ, ??, used: ??
                                              half, 217, 337, ??, ??, ??, ??, ??, ??, ??,
G_-Zh1h2, ??, used: ??
                                                     ??, ??, ??, ??, ??, ??, ??, 332, used:
G_-Zh1h3, ??, used: ??
                                                     G_Zh2h3, ??, used: ??
                                                     ??, ??, ??, ??, ??
G_ZHEH, ??, used: ??
                                              Half, ??, ??, ??, ??, used: ??, ??, ??, ??,
G_{-}ZHpHm, ??, used: ??
                                                     ??, ??, ??, ??, ??, ??, ??, ??, ??
G_{-}ZHPsipp, ??, ??, used: ??, ?? G_{-}ZHTHT, ??, ??, used: ??, ??
                                              harvest, 33, 38, 31, used: 38, 38, 39, ??
                                              harvest\_goldstones, ??, used:
G_-zhthth, ??, used: G_-ZLQ, ??, used: ??
                                              harvest_list, 33, 38, 31, used: ??, ??
                                              hash, 621, 621, ??, ??, ??, 620, used: ??,
G_{-}ZPLQLQ, ??, used: ??
                                                     ??, ??, ??
G_{-}ZPSFSF, ??, ??, ??, used: ??, ??, ??
                                              hash (type), 620, 621, 620, used: 621, 621, 620
G_{-}ZPsip, ??, ??, used: ??, ??
                                              Hat, 288, 284, used: 288
G_ZPsipp, ??, ??, used: ??, ??
                                             HAT, ??, ??, used: ??, 57
G_{-}ZSF, ??, ??, ??, used: ??, ??, ??
                                              HAT (camlyacc token), 57, used: 57, 58
                                              HC1, ??, used: ??, ??, ??, ??, ??, ??, ??,
G_{ZTHT}, ??, ??, used: ??, ??
G_{-}ZZLQLQ, ??, used: ??
                                                     ??, ??, ??, ??, ??, ??, ??, ??, ??
G_{-}ZZSFSF, ??, ??, ??, used: ??, ??, ??, ??, ??
                                              HC1c, ??, used: ??, ??, ??, ??, ??, ??, ??, ??,
G_{-}ZZWW, ??, ??, ??, ??, ??, ??, ??, ??,
                                                     ??, ??, ??, ??, ??, ??, ??, ??, ??
       HC2, ??, used: ??, ??
       used: ??, ??, ??, ??, ??, ??, ??, ??,
                                              HC2c, ??, used: ??, ??
       HC3, ??, used: ??, ??
       ??, ??
                                              HC3c, ??, used: ??, ??
```

```
HC4, ??, used: ??, ??
                                                   higgs_hmm, ??, ??, ??, ??, ??, ??, ??, ??,
HC4c, ??, used: ??, ??
                                                           ??, ??, ??, ??, ??, ??, used: ??, ??,
HC5, ??, used: ??, ??
HC5c, ??, used: ??, ??
                                                   higgs_matrix, ??, ??, ??, used: ??, ??, ??,
                                                           ??, ??, ??, ??, ??
hdn, 625, 623, used: 625, 633
                                                   higgs_neutr, ??, ??, ??, used: ??, ??, ??
hdtl, 678, used: 678, 678
                                                   higgs_neutr', ??, used: ??
header, 599, used: 599
header (field), 705, 700, used: 705, 706, 599,
                                                   higgs\_neutr'', ??, used: ??
        600,600
                                                   higgs_neu_gravitino, ??, used: ??
                                                   higgs_SCC, ??, ??, used: ??, ??
header_sans_color, 599, used: 600, 600
                                                   higgs_sfermion, ??, ??, ??, used: ??, ??, ??
Heavy, ??, used: ??, ??, ??, ??, ??
Heavy2, ??, used: ??, ??
                                                   higgs_sfermion', ??, ??, used: ??, ??, ??
                                                   higgs_sfermion", ??, ??, ??, used: ??, ??, ??
heavy_currents, ??, used: ??
heavy_gauge_higgs, ??, ??, ??, used: ??, ??, ??
                                                                   ??, used: ??
                                                   higgs\_sfermion4,
heavy\_gauge\_higgs4, ??, used: ??
                                                                    ??, used: ??
                                                   higgs_sfermion4',
heavy_quartic_gauge, ??, ??, ??, used: ??, ??, ??
                                                   higgs\_sfermion\_P, ??, ??, used: ??, ??
heavy_top_currents, ??, ??, used: ??, ??
                                                   higgs\_sfermion\_S, ??, ??, used: ??, ??
                                                   higgs_sneutrino, ??, ??, ??, used: ??, ??, ??
heavy_triple_gauge, ??, ??, ??, used: ??, ??, ??
                                                   higgs_sneutrino', ??, ??, ??, used: ??, ??, ??
height, 645, used: 645, 645, 646
                                                   higgs_sneutrino", ??, ??, ??, used: ??, ??, ??
Helac (module), 27, ??, 18, 124, ??, used: ??,
                                                   higgs_sneutrino4, ??, used: ??
higgs_sneutrino4', ??, used: ??
Helac_Binary (module), 27, 18, 124, used: 606
                                                   higgs_SPP, ??, ??, used: ??, ??
Helac_Binary_Majorana (module), 124, used:
                                                   higgs_squark, ??, ??, ??, used: ??, ??, ??
Helac\_Majorana \text{ (module)}, ??, 124, ??, used:
                                                   higgs\_squark', ??, ??, used: ??, ??, ??
                                                   higgs_squark4, ??, used: ??
                                                   higgs\_squark4', ??, used: ??
Helac\_Mixed23 (module), 124, used: 606
                                                   higgs\_squark\_a, ??, ??, used: ??, ??, ??
Helac\_Mixed23\_Majorana \text{ (module)}, \quad 124, \text{ used:}
                                                   higgs_squark_b, ??, ??, ??, used: ??, ??, ??
helicities, ??, ??, 125, ??, used:
                                                   higgs_SSS, ??, ??, used: ??, ??
helicities (field), ??, used: ??, ??
                                                   higgs_triangle, ??, ??, ??, ??, ??, ??, ??, ??,
                                                           helicities (type), ??, used:
helicity_table, ??, used: ??
                                                           ??, ??, ??, ??, ??, ??, ??, ??, used:
hgg, ??, used: ??, ??, ??, ??, ??
                                                           ??, ??, ??, ??, ??, ??
     ??, used: ??, ??, ??, ??, ??, ??, ??, ??,
                                                   higgs_triangle_vertices, ??, ??, ??, used: ??, ??,
Hh,
        ??, ??, ??, ??
     ??, used: ??, ??, ??, ??, ??, ??, ??, ??,
HH,
                                                   Hit,
                                                        620, 622, 620, used: 622
        ??, ??, ??, ??
                                                   Hm,
                                                        ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                           hhgg,
      ??, used: ??, ??
      higgs,
        ??, ??, ??, ??, ??, ??, used: ??, ??,
                                                           ??, ??, ??, ??, ??, ??, ??, ??
        ??, ??, ??, ??, ??
                                                   Hn \text{ (module)}, ??, \text{ used: } ??, ??, ??, ??
higgs (type), ??, used: ??, ??
                                                   homogeneous, 628, 623, used: 628, ??, ??
       homogeneous\_final\_state, 280, used: 280, 282
higgs4,
        ??, ??, ??, ??, ??, ??, used: ??, ??,
                                                   hook_lengths_product, 696, 692, used: 696, 696,
        ??, ??, ??, ??, ??
                                                   hook_lengths_table, 696, used:
higgs_anom, ??, ??, ??, ??, ??, ??, ??, ??,
                                                   Hp, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                           ??, ??, ??, ??, ??, ??, used: ??, ??,
        ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                           higgs_charg_neutr, ??, ??, ??, used: ??, ??, ??
                                                           higgs_ch_gravitino, ??, used: ??
                                                           ??, ??, ??, ??, ??, ??, ??, ??, ??
higgs_gold, ??, used: ??
                                                   HSExt \text{ (module)}, ??, ??, used: ??, ??
higgs\_gold4, ??, used: ??
                                                   hs\_of\_flavor, ??, used: ??, ??
                                                   hs\_of\_flavors, ??, used: ??, ??
higgs_gold_sfermion, ??, used: ??
higgs_gold_sfermion', ??, used: ??
                                                   hs\_of\_lorentz, ??, used: ??, ??
higgs_gold_sneutrino, ??, used: ??
                                                   H_{-}Heavy, ??, used: ??, ??, ??, ??, ??, ??, ??,
higgs_gold_sneutrino', ??, used: ??
                                                           higgs\_gold\_squark, ??, used: ??
                                                           ??, ??, ??, ??, ??
higgs_gold_squark', ??, used: ??
```

```
H_Light, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                       Index (sig), 315, 341, 333, used: 322, 323, 342,
        333
        ??, ??, ??, ??, ??
                                                       INDEX, ??, ??, used: ??, 297
   214, 215, 215, 213, used: 215, 215, 218, 219,
                                                       INDEX (camlyacc token), 298, used: 300
        220, 220, 221, 221, 221, 297, 331
                                                       index_attribute (camlyacc non-terminal), 300,
   315, 316, 317, 320, 203, used: 317, 320, 321,
                                                               used: 300
        326, 339, ??, ??, ??, ??, ??
                                                       index_attributes (camlyacc non-terminal), 300,
                                                               used: 300, 300
I (module), ??, 307, 298, used: ??, 309, 310,
        310, 300, 300
                                                       index\_mismatches, 313, used: 314
i2s, 351, used: 351, 351
                                                       index\_space, 309, used: 310
id, 324, ??, used: 324, ??
                                                       index\_string, 641, 640, used:
ID, ??, ??, ??, ??, used: ??, ??, ??, ??, 331
                                                       indices, 305, 317, 322, 324, used: 306, 319, 319,
ID (camlyacc token), 331, 361, used: 332, 362,
                                                                319, 324
        362, 362
                                                       indices (field), 364, used:
Identity, 348, 349, 355, 356, 334, 335, used:
                                                       indices_ok, 311, 312, used: 312
        350, 352, 356, 356
                                                       Infinity, 703, used: 703
                                                       init, ??, ??, 211, used: 631, 731, 732, 687, 709,
Identity8, 355, 356, 335, used: 356
IG\_s, ??, used: ??
                                                               ??
imag, 716, used: 218
                                                              321, used: 322
                                                       init2,
IMap (module), 24, 354, used: 24, 24, 24, 24, 24,
                                                       init3, 321, used: 322
        25, 25, 354, 355
                                                       init4, 322, used: 322
import, 61, used: 62
                                                       init_position, ??, ??, ??, ??, ??, ??, 296, 330,
import_prefixed, 593, used: 593, 593
                                                               used: 303, 304, 336
                                                               ??, used: ??, ??, ??, ??, ??, ??
impossible, 691, 8, used: 691, 8, 8
                                                       inject,
                                                       injectl, ??, used: ??, ??
Impossible (exn), 691, 8, 20, 35, 39, ??, ??, ??,
        ??, used:
                                                       inner (field), 705, 700, used: 705, 600, 600, 600
                                                       inner_product_functions, 423, used:
in1 (field), 590, used:
in2 (field), 590, used:
                                                       input, ??, ??, used: ??, 336
Include, 295, 287, used: ??, 299
                                                       input (field), 208, used: ??, ??, ??, ??, ??, ??,
INCLUDE, ??, ??, used: ??, 297
                                                               INCLUDE (camlyacc token), 298, used: 299
                                                                ??, ??, ??, ??, ??, ??, ??, ??, 609
include_anomalous, ??, ??, ??, ??, ??, ??, ??,
                                                       input (camlyacc non-terminal), 332, used: 332
        ??, ??, used:
                                                       input\_line\_opt, 446, used: 446
include_ckm, ??, ??, ??, ??, ??, ??, ??,
                                                       input_parameters, ??, ??, ??, ??, ??, ??, ??,
                                                               ??, ??, ??, ??, ??, ??, ??, used: ??,
        used: ??, ??
include_four, ??, ??, ??, ??, ??, ??, ??,
                                                               used: ??
                                                               ??, ??, ??, ??
include_goldstone, ??, ??, ??, ??, ??, ??, ??,
                                                       insert, 679, 310, used: 679, 679, 310
        used: ??, ??, ??, ??
                                                       insert_inorder_signed, 680, used: 680
include_goldstones, 600, used: 600, 603
                                                       insert\_in\_unfinished\_decays, 586, used: 587
include_hf, ??, ??, ??, ??, ??, ??, ??, used:
                                                       insert\_kind, 310, used: 310
        ??, ??
                                                       insert_signed, 679, used: 679, 679
incoming, 44, 47, 48, 52, 52, 278, 278, ??, ??,
                                                       insert_stem, 310, used: 310
        ??, ??, 42, 43, 277, 123, ??, used: 52,
                                                       inspect_partition, 18, 18, 20, 22, 22, 27, 27, 16,
        ??, ??, ??
incoming (field), 705, ??, ??, 700, used: 705,
                                                       Int (module), ??, 23, ??, ??, 17, used: 630, 630,
                                                               343, 343, 343, 355, 355, 282, 282, ??, ??
        706, ??, ??, ??, ??, ??
Incompatible\_factors (exn), 354, used:
                                                             ??, ??, ??, ??, ??, used: ??, ??, ??, 57,
Incomplete (exn), 584, 585, 583, used:
Inconsistent\_vertex (exn), 327, used:
                                                       INT (camlyacc token), 57, 331, 361, used: 58,
increasing, 694, used: 695
                                                               332, 362, 363, 363, 363
index, 305, used: 641, 306, ??
                                                               291, 330, 285, 329, used: ??, ??, 336,
                                                       integer,
index (type), 315, 317, 322, 323, used: 316,
                                                               301, 301, 332
        316, 316, 317
                                                       integer (type), 23, 23, 17, used: 23, 24, 24, 17,
index (camlyacc non-terminal), 300, used: 299
                                                                17, 17, 17
       295, 296, 307, 307, 287, 287, used: 296,
                                                       integer (camlyacc non-terminal), 301, used: 301,
        ??, 310, 299
Index (module), 294, 322, 323, 342, 287, 333,
                                                       Integer, 291, 329, 338, 203, 285, 329, 359,
        used: 295, 296, 296, ??, 307, 322, 323,
                                                                used: 291, 291, 291, 291, 330, ??, 337,
        343, 343, 348, 351, 352, 352, 354, 355,
                                                                339, 339, 340, 346, 350, ??, ??, ??, ??,
        287, 287, 298
```

```
is_gauss, 60, 63, ??, ??, ??, ??, 59, 124, ??,
        used: ??, ??
                                                         is_gauss (field), 63, ??, ??, used: 63, 63, 67, ??,
        ??, ??, ??
        ??, ??, ??, ??, ??, ??, ??, ??, ??, 362
                                                         is\_goldstone\_of, ??, used:
                                                         is_integer, 715, 716, used: 346
Integer (sig), 22, 17, used: 23, 23, 17, 17
integers (camlyacc non-terminal), 363, used:
                                                         is\_negative, 715, 716, used:
        362, 363
                                                         is_node, 33, 37, 31, used: ??, ??, ??
Integer_List, 359, used: ??, 362
                                                         is_null, 249, 249, 249, 249, 249, 249, 715,
integer\_or\_id, 352, used: 352
                                                                  716, 716, 718, 248, used: 249, 249, 344,
integer_range, 282, used: 282
                                                                  278, ??
interleave, 629, 624, used: 629
                                                         is\_numeric, 642, used: 642
\begin{array}{ll} interleave\_nearest, & \textbf{629}, \, \textbf{624}, \, \text{used: 291} \\ interleave\_spaces, & \textbf{291}, \, \text{used: 291} \end{array}
                                                         is\_offspring, 33, 37, 31, used: 38 is\_positive, 715, 716, used:
internal_edges, 707, used: 708
                                                         is_real, 716, used: 346
                                                         is_singleton, 711, 711, used: ??
internal\_edges\_from, 707, used: 707, 707
int_edges (field), 707, used: 708, 708, 710
                                                         is\_skip, ??, used: ??
int_incidence, 708, used: 709
                                                         is\_source, ??, used: ??, ??, ??
int\_list2\_to\_string, 631, used: 632
                                                         is_sterile, 33, 37, 31, used: 38, 38, ??
                                                         is\_success, ??, used:
int_list_to_string, 56, used:
int_node, 704, used: 704
                                                         is\_todo, ??, used: ??
                                                         is_unit, 343, 351, 356, 715, 716, 717, 718,
int_nodes (field), 707, used: 708, 708, 709, 710,
        710, 710
                                                                  334, used: 344, 345, 351
int_of_char, ??, ??, ??, 296, used: ??, ??,
                                                         It (module), 250, used: ??, ??, ??, ??, 597, 124,
        126, ??, ??
        ??, 297
                                                         iter, 645, 649, 652, 658, 658, 659, 660, 661, 7,
                                                                  9,\, 10,\, 12,\, 14,\, 33,\, 35,\, 36,\, 37,\, 39,\, 644,
int\_of\_csign, ??, used: ??, ??, ??, ??, ??, ??
int\_of\_gen, ??, used: ??
                                                                  656, 656, 657, 5, 31, used: ??, ??, ??,
int_orders, ??, used: ??
                                                                  ??, ??, ??, 649, 652, 446, 447, 447,
                                                                  447, 448, 659, 660, 661, 662, 711, 712,
inv, 715, 716, used: 346
                                                                  704, 705, 705, 706, 706, 706, 706, 707,
invalid\_flavor, 609, used: 609
invalid\_group, 293, used: 293, 293, 294
                                                                  708, 708, 710, 710, 710, 14, 36, 37, 37, 39,
invalid_parameter_attr, ??, ??, ??, 298, 331,
                                                                  ??, ??, 592, 593, 593, 594, 595, 304, 310,
        361, used: ??, 300
                                                                  312, 325, 326, 327, ??, ??, ??, ??, ??,
inverse, 684, 685, 686, 683, used: 685, 686, 688,
                                                                  603, 604, 604, 604, 604, 605
                                                         iter', 659, 661, used: 659, 661
inverse (field), 365, used:
                                                         iteri, 627, 624, used: 627, 695
Inverse, 349, 349, 335, used: 350, 350, 351
                                                         iteri2, 627, 624, used:
inverse_ok, 688, used: 688
                                                         iter\_edges, 710, 701, used:
invert, 343, 351, 356, 334, used: 346
                                                         iter\_incoming, 710, 701, used:
invertible, 343, 351, 356, 334, used: 346
                                                         iter_internal, 710, 701, used:
invert_array, 708, used: 708
                                                         iter_nodes, 33, 37, 31, used: ??
invert_array_unsafe, 708, used:
                                                         iter_outgoing, 710, 701, used:
                                                         I\_Dim6\_AWW\_DP, ??, used: ?? I\_Dim6\_AWW\_DW, ??, used: ??
InvPi (module), 668, used: 668, 668, 668,
        668, 668
inv_pi, 667, 668, 668, 669, 667, 667, used:
                                                         I\_Dim6\_AWW\_Gauge, ??, used:
                                                         I\_Dim6\_AWW\_GGG, ??, used: ??
        668, 669, 669, ??, ??, ??
IPowSet (module), 66, 282, used: 66, 66, 282
                                                         I_Dim6_WWZ_D, ??, used: ??
iset\_list\_union, 630, used: 630
                                                         I_Dim6_WWZ_DPWDW, ??, used: ??
isospin (type), ??, used: ??, ??
                                                         I\_Dim6\_WWZ\_DW, ??, used: ??
                                                         I\_Dim6\_WWZ\_W, ??, used: ??
Iso_down, ??, used: ??, ??, ??
                                                         I_{-}G1_{-}AWW, ??, ??, ??, ??, used: ??, ??,
Iso_up, ??, used: ??, ??
                                                                  ??, ??, ??
is_alpha, 642, used: 642, 642, 642
is\_alphanum, 642, used: 642, 642
                                                         I\_G1\_minus\_kappa\_minus\_G4\_AWW, ??, ??,
                                                                  ??, ??, ??, used: ??, ??, ??, ??, ??
is_backslash, 290, used: 291
is\_char, 290, used: 291
                                                         I_G1_minus_kappa_minus_G4_ZWW, ??, ??,
                                                                  ??, ??, ??, used: ??, ??, ??, ??, ??
is_empty, 645, 645, 650, 671, 671, 658, 658,
        658, 644, 670, 656, 656, used: 671, 671,
                                                         I_G1_minus_kappa_plus_G4_AWW, ??, ??, ??,
                                                                  ??, ??, used: ??, ??, ??, ??, ??
        671, 672, 658, 37, 38, 66, 281, 603
is\_error, ??, used: ??
                                                         I_-G1_-minus_-kappa_-plus_-G4_-ZWW, ??, ??, ??,
is_failure, ??, used: ??
                                                                  ??, ??, used: ??, ??, ??, ??, ??
```

```
I_G1_plus_kappa_minus_G4_AWW, ??, ??, ??,
                                                     I_{-kappa5\_AWW}, ??, ??, ??, ??, used: ??,
        ??, ??, used: ??, ??, ??, ??, ??
                                                             ??, ??, ??, ??
                                                     I_{-kappa5_{-}ZWW}, ??, ??, ??, ??, used: ??,
I_G1_plus_kappa_minus_G4_ZWW, ??, ??, ??,
        ??, ??, used: ??, ??, ??, ??, ??
                                                             ??, ??, ??, ??
I_G1_plus_kappa_plus_G4_AWW, ??, ??, ??,
                                                     I_{-lambda5\_AWW}, ??, ??, ??, ??, used: ??,
                                                             ??, ??, ??, ??
        ??, ??, used: ??, ??, ??, ??, ??
I_G1_plus_kappa_plus_G4_ZWW, ??, ??, ??,
                                                     I_{-lambda5}ZWW, ??, ??, ??, ??, used: ??,
        ??, ??, used: ??, ??, ??, ??, ??
                                                             ??, ??, ??, ??
I_{-}G1_{-}ZWW, ??, ??, ??, ??, used: ??, ??,
                                                     I\_lambda\_AWW, ??, ??, ??, ??, ??, used: ??,
                                                             ??, ??, ??, ??
        ??, ??, ??
I_{-}G3, 607, used: 609
                                                     I_{-lambda\_ZWW}, ??, ??, ??, ??, used: ??,
I\_Gauge\_Gauge\_Gauge, 180, used:
                                                             ??, ??, ??, ??
I_{-}Q_{-}H, ??, ??, ??, used:
                                                     I_{-}Q_{-}W, ??, ??, ??, ??, ??, ??, ??, ??, ??.
        ??, ??, ??, ??, ??, used: ??, ??, ??,
        ??, ??, ??, ??, ??, ??, ??, ??, ??,
        ??, ??, ??, ??, ??, ??, ??, ??
                                                             used: ??, ??, ??, ??, ??, ??, ??, ??,
I\_GsRt2, ??, used: ??
                                                             I_{-}G_{-}Abb, ??, used:
                                                             I\_G\_Acc, ??, used:
                                                             ??, ??, ??, ??, ??, ??, ??, ??, ??
I_{-}G_{-}AHWHW, ??, ??, used: ??, ??
                                                     I_{-}Q_{-}W_{-}K, ??, used: ??
I_{-}G_{-}AHWHWH, ??, ??, used: ??, ??
                                                     I_{-}Q_{-}ZH, ??, used: ??
I_-G_-AHWW, ??, ??, used: ??, ?? I_-G_-Amumu, ??, used:
                                                     J, 316, 320, used: 321
                                                     Java (module), 448, used:
I\_G\_Atautau, ??, used:
                                                     join, 646, 619, 619, used: 646, 646, 646, 648,
I_{-}G_{-}Att, ??, used:
                                                             649, 619
I\_G\_CC, ??, used: ??
                                                     join_signs, 680, used: 680
I\_G\_Psi0ppWHW, ??, ??, used: ??, ??
                                                     K, 316, 318, used: 318, 327, 327
I_-G_-Psi\theta ppWHWH, ??, ??, used: ??, ??
                                                     K6, 355, 356, 335, used: 356, 356
I_-G_-Psi0ppWW, ??, ??, used: ??, ??
                                                     K6Bar, 355, 356, 335, used: 356, 356
I_{-}G_{-}Psi\partial pWA, ??, ??, used: ??, ??
                                                     ket, ??, ??, ??, ??, 123, ??, used: ??, 603
I_-G_-Psi\partial pWAH, ??, ??, used: ??, ??
                                                     ket (field), 364, used:
I_-G_-Psi\partial pWHA, ??, ??, used: ??, ??
                                                     ket_to_channel, ??, used: ??
I_-G_-Psi\theta pWHAH, ??, ??, used: ??, ??
                                                     key (type), 620, 621, 658, 658, 658, 35, 36, 39,
I_G_Psi0pWHZ, ??, ??, used: ??, ??
                                                             620, 656, 656, used: 621, 658, 658, 658,
I_G_Psi0pWHZH, ??, ??, used: ??, ??
                                                             658, 660, 35, 36, 36, 39, ??, 620, 620, 656,
I_G_Psi0pWZ, ??, ??, used: ??, ??
                                                             656, 656, 656, 656, 656, 657
I_{-}G_{-}Psi\partial_{p}WZH, ??, ??, used: ??, ?? I_{-}G_{-}S, ??, ??, ??, used: ??, ??, ??
                                                     Key \text{ (sig)}, \quad \textbf{621}, \, \textbf{620}, \, \text{used: } 621, \, 620
                                                     keys, 36, used: 36
I_{-}G_{-}weak, ??, ??, ??, ??, used: ??, ??, ??,
                                                     keystone, 20, used: 20
        keystones, 678, 18, 20, 21, 22, 22, 23, 26, 27,
        ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                             28, 673, 16, 17, used: 678, 27, ??
I_{-}G_{-}WWW, ??, ??, ??, ??, ??, used: ??, ??
                                                     keystones', 21, 27, 28, used: 21, 22, 22, 27, 28
I_{-}G_{-}Z1, ??, used: ??
                                                     kind, 212, 213, used:
I_{-}G_{-}Z2, ??, used: ??
                                                     kind (field), 359, used: ??, 362
                                                     kind (type), 307, 307, 212, 213, used: 307, 307,
I_{-}G_{-}Z3, ??, used: ??
I_{-}G_{-}Z_{4}, ??, used: ??
                                                             307, 308, 212, 213
I_-G_-Z_5, ??, used: ??
                                                     kind\_conflicts, 313, used: 314
I_{-}G_{-}Z6, ??, used: ??
                                                     kind_of_stem, 307, 308, used: 308
I_{-}G_{-}ZHWHWH, ??, ??, used: ??, ??
                                                     kind\_of\_symbol, 307, 308, used: 308
I_{-}G_{-}ZHWW, ??, ??, used: ??, ??
                                                     kind\_of\_symbol\_or\_stem, 307, 308, used: 312
kind_to_string, 308, used: 308, 308, 310
                                                     kk2 (type), ??, used: ??
        kkmode (type), ??, used: ??, ??, ??
        used: ??, ??, ??, ??, ??, ??, ??, ??,
                                                     kmatrix_cuts, ??, used: ??, ??, ??
        k_{-}matrix, ??, ??, ??, ??, ??, ??, ??, ??, ??,
        ??, ??, ??, ??, ??, ??, ??, ??
                                                             ??, ??, ??, ??, ??, ??, used: ??, ??,
I_-G_-ZWW_-K1, ??, used: ??
                                                             ??, ??, ??
I_-G_-ZWW_-K2, ??, used: ??
                                                     k_{matrix\_2scalar\_2gauge}, ??, ??, used: ??, ??
I_-G_-ZWW_-K3, ??, used: ??
                                                     k_{matrix\_2scalar\_2gauge\_m}, ??, ??, used: ??,
```

```
k_{matrix\_2scalar\_2gauge\_m\_1}, ??, ??, used:
                                                                                                              LBRACKET (camlyacc token), 57, 298, 361,
                 ??. ??
                                                                                                                               used: 58, 299, 299, 362
k_matrix_2scalar_2gauge_m_7, ??, ??, used:
                                                                                                              leaf,
                                                                                                                          711, 701, 711, 699, used: 38, 38
                                                                                                              Leaf, 711, 701, 584, used: 711, 701, 702, 585,
                 ??, ??
k_matrix\_4scalar, ??, ??, used: ??, ??
                                                                                                                               585, 585, 586, 586
\begin{array}{ll} K\_Matrix\_Coeff, & ??, ??, ??, ??, ??, used: \\ K\_Matrix\_Pole, & ??, ??, ??, ??, ??, used: \end{array}
                                                                                                              leafs, 701, 699, used: 701, 704, 707, ??
                                                                                                              leafs\_and\_nodes, 707, used: 708
k\_matrix\_quartic\_gauge, \quad \ref{eq:constraint}, \ref{eq:constraint}
                                                                                                              leaf_label, 703, 703, used: 704, 704
                used: ??, ??, ??, ??, ??
                                                                                                              leaf\_node, 704, used: 704, 704
k_{-}matrix_{-}quartic_{-}gauge_{-}m_{-}0, ??, ??, used: ??,
                                                                                                              leaf\_or\_int\_node, 704, used: 704
                                                                                                              leaf\_to\_string, 702, used: 703
k_{-}matrix_{-}quartic_{-}gauge_{-}m_{-}1, ??, ??, used: ??,
                                                                                                              left_to_right, 709, 700, used:
                                                                                                              k_{matrix\_quartic\_gauge\_m\_7}, ??, ??, used: ??,
                                                                                                                               used: ??, ??, ??, ??, ??, ??, ??, ??,
k_{matrix\_quartic\_gauge\_t\_0}, ??, ??, used: ??,
                                                                                                                               ??, ??
k_{matrix\_quartic\_gauge\_t\_1},
                                                           ??, ??, used: ??,
                                                                                                              Lepton, ??, used: ??, ??, ??, ??, ??
                                                                                                              lep_family, ??, used:
k_matrix_quartic_qauge_t_2, ??, ??, used: ??,
                                                                                                              less, 44, 47, 51, 42, used: 585, 586
                                                                                                              less', 47, used: 47, 47
                                                                                                              lesseq, 44, 47, 51, 42, used: 51, 585, 586, 60
k_{-}matrix_{-}quartic_{-}gauge_{-}t_{-}rsi, ??, ??, used: ??,
                                                                                                              lesseq', 47, used: 47, 47
k_{-}matrix_{-}tm, ??, ??, ??, used: ??, ??
                                                                                                              Lexical\_Error (exn), ??, ??, used:
      lexicographic, 630, 625, used:
                 ??, ??, ??, ??, ??, ??, ??, used: ??,
                                                                                                              lhs, ??, ??, ??, ??, 123, ??, used: ??, ??, ??, ??,
                 ??, ??
                 Lie (module), 293, 286, used: 306, 307
                 Light, ??, used: ??, ??, ??, ??, ??, ??
                 Light2, ??, used: ??, ??, ??
                 Light\_Heavy, ??, used: ??, ??
                 Linalg (module), 731, 731, used: 709, 709
                 linear, 718, used:
                 linear (type), 343, 344, 333, used: 343, 344, 333
                 Linear, 343, 344, 333, used: 345, 345, 346, 346,
                 346, 346
                 Linear (sig), 718, used: 719
                 linear\_to\_string, 344, used: 344
                 lines_of_file, 446, used: 446
                lines_to_string, ??, ??, used: ??, ??
                 ??, ??, ??, ??, ??
                                                                                                              list, 664, 684, 685, 686, 289, 663, 683, 284,
L (module), 307, used: 307, 307, 309
label, 584, used: 584, 585, 585, 586, 587, 587
                                                                                                              list (camlyacc non-terminal), 362, used:
label (field), 703, 700, used: 704, 704, 705, 705,
                                                                                                              List, 288, 284, used: 289, 289, 289, 290
                                                                                                              List (module), 650, 644, used: ??, ??, ??, ??,
                598
Label, ??, ??, used: ??, ??, ??
                                                                                                                               ??, ??, ??, ??, ??, ??, ??, ??, ??,
Lambda, ??, used:
                                                                                                                               736, 736, 736, 736, 736, 737, 737, 737,
last_begin (field), 616, used: 617, 617, 617, 618
                                                                                                                               446, 446, 447, 447, 447, 448, 625, 625,
                                                                                                                               626, 626, 626, 626, 627, 627, 627, 627,
last\_char, 290, used: 291
last_reset (field), 616, used: 617, 617, 618
                                                                                                                               627, 627, 627, 627, 628, 629, 629, 629,
LaTeX (module), 448, used:
                                                                                                                               630, 630, 630, 630, 630, 630, 630, 630,
latex_section, 705, used: 705, 706
                                                                                                                               630, 631, 631, 631, 631, 632, 636, 637,
                                                                                                                               668, 671, 671, 671, 671, 671, 671, 619,
Laurent (module), 717, used:
Laurent (sig), 716, used: 717
                                                                                                                               621, 734, 711, 711, 711, 711, 711, 712,
layout, 709, 700, used:
                                                                                                                               615, 615, 663, 664, 664, 664, 664, 664,
layout (type), 709, 700, used: 700, 701
                                                                                                                               664, 676, 676, 676, 676, 676, 677, 677,
LBRACE, ??, ??, ??, used: ??, ??, 297
                                                                                                                               677, 677, 677, 677, 678, 678, 678, 678,
LBRACE (camlyacc token), 298, 361, used: 299,
                                                                                                                               678, 678, 678, 679, 679, 679, 679,
                 299,\,301,\,301,\,301,\,363
                                                                                                                               679, 679, 679, 680, 680, 680, 680, 680,
\mathit{LBRACKET}, \quad \ref{loop}, \ref{l
                                                                                                                               680, 680, 681, 682, 683, 684, 684, 684,
                 ??, ??, 57, 297
                                                                                                                               684, 685, 685, 685, 685, 685, 685, 685,
```

```
686, 686, 686, 686, 686, 686, 687, 689,
                                     list2, 664, 663, used: 9, 12, 346, 346, ??, ??, ??,
                                           690, 690, 691, 701, 702, 702, 702, 702,
                                           702, 703, 704, 704, 705, 705, 706, 706,
706, 706, 707, 707, 708, 708, 708, 709,
                                           709, 709, 710, 694, 694, 694, 694, 695,
                                           695, 695, 695, 695, 695, 696, 696, 8, 8, 9,
                                           ??, ??, ??, ??, ??, ??, ??, ??, ??
9, 10, 11, 13, 13, 13, 14, 14, 14, 14, 14, 15,
15, 18, 20, 20, 21, 21, 22, 22, 24, 24, 24,
                                     list2\_opt, 664, 663, used:
26, 26, 26, 26, 26, 27, 27, 27, 27, 27, 28,
                                     list3, 664, 663, used: 10, 12, ??, ??, ??, ??, ??,
28, 28, 38, 38, 38, 39, 45, 45, 46, 46, 49,
                                            ??, ??, ??
50, 50, 50, 583, 584, 586, 586, 586, 587,
                                     list3\_opt, 664, 663, used:
587, 587, 588, 249, 249, 249, 249, ??, ??,
                                     ListItem, ??, ??, used: ??, ??, ??
lists, 33, 38, 31, used: ??, ??
590, 590, 590, 590, 590, 590, 591, 591,
                                     Lists (module), 44, 43, used: 54, 43
591, 591, 591, 591, 591, 592, 592, 593,
                                     ListSimpleMake (module), ??, ??, used:
593, 593, 594, 595, 217, 289, 289, 289,
                                     ListsW (module), 54, 43, used:
290, 290, 290, 291, 292, 293, 293, 295,
                                     list\_of\_array\_row, 695, used: 695
295, 296, 296, 304, 309, 309, 310, 310,
                                     list\_opt, 664, 663, used:
                                     list_to_string, 290, 312, 341, 342, 285, 333,
311, 312, 312, 312, 312, 312, 313, 315,
318, 319, 319, 320, 320, 321, 321, 322,
                                           used: 290, 290, 292, 293, 295, 295, 309,
324, 324, 324, 324, 324, 325, 325, 326,
                                           312.342
327, 330, 330, 336, 336, 336, 338, 339.
                                     Literal, 358, used: ??, 363, 363
340, 341, 341, 342, 342, 344, 344, 344,
                                     literal_string_expr (camlyacc non-terminal), 363,
345, 345, 345, 345, 345, 345, 345, 345,
                                           used: 363
346, 346, 346, 346, 348, 348, 355, 355,
                                     Littlest (module), ??, ??, used: ??, ??
                                     littlest\_gauge\_higgs4, ??, ??, used:
355, 355, 358, 56, 56, 60, 61, 61, 61, 62,
62, 62, 62, 63, 63, 64, 64, 64, 65, 65, 65,
                                     Littlest_Tpar (module), ??, ??, used: ??
66, 66, 66, 67, 278, 278, 279, 279, 281,
                                     LMOM, 180, used:
281, 281, 281, 281, 281, 281, 281, 282,
                                     load, 307, 310, used: 312
282, 282, 283, 283, ??, ??, ??, ??, ??, ??,
                                     Lodd, ??, used: ??, ??
Log, 337, 203, used: 337, 339
Log10, 337, 203, used: 337, 339
longest, 658, 659, 660, 662, 656, 657, used:
longest', 659, 662, used: 659, 662
??, ??, ??, ??, ??, ??, ??, ??, ??,
                                     longest_first, 678, used: 678
                                     lookup2, 321, used: 322
597, 598, 598, 598, 598, 599, 599, 599,
599, 599, 600, 600, 600, 600, 600, 600,
                                     lookup3, 322, used: 322
600, 601, 601, 602, 602, 603, 603, 603,
                                     lookup4, 322, used: 322
                                     lookup_fuse2, ??, used: ??, ??
604, 604, 604, 604, 605, 606, ??, ??, ??,
lookup_fuse3, ??, used: ??, ??
lookup_fusen, ??, used: ??, ??, ??, ??
lookup_ketset, ??, used: ??
??, ??, ??, ??, ??, ??, ??, ??, ??, ??.
                                     loop\_cs, ??, used: ??, ??
loop_gen, ??, used: ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??, ??, ??, ??, ??, ??.
                                            ??, ??, ??, ??
                                     loop\_iso, ??, used: ??, ??, ??, ??, ??, ??, ??, ??.
??, ??
                                     loop_{-}kk, ??, used: ??, ??, ??, ??, ??, ??, ??, ??,
??, ??, ??, ??, ??
loop_kk2, ??, used: ??, ??, ??, ??
Loop_Tags (module), ??, used:
                                           7?, ??, ??, ??, ??, ??, ??, ??, ??, ??,
lorentz,
??, ??, ??, ??, ??, ??, ??, ??, ??,
607, 210, 422, used: ??, ??, ??, 324,
325, 325, ??, ??, ??, ??, ??, ??
lorentz (field), 311, 324, 378, used: 311, 311,
311, 312, 312, 324, 324, 326, 326, 326,
326, 327, 327, 327, 327
??, ??, ??, ??, ??, ??, ??, 607, 607,
                                     lorentz (label), ??, ??, 211, used:
607, 607, 608, 609, 609, 609, 609, 609, 611
```

```
lorentz (type), 178, used: 315, 350, 355, 178,
                                          L_K1_L, ??, used: ??, ??, ??
                                          L_{-}K1_{-}R, ??, used: ??, ??, ??
      178, 210, 211, 335, 365, 422, 422, 378
Lorentz, 294, 295, 306, 307, 287, 287, used:
                                          L_{-}K2_{-}L, ??, used: ??, ??, ??
      ??, 309, 309, 300, 301
                                          L_{-}K2_{-}R, ??, used: ??, ??, ??
                                         L_-NC, ??, used:
Lorentz (module), 294, 317, 355, 286, 335,
                                          L_NCH, ??, used:
      used: 324, 324, 324, 325, 325, 326, 326,
                                         M, 316, 320, ??, ??, ??, ??, ??, ??, ??, ??,
      326, 326, 326, 326, 326, 327, 327,
      327, 327, 365, 378
                                                ??, ??, ??, ??, ??, ??, ??, used: 320,
                                                321, 321, ??, ??, ??, ??, ??, ??, ??, ??,
Lorentz (sig), 315, used: 317
                                                LORENTZ, ??, ??, used: ??, 297
                                                LORENTZ (camlyacc token), 298, used: 300,
                                                lorentz3 (type), 178, used:
                                                lorentz4 (type), 178, used:
                                                lorentzn (type), 178, used: 184, 422, 378
Lorentz_Atom (module), 349, 335, used: 350,
                                                350, 350, 335, 365
                                                Lorentz_Atom (sig), 348, 334, used: 335
                                                Lorentz\_Atom' (module), 350, used: 355
lorentz_aux, ??, ??, ??, ??, used: ??, ??, ??,
                                                ??. ??
                                                lorentz\_module, 378, used:
                                                ??, ??
lorentz_ordering, ??, used: ??
lorentz_reps (field), 315, used: 317, 317, 318, 324
                                          M (module), 736, 707, 39, used: 736, 736, 736,
lower (field), 705, 700, used: 706
                                                736, 736, 708, 708, 708, 708, 39
lower (camllex regexpr), 57, 297, 331, used: 57,
                                          M1, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
      297, 331
                                                lowercase, 642, 640, used: 642
LPAREN, ??, ??, ??, ??, ??, ??, ??, used:
                                                ??, ??, ??, ?7, 297, 331
                                                LPAREN (camlyacc token), 57, 298, 331, 361,
      used: 58, 301, 302, 332, 362, 363, 364
                                                ??, used: ??, ??, ??, ??, ??, ??, ??, ??,
      ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                ??, ??, ??, ??
      ??, used: ??, ??, ??, ??, ??, ??, ??,
                                          M1 (module), 36, used: 36, 36, 36, 36
LQino,
      ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                          M2, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                lqino\_lq\_gg, ??, used: ??
                                                lqino\_lq\_gg', ??, used: ??
lqino\_lq\_neu, ??, used: ??
                                                lgino\_lq\_neu', ??, used: ??
                                                lgino\_lq\_neu2, ??, used: ??
                                                lqino\_lq\_neu2', ??, used: ??
                                                lq\_gauge4, ??, used: ??
                                                lq\_gauge4', ??, used: ??
                                                ??, ??, ??, ??, ??, ??
\begin{array}{ll} lq\_gg\_gauge2, & ??, \text{ used: }?? \\ lq\_gg\_gauge2', & ??, \text{ used: }?? \end{array}
                                          M2 (module), 36, used: 36, 36, 36
                                          macro, 359, used: ??, 362
lq\_neutr\_Z, ??, used: ??
                                          Macro, 358, used: ??, 363, 363
lq\_phiggs, ??, used: ??
                                          macro_string_expr (camlyacc non-terminal),
lq\_phiggs', ??, used: ??
lq\_se\_su, ??, used: ??
                                                363, used: 363, 363
                                               ??, 596, 601, ??, 596, used: 61, 606, ??,
                                          main,
lq\_se\_su', ??, used: ??
                                                ??, ??, 607, ??, ??, ??, ??, ??, ??, ??,
lq_shiggs, ??, used: ??
                                                lq\_shiggs', ??, used: ??
                                                lq\_snu\_sd, ??, used: ??
                                                lq\_snu\_sd', ??, used: ??
                                                ??, ??, ??, ??, ??, ??, ??, 611, ??,
                                                ??, ??, ??, ??
lu\_backsubstitute, 733, used: 734, 734
lu\_decompose, 733, 731, used:
                                          main (camlyacc non-terminal), 58, used: 58
lu\_decompose\_in\_place, 732, used: 733, 734, 734
                                          Maj, 353, used: 353, 354
                                          Majorana, 294, ??, 178, 286, used: 355, ??, ??,
lu\_decompose\_split, 733, used: 733
                                                603, ??, ??, ??, ??, 607
L_-CN, ??, used:
L\_CNG, ??, used:
                                          Majorana (exn), 124, used:
```

```
Majorana (module), 221, 214, used:
                                                       Majorana_R (module), 220, used: 221
                                                       Maj\_Ghost, 178, used:
make, 715, 716, used: ??, ??, 445, 636, 731, 732,
                                                       686, 686, 215, 217, ??, ??, ??, ??, ??, ??,
                                                       ??, 346, ??, 598, ??, ??, ??, ??, ??, ??,
       ??, ??, ??, ??, 296, 330
                                                       Make (module), 735, 668, 671, 621, 658, 40,
                                                       584, 589, 215, 60, 278, ??, 596, 735,
                                                       667, 670, 620, 657, 32, 583, 589, 59,
       278, used: ??, ??, 736, 736, ??, ??, ??,
                                                       ??, ??, ??, ??, ??, ??, ??, ??, ??,
       668, 671, 671, 660, 615, 24, 36, 36, 36, 36,
       39, ??, ??, ??, ??, 589, 590, 592, 221,
                                                       607, 607, 607, 607, 608, 609, 609, 609,
       308, 321, 321, 322, 324, 336, 354, 66, 281,
                                                       609, 609, 611
       282, 282, 283, ??, ??, ??, ??, ??, ??, ??,
                                                map2,
                                                       7, 9, 10, 12, 14, 5, used: 736, 630, 664,
       709, 14, 249, 249, 590, 591, 325, 281, ??
       ??, ??, ??, 597, 597, 597, 597, 606,
                                                Map2 \pmod{1}, used: 321, 321
       606, 124, ??
                                                map3, 627, 624, used: 627
MakeMap (module), 660, 657, used:
                                                Map3 (module), 321, used: 321, 322
                                                Map4 (module), 322, used: 322, 322
MakePoly (module), 661, 657, used:
Maker (sig), ??, 247, 124, ??, used: 589, ??, ??,
                                                mapi, ??, 645, 649, 652, 627, 658, 658, 659,
       596, 606, 606, 247, 589, 124, 124, 124,
                                                       660, 661, 644, 624, 656, 656, 657,
       124, 126, ??, ??, ??, ??, ??, 596, 596,
                                                       used: ??, 649, 652, 659, 661, 709, ??
       596, 448, 448, 448
                                                mapi', 659, 661, used: 659, 661
make\_external\_dag, ??, used: ??
                                                mappable, 592, used: 592
Make\_Linear (module), 719, used:
                                                Mappable (module), 592, used: 592, 592
Make\_Ring (module), 719, used:
                                                map\_amplitude\_wfs, ??, used:
map, 645, 649, 652, 658, 658, 659, 660, 661,
                                                map\_atom, 364, used:
       702, 7, 9, 10, 12, 14, 33, 37, 584, 586,
                                                map_atoms, 343, 345, 333, used: 345, 345
                                                map\_coeff, 343, 345, 333, used: 346
       336, 644, 656, 656, 657, 717, 718, 699,
       5, 31, 583, used: ??, 736, 736, 736, 737,
                                                map\_conjspinor, 317, used: 318
       737, 737, 649, 652, 642, 627, 628, 629,
                                                map_decay, 584, 585, 583, used: 585, 585, 586
       630, 630, 630, 630, 631, 636, 637, 671,
                                                map\_fermion\_lines, 365, used:
                                                map\_index, 317, used: 317
       671, 671, 659, 661, 734, 711, 711, 711,
       615, 664, 664, 677, 677, 678, 678, 678,
                                                map_indices, 343, 343, 345, 350, 356, 333,
       678, 678, 678, 679, 679, 679, 679, 680,
                                                       334, 365, used: 345, 356
       680, 681, 685, 685, 685, 685, 685, 685,
                                                map\_indices\_dirac, 350, used: 350, 350
       685, 686, 686, 686, 686, 686, 687, 689,
                                                map\_indices\_scalar, 349, 349, 350, 335, used:
       691, 702, 702, 702, 702, 702, 703, 704,
       707, 708, 709, 709, 695, 695, 696, 8, 8, 14,
                                                map_indices_vector, 349, 349, 335, used: 350
       14, 15, 18, 20, 21, 22, 22, 27, 27, 27, 28,
                                                map\_nodes, 33, 37, 31, used:
       28, 38, 45, 49, 50, 586, 587, 587, 587, 588,
                                                map\_numerators, 345, used: 345
       ??, ??, ??, ??, 590, 590, 591, 591, 591,
                                                map\_position, 341, 342, 333, used:
       591, 591, 593, 289, 289, 290, 290, 290,
                                                map_primitive, 318, 322, 323, used: 320, 324
       291, 292, 293, 293, 295, 295, 296, 310,
                                                map\_ratios, 345, used: 345, 346, 348
                                                Map_S (sig), 658, 656, used: 658, 660, 657
       312, 312, 312, 312, 313, 319, 320, 324,
       324, 324, 324, 325, 325, 330, 330, 336,
                                                map\_spinor, 317, used: 318
       336, 336, 338, 339, 340, 342, 342, 344,
                                                map\_summands, 345, used: 345, 346
       344, 344, 345, 345, 345, 345, 346, 346,
                                                map_tensor, 316, 320, 322, 324, used: 324
                                                map\_vector, 317, used: 318
       346, 348, 348, 355, 355, 56, 56, 60, 61, 62,
       62, 62, 62, 63, 63, 66, 66, 67, 278, 281,
                                                mask, 49, used: 49, 50, 53, 53, 54
       281, 281, 281, 282, 282, 283, 283, ??, ??,
                                                mask2, 49, used:
       mask21, 49, used: 49
       mask5, 49, used: 49
       ??, ??, ??, ??, ??, ??, ??, ??, 597,
                                                maskb, 49, used: 49
       598, 598, 598, 599, 599, 599, 599, 599,
                                                maskd, 49, used: 49
                                                maskr, 49, used: 49
       600, 600, 600, 600, 600, 600, 600, 601,
       602, 602, 603, 603, 604, 605, 606, ??, ??,
                                                mask_in, 51, used: 52, 52, 52, 52, 52
       mask\_in1, 51, used: 51, 52, 52, 52
       mask\_in2, 51, used: 51
```

```
Mass, 292, 349, 349, ??, ??, ??, ??, ??, ??, ??,
                                              mem, 645, 647, 651, 658, 658, 659, 660, 661,
       ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                                      35, 36, 39, 644, 656, 656, 657, used:
       285, 335, used: ??, 349, 350, 352, ??, ??,
                                                      ??, 647, 651, 631, 631, 672, 659, 661, 708,
       ??, ??, ??, ??, 300
                                                     36, 37, 37, 37, 39, ??, ??, ??, 592, 296,
MASS, ??, ??, used: ??, 297
                                                     355, 64, 64, 65, 65, ??, ??, 599, 599
MASS (camlyacc token), 298, used: 300, 300
                                                     646, 589, 591, 589, used: 646, 647, 647,
                                              merge,
massive, 212, 213, used:
                                                      647, 647, 604
massive (type), 212, 212, used: 212, 212, 213,
                                              merge_overlapping, 281, used: 281, 281
                                              merge_partitions, 282, used: 282
       213
Massive, 212, 213, used:
                                              merge_sets, 591, used: 591
Massive_Vector, 178, used: ??, ??, ??, ??, ??,
                                              Metric, 348, 349, 335, used: 349, 349, 352
                                              ??, ??, ??, ??, ??
                                                      ??, ??, ??, ??, used: ??, ??, ??, ??, ??,
                                                      massless, 212, 213, used:
massless (type), 212, 212, used: 212, 213
                                                      Massless, 212, 213, used:
mass_symbol, ??, ??, ??, ??, ??, ??, ??, ??,
                                                     ??, ??, ??, ??
       mhm, ??, used: ??, ??, ??
       611, 211, used: ??, ??, ??, ??, ??
                                              min, 23, used: 626, 696, 696, 14, 335
mass\_symbol (label), ??, ??, 211, used:
                                              minimal_fusion_tower, ??, used: ??
Matched, 345, used: 345
                                              minus. 364. used:
matching, 45, used:
                                              Minus, 364, used:
match_all, 637, 635, used:
                                              MINUS, ??, ??, ??, ??, ??, used: ??, ??, ??,
match_coupling, 65, used: 66, 66
                                                      297, 331
match_coupling_wildcard, 66, used: 66
                                              MINUS (camlyacc token), 298, 331, 361, used:
match\_first, 637, 635, used:
                                                      299, 301, 302, 303, 332, 332
mathematica_symbol, ??, ??, used: ??, ??, ??,
                                              minus_gauge4, ??, ??, ??, ??, ??, ??, ??, ??,
                                                      ??, 331
matmul, 731, 731, used:
                                                      ??, used: ??, ??, ??, ??, ??, ??, ??,
matmulv, 731, 731, used: 709
                                                      matrix (type), 215, used: 215, 215, 219, 220, 220
                                                      ??, ??, ??, ??, ??, ??
Matter, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                              minus_i, 214, 215, 215, 213, used: 215, 220,
       ??, ??, ??, ??, ??, 212, used: ??, ??,
                                                      220, 221, 221, 221, 221
       minus_one, 214, 215, 215, 213, used: 215, 219,
       ??, ??, ??
                                                     220, 220, 220, 220, 220, 221
matter_field, ??, ??, ??, ??, ??, ??, ??, ??,
                                              min_max_rank, 35, 36, 37, 39, 39, 33, used: 37,
       ??, ??, ??, ??, ??, ??, 212, used: ??,
       mismatch, 45, 50, used: 45, 45, 46, 46, 46, 47,
       ??, ??, ??, ??, ??, ??, ??
                                                     47, 47, 48, 50, 50, 51, 51, 51, 51, 52
matter_field (type), ??, ??, ??, ??, ??, ??, ??,
                                              Mismatch (exn), 621, 622, 44, 45, 50, ??, ??,
       ??, ??, ??, ??, ??, ??, ??, ??, ??, 212,
                                                      620, 41, 125, ??, used:
       used: ??, ??, ??, ??, ??, ??, ??, ??,
                                              Mismatched_arity (exn), 8, 9, 10, 12, 14, 6,
       used:
       ??, ??, ??, ??, ??, ??, ??, 212, 212
                                                    620, 622, 620, used: 622
                                              Miss.
maxabsval, 731, used: 732
                                              Missing\_Space (exn), 307, 309, used:
max_arity, 7, 8, 10, 11, 13, 13, 15, 21, 27, 606,
                                              Mixed23 (module), 11, 22, ??, 606, 6, 16, 124,
       5, 7, used: 13, 14, 15, 21, 27
                                                      ??, 596, used: 22, ??, 606, ??, ??, 607,
max\_clines, 444, used: 445
                                                      max_degree, ??, ??, ??, ??, ??, ??, ??, ??,
       ??, ??, ??, ??, ??, 611, 16
       609, 210, used: ??, ??, ??, ??, 606, ??
                                              Mixed23 (sig), 11, 6, used: 6
max_degree_of_vertices, ??, used: ??
                                              Mixed23_Majorana (module), ??, 606, 124, ??,
max_generations, ??, used: ??
                                                      596, used: 606, ??
max_subtree, 18, 20, 21, 22, 22, 27, 28, 16,
                                              Mixed23_Majorana_vintage (module), 606, 596,
       used: 22, 22, ??
                                                      used: ??, ??, ??, ??, ??, ??, ??, ??,
maybe_backtrace, ??, used: ??, ??
                                                      ??
Maybe_Graded (module), 36, used: 39, 40
                                              mixed_fold_left, ??, used: ??
maybe_parentheses, 339, used: 338, 339
                                              mk\_and, 56, 55, used: ??, 62, 58
maybe_read, 621, 622, 620, used:
                                              mk\_any\_flavor, 56, 55, used: ??, 58
```

```
mk\_false, 56, 55, used:
                                               momentum_list (camlyacc non-terminal), 58,
mk\_gauss, 56, 55, used: ??, 58
                                                      used: 58, 58
mk\_gauss\_not, 56, 55, used: ??, 58
                                               momentum\_to\_string, 62, used: 62
                                               momentum\_to\_TeX, 600, used: 600, 600, 600,
mk\_off\_shell, 56, 55, used: ??, 58
mk\_off\_shell\_not, 56, 55, used: ??, 58
                                                      600
mk\_on\_shell, 56, 55, used: ??, 58
                                               MOML, 180, used:
mk\_on\_shell\_not, 56, 55, used: ??, 58
                                               MOMR, 180, used:
mk\_true, 56, 55, used: ??, 58
                                               Mono (sig), 7, 5, used: 8, 10, 6, 6
mk_{-}x_{-}flavor, 56, 55, used: ??, 58
                                               MSSM (module), ??, ??, used: ??, ??, ??, ??
mk_x_vertex, 56, 55, used: ??, 58
                                               MSSM\_flags (sig), ??, ??, used: ??, ??, ??, ??,
                                                      ??, ??, ??, ??
Model (module), 208, 378, used: ??, ??, ??, ??,
       ??, 589, 324, 60, 278, ??, ??, ??, ??, ??,
                                               MSSM\_goldstone (module), ??, ??, used:
                                               MSSM_Grav (module), ??, ??, used: ??
MSSM_Hgg (module), ??, ??, used: ??
       ??, ??, ??, ??, 596, 606, 606, 606, ??, ??,
       ??, ??, 247, ??, ??, ??, ??, ??, 589, 303,
       378, 59, 250, 250, 278, 124, 126, 126, ??,
                                               MSSM\_no\_4 (module), ??, ??, used: ??
       ??, ??, 596, 596, 596, ??, ??, ??, ??, ??,
                                               MSSM\_no\_4\_ckm (module), ??, ??, used: ??
       ??, ??, ??, ??, ??, ??, ??, ??
                                               MSSM_no_goldstone (module), ??, ??, used:
Modelfile (module), 313, used:
                                               MT (module), 597, used: 597
Modelfile_Test (module), 313, 303, used:
                                               mu, 326, used: 327, 327, 327, 327
Modellib_BSM (module), ??, ??, used: ??, ??,
                                                    ??, ??, used:
                                               Mu,
       mul, 214, 216, 715, 716, 717, 717, 718, 214,
       ??, ??, ??, ??, ??, ??, ??
                                                      used: 216, 216, 217, 217, 219, 219, 346
Modellib_MSSM (module), ??, ??, used: ??, ??,
                                               mult, 291, 285, used: 292, ??, 301
                                               Multi (module), ??, 126, ??, used: 597, 247
       ??, ??
Modellib_NMSSM (module), ??, ??, used: ??,
                                               Multi (sig), ??, 125, ??, used: ??, 126, ??
                                               multinomial, 676, 673, used:
       ??, ??
Modellib_NoH (module), ??, ??, used: ??, ??
                                               multiplicity, 24, ??, ??, 125, ??, used: 24, 25, 25
Modellib_PSSSM (module), ??, ??, used: ??
                                               multiplicity (field), ??, used: ??, ??
Modellib_SM (module), ??, ??, used: 606, ??, ??, ??, ??, 607, ??, ??, ??, ??, ??, ??, ??, ??,
                                               multiply, 630, 330, 346, 624, 329, used: ??,
                                                      346, 346, 332
       ??, ??, ??, ??, ??, ??, ??, ??, ??
                                               multiply1, 346, used: 346
Modellib_WZW (module), ??, ??, used: ??
                                               multiply2, 346, used: 346
Modellib_Zprime (module), ??, ??, used: ??, ??
                                               multi_choose, 679, 674, used:
Modeltools (module), ??, ??, used: 61, 65, 597,
                                               Multi_Maker (sig), ??, 126, ??, used: 126, ??
       multi_split, 677, 674, used: 677, 679
                                               multi_split', 677, used: 677, 677, 677
       ??, ??, ??, ??, ??, 608
                                               Muon, ??, used: ??, ??, ??, ??
modname, ??, used: ??, ??, ??, ??
                                               Mutable (module), ??, ??, used:
mod_float2, 616, used: 616
                                               Mutable (sig), 211, used: ??, ??, ??
mom, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                               M_-N, ??, used:
       used: ??, ??, ??, ??, ??, ??, ??, ??,
                                               M\_SF, ??, used:
       M_{-}U, ??, used:
       ??, ??, ??, ??
                                               M_{-}V, ??, used:
MOM, 180, used:
                                               n (field), 586, used:
                                               MOM5, 180, used:
                                                       ??, ??, ??, ??, ??, ??, ??, used: ??,
Momenta, 246, used: 601
momentum, ??, ??, ??, 122, ??, used: 590,
                                                      ??, ??, 606
                                                      momentum (field), ??, ??, used: ??, ??, ??, ??,
                                                      ??, ??, ??, ??, ??
                                                      momentum (type), 584, 584, 583, used: 584,
                                                      585, 585, 586, 583, 583, 583
momentum (camlyacc non-terminal), 58, used:
                                                      Momentum (module), 44, 41, used: 584, 589, 60,
       ??, ??, ??, ??, ??, 596, 596, 606, 247,
                                                      ??, ??, ??, ??, ??
       583, 589, 59, 124, 126, 126, ??, ??, ??,
                                               N1, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                       momentum_list, ??, ??, ??, 122, ??, used:
                                                      ??, ??, ??
       ??, 598, 598
```

```
N10, ??, used: ??, ??
                                                   Negative\_QCD\_order (exn), ??, used:
N11, ??, used: ??, ??
                                                   nested_to_triple, ??, used: ??
                                                   neu (type), ??, ??, ??, used: ??, ??, ??, ??, ??,
N2, ??, ??, wsed: ??, ??, ??, ??, ??, ??, ??,
        ??, ??, ??
                                                   Neutral, 292, 307, 307, 285, used: ??, 310, 299
                                                   NEUTRAL, ??, ??, used: ??, 297
N3,
    ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
       NEUTRAL (camlyacc token), 298, used: 299
                                                   Neutralino, ??, ??, ??, used: ??, ??, ??, ??, ??,
       ??, ??
                                                           ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
       ??, ??, ??, ??, ??
    ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??,
                                                   neutral_A, ??, used: ??
N5,
        ??, ??
                                                   neutral_currents, ??, ??, ??, ??, ??, ??, ??, ??,
     ??, used: ??, ??
                                                           N6,
                                                           ??, used: ??, ??, ??, ??, ??, ??, ??,
N7,
    ??, used: ??, ??
N8, ??, used: ??, ??
                                                           N9, ??, used: ??, ??
                                                           ??
name (field), 292, 293, 295, 295, 313, 285,
                                                   neutral_currents', ??, used: ??
       286, 287, 287, 359, 378, 378, used: 293,
                                                   neutral_heavy_currents, ??, ??, ??, ??, used: ??,
       293, 295, 295, ??, 309, 309, 310, 310, ??,
                                                           ??, ??, ??
                                                   neutral_sfermion_currents, ??, ??, ??, used: ??,
       299, 300, 300, 300, 362
name (type), 292, 285, 358, used: 292, 285, 358,
                                                           ??. ??
       359, 359
                                                   neutral_sfermion_currents', ??, ??, ??, used: ??,
                                                           ??, ??
name (camlyacc non-terminal), 302, 362, used:
       302, 362, 362, 362, 363, 363, 363, 363
                                                   neutral_up_type_currents, ??, used: ??
       359, used: ??, 362
                                                   neutral_Z, ??, ??, used: ??, ??
Name,
named (field), 343, used: 343, 343
                                                   neutral_Z_1, ??, used: ??
                                                   neutral_Z_2, ??, used: ??
named_summation, 342, 343, 333, used: 352
names (camlyacc non-terminal), 363, used: 362,
                                                   neutr_lqino_current, ??, used: ??
       363, 364
                                                   newline, 446, 444, used: 448
Name_List, 359, used: ??, 362
                                                   NH, ??, used: ??, ??, ??, ??, ??
name\_to\_string, 292, used: 293
                                                   nl, 572, ??, used: 572, ??
Nary (module), 13, 20, ??, 606, 7, 16, 124, ??,
                                                   nlist, ??, used: ??, ??
                                                   NMSSM (module), ??, ??, used: ??
       596, used: 15, 21, 21, 27, 27, ??, ??, 606,
       ??, ??, 16, 18
                                                   NMSSM_CKM (module), ??, ??, used: ??
                                                   NMSSM_flags (sig), ??, ??, used: ??, ??, ??,
Nary (sig), 13, 7, used: 7, 7
Nary4 (module), 21, used: 22, 22
Nary_Majorana (module), ??, 606, 124, ??,
                                                   NMSSM\_func \text{ (module)}, ??, ??, used: ??, ??, ??
       596, used: 606, ??, ??
                                                   NMSSM\_Hgg \text{ (module)}, ??, ??, used: ??
    Nodd, ??, used: ??
       node, 701, 699, used:
       ??, ??, ??, ??, ??, ??, ??, 607, 608,
                                                   node (field), 707, used: 707
       210, used: ??, ??, ??, ??, 608, 608
                                                   node (type), ??, 33, 33, 34, 34, 35, 36, ??, ??,
nc (label), ??, ??, 211, used:
                                                           30, 30, 32, used: ??, 33, 33, 34, 34, 34,
nc_coupling, ??, ??, ??, ??, ??, ??, ??, ??,
                                                           36, 39, ??, ??, ??, 30, 30, 30, 30, 30, 31,
       ??, ??, ??, ??, ??, ??, used: ??, ??,
                                                           31, 31, 31, 31, 31, 31, 31, 31, 32, 32,
       32, 32, 32, 33, 33, 33
       ??, ??, ??
                                                   Node,
                                                          645, 711, 701, used: 645, 645, 645, 647,
neg, \quad \mathbf{44},\, \mathbf{45},\, \mathbf{50},\, \mathbf{214},\, \mathbf{215},\, \mathbf{715},\, \mathbf{716},\, \mathbf{717},\, \mathbf{718},\,
                                                           648, 649, 649, 649, 711, 701, 702, 702,
       42, 214, used: 45, 46, 47, 48, 50, 51, 51,
                                                           702, 702, 702, 703
       52,\,587,\,215,\,215,\,217,\,218,\,219,\,219,\,60,
                                                          701, 699, used: 701, 707
                                                   nodes,
                                                   Nodes (module), 33, 34, 34, 35, ??, 30, 32,
       64, 64, 65, ??
     ??, 203, used: ??, ??, ??, ??, ??, ??, ??,
Neq,
                                                           used: 33, 34, 34, 35, 36, 36, 37, 37, 38, 39,
       ??, 30, 32, 33, 33
       node\_to\_string, 703, used: 703
       node_with_tension (type), 707, used:
                                                   NoH (module), ??, ??, used: ??
       ??, ??, ??, ??, ??, ??, ??
                                                   NoH_flags (sig), ??, ??, used: ??, ??, ??, ??, ??
neg', 45, used: 45, 45
                                                   NoH_k\_matrix (module), ??, ??, used: ??, ??
Negative (exn), 44, 46, 51, 41, used:
Negative\_EW\_order (exn), ??, used:
                                                   non\_decreasing, 694, used: 695
Negative\_QCD\_EW\_orders (exn), ??, used:
                                                   non\_increasing, 694, used: 694
```

```
normalization, 696, 693, used: 697
                                                  NOT, ??, ??, used: ??, 57
                                                  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
NOT (camlyacc token), 57, used: 57, 58, 58
not_arg_or_token_list (camlyacc non-terminal),
                                                  ??, 611, ??, ??, ??, ??, used: 606, ??,
                                                  ??, ??, 607, ??, ??, ??, ??, ??, ??, ??,
      303, used: 301
                                                  Not_invertible (exn), 708, used:
not_positive, 341, used: 342
                                                  Not_Square (exn), 731, 731, used:
                                                  ??, ??, ??, ??, ??, ??, ??, 611, ??,
no_cascades, 60, 63, 59, used: 67, 606
no_exclusions, ??, ??, ??, ??, ??, ??, 122, 125,
                                                  ??, ??, ??
      ??, ??, used: ??, ??, 603, 604, 606
                                           Ocaml (module), 448, used:
No_Tags (module), ??, used: ??
                                           odd, 684, 685, 686, 683, used:
No_termination (exn), 8, 9, 12, 14, 6, used:
                                           of2, 8, 8, 11, 11, 13, 13, 6, 6, 7, used: 9, 9, 13,
nu, 326, used: 327
                                                  13, 15, 20, 22, 28
null, 214, 215, 715, 716, 716, 718, 718, 213,
                                           of2_kludge, 7, 9, 11, 13, 15, 6, used: ??
      used: 249, 215, 217, 217, 217, 217, 346
                                           of3, 10, 10, 11, 11, 13, 13, 6, 6, 7, used: 11, 13,
Null (module), 249, 248, used: ??, ??, ??, 608,
                                                  22, 22
                                           offset_map, 684, used: 684
null_coupling, ??, ??, ??, ??, 126, ??, used: ??,
                                           OFFSHELL, ??, ??, used: ??, 57
                                           OFFSHELL (camlyacc token), 57, used: 58
      ??
null\_wf, ??, ??, ??, ??, 126, ??, used: ??, ??,
                                           Offspring (module), 36, used: 37, 37, 37, 37, 37,
      ??. ??
                                                  37, 37, 37, 38, 38, 38, 38, 38
numerator (field), 378, 378, used:
                                           Off_shell, 55, 61, 55, used: 56, 61
                                           Off_shell_not, 55, 61, 55, used: 56, 61
numerators, 348, used: 348
                                           of_array, 684, 684, 686, 683, used:
num_cells_diagram, 695, 692, used: 696
num\_cells\_tableau, 695, 693, used:
                                           of_expr, 340, 343, 343, 346, 348, 352, 357,
num_columns, 637, 635, used: 639, 639, 639
                                                  332, 334, 334, used: 340, 346, 348, 348,
num_{-}ext, 24, used: 24
                                                  348
num\_prop, 24, used: 24
                                           of \_expr', 340, 346, used: 340, 346
num\_rows, 637, 635, used:
                                           of \_expr1, 356, used: 357
num_standard_tableaux, 696, 692, used: 697
                                           of_int, 22, 23, 315, 315, 315, 323, 17, used:
N_{-}K1, ??, used: ??, ??, ??
                                                  23, 26, 27, 324, 326, 326, 326, 326, 326,
N_{-}K2, ??, used: ??, ??, ??
                                                  326, 326, 327, 327, 327, 327
O, 293, 316, 320, ??, ??, ??, ??, ??, ??, ??, ??,
                                           of_ints, 44, 45, 50, 41, used: 48, 54, 587, 587,
      ??, ??, ??, ??, ??, ??, ??, used: 293,
                                                  588, 61, ??, ??
      320, 320, 321, 321, ??, ??, ??, ??, ??, ??,
                                                  ??, ??, ??, 735, 736, 667, 668, 668,
                                           of \_list,
      669, 684, 684, 686, 13, 13, ??, 735,
                                                  667, 667, 683, 7, used: ??, 736, 736,
      736, 737, 737, 630, 630, 636, 669, 671,
      671, 681, 684, 685, 685, 685, 685, 685,
      685, 686, 686, 686, 686, 686, 686, 686,
      686, 686, 687, 687, 687, 687, 688, 688,
      ??, ??, ??, ??, ??, ??, ??, ??, ??, ??.
                                                  688, 688, 688, 689, 708, 709, 709, 696,
      696, 15, 21, 27, 324, 282, ??, ??, 603
      ??, ??, ??, ??, ??, ??, ??, ??, ??, ??.
                                           of_lists, 735, 736, 671, 671, 684, 685, 686,
      735, 670, 683, used: 736, 737, 737, 688,
      324, 66, 282
      of_lists_list, 684, used: 685, 686
                                           of\_list\_gap, 687, used: 687
      of\_list\_ok, 687, used: 687
      of\_list\_overlap, 687, used: 687
      of_momenta, 584, 587, 583, used:
      of_momentum, 53, 53, 54, 43, used: 54, 590,
                                                  591, 592
      of\_propagator\_UFO, 378, used:
      of_string, ??, ??, 311, 312, 336, 343, 348, 61,
      ??, 332, 334, used: 313, 336, 348, 61, 62
      of_strings, 336, 343, 348, 332, 334, used: 348
      ??, ??, ??, ??, ??, ??, ??, ??
                                           of_string_and_back, 311, 313, used: 313, 313,
O (module), 606, ??, ??, ??, 607, ??, ??, ??,
      of_string_list, 60, 62, 59, used: 603, 604
```

```
of\_subarray, 626, 623, used:
                                                     Omega\_SSC\_AltT (module), ??, used:
of_vertices, ??, ??, ??, used: ??, 66, ??, ??, ??,
                                                     Omega\_SYM (module), 607, used:
        Omega_Template (module), ??, used:
        Omega_THDM (module), ??, used:
        ??, 609
                                                     Omega\_THDM\_CKM (module), ??, used:
ok, 324, used: 327, 327, 327, 328, 328
                                                     Omega\_THDM\_CKM\_VM \text{ (module)}, ??, used:
       343, 343, 348, 355, 358, 334, 334,
                                                     Omega\_THDM\_VM \text{ (module)}, ??, used:
omega,
                                                     Omega_Threeshl (module), ??, used:
        used: 348
Omega (module), 596, 596, used: 606, ??, ??,
                                                     Omega_Threeshl_nohf (module), ??, used:
        ??, 607, ??, ??, ??, ??, ??, ??, ??, ??,
                                                     Omega_UED (module), ??, used:
                                                     Omega_UFO (module), ??, used:
        Omega_UFO_Dirac (module), ??, used:
        Omega_UFO_Majorana (module), ??, used:
        ??, ??, ??, ??, ??, ??, 611, ??, ??,
                                                     Omega_WZW (module), ??, used:
Omega_Xdim (module), ??, used:
        ??, ??, ??
                                                     Omega_Zprime (module), ??, used:
Omega\_AltH \text{ (module)}, ??, used:
Omega_Dirac (module), ??, used: ??
                                                     Omega\_Zprime\_VM \text{ (module)}, ??, used:
Omega\_GravTest \text{ (module)}, ??, used:
                                                     one, 22, 23, 214, 215, 215, 17, 213, used: 24,
Omega_HSExt (module), ??, used:
                                                             25, 26, 27, 215, 215, 215, 219, 219, 220,
                                                             220, 220, 220, 220, 220, 221, 65
Omega\_HSExt\_VM \text{ (module)}, ??, used:
                                                     one\_compatible, 60, used: 60, 64
Omega_Littlest (module), ??, used:
                                                     Only_Insertion, 178, used: ??, ??, ??, ??, ??,
Omega\_Littlest\_Eta (module), ??, used:
Omega\_Littlest\_Tpar  (module), ??, used:
                                                             ??, ??
Omega_Majorana (module), ??, used: ??
Omega_MSSM (module), ??, used:
                                                     only\_wf, 61, used: 61
Omega\_MSSM\_CKM \text{ (module)}, ??, used:
                                                     ONSHELL, ??, ??, used: ??, 57
                                                     ONSHELL (camlyacc token), 57, used: 58
Omega\_MSSM\_Grav \text{ (module)}, ??, used:
Omega_MSSM_Hgg (module), ??, used:
                                                     on_shell, 60, 63, ??, ??, ??, ??, 59, 124, ??,
Omega\_NMSSM (module), ??, used:
                                                             used: ??, ??
Omega\_NMSSM\_CKM  (module), ??, used:
                                                     on_shell (field), 63, ??, ??, used: 63, 63, 67, ??,
Omega\_NMSSM\_Hgg  (module), ??, used:
                                                             ??, ??, ??
Omega\_NoH\_rx \text{ (module)}, ??, used:
                                                     On_shell, 55, 61, 55, used: 56, 61
Omega\_PSSSM (module), ??, used:
                                                     On_shell_not, 55, 61, 55, used: 56, 61
Omega\_QCD (module), ??, used:
                                                     openmp, 614, used:
Omega\_QCD\_VM \text{ (module)}, ??, used:
                                                     open_file, 617, 616, used:
                                                     Open_File, 616, used: 617
Omega\_QED (module), 606, used:
Omega\_QED\_VM (module), ??, used:
                                                     open\_in\_bin\_first, 621, used: 622
Omega_Simplest (module), ??, used:
                                                     open\_out\_bin\_last, 621, used: 622
Omega\_Simplest\_univ \text{ (module)}, ??, used:
                                                     optional, 289, used: 289
Omega\_SM (module), 607, used:
                                                     optional_scripts (camlyacc non-terminal), 302,
Omega\_SM\_ac (module), ??, used:
                                                             used: 302, 302
                                                     options, ??, ??, ??, ??, ??, ??, ??, ??, ??,
Omega\_SM\_ac\_CKM \text{ (module)}, ??, used:
Omega\_SM\_CKM (module), ??, used:
                                                             Omega\_SM\_CKM\_VM (module), ??, used:
Omega\_SM\_dim6 (module), ??, used: Omega\_SM\_Higgs (module), ??, used:
                                                             ??, ??, ??, 607, 211, 246, 122, 125, ??,
                                                             ??, used: ??, ??, ??, ??, 601, ??, ??
Omega\_SM\_Higgs\_CKM \text{ (module)}, ??, used:
                                                     Options (module), 615, 615, used: ??, ??, ??,
Omega\_SM\_Higgs\_CKM\_VM \text{ (module)}, ??,
                                                             ??, ??, 601, ??, ??, ??, ??, ??, ??, ??,
                                                             used:
Omega\_SM\_Higgs\_VM \text{ (module)}, ??, used:
                                                             ??, ??, ??, ??, ??, ??, 607, 211, 246,
Omega_SM_Majorana (module), ??, used:
                                                             122, 125, ??, ??
                                                     opt\_array\_to\_list, 597, used: 597
Omega\_SM\_Majorana\_legacy  (module), ??,
                                                     Ord (sig), 33, 29, used: 33, 34, 34, 34, 35, 35, 35,
       used:
Omega\_SM\_rx \text{ (module)}, ??, used:
                                                             30, 30, 32, 32
Omega\_SM\_top \text{ (module)}, ??, used:
                                                     order (camlyacc non-terminal), 363, used: 363
Omega\_SM\_top\_anom (module), ??, used:
                                                     ordered_multi_choose, 679, 674, used:
Omega\_SM\_tt\_threshold (module), ??, used:
                                                     ordered_multi_split, 677, 674, used: 679
Omega\_SM\_ul \text{ (module)}, ??, used:
                                                     ordered_partitions, 678, 673, used:
Omega\_SM\_VM \text{ (module)}, ??, used:
                                                     ordered_partitions', 678, used: 678, 678
Omega\_SSC \text{ (module)}, ??, used:
                                                     ordered_split, 676, 674, used: 677, 678
Omega\_SSC\_2 (module), ??, used:
                                                     ordered_split_unsafe, 676, used: 676, 677
```

```
Ordered_Type (sig), 670, 670, used: 671, 670
                                                   P12, 349, 349, 335, used: 349, 350, 350
P2, 349, 349, ??, ??, 180, 335, used: 349, 350,
        350, ??, ??, ??, ??, ??, ??, ??, ??, ??.
        ??, ??, ??, 607, used:
                                                           ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
orders (type), ??, ??, ??, ??, ??, ??, ??, ??,
                                                        ??, 598, used: ??, ??, 598, 600
                                                   p2s,
        P3, 22, ??, used: 22, ??, ??, ??
        ??, ??, ??, ??, 607, used:
                                                        22, ??, used: 22, ??, ??, ??
                                                   P4,
                                                   P5, ??, used: ??, ??, ??
orders (camlyacc non-terminal), 363, used: 363,
                                                   P6, ??, used: ??, ??, ??
        363
Order_Dictionary, 359, used: ??, 363
                                                   P7, ??, used: ??, ??, ??
order_flavor, ??, used: ??
                                                   pack, 341, 342, 333, used: 342, 355
order_spin_table, ??, used: ??
                                                   pack_map, 591, used: 591
order_spin_table1, ??, used: ??, ??
                                                   pack_tree, 590, used: 591
Order_Tags (module), ??, used:
                                                   pair, ??, used: ??
order_wf, ??, ??, used: ??, ??, ??, ??, ??, ??
                                                   pairs, 629, 690, ??, ??, 623, 690, used: 9, 13,
18, 28, ??, ??, ??, ??, ??, ??, ??, ??
       ??, ??, ??, ??, ??, 212, used: ??, ??, ??,
                                                   pairs', 628, 690, used: 628, 629, 690, 690
        pair_or_triple (type), 11, 6, used: 11
        ??, ??, ??
                                                   parameter (camlyacc non-terminal), 300, used:
other (field), 311, used: 311, 311, 311, 312
other (type), ??, ??, ??, ??, ??, ??, ??, ??,
                                                   Parameter, 293, 295, 296, 307, 307, 286, 287,
        ??, ??, ??, ??, ??, ??, <mark>212</mark>, used: ??,
                                                           287, used: 296, ??, 310, 299, 300
        Parameter (module), 293, 286, used: 295, 296,
        296, ??, 307, 287, 287, 298
       ??, ??, ??, ??, 212, 212
                                                   PARAMETER, ??, ??, used: ??, 297
PARAMETER (camlyacc token), 298, used: 300
                                                   parameters, ??, ??, ??, ??, 305, ??, ??, ??, ??,
        ??, ??, ??, ??, ??, 212, used: ??, ??, ??,
        ??, ??
OUnit (module), ??, ??, used: ??, 736, 736, 446,
                                                           ??, 609, 210, used: ??, ??, ??, 306, ??,
        642, 631, 637, 681, 687, 687, 697, 697,
       214, 217, 303, 304, 313, 327, 358, 735,
                                                   parameters (field), 378, used:
        444, 640, 625, 635, 715, 675, 684, 693,
                                                   parameters (label), ??, ??, 211, used:
        214, 303, 335, 365, 423, 379
                                                   parameters (type), 208, used: 210, 211
OUnitDiff (module), ??, ??, used:
                                                   parameters_to_channel, 247, used: 603
out (field), 590, used:
                                                   parameter_attribute (camlyacc non-terminal),
outer, 665, 663, used:
                                                           300, used: 300
outer (field), 705, 700, used: 705, 600, 600, 600
                                                   parameter_attributes (camlyacc non-terminal),
                                                           300, used: 300, 300
outer\_self, 665, 663, used:
outgoing, 44, 47, 48, 52, 52, 278, 278, ??, ??,
                                                   Parents (module), 36, used: 37, 37, 37, 37, 37,
        ??, ??, 43, 43, 277, 123, ??, used: ??,
                                                           37, 37, 37, 37, 37, 37, 37, 37, 38, 38,
        ??, ??
                                                           38. 38. 38. 39.
outgoing (field), ??, ??, used: ??, ??, ??, ??, ??
                                                   parse, 615, 615, 365, used: ??, 601
                                                   Parser_Test (module), 304, 303, used:
Out\_of\_bounds (exn), 630, 624, used:
Out\_of\_range (exn), 315, 315, used:
                                                   parse_decay, 278, 279, 277, used: 602
overlapping, 281, used: 281
                                                   parse_directory, 378, 378, used:
overlapping_flavor_sums, 281, used: 282
                                                   parse_error, ??, 304, ??, ??, ??, 57, 298, 331,
Overlapping_indices (exn), 630, 624, used:
                                                           361, used: ??, ??, ??, 304, 304, ??, ??,
p (field), 44, used: 45, 45, 46, 46, 46, 47, 47, 47
                                                           ??, ??, ??, ??, ??, 298, 299, 299, 299,
p (type), 60, 60, ??, ??, ??, ??, 59, 122, ??,
                                                           299, 301, 301, 301, 301, 302, 302, 331, 361
        used: 60, 60, 63, ??, ??, ??, ??, ??, ??,
                                                   parse_file, 304, 303, 377, used: 314
        ??, ??, ??, 59, 59, 59, 59, 59, 122, 124,
                                                   parse_process, 278, 279, 277, used: 279, 279
        126, ??, ??, ??
                                                   parse_scattering, 278, 279, 277, used: 602
P, 316, 318, 348, 349, 180, 335, used: 318,
                                                   parse_string, 303, 303, 377, used: 304, 304, 305,
        349, 349, 352, ??, ??, ??, ??, ??, ??, ??,
                                                           306, 312
        ??, ??, ??, ??, ??, ??
                                                   parse_token, 306, used: 306
P (module), 736, ??, 307, 596, 298, used: 736,
                                                   partial, 718, used:
        736, 736, 737, 737, 737, 737, ??, 310, 310,
                                                   Partial (module), 735, 735, used: 324
        597, 597, 299, 300
                                                   particle (field), 311, used: 311, 311, 311, 312
P1, ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??,
                                                   particle (camlyacc non-terminal), 299, used: 299
        ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                   Particle, 295, 296, 287, 287, used: 296, ??, 299
```

```
Particle (module), 292, 285, used: 295, 296, 296,
                                          Phasespace (module), 583, 583, used:
      ??, 307, 287, 287, 298
                                          phase_space_braket, ??, used: ??
particles, 305, used: 306
                                          phase_space_channels, ??, ??, 124, ??, used: 604
particles (field), 378, used:
                                          phase_space_channels_flipped, ??, ??, 124, ??,
particle_attribute (camlyacc non-terminal), 300,
                                                 used: 604
      used: 299
                                          phase_space_channels_transformed, ??, used: ??,
particle_attributes (camlyacc non-terminal), 299,
      used: 299, 299
                                          Phi, ??, ??, 607, used: ??, ??, ??, ??, ??, ??,
partition, 20, 27, 60, 63, 59, used: 672, 21, 27,
                                                 ??, 607, 609, 609
                                               Phi0.
                                                 ??, ??, ??, ??, ??, ??, used: ??, ??,
partition (field), 63, used: 63, 63
partition (type), 18, 18, 20, 22, 22, 27, 27, 16,
                                                 used: 18, 16, 16
                                                 Partition (module), 690, 690, used: 9, 11, 13,
                                                 15, 18, 20, 27, 28
                                                 partitioned_sort, 630, 624, used: 282
partitions, 678, 18, 19, 21, 22, 22, 27, 28, 673,
                                                16, used: 678, 20, 21, 22, 22, 27, 27, 28
                                                 ??, ??, ??, ??, ??, ??, ??, ??
partitions', 677, 18, 21, 27, used: 677, 678, 678,
                                          Phi3 (module), ??, ??, used:
                                          Phi4 (module), ??, ??, used:
      18, 19, 21, 21, 27, 27
                                          phiggs (type), ??, ??, used: ??, ??, ??, ??
partition_to_string, 67, used: 67
                                                 ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
part_to_string. 66, used: 67
                                          PHiqqs,
path (type), ??, ??, used: ??, ??, ??, ??, ??
                                                 PBP, 180, used: ??, ??, ??, ??, ??, ??, ??
                                                 ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
pcompare, 625, 635, 675, 694, 288, 303, ??,
                                          phiggs_neutr, ??, ??, used: ??, ??
      used: 628, 629, 630, 630, 630, 631, 636,
                                          680, 681, 681, 694, 696, 291, 321, 321,
                                                ??, ??, ??, ??, ??, ??, used: ??, ??,
      322, ??
                                                pdq,
      ??, ??, ??, ??, ??, ??, ??, ??, 611,
                                                 210, used: ??, ??, ??, ??, ??, ??, ??,
                                                 ??, ??, ??, ??, ??, ??, ??, ??, ??
      ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                ??, used: ??, ??, ??, ??
                                          Phino,
pdg (label), ??, ??, 211, used:
                                               Phip,
                                                 ??, ??, ??, ??, ??, ??, used: ??, ??,
       ??, ??, ??, ??, used: ??, ??, ??, ??, ??,
pdg\_mw,
      ??, ??, ??
                                                 perform_test, ??, ??, used: ??
permutation, ??, ??, used: ??, ??
                                                Permutation (module), 684, 324, 683, used:
                                                324, 324, 422
permutations, ??, ??, 325, used: ??, 326, 327
                                                ??, ??, ??, ??, ??, ??, ??, ??, ??
permutation_symmetry, 26, used: 26
                                          Photon, ??, used: ??, ??, ??, ??
permute, 679, 324, 674, used: 680, 681, 681,
      683, 685, 686, ??, ??, ??, 325
                                          PHS (module), 597, used: 604
                                             668, 668, 669, 283, ??, 666, 667, used: 669,
permute3, ??, used: ??
permute4, ??, used: ??
                                                 669, 669
permuten, ??, ??, used: ??
                                             Pi,
permute_cyclic, 680, 674, used: 681, 685, 686
                                                 ??, ??, ??, ??, ??, used: ??, ??, ??,
permute_cyclic_signed, 680, 674, used: 681
                                                ??, ??
permute_even, 679, 674, used: 685, 686
                                          pivot_column, 732, used: 732
permute_odd, 679, 674, used: 685, 686
                                          plist, ??, used: ??
permute_signed, 679, 674, used: 679, 679, 680,
                                          PLUS, ??, ??, ??, ??, ??, ??, ??, used: ??,
      682, 685, 686
                                                 ??, ??, ??, 57, 297, 331
permute_signed', 679, used: 679, 679
                                          PLUS (camlyacc token), 57, 298, 331, 361,
permute_tensor, 680, 674, used:
                                                 used: 57, 58, 299, 301, 302, 303, 332, 332,
                                                 363, 363, 363
permute_tensor_even, 680, 674, used:
permute_tensor_odd, 680, 674, used:
                                          PM (module), 684, 324, used: 684, 684, 324
permute_tensor_signed, 680, 674, used: 680, 680
                                          Pmap (module), 645, 644, used: 636, 661, 684,
pfx (camllex regexpr), 297, used: 297
                                                 707, 657
```

```
PMap (module), 636, used: 636
                                                     prefixes (camlyacc non-terminal), 302, used: 302,
poles, ??, ??, 124, ??, used: 590, ??
                                                             302, 302
Poles (module), 589, used: 590, 590, 590, 591
                                                     prefix_of_string, 288, 284, used: 289, 311
poles', ??, used: ??
                                                     prefix_to_string, 288, 284, used: 290, 312
                                                     prepend_to_fst_unsorted, 677, used: 677
poles_beneath, ??, used: ??
poles_to_whizard, 590, used: 590
                                                     Pre_Bundle (module), 280, used: 281
Pole_Map (module), 590, used: 590, 591, 591
                                                     Prime, 288, 284, used: 288
Poly (sig), 660, 8, 657, 6, used: 661, 8, 11, 13,
                                                     PRIME, ??, ??, used: ??, 297
        34, ??, ??, ??, ??, ??, 657, 6, 6, 7, 30,
                                                     PRIME (camlyacc token), 298, used: 302
                                                     primes (camlyacc non-terminal), 302, used: 302
Pos. ??, used: ??, ??, ??, ??, ??, ??, ??, ??,
                                                     prime_list (camlyacc non-terminal), 302, used:
        ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                             302, 302
position, 341, 342, 333, used: 342
                                                     primitive (type), 316, 318, 322, 323, used: 316,
positive, 341, used: 342
                                                             320, 322, 324
Positron, ??, used: ??, ??, ??, ??
                                                     primitive_conjspinor_indices, 319, used: 319
                                                     primitive_indices, 322, 324, used: 324
POT, 180, used: ??
pow, 715, 716, 717, used: 346
                                                     primitive_ok, 318, 323, used: 320, 324
Pow, 203, used: 339
                                                     primitive_spinor_indices, 319, used: 319
power, 627, 664, 7, 9, 10, 13, 14, 330, 346,
                                                     primitive_vector_indices, 319, used: 319
                                                     print, 572, ??, 572, ??, used:
        624, 717, 663, 6, 329, used: 627, 632,
        ??, 346, 346, 607, 607, 607, 332
                                                     printf, 446, 444, used: ??, ??, ??, 446, 448, 615,
Power, 329, 338, 329, used: 330, 336, 340
                                                             687, 14, 218, 218, 325, 572, ??
POWER, ??, ??, used: ??, 331
                                                     print_time, 687, used: 689
POWER (camlyacc token), 331, used: 332, 332
                                                     print\_token, 306, used:
                                                     Proc (module), ??, 597, used: ??, ??, 602
power_fold, 7, 9, 10, 12, 14, 6, used: 13, 14, ??
                                                     process (type), 278, 278, ??, ??, 277, 125, ??,
power\_fold2, 12, used: 12
                                                             used: 278, ??, ??, ??, 277, 125, 125, 125,
power_fold23, 12, used: 12
                                                             ??, ??, ??
PowSet (module), 670, 670, used: 66, 282
PowX, 203, used: 339
                                                     Process (module), 278, 277, used: ??, 597
                                                     processes, ??, ??, 125, ??, used:
pp, 214, 217, 717, 214, used:
pp\_comma\_separator, ??, ??, used:
                                                     processes (field), ??, used: ??, ??
pp\_diff, ??, ??, ??, used: ??, ??
                                                     Process_Bundle (module), 281, used: 282
pp_display_blanks, 445, used: 445, 445
                                                     process_sans_color, ??, used: ??, ??
pp_display_newline, 445, used: 445
                                                     process_table, ??, ??, 125, ??, used: 597
pp_flush, 446, 444, used: 446, 447
                                                     process_table (field), ??, used: ??, ??, ??
                                                     process_to_string, ??, used: ??
pp\_fortran\_newline, 445, used: 445
                                                     Prod, 203, used: 339, ??, ??, ??, ??, ??, ??, ??,
pp_newline, 445, 444, used: 446, 447, 447, 447
pp_output_function, 444, used: 445, 445, 445
                                                              pp_printer, ??, ??, ??, ??, ??, used: ??, ??,
                                                             ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
        ??, ??, ??, ??
                                                     product, 671, 694, 7, 9, 10, 12, 14, 214, 217,
pp_print_gen, ??, used: ??, ??, ??
                                                              354, 717, 717, 718, 5, 214, used: 672,
pp_print_sep, ??, ??, used: ??, ??, ??
                                                             694, 696, 9, 10, 218, 219, 219, 354, ??, ??
pp_setup, 445, used: 445, 445, 446, 446, 446
                                                     Product, 291, 329, 338, 285, 329, used: 291,
                                                              330, 336, 338, 340, 346
pp_set_formatter_out_channel, 446, 444, used:
        446, 446
                                                     Product (module), 663, 663, used: 680, 680, 702,
pp_switch_line_continuation, 445, 444, used:
                                                             9, 9, 10, 10, 12, 14, 14, 14, 15, 38, 591,
                                                             346, 346, 61, 280, 280, ??, ??, ??, ??, ??,
        445, 446
pred, 22, 23, 17, used: 642, 625, 625, 625, 626,
                                                             626, 626, 629, 630, 630, 636, 731, 731,
                                                             732, 733, 733, 675, 675, 676, 676, 677,
                                                             677, 678, 685, 685, 685, 685, 685, 686,
                                                             686, 686, 686, 686, 686, 686, 690, 690,
        708, 710, 694, 695, 696, 696, 9, 10, 14, 14,
                                                             15, 21, 23, 24, 25, 26, 27, 27, 27, 28, 50,
                                                             ??, ??, ??, ??, ??, ??, 607, 607, 607
        50, 53, 53, 343, 346, 352, 352, 282, 282,
                                                     product_fold, 7, 9, 10, 12, 14, 5, used: 9, 10, 12,
        ??, 597, 606, ??, ??
                                                              12, 14
prefix (field), 288, 311, 284, used: 288, 290, 311,
                                                     Progress (module), 616, 616, used: ??, ??, ??
        311, 311, 312
                                                     progress_mode (type), ??, used:
prefix (type), 288, 284, used: 288, 311, 284, 284
                                                     progress_option, ??, used: ??, ??
PREFIX, ??, ??, used: ??, 297
                                                     project, ??, used: ??, ??, ??, ??, ??, ??, ??
```

Projection (sig), 668, 666, used: 668, 667

PREFIX (camlyacc token), 298, used: 302

```
project_factors, 355, used: 355
                               ProjM, 348, 349, 334, 364, used: 350, 352
                               ProjP, 348, 349, 334, 364, used: 350, 352
                               ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
prop (field), 24, used: 24, 24, 25, 25, 25
                               propagator, ??, ??, ??, ??, ??, ??, ??, ??,
                               608, 210, 422, used: ??, ??, ??, ??, 599,
                               599, ??, ??
                               propagator (label), ??, ??, 211, used:
propagator (type), 178, used: 210, 211
                               Propagator (module), 378, used:
                               propagators (field), 378, used:
                               Propagator_UFO (module), 378, used: 378
prop_aux, ??, ??, ??, ??, used: ??, ??, ??,
                                Prop\_Col\_Feynman, 178, used:
                               ??, ??, ??, ??, 608
                               ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
Prop\_Col\_Majorana, 178, used:
                           Psi0,
                                ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
Prop\_Col\_Scalar, 178, used:
                               ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
Prop_Col_Unitarity, 178, used:
                           Psi1,
                                ??, ??, ??, ??, ??
Prop_ConjSpinor, 178, used: ??, ??, ??, ??, ??.
    180, used: ??, ??, ??, ??, ??, ??, ??,
                           Psibar.
                                ??, ??, ??, ??, ??, ??, ??, ??, ??, 608
Prop_Feynman, 178, used: ??, ??, ??, ??, ??,
                                ??, ??, ??, ??, ??, ??, 608
                               Prop_Gauge, 178, used: ??
Prop\_Ghost, 178, used:
                               Prop_Majorana, 178, used: ??, ??, ??, ??, 608
Prop_Rxi, 178, used: ??
                               Prop_Scalar, 178, used: ??, ??, ??, ??, ??, ??,
                               ??, ??, ??, ??, ??, 608
                               prop_spinor, ??, ??, ??, ??, ??, ??, ??, ??.
                               used: ??, ??, ??, ??, ??, ??, ??, ??,
                               ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                               Prop_Spinor, 178, used: ??, ??, ??, ??, ??, ??,
                               ??, ??, ??, ??, ??, ??, ??, 608
                               Prop_Tensor_2, 178, used: ??, ??, ??, ??, ??
                               Prop_Tensor_pure, 178, used: ??
Prop_UFO, 178, used:
                               Prop_Unitarity, 178, used: ??, ??, ??, ??, ??,
                               ??, ??, ??, ??, ??
                               Prop\_Vectorspinor, 178, used: ??
Prop\_Vector\_pure, 178, used: ??
                               ??, ??, ??, 608
PSet (module), 282, used: 282, 282
                           Psim, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
pseudo, 316, 316, 320, 320, 321, 322, used: 322
                                ??, ??, ??, ??, ??, ??, ??
pseudo\_map, 322, used: 322
                               ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                           Psimm,
Psi, 180, used: ??, ??, ??, ??, ??, ??, ??.
                                ??, ??, ??, ??, ??
    Psip, ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??,
    ??, ??, ??, ??, ??, ??
    ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                           Psipp,
    ??, ??, ??, ??, ??
    PSSSM (module), ??, ??, used: ??
    PSSSM\_QCD (module), ??, ??, used:
    PV, 180, used:
    P-NNA, ??, used:
    P\_NNG, ??, used:
```

```
P_-NNH1, ??, used:
                                                Quot, 203, used: 339, 339, ??, ??, ??, ??, ??,
                                                       P_NNH2, ??, used:
p_to_string, 587, used: 587, 588
                                                       P_Whizard (module), 596, used: 597
                                                       ??, ??, ??
q (type), 716, used: 716, 716, 716
Q, ??, 607, used: ??, ??, 607, 608, 608, 609
                                                quote, 642, 640, used: 602
Q (module), 249, 215, 307, 337, 344, 716, 248,
                                                quoted, 330, 329, used: ??, 332
       used: 215, 215, 217, 218, 310, 310, 337,
                                                Quoted, 329, 329, used: 330
                                                QUOTED, ??, ??, used: ??, 331
       338, 338, 339, 344, 346, 351, 716
QBUB, ??, ??, ??, ??, used: ??, ??, ??, ??,
                                                QUOTED (cambrace token), 331, used: 332
       quoted\_index, 352, used: 352
       ??, ??, ??, ??, ??
                                                Quotient, 329, 338, 329, used: 330, 336, 337,
qc (type), 214, 215, 215, 219, 220, 220, 213,
       used: 214, 213, 213, 214
                                                QUQD1L, ??, ??, used: ??, ??, ??, ??, ??, ??,
QC (module), 215, 344, 716, used: 214, 215,
                                                       ??, ??
                                                QUQD1R, ??, ??, used: ??, ??, ??, ??, ??, ??,
       215, 215, 215, 216, 216, 216, 216, 216,
                                                       ??, ??
       217, 217, 217, 218, 343, 344, 344, 344,
       346, 346, 346, 346, 346, 65, 184, 717, 213,
                                                QUQD8L, ??, ??, used: ??, ??, ??, ??, ??, ??,
                                                       ??, ??
       333, 333, 365, 422, 378
QCD (module), ??, ??, used: ??, ??
                                                QUQD8R, ??, ??, used: ??, ??, ??, ??, ??, ??,
qcd\_ew\_check, ??, used:
                                                       ??. ??
qcd\_order, ??, used: ??, ??
                                                QW,
                                                     ??, ??, ??, ??, ??, used: ??, ??, ??, ??, ??,
qcd\_order (type), ??, used:
                                                       QCD_order, ??, used: ??, ??
                                                       ??, ??, ??, ??
QComplex (module), 716, used:
                                                Q_{-charg}, ??, ??, used: ??, ??, ??
                                                Q_{-down}, ??, ??, ??, ??, ??, ??, ??, ??, ??,
QComplex (sig), 716, used: 716
QED (module), ??, ??, used: 606, ??
                                                       ??, ??, ??, ??, ??, ??, ??, ??, ??,
used: ??, ??, ??, ??, ??, ??, ??, ??,
       ??, ??, ??, ??, used: ??, ??, ??, ??, ??,
                                                       ??, ??, ??, ??, ??, ??, ??
       ??, ??, ??
                                                Q_{-lepton}, ??, ??, ??, ??, ??, ??, ??, ??, ??,
QGUG, ??, ??, ??, ??, used: ??, ??, ??, ??,
                                                       ??, ??, ??, ??, ??, ??, ??, ??, ??,
       used: ??, ??, ??, ??, ??, ??, ??, ??,
       ??, ??, ??, ??, ??
                                                       QH, ??, used: ??, ??, ??, ??, ??, ??, ??, ??
                                                       ??, ??, ??, ??, ??, ??
                                                       ??, used: ??
qn\_baryon, ??, used: ??
                                                Q\_top,
qn\_charge, ??, used: ??
                                                       Q_{-}up,
qn\_generation, ??, used: ??
                                                       ??, ??, ??, ??, ??, ??, ??, ??, used:
qn\_lepton, ??, used: ??
                                                       QQ (module), 249, 248, used: ??, ??, ??, ??,
                                                       ??, ??, ??, ??, ??
       Q_{Z_up}, ??, ??, used: ??, ??
       ??, ??, ??, ??, ??, ??, ??, ??
                                                r (field), 44, used: 45, 45, 45, 45, 45, 46, 46,
quark, ??, used: ??, ??
                                                       46, 47, 47, 47, 48, 48, 48, 49
Quark, ??, used: ??, ??, ??, ??, ??, ??, ??
                                                r (type), 343, 343, 348, 353, 357, 334, 334,
quark\_current, 608, used: 609
                                                       used: 343, 343, 344, 348, 354, 334, 334,
quark_currents, ??, used: ??
                                                       334
quartic_anom, ??, ??, ??, ??, ??, ??, ??, ??,
                                                R \text{ (sig)}, 215, \text{ used: } 215, 219, 220, 220
       ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                                raises, ??, used: ??
       used: ??, ??, ??, ??, ??
                                                random\_diagram, 697, used:
quartic_gauge, ??, ??, ??, ??, ??, ??, ??, ??,
                                                random\_int, 697, used: 697
       random_int_list, 631, 687, used: 633, 688
       used: ??, ??, ??, ??, ??, ??, ??, ??,
                                                range, 626, 623, used: 630, 631, 632, 681, 682,
       ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                       683, 684, 685, 685, 685, 685, 685, 686,
quartic_gravitino, ??, used: ??
                                                       686, 686, 686, 686, 686, 687, 688, 688,
quartic_grav_char, ??, used: ??
                                                       688, 689, 14, 15, 49, 324, 325, 60, ??, ??,
quartic_grav_gauge, ??, used: ??
                                                       ??, ??, 607
quartic_grav_neu, ??, used: ??
                                                Range (exn), 44, 44, 50, 41, used:
quartic_grav_sneutrino, ??, used: ??
                                                rank, 34, 35, 35, 35, 37, 39, 44, 45, 49, ??, 32,
Quiet, ??, used: ??
                                                       33, 41, used: 36, 37, 50, 50, 50, 52, 52,
                                                       52, 52, 52, 52, 52, 52, 53, ??, ??, ??
```

```
rank (type), 35, 36, 36, 39, 39, 33, used: 35, 36,
                                                          result (type), 620, 622, 345, 620, used: 621, 620
        36, 39, 39, 33, 33
                                                          result_flavour, ??, used: ??
                                                          result\_msq, ??, used: ??
        49, used: 49, 50, 50, 51, 51, 51, 51
                                                          result_path, ??, used: ??
ranked, 35, 36, 37, 39, 39, 33, used: 37, ??
ranks, 35, 36, 37, 39, 39, 33, used: 37
                                                          rev, 311, used: ??, 446, 625, 625, 626, 626, 626,
rank\_set, 36, used: 36
                                                                  626, 627, 627, 629, 629, 630, 630, 636,
                                                                  621, 664, 664, 664, 664, 676, 677, 678,
Rat (module), 249, used: 249, 249
Ratio, 291, 285, used: 292
                                                                  678, 678, 680, 680, 684, 690, 702, 703,
Rational, 338, used:
                                                                  694, 14, 49, 50, ??, 289, 311, 312, 345,
Rational (sig), 715, used: 716, 716, 718, 719
                                                                  279, 279, 281, ??, 601, 602
Ratios, 343, 344, 333, used: 345, 345, 346, 346,
                                                          rev (field), 703, 700, used: 704, 704, 704, 705,
                                                                  705, 598, 599, 599
                                                          revmap, ??, used: ??, ??
raw (field), 615, used: 615, 615, 615
RBRACE, ??, ??, ??, used: ??, ??, 297
                                                          revmap2, ??, used: ??
RBRACE (camlyacc token), 298, 361, used:
                                                          rev_flatmap, 627, 624, used: 679
        299, 299, 299, 301, 301, 301, 301, 302,
                                                          rev\_multiply, 630, used: 630, 630
                                                          Rf, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
        302, 303, 363
RBRACKET, ??, ??, ??, ??, ??, used: ??,
                                                                   ??, ??, ??, ??, ??, ??, ??, ??, ??
        ??, ??, 57, 297
                                                          rf3, ??, ??, ??, used: ??, ??, ??
                                                          rf3cf, ??, ??, used: ??
RBRACKET (camlyacc token), 57, 298, 361,
                                                          rf3h, ??, ??, used: ??
        used: 58, 299, 299, 301, 301, 303, 362
                                                         rf3t, ??, ??, ??, used: ??, ??, ??
read, 621, 622, 620, used: 622
read_generation, ??, used: ??
                                                          RFailure, ??, ??, used: ??
read_lines, 446, 601, used: 446
                                                          rfddphi3cf, ??, used: ??
                                                          Rff, ??, used: ??, ??, ??, ??
read_lines_rev, 601, used: 601, 601
real, 716, used: 218, 346
                                                          rff3cf, ??, used: ??
                                                          rff3h, ??, used: ??
Real, 338, 208, used: 338, 339, 340, ??, ??, ??,
                                                          Rfphi, ??, used: ??, ??, ??, ??, ??
        ??, ??
                                                          rfphi3cf, ??, used: ??
Real_Array, 208, used: ??, ??, ??, ??, ??, ??,
         ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                          Rfv, ??, used: ??, ??, ??, ??
                                                          rfv3cf, ??, used: ??
Rec, 203, used:
                                                          rhs, ??, ??, ??, ??, 123, ??, used: ??, 603
remove, 645, 647, 651, 658, 658, 659, 660,
        661, 24, 644, 656, 656, 657, used: ??,
                                                          rhs (type), ??, ??, ??, ??, 122, ??, used: ??, ??,
                                                                   647, 651, 659, 659, 661, 661, 24, 25, 25,
        282
                                                                  123, 123, 123, ??, ??, ??, ??, ??
remove_duplicate_final_states, 278, 282, 277,
                                                          Ring (sig), 718, used: 718, 719
        used: ??
                                                          RMOM, 180, used:
                                                          RPAREN, ??, ??, ??, ??, ??, ??, ??, used:
rename, 336, 332, used:
                                                                  ??, ??, ??, 57, 297, 331
rename1, 336, used: 336
rename_indices, 343, 343, 345, 350, 356, 333,
                                                          RPAREN (camlyacc token), 57, 298, 331, 361,
        334, used: 345
                                                                  used: 58, 301, 302, 303, 332, 362, 363, 364
                                                                  ??, ??, ??, used: ??, ??, ??
rename_indices2, 350, used: 350, 355
                                                          rphi3.
                                                          rphi3h, ??, ??, used: ??
rename_indices_dirac, 350, used: 350
                                                          Rphim, ??, ??, ??, used: ??, ??, ??, ??, ??, ??,
rename_indices_vector, 349, 349, 335, used: 350
rep (type), 294, 294, 286, 286, used: 294, 286,
                                                                   ??, ??, ??
                                                          Rphimm, ??, ??, ??, used: ??, ??, ??, ??, ??,
        286
                                                                  ??, ??, ??, ??
repeat, 689, used: 689
rep_conjugate, 343, 343, 348, 354, 358, 334,
                                                                  ??, ??, ??, used: ??, ??, ??, ??, ??, ??,
                                                          Rphin,
                                                                  ??, ??, ??
        334, used: 348
rep\_of\_int, \quad \mathbf{343}, \, \mathbf{343}, \, \mathbf{348}, \, \mathbf{353}, \, \mathbf{357}, \, \mathbf{334}, \, \mathbf{334},
                                                                  ??, ??, ??, used: ??, ??, ??, ??, ??, ??,
                                                          Rphip,
                                                                  ??, ??, ??
        used: 348
                                                                   ??, ??, ??, used: ??, ??, ??, ??, ??, ??
rep_of_string, 294, 286, used: 309
                                                          Rphipp,
rep_to_string, 294, 343, 343, 348, 353, 357,
                                                          Rphisn,
                                                                   ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
        286, 334, 334, used: 307, 348, 354
                                                                   ??
                                                          Rsigma, ??, ??, ??, used: ??, ??, ??, ??, ??, ??,
rep_to_string_whizard, 343, 343, 348, 353, 357,
                                                                  ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
        334, 334, used: 348
                                                          rsigma3, ??, ??, ??, used: ??, ??, ??
rep_trivial, 343, 343, 348, 353, 357, 334, 334,
                                                          rsigma3h, ??, ??, used: ??, ??
        used: 348
                                                          rsigma3t, ??, ??, ??, used: ??, ??, ??
required_space, 291, used: 291
                                                          rsigma4, ??, ??, ??, used: ??
RError, ??, ??, used: ??
                                                          RSkip, ??, ??, used: ??
reset, 617, 616, used:
```

```
RSuccess, ??, ??, used: ??
                                            sanitize\_questionmark, 643, used: 643
rt3, ??, ??, used: ??, ??, ??
                                            sanitize_underscore, 643, used: 643
rt3cf, ??, ??, used: ??, ??
                                            sanitize_underscore_alpha, 643, used: 643
Rtm, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                            sanitize_valid, 643, used: 643
      ??, ??, ??, ??
                                            sans_colors, ??, used: ??
      ??, ??, ??, used: ??, ??, ??, ??, ??, ??,
                                            Sbar, ??, used: ??, ??, ??, ??
Rtmm,
      ??, ??, ??, ??, ??
                                            scalar, 316, 316, 320, 320, 320, 322, 343, 351,
   ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                   356, 334, used: 322, 346, 351
Rtn.
      ??, ??, ??, ??
                                            scalar (field), 365, used:
                                            scalar (type), 349, 349, 335, used: 349, 349,
RTodo, ??, ??, used: ??
    ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                   349, 335, 335, 365
Rtp,
      ??, ??, ??, ??
                                                   294, 349, 349, 178, 286, 335, used: 350,
                                            Scalar,
     ??, ??, ??, used: ??, ??, ??, ??, ??, ??,
                                                   350, 350, 351, 352, 355, ??, ??, ??, ??,
Rtpp,
                                                   Rtsn, ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??
                                                   ??, ??, ??, ??, ??, ??, ??, 607
Running, 180, used: ??, ??, ??, ??, ??, ??, ??,
                                            Scalar2_Vector2, 182, used: ??, ??, ??, ??, ??,
      run\_test\_tt, ??, ??, used: ??
                                                   run\_test\_tt\_main, ??, ??, used:
                                                   R_-CN, ??, used:
R_{-}CNG, ??, used: R_{-}NC, ??, used:
                                                   ??, ??, ??, ??, ??, ??, ??, ??, 609
                                            Scalar4,
                                                  182, used: 65, ??, ??, ??, ??, ??,
                                                   R\_NCH, ??, used:
                                                   r\_omega (type), 343, 343, 348, 355, 358, 334,
                                                   ??, ??, ??, ??, ??, ??, ??
      334, used: 343, 343, 344, 348, 350, 356,
      334, 334, 335, 335
                                            scalar\_map, 322, used: 322
S, 316, 318, 353, 357, ??, ??, 180, used: 318,
                                            Scalar_Scalar_Scalar, 180, used: 65, ??, ??, ??,
                                                   353, 354, 357, 358, ??, ??, ??, ??, ??, ??,
      ??, ??, ??, ??, ??
                                                   ??, ??, ??, ??
S (module), 311, 337, 344, 351, 356, ??, used:
                                            scalar\_to\_string, 351, used: 351
                                            Scalar_Vector_Vector, 180, used: ??, ??, ??, ??,
      312, 346, ??, ??
                                                   S \text{ (sig)}, ??, ??, \text{used: } ??, ??, ??, ??, ??
   ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                   ??, ??
                                                   S1 (module), 36, used: 36
S2, ??, ??, 180, used: ??, ??, ??, ??, ??, ??, ??,
                                            Scalar\_Vector\_Vector\_t, 180, used: ??, ??, ??,
      ??
      ??, ??, ??, ??, ??, ??
                                            scale, 717, 718, 718, used:
S2 (module), 36, used: 36
                                            scattering (type), 278, 278, 277, used: 278, 278,
S2L, 180, used:
S2LR, 180, used: ??, ??
                                            Scattering, 278, 278, 277, used: 279, 279, 279
S2P, 180, used:
                                            Scattering (module), 44, 47, 52, 42, used: 586,
S2R, 180, used:
                                                  587, 590, 590, 590, 592, ??, ??, ??, ??
S3, ??, ??, used: ??, ??, ??, ??, ??, ??, ??, ??,
                                            Sccs, ??, ??, used:
      scripted, 289, 284, used: ??, 302, 302
      ??, ??
                                            scripted (type), 288, 284, used: 288, 284, 284,
   ??, used: ??, ??, ??
   ??, used: ??, ??, ??
                                            Scripted, 288, 284, used: 289, 289, 290
   ??, used: ??, ??, ??
S6.
                                            scripted_token (camlyacc non-terminal), 302,
S7, ??, used: ??, ??, ??
                                                   used: 299, 301, 301
                                            scripted_to_string, 290, 285, used: 290, 291
S8, ??, used: ??, ??, ??
S9, ??, used: ??, ??, ??
                                            SD, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                   same_elements, 684, used: 684
                                                   ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
sanitize\_digit, 643, used: 643
                                                  ??, ??, ??, used: ??, ??, ??, ??, ??, ??,
sanitize\_digit\_alpha, 643, used: 643
                                            Sdown,
sanitize_fortran_id, 642, 640, used: 643, 643,
                                                   643, 643, 643, 643
```

```
set_nc, ??, used: ??
        set\_of\_flavors, 590, used: 590
        set\_of\_list, 671, used:
        ??, ??, ??, ??, ??, ??, ??, ??
                                                     set_parameters, ??, used: ??
sdown4\_1gen, ??, used: ??
                                                     set\_pdg, ??, used: ??
sdown4_1gen', ??, used: ??
sdown4_2gen, ??, used: ??
                                                      set_propagator, ??, used: ??
                                                     set_to_string, 671, used: 671
sdown4\_2gen', ??, used: ??
                                                     set_vertices, ??, used: ??
search_path, 620, used: 621, 622, 622
                                                     set\_width, ??, used: ??
Sec. 337, used: 337
                                                     set_width_symbol, ??, used: ??
                                                     sff (type), ??, ??, ??, used: ??, ??, ??, ??
selectors (type), 60, 63, ??, ??, ??, ??, 59, 123,
                                                      sfm (type), ??, ??, ??, used: ??, ??, ??, ??,
        125, ??, ??, used: 60, 60, 60, 60, ??, ??,
        ??, ??, ??, ??, 59, 59, 59, 59, 59, 59,
        59, 59, 123, 124, 125, 126, ??, ??, ??, ??
                                                      SG, 607, used: 607, 609
select_p, 60, 63, 59, used: ??
                                                      shift\_left\_pred, 281, used: 282
select_p (field), 63, used: 63, 63, 67
                                                      shift\_left\_pred', 281, used: 281
                                                      shiggs (type), ??, ??, used: ??, ??, ??, ??
select_vtx, 60, 63, 59, used: ??
                                                      SHiggs, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
select\_vtx (field), 63, used: 63, 63, 67
select_wf, 60, 63, 59, used: ??
                                                              select_wf (field), 63, used: 63, 63, 67
                                                              ??. ??
self_adjoint, 218, used: 219
                                                      shiggs_neutr, ??, ??, used: ??, ??
semistandard\_tableau, 695, 693, used: 696
seq (type), 675, 673, used: 673, 673, 674,
                                                      shortest, 658, 660, 660, 662, 656, 657, used:
        674, 674, 674, 674, 674, 674, 674, 674,
                                                      shortest', 660, 662, used: 660, 662
        674, 675
                                                      shuffle, 687, used: 688, 688, 688, 688, 689
                                                      Sigma, 348, 349, 334, 364, used: 350, 352
series, 293, used: 294
                                                      sign, 681, ??, ??, ??, ??, ??, 674, 122, ??,
set, 282, used: 630, 636, 636, 731, 731, 732, 732,
                                                              used: 683, ??, ??
        733, 733, 686, 686, 687, 708, 708, 709,
                                                     sign (field), ??, ??, used: ??, ??, ?? sign (type), ??, ??, used: ??, ??
        695, 695, 696, 215, 216, 216, 216, 216,
        216, 216, 217, 217, 217, 282, ??, ??
set (type), 671, 665, 663, used:
                                                      sign2, 681, used: 683
Set (module), ??, used: ??, ??, ??, ??, ??, ??,
                                                      signed, 684, 685, 686, 683, used:
        ??, ??, ??, ??, 668, 671, 671, 36, 36, 36,
                                                      sign\_sign2, 683, used: 683
        ??, ??, ??, 590, 592, 308, 281, 282, ??,
                                                      Simplest (module), ??, ??, used: ??, ??
        ??, ??, ??, 597, ??
                                                      Sin, 337, ??, 203, used: 337, 339
SetMake (module), ??, ??, used:
                                                      Sin2al, ??, used:
Sets (module), ??, ??, used: 630, 630, 330, 343,
                                                      Sin2am2b, ??, used:
                                                      Sin2be, ??, used:
        343, 343, 345, 345, 345, 345, 355, 355,
                                                      Sin2thw, ??, ??, ??, ??, ??, ??, ??, ??, ??,
        282, 282, 603, 329, 332, 378
                                                              ??, ??, ??, ??, ??, ??, ??, used: ??,
SetTestPath (module), ??, used: ??
setup, ??, ??, 211, used:
                                                              set_color, ??, used: ??
                                                              ??, ??, ??, ??, ??, ??
set_conjugate, ??, used: ??
                                                      Sin4al, ??, used:
set_constant_symbol, ??, used: ??
                                                              ??, used:
                                                      Sin 4be,
set_external_flavors, ??, used: ??
                                                      Sinamb, ??, used:
                                                      Sinapb, ??, used:
set_fermion, ??, used: ??
set_flavors, ??, used: ??
                                                      singleton, 645, 645, 650, 44, 45, 50, 644, 718,
set_flavor_of_string, ??, used: ??
                                                              42, used: 330, 330, 281, ??, ??
set\_flavor\_symbol, ??, used: ??
                                                      Singular (exn), 731, 731, used:
set_flavor_to_string, ??, used: ??
                                                      Sinh, 337, 203, used: 337, 339
set\_flavor\_to\_TeX, ??, used: ??
                                                      Sinpsi, ??, ??, used:
                                                      Sinthw, ??, ??, ??, ??, ??, ??, ??, ??, ??,
set_formatter_out_channel, 446, 444, used: 446,
                                                              ??, ??, ??, ??, ??, ??, used: ??, ??,
        448
set_fuse, ??, used: ??
                                                              ??, ??, ??
set\_fuse2, ??, used: ??
                                                      size, 33, 38, 31, used:
set\_fuse3, ??, used: ??
                                                      Skip (exn), ??, used:
                                                     skip_if, ??, ??, used: ??, 219
set_gauge_symbol, ??, used: ??
                                                     SL, ??, ??, ??, ??, 180, used: ??, ??, ??, ??, ??,
set_goldstone, ??, used: ??
set_lorentz, ??, used: ??
                                                              set_mass_symbol, ??, used: ??
                                                              set_max_degree, ??, used: ??
```

```
sneu2\_slep2\_2, ??, used: ??
       ??, ??, ??, ??, ??, ??, ??, ??, ??
                                              sneu2\_slep2\_2', ??, used: ??
Slepton, ??, ??, ??, used: ??, ??, ??, ??, ??, ??,
                                              sneu2_squark2, ??, used: ??
       sneu2\_squark2', ??, used: ??
       Sneutrino, ??, ??, ??, used: ??, ??, ??, ??, ??,
                                                     ??, ??, ??
slepton2_squark2, ??, used: ??
slepton2_squark2',
                ??, used: ??
                                              sneutrino4, ??, used: ??
slepton2_squark2", ??, used: ??
                                              sneutrino4', ??, used: ??
slepton4_1gen, ??, used: ??
                                              SO, 293, used: 293
slepton4_1gen', ??, used: ??
slepton4_2gen, ??, used: ??
                                              solve, 734, 731, used:
                                              solve_destructive, 734, used: 734
slepton4\_2gen', ??, used: ??
                                              solve_many, 734, 731, used: 709
slep_sneu_squark2, ??, used: ??
                                              solve_many_destructive, 734, used: 734
slep_sneu_squark2', ??, used: ??
                                              sort, 702, 584, 586, 699, 583, used: 629, 630,
slep_sneu_squark2", ??, used: ??
                                                      630, 630, 632, 711, 711, 678, 678, 681,
slist, ??, used: ??
                                                      684, 684, 685, 686, 702, 702, 703, 696, 38,
                                                      39, 45, 586, 587, 590, 591, 293, 309, 309,
slow_binomial, 676, used: 676
SLR, 180, used: ??, ??, ??, ??, ??, ??, ??.
                                                     315, 342, 346, 348, 358, 56, 61, ??, ??, ??,
       598, 602
       sort', 702, used: 702, 702
       sort3, 587, used: 587
       sorted\_keys, 36, used:
       ??, ??, ??
                                              sort_2i, 703, used: 704, 707
SLRV, 180, used: ??, ??, ??
                                              sort_2i', 703, used: 703, 703
SLV, 180, used: ??, ??
                                              sort\_children, 711, used: 711
SM (module), ??, ??, used: ??, ??, ??, ??,
                                              sort_decay, 584, 584, 583, used: 584, 586
       ??, ??, ??, 607, ??, ??, ??, ??, ??, ??,
                                              sort_momenta, 586, used: 586, 587
       ??, ??, ??, ??, ??, ??, ??, ??, ??
                                              sort\_section, 630, used: 630
Small_Rational (module), 716, used: 249, ??,
                                              sort_signed, 680, 675, used: 681, 682, 682, 682,
       682, ??, ??
       ??, ??, ??, ??, ??, ??, ??, ??, ??, 248,
                                              sort\_signed\_all, 682, used: 683
                                              sort\_signed\_even, 682, used: 683
                                              sort_signed_not_unique, 682, used: 683
SMap (module), 336, used: 336, 336, 343, 343,
                                              sort\_signed\_odd, 682, used: 683
SM\_anomalous (module), ??, ??, used: ??
                                              sort_spin_table, ??, used: ??
SM\_anomalous\_ckm \text{ (module)}, ??, ??, used: ??
                                              Sp, 293, 353, used: 293, 353, 354, 354
                                              SP, 180, used: ??, ??, ??, ??, ??, ??, ??, ??,
SM\_anomalous\_top \text{ (module)}, ??, ??, used: ??
SM\_clones (module), ??, ??, used:
                                                      ??, ??
SM\_dim6 (module), ??, ??, used: ??
                                              space (type), 306, 307, used: 307, 307
SM_flags (sig), ??, ??, ??, ??, ??, ??, ??, used:
                                              spacelike, 44, 48, 48, 52, 52, 43, 43, used: 586,
       space\_to\_string, 307, used: 308
       ??, ??
                                              species, 703, used: 704, 704, 704
                                              Spin, 292, 285, used: ??, 300
SM\_gluons (module), ??, used:
SM\_Higgs (module), ??, ??, used: ??, ??
                                              SPIN, ??, ??, used: ??, 297
SM_Higgs_CKM (module), ??, ??, used: ??, ??
                                              SPIN (camlyacc token), 298, used: 300, 300
SM_k\_matrix (module), ??, ??, used: ??, ??
                                              SM_no_anomalous (module), ??, ??, ??, ??, ??,
                                                      ??, ??, ??, ??, ??, ??, ??, ??, used:
       ??, ??, used: ??, ??, 607, ??, ??, ??, ??,
                                                      ??, ??, ??
                                                      ??, ??, ??, ??, ??, ??, ??, ??
SM\_no\_anomalous\_ckm  (module), ??, ??, used:
                                              spinor (type), 316, 317, used: 316, 318
       ??, ??
                                                     316, 317, 178, used: 317, 326, 355, ??,
                                              Spinor,
                                                      SM Rxi (module), ??, ??, used:
                                                     SM_{tt-threshold} (module), ??, ??, used: ??
SN, ??, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                      ??, ??, 607
       ??, ??, ??, ??, ??, ??, ??, ??, ??
                                              spinor\_contraction\_ok, 319, used: 319
sneu2\_slep2\_1, ??, used: ??
                                              spinor_field, 326, used: 326, 326, 326, 326, 327,
sneu2\_slep2\_1', ??, used: ??
                                                      327
```

```
spinor\_index, 352, used: 352
                                                                                        status, 614, used:
                                                                                        Stat\_Dirac \text{ (module)}, ??, used: ??, ??, ??, ??
spinor_indices, 319, used: 319
                                                                                        stat_fuse, ??, ??, ??, ??, used: ??, ??, ??, ??
spinor\_ok, 317, used: 318
spinor_sandwitch_ok, 318, used: 318
                                                                                        stat_keystone, ??, used: ??
spins (field), 378, used:
                                                                                        Stat_Majorana (module), ??, used: ??, ??, ??,
split, 646, 619, 677, 7, 9, 10, 12, 14, 44, 49,
                                                                                        Stat\_Maker (sig), ??, used: ??
             53, 619, 674, 5, 42, used: 646, 648, 649,
             619, 677, 680, 709, 14, 28, ??, ??, 590, ??,
                                                                                        Stat\_Map (module), ??, used: ??
                                                                                        stat_sign, ??, ??, ??, used: ??, ??, ??
split', 619, 676, used: 619, 619, 676, 676, 677
                                                                                        stat\_to\_string, ??, ??, used:
splitn, 625, 623, used:
                                                                                        std_formatter, 445, 444, used: 445, 446, 446
splitn', 625, used: 625, 625
                                                                                        stem, 289, 285, used: 289, 306, 308, 308, 308,
split_last, 625, 623, used: 632
                                                                                                     309, 331
SPM, 180, used: ??, ??, ??
                                                                                        stem (field), 288, 311, 284, used: 288, 289, 289,
SQ, 607, used: 607, 608, 609, 609, 609
                                                                                                     290, 290, 311, 311, 312
Sqrt, 337, 203, used: 337, 339, 340, ??, ??, ??,
                                                                                        stem_kinds (field), 308, used: 308, 308, 309, 310
                                                                                        step (field), 616, used: 617, 617, 618
             ??, ??
squark\_current, 609, used: 609
                                                                                        steps (field), 616, used: 617, 618
squark_seagull, 609, used: 609
                                                                                        string (camlyacc non-terminal), 363, used:
SR, 180, used: ??, ??, ??, ??, ??, ??, ??, ??,
                                                                                                     359, used: ??, ??, ??, 601, ??, ??, ??,
                                                                                        String,
                                                                                                     ??, ??, ??, ??, ??, ??, ??, <mark>362</mark>, <mark>362</mark>
             ??, ??, ??
SRV, 180, used:
                                                                                        String (module), ??, ??, used: ??, ??, ??, ??, ??,
SS (module), 308, used: 308, 308, 309, 310
                                                                                                     445, 640, 641, 641, 641, 641, 641, 642,
SSC (module), ??, ??, used: ??, ??
                                                                                                     642, 642, 642, 642, 631, 671, 671, 711,
SSC\_AltT (module), ??, ??, used: ??
                                                                                                     711, 685, 687, 703, 704, 45, 50, 587, 588,
SSC_flags (sig), ??, ??, used: ??, ??, ??, ??, ??,
                                                                                                     591, 591, 591, 593, 290, 290, 290, 291,
             ??, ??, ??
                                                                                                     292, 293, 293, 295, 295, ??, 303, 304, 310,
SSC_kmatrix (module), ??, ??, used: ??
                                                                                                     312, 312, 312, 313, 319, 325, ??, ??, 335,
                                                                                                     338, 342, 342, 344, 344, 344, 345, 345,
SSC_kmatrix_2 (module), ??, ??, used: ??, ??
ST (module), 308, used: 308, 308, 308, 308, 308,
                                                                                                     345, 345, 352, 352, 56, 56, ??, 62, 62, 62,
             308, 308, 309, 310
                                                                                                     63, 63, 66, 67, 279, 279, 279, 279, 281,
           620, 622, 620, used: 622
                                                                                                     281, ??, ??, ??, ??, 598, 598, 599, 599,
standard\_gauge\_higgs, ??, ??, ??, ??, ??, ??, ??,
                                                                                                     600, 601, 602, 603, 603, 604, ??, 609, 611,
                                                                                                     378, 57, 296, 330
             ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
             ??, ??
                                                                                        STRING, ??, ??, ??, used: ??, ??, ??, ??, 57
standard_gauge_higgs', ??, ??, ??, used: ??, ??,
                                                                                        STRING (camlyacc token), 57, 361, used: 58,
                                                                                                     362, 363, 363, 364
standard\_gauge\_higgs4, \quad \ref{eq:standard}, \quad \ref{eq:standard}, \\ \ref{eq:standard}, \\
                                                                                        string1, ??, used: ??, ??, ??, ??
             ??, ??, ??, used: ??, ??, ??, ??, ??, ??,
                                                                                        string2, ??, used: ??, ??, ??, ??
             ??, ??, ??
                                                                                        strings (camlyacc non-terminal), 363, used: 362,
standard_higgs, ??, ??, ??, ??, ??, ??, ??,
             ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                                                        string\_atom (type), 358, used: 359
                                                                                        String_Caseless (module), ??, ??, used: 330, 329,
standard_higgs4, ??, ??, ??, ??, ??, ??, ??, ??,
                                                                                        string_expr (camlyacc non-terminal), 363, used:
             ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                                                                     362, 362, 363
standard_quartic_gauge, ??, ??, ??, ??, ??, ??,
                                                                                        String\_Expr, 359, used: ??, 362, 362
             ??, ??, ??, ??, used: ??, ??, ??, ??,
                                                                                        String_Hash (module), ??, used: ??, ??, ??
             ??, ??, ??, ??, ??, ??, ??
                                                                                        String_Key (module), ??, used: ??
standard_tableau, 696, 693, used:
                                                                                        string_list (camlyacc non-terminal), 58, used: 58,
standard_triple_gauge, ??, ??, ??, ??, ??, ??,
                                                                                                     58, 58, 58, 58
             ??, ??, ??, ??, used: ??, ??, ??, ??,
                                                                                        String_List, 359, used: ??, 362
             ??, ??, ??, ??, ??, ??, ??
                                                                                        string_lists (camlyacc non-terminal), 58, used:
Star, 288, 284, used: 288
                                                                                                     58, 58
stat, ??, ??, ??, used: ??, ??
                                                                                        string\_of\_angle, ??, used: ??
stat (type), ??, ??, ??, ??, used: ??, ??, ??, ??,
                                                                                        string_of_char, ??, ??, ??, ??, ??, ??, 296, 330,
             ??, ??, ??
                                                                                                     used: ??, ??, ??, ??, ??, ??, ??, ??,
Stat (sig), ??, used: ??, ??, ??
                                                                                                     ??, ??, ??, 296, 297, 331
                                                                                        string_of_fermion_gen, ??, ??, used: ??, ??
state (type), 616, used: 616
Static (module), ??, ??, used:
                                                                                        string_of_fermion_type, ??, ??, used: ??, ??
```

```
string\_of\_gen, ??, ??, used: ??, ??, ??
                                                     suite_hook_lengths_product, 697, used: 697
string\_of\_higgs, ??, used: ??, ??
                                                     suite\_inverse, 688, used: 689
                                                     suite_long, 697, 697, 693, used:
string\_of\_int\_list, 611, used: 611
string_of_neu, ??, ??, ??, used: ??, ??, ??, ??,
                                                     suite_normalization, 697, used: 697
        ??, ??, ??, ??
                                                     suite_num_columns, 639, used: 639
string_of_node, ??, ??, used: ??
                                                     suite_num_standard_tableaux, 697, used: 697
string\_of\_path, ??, ??, used: ??, ??, ??
                                                     suite\_of\_list, 687, used: 689
string_of_phiggs, ??, ??, used: ??, ??, ??, ??, ??
                                                     suite\_of\_lists, 688, used: 689
string_of_sff, ??, ??, ??, used: ??, ??, ??, ??,
                                                     suite_permute, 681, used: 683
                                                     suite_power, 632, used: 634
string_of_sfm, ??, ??, ??, used: ??, ??, ??, ??
                                                     suite\_sanitize, 643, used: 643
string_of_shiggs, ??, ??, used: ??, ??, ??, ??, ??
                                                     suite\_sort\_signed, 683, used: 683
strip, 289, used: 289
                                                     suite\_split, 632, used: 634
strip\_from\_first, 641, 640, used:
                                                     sum, 709, 694, 249, 249, 249, 249, 249, 249,
strip_from_last, 641, 640, used:
                                                             715, 716, 717, 718, 718, 248, used: 709,
strip_prefix, 640, 640, used:
                                                             694, 695, 346, 278, ??
strip\_prefix\_star, 641, 640, used:
                                                     Sum,
                                                           291, 329, 338, 203, 285, 329, used: 291,
strip_required_prefix, 641, 640, used:
                                                             330, 336, 339, 340
sty, 705, 700, used:
                                                     summary, 618, 616, used: ??
                                                     summation, 342, 342, 333, used:
style, 703, used: 703
style (field), 703, 700, used: 703, 705, 705, 598
                                                     SUM_{-1}, ??, used:
SU, 293, ??, ??, ??, used: 293, 293, ??, ??, ??,
                                                     sup (field), 702, 703, used: 702, 703, 703
        Sup, 703, ??, ??, ??, used: 703, ??, ??, ??, ??,
        ??, ??, ??, ??
                                                             sub, 44, 46, 51, 214, 216, 291, 715, 716, 718,
                                                             718, 42, 214, 285, used: ??, 640, 641,
                                                             641, 641, 641, 641, 709, 46, 51, 585, 587,
        216, ??, 303, 335, 352, 352, ??, 279, ??,
                                                             ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                     sup2\_sdown2, ??, used: ??
        609, 57, 301
                                                     sup2\_sdown2\_1, ??, used: ??
sub (field), 288, 284, used: 288, 290, 312
                                                     sup2\_sdown2\_2, ??, used: ??
sub (camlyacc non-terminal), 302, used: 302
SUB, ??, ??, used: ??, 297
                                                     sup2\_sdown2\_3, ??, used: ??
                                                     sup4_1gen, ??, used: ??
SUB (camlyacc token), 298, used: 302
                                                     sup4_1gen', ??, used: ??
sub', 46, used: 46, 46
                                                     sup4_2gen, ??, used: ??
substitute, 336, 332, used:
                                                     sup4_2gen', ??, used: ??
subtract, 330, 329, used: ??, 332
succ, 22, 23, 17, used: 445, 640, 641, 641, 641,
                                                     super (field), 288, 284, used: 288, 290, 312
        626, 627, 627, 627, 629, 631, 636, 636,
                                                     super (camlyacc non-terminal), 302, used: 302
        637, 616, 617, 660, 662, 732, 732, 733,
                                                     SUPER, ??, ??, used: ??, 297
        733, 684, 684, 687, 687, 690, 690, 704,
                                                     SUPER (camlyacc token), 298, used: 302
                                                     Supp, ??, ??, used:
        705, 706, 708, 708, 694, 695, 697, 13, 18,
        21, 23, 25, 26, 27, 27, 38, 45, 47, 50, 53,
                                                     Supp2, ??, ??, used:
        590, 60, 279, 279, 279, 281, ??, ??, ??,
                                                     supremum_or_infinity (type), 703, used: 703
        ??, ??, ??
                                                     SV, 180, used:
suite, 736, 737, 447, 643, 634, 639, 683, 687,
                                                     swap, 732, used: 732, 733
        689, 697, 697, 303, 306, 314, 328, 358,
                                                     switch_line_continuation, 446, 444, used:
        735, 444, 640, 625, 635, 715, 675, 684,
                                                     SYM \pmod{1}, 607, used: 611
        693, 303, 335, 365, 423, 379, used:
                                                     Symbol (module), 307, used: 311, 311, 312
suite\_alist\_of\_list, 633, used: 634
                                                     Symbol (sig), 306, used: 307
suite_apply, 737, 688, used: 737, 689
                                                     symbol_kinds (field), 308, used: 308, 308, 309,
suite_apply_with_fallback, 737, used: 737
                                                             310
                                                     symmetry, 676, 26, ??, ??, ??, ??, 673, 124, ??,
suite_auto, 737, used: 737
suite_compare, 638, used: 639
                                                             used: 26, ??, ??
                                                     symmetry (field), ??, ??, used: ??, ??, ??
suite_complement, 634, used: 634
suite_compose, 689, used: 689
                                                     syntax\_error, 304, used: 305
suite\_cycle, 633, used: 634
                                                     Syntax_Error (exn), 288, 329, 56, 284, 329,
suite_filtermap, 631, used: 634
                                                             358, 55, used:
suite_find_all, 639, used: 639
                                                     s_channel, 44, 48, 52, ??, ??, ??, 43, 124, ??,
suite\_find\_first, 638, used: 639
                                                             used: 590, 592
```

 $suite_fortran$, **643**, used: 643

```
s\_channel\_in, 44, 47, 52, 43, used: 48, 48, 52,
s\_channel\_out, 44, 48, 52, 43, used: 48, 52
s\_channel\_out', 47, used: 47, 48
S\_NNA, ??, used:
S\_NNG, ??, used:
S\_NNH1, ??, used:
S\_NNH2, ??, used:
t (type), ??, ??, ??, ?35, 736, 736, 645,
        645, 650, ??, ??, ??, 667, 668, 668,
        668, 669, 670, 671, 671, 621, 621, 616,
        658, 658, 658, 660, 661, 711, 615, 615,
        684, 684, 685, 701, 7, 8, 10, 11, 13, 22,
        23, 24, 33, 33, 33, 34, 34, 35, 35, 35,
        35, 36, 37, 39, 44, 44, 49, 53, 53, 54,
        584, 586, 249, 249, 249, 249, 249, 249,
        ??, ??, ??, ??, ??, ??, ??, 589, 589,
        590, 590, 592, 214, 215, 215, 219, 220,
        220, 288, 291, 292, 293, 294, 295, 295,
        295, 296, 306, 307, 311, 311, 315, 315,
        315, 316, 316, 320, 320, 320, 321, 321,
        322, 323, 324, 324, 336, 336, 338, 341,
        342, 343, 343, 344, 349, 349, 350, 354,
        355, 356, 356, 55, 60, 61, 66, 278, 278,
        281, 282, 282, ??, ??, ??, ??, ??, ??, ??,
        ??, ??, ??, ??, ??, ??, ??, 597, 184,
        ??, ??, 735, 644, 667, 667, 670, 670,
        620, 620, 616, 656, 656, 657, 711, 715,
        716, 716, 717, 718, 718, 615, 683, 699,
        5, 17, 29, 30, 30, 32, 41, 43, 583, 248,
        ??, ??, 589, 213, 284, 285, 285, 286,
        286, 287, 287, 287, 288, 359, 332, 332,
        333, 333, 334, 335, 335, 365, 378, 378,
        378, 378, 55, 59, 277, 57, 58, 298, 298,
        299, 331, 332, 361, 362, used: ??, ??,
        ??, ??, ??, ??, ??, ??, ??, ??, ??, 735,
        735, 736, 645, 645, 667, 668, 668, 669,
        670, 671, 671, 671, 671, 621, 658, 658,
        658, 660, 661, 711, 615, 684, 701, 705, 7,
        8, 10, 11, 13, 19, 21, 22, 22, 22, 23, 24, 27,
        28, 33, 33, 33, 34, 34, 34, 34, 34, 35, 35,
        36, 36, 36, 36, 36, 37, 39, 39, 39, 44, 45,
        50, 53, 53, 54, 584, 584, 249, 249, 249,
        249, 249, 249, ??, ??, ??, ??, 589, 589,
        589, 590, 590, 214, 215, 215, 288, 288,
        291, 292, 292, 293, 293, 294, 295, 295,
        295, 295, 296, ??, 306, 306, 307, 307, 307,
        307, 307, 307, 307, 308, 309, 311, 311,
        311, 313, 315, 315, 315, 315, 316, 320,
        322, 323, ??, 338, 341, 342, 342, 343, 343,
        343, 344, 344, 349, 349, 350, 350, 356,
        356, 358, 55, ??, 60, 60, 60, 60, 60, 61, 63,
        278, 280, 282, 282, ??, ??, ??, ??, ??, ??,
        ??, 596, 597, 606, 184, 210, 210, 210, 211,
        211, 213, 213, 246, ??, ??, ??, ??, ??, ??,
        ??, ??, ??, 735, 735, 735, 735, 735, 644,
        444, 667, 667, 667, 667, 667, 667, 670,
        670, 670, 670, 670, 670, 670, 670, 620,
        616, 656, 656, 656, 656, 656, 656, 656,
```

```
656, 657, 657, 657, 657, 657, 657, 711,
        711, 715, 715, 716, 716, 716, 716, 716,
        716, 716, 716, 716, 717, 717, 717, 717,
        717, 717, 717, 717, 717, 717, 718, 718,
        718, 718, 718, 718, 718, 718, 718, 718,
        719, 719, 615, 615, 683, 683, 683, 683,
        683, 683, 683, 699, 699, 699, 699, 699,
        699, 699, 699, 700, 700, 700, 5, 5, 5, 5, 5,
        5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 7, 16, 17, 17, 18,
        18, 29, 30, 30, 30, 30, 30, 30, 30, 30, 31,
       31, 31, 31, 31, 31, 31, 31, 31, 31, 32,
       32, 32, 32, 32, 33, 33, 33, 41, 41, 41,
       43, 43, 583, 583, 583, 583, 583, 248, 248,
        248, 248, 248, 248, 27, 27, 28, 589,
        589, 213, 213, 214, 214, 214, 214, 284,
        284, 284, 284, 285, 285, 285, 285, 285,
        285, 285, 285, 285, 286, 286, 286, 286,
        287, 287, 287, 287, 287, 287, 287, 287,
        287, 288, 288, 288, ??, 303, 329, 359, ??,
        334, 335, 335, 335, 365, 365, 365, 365,
        365, 365, 365, 365, 422, 422, 422, 422,
        55, 55, ??, 59, 59, 59, 59, 277, 277, 277,
        277, 277, 278, 122, 122, 123, 124, 124,
        125, 125, 126, 126, ??, ??, ??, ??, ??, ??,
        ??, ??, ??, 596, 58, 299, 362
T, 316, 318, 322, 323, 353, 355, 356, ??, 335,
        used: 318, 323, 326, 326, 326, 326, 327,
        327, 353, 354, 354, 356, 356, ??, ??, ??,
        ??
T (module), 293, ??, 306, 307, 311, 597, 298,
        used: ??, 306, 306, 307, 308, 308, 308,
        308, 308, 309, 309, 310, 311, 311, 311,
        312, 312, 312, 312, 312, 313, 601, 603,
        603, 301, 301, 302, 302, 302, 302, 302, 302
T (sig),
        735, 645, 668, 671, 620, 658, 684, 18,
        33, 44, 584, 249, 589, 214, 60, 278, ??,
        596, 208, 246, 735, 644, 666, 670, 620,
        656, 683, 16, 30, 41, 583, 248, 589,
        213, 422, 59, 277, 122, ??, 596, used:
        735, 658, 661, 684, 685, 687, 39, 584, 249,
        249, 249, 249, 249, ??, ??, ??, ??, 589,
        215, 324, 60, 278, ??, ??, ??, ??, ??, ??,
        ??, ??, 596, 606, 606, 606, 210, 211, 212,
        213, 247, 735, 644, 644, 667, 670, 620,
        657, 657, 683, 684, 684, 16, 18, 18, 32, 33,
        43, 583, 248, 248, 248, 248, 248, ??, ??,
        ??, ??, 589, 214, 303, 423, 378, 59, 250,
        278, 124, 126, 126, ??, ??, ??, 596, 596,
       596, ??, ??, ??, ??, ??, ??, ??
t' (type), 293, 316, 320, 282, 286, used: 293,
        316, 316, 316, 320, 320, 320, 282, 286
    11, 6, used: 11, 12, 12, 12, 12, 12, 12
     11, 6, used: 11, 12, 12, 12, 12, 12
    355, 356, 335, used: 356, 356
```

T2,

T3,

T6,

```
??, ??, ??, ??, ??, 609, used: ??, ??, ??,
                                                      Tensor (module), 295, 344, 287, used: 295, 296,
        296, ??, 307, 355, 358, 287, 287, 298
                                                      Tensor (sig), 343, 333, used: 344, 335, 335
        ??, ??, ??, 609
                                                      TENSOR, ??, ??, used: ??, 297
table (type), 694, 307, 308, used: 307, 307, 307,
                                                      TENSOR (camlyacc token), 298, used: 300
        307, 307
                                                      tensors, 305, used: 306
                                                      TensorScalar_Scalar_Scalar, 180, used: ??
tableau (type), 694, 693, used: 693, 693, 693,
                                                      TensorScalar_Scalar_Scalar_cf, 180, used: ??
        693, 693, 693
tableau\_columns, 694, used: 695
                                                      TensorScalar_Vector_Vector, 180, used: ??
tableau_of_array, 695, used: 695, 696
                                                      TensorScalar_Vector_Vector_cf, 180, used: ??
tableau_of_diagram, 695, used: 695, 696
                                                      TensorVector_Scalar_Scalar, 180, used: ??
tableau\_rows, 694, used: 695
                                                      TensorVector_Scalar_Scalar_cf, 180, used: ??
tables_filled, ??, used: ??
                                                      TensorVector_Vector_Vector, 180, used: ??
tag (field), 621, used: 622, 622
                                                      TensorVector_Vector_Vector_cf, 180, used: ??
tagged (type), 621, used:
                                                      Tensor_1, 178, used: ??, ??, ??, ??, ??
Tagged (module), ??, used: ??, ??
                                                      Tensor_2, 178, used: 355, ??, ??, ??, ??, ??
Tagged\_Binary \text{ (module)}, ??, 126, ??, used:
                                                      Tensor_2\_Scalar\_Scalar, 180, used: ??, ??, ??
Tagged_Coupling (module), ??, used: ??, ??, ??,
                                                      Tensor_2\_Scalar\_Scalar\_cf, 180, used: ??, ??,
        ??, ??
Tagged\_Coupling (sig), ??, used: ??
                                                      Tensor_2_Vector_Vector, 180, used: ??, ??, ??,
Tagged_Maker (sig), ??, 126, ??, used: 126, ??
                                                              ??. ??
Tagger (sig), ??, 126, ??, used: ??, ??, ??, 126,
                                                      Tensor_2_Vector_Vector_1, 180, used: ??, ??,
                                                              ??, ??, ??, ??
Tags (module), ??, ??, ??, used: ??, ??, ??, ??,
                                                      Tensor_2 - Vector_Vector_cf, 180, used: ??, ??,
        ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                              ??, ??, ??
Tags (sig), ??, 126, ??, used: ??, ??, 126, ??
                                                      Tensor_2 - Vector_1 - Vector_1, 180, used: ??, ??,
take\_column, 694, used: 694
Tan, 337, 203, used: 337, 339
                                                      tensor_attribute (camlyacc non-terminal), 301,
Tana, ??, used:
                                                              used: 300
Tanb, ??, used:
                                                      tensor_attributes (camlyacc non-terminal), 300,
Tanh, 337, 203, used: 337, 339
                                                              used: 300, 300
Target (module), 246, used: 596, 606, 606, 596,
                                                      tensor\_map, 322, used: 322
        596, 596, 448, 448, 448
                                                      tensor_ok, 316, 320, 322, 324, used: 324
Targets (module), 448, 378, 448, used: 606, ??,
                                                      tensor\_space, 309, used: 310
        ??, ??, 607, ??, ??, ??, ??, ??, ??, ??,
                                                      term (type), 364, used: 364, 365
        Term (module), 719, used:
        Term (sig), 717, used: 719, 719
        term\_to\_string, 344, used: 344
        ??, ??, ??, ??, ??, ??, ??, 611, ??,
                                                      Ternary (module), 10, 22, 6, 16, used: 13, 22, 16
        ??, ??, ??, ??
                                                      Ternary (sig), 10, 6, used: 6
                                                      test (type), ??, ??, used: ??, 736, 687, 697, 214,
Targets_Kmatrix (module), 572, 572, used:
                                                              303, 358, ??, ??, ??, ??, ??, ??, ??, ??,
Targets_Kmatrix_2 (module), ??, ??, used:
Tau, ??, used: ??, ??, ??, ??
                                                              ??, ??, ??, 735, 444, 640, 625, 635, 715,
Tbar, ??, used: ??, ??, ??, ??
                                                              675, 684, 693, 214, 303, 335, 365, 423, 379
tbar\_gl\_t, 326, used: 326, 326, 326, 326, 326, 327
                                                      Test (module), 736, 446, 642, 631, 637, 681,
TBWA, ??, ??, ??, ??, used: ??, ??, ??, ??,
                                                              687, 697, 324, 735, 444, 640, 625, 635,
        ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                              715, 716, 717, 675, 684, 693, 303, 365,
TBWZ, ??, ??, ??, ??, used: ??, ??, ??, ??,
                                                              423, 379, used:
        ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                      Test (sig), 697, 303, 358, 715, 693, 303, 335,
TCGG, ??, used: ??, ??, ??
                                                              365, 423, 379, used: 304, 324, 715, 716,
Template (module), ??, ??, used: ??
                                                              717, 693, 303, 303, 365, 423, 379
                                                      TestCase, ??, ??, used: ??, ??, ??
tension (field), 703, 707, 700, used: 705, 707,
                                                      TestLabel, ??, ??, used: ??, ??, ??
tension_to_equation, 709, used: 709
                                                      TestList, ??, ??, used: ??, ??, ??
                                                      test\_ac, 217, used:
tensor, 316, 316, 320, 320, 321, 322, used: 322
tensor (type), 316, 320, 322, 324, used: 316,
                                                      test\_case\_count, ??, ??, used: ??, ??, ??
                                                      test\_case\_paths, ??, ??, used: ??
        316.324
tensor (camlyacc non-terminal), 300, used: 299
                                                      test\_compare\_empty, 637, used: 638
Tensor, 295, 296, 307, 307, 287, 287, used:
                                                      test\_compare\_equal, 638, used: 638
        296, ??, 310, 299
                                                      test\_compare\_less, 638, used: 638
                                                      test_compare_longer, 638, used: 638
```

```
test\_compare\_more, 638, used: 638
                                                         test_compare_shorter, 637, used: 638
test_decorate, ??, ??, used: ??
                                                         ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
test_event (type), ??, ??, used: ??
                                                         test_filter, ??, ??, used: ??
                                                         607, 607
test\_find\_all\_empty, 638, used: 639
                                                  ThoString (module), 640, 640, used: ??, 602
test\_find\_all\_first, 638, used: 639
                                                  thread, 664, 663, used:
test\_find\_all\_last, 639, used: 639
                                                  thread\_unfinished\_decays, 587, used: 587
test\_find\_all\_not\_last, 639, used: 639
                                                  thread\_unfinished\_decays', 587, used: 587, 587,
test_find_first_first, 638, used: 638
test\_find\_first\_last, 638, used: 638
                                                  three, 23, used: 24, 24, 25, 25, 25
test_find_first_not_found, 638, used: 638
                                                  Threeshl (module), ??, ??, used: ??, ??
test_find_first_not_last, 638, used: 638
                                                  Threeshl_ckm (module), ??, ??, used:
test_fun (type), ??, ??, used: ??, ??, ??, ??
                                                  Threeshl\_ckm\_no\_hf (module), ??, ??, used:
                                                  Threeshl_diet (module), ??, ??, used:
test\_list, 633, used: 633
test_num_columns_not_ok, 639, used: 639
                                                  Threeshl\_diet\_no\_hf (module), ??, ??, used:
                                                  Threeshl\_no\_ckm \text{ (module)}, ??, ??, used: ??
test\_num\_columns\_ok\theta, 639, used: 639
test\_num\_columns\_ok2, 639, used: 639
                                                  Threeshl\_no\_ckm\_no\_hf (module), ??, ??, used:
test_result (type), ??, ??, used: ??, ??, ??,
       ??
                                                  Threeshl_options (sig), ??, ??, used: ??, ??, ??,
                                                         ??, ??, ??, ??, ??
test\_rhs, ??, used: ??
test\_stem, 306, used: 306
                                                  three_gluon, ??, 609, used: ??, 609
test_suite, 214, 219, 214, used:
                                                  Thw, ??, used:
                                                  Th\_SF, ??, used:
test\_sum, 48, 52, used: 49, 52
                                                  Tilde, 288, 284, used: 288
TeX, 292, 293, 285, 286, used: ??, 300, 300
TEX, ??, ??, used: ??, 297
                                                  time, 737, 687, 687, 689, 684, used: 617, 617,
TEX (camlyacc token), 298, used: 300, 300
                                                         617, 618, 618, 687, 687
TeX\_Anti, 292, 285, used: ??, 300
                                                  timelike, 44, 48, 48, 52, 52, 43, 43, used: 48,
tex\_lbl, 703, used: 704, 704, 704
                                                         48, 52, 52, 590, 592, ??, ??, ??, ??
Timelike, 180, used: ??, ??, ??, ??, ??, ??, ??,
       ??, ??, ??, ??, used: ??, ??, ??, ??, ??,
                                                         ??, ??, ??, ??, ??, ??, 608
       ??, ??
                                                  timelike_map, 590, used: 590, 591
THDM (module), ??, ??, used: ??, ??
                                                  times, 214, 216, 214, used: 217, 218, 219
THDM\_CKM (module), ??, ??, used: ??, ??
                                                  TIMES, ??, ??, ??, used: ??, ??, 297, 331
THDM\_flags (sig), ??, ??, used: ??, ??, ??, ??
                                                  TIMES (camlyacc token), 298, 331, used: 299,
this (field), 705, 700, used: 706
                                                         301, 302, 303, 332, 332
ThoArray (module), 635, 635, used:
                                                  times\_minus\_cc, 364, used:
                                                  time\_fun, ??, used: ??
ThoFilename (module), 619, 619, used:
ThoList (module), 625, 623, used: 637, 664, 676,
                                                  time_to_string, 616, used: 618, 618
       678, 678, 678, 678, 679, 679, 681, 681,
                                                  tln, 625, 623, used: 625, 633
       682, 683, 684, 685, 685, 685, 685, 685,
                                                  TLR, 180, used: ??, ??, ??, ??, ??, ??, ??, ??,
                                                         ??, ??
       685, 686, 686, 686, 686, 686, 686, 687,
                                                  TLRM, 180, used: ??, ??, ??, ??, ??
       688, 688, 688, 688, 689, 701, 701, 707,
                                                  todo, ??, ??, used:
       707, 707, 707, 695, 13, 14, 14, 15, 15, 20,
       21, 27, 28, 49, ??, ??, ??, ??, ??, 591,
                                                  Todo (exn), ??, used:
                                                  token, 289, ??, ??, ??, ??, 284, ??, ??, ??, ??,
       291, 309, 309, 312, 315, 319, 319, 319,
       319, 319, 324, 324, 324, 325, 342, 346,
                                                         used: ??, ??, ??, ??, ??, ??, ??, ??,
       348, 348, 60, 61, 280, 280, 280, 282, 282,
                                                         ??, ??, 303, 304, ??, ??, ??, ??, ??, ??,
       283, 283, ??, ??, ??, ??, ??, ??, ??, ??,
                                                         ??, ??, ??, 336, ??, ??, 61, 57, 297, 331,
       ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                                         302, 302, 302
       598, 598, 601, 602, 603, ??, ??, ??, ??,
                                                  token (type), ??, ??, ??, ??, ??, ??, ??, used:
       ??, ??, ??, ??, ??, ??, ??, ??
       token (camllex parsing rule), 57, 297, 331, used:
       token (camlyacc non-terminal), 301, used: 302,
       302, 302
       Token,
                                                         288, 284, used: 289, 291
       Token (module), 288, 284, used: 291, 292, 292,
       292, 292, 292, 293, 293, 293, 293, 293,
       294, 295, 295, 295, 295, 295, 295, 295,
```

```
295, 296, 296, ??, 306, 306, 307, 311, 285,
                                                     to\_select\_vtx, 66, used: 67
        285, 285, 285, 286, 286, 287, 287, 287,
                                                     to\_select\_wf, 64, used: 67
        287, 287, 287, 298
                                                     to_string, 631, 670, 671, 671, 711, 681, 684,
TOKEN, ??, ??, used: ??, 297
                                                             685, 687, 703, 22, 23, 44, 45, 50, 588,
                                                             290, 292, 293, 293, 295, 295, 311, 313,
TOKEN (camlyacc token), 298, used: 302, 302,
                                                             338, 341, 342, 343, 343, 344, 351, 357,
tokens, 306, used: 306
                                                             56, 60, 63, 66, 281, 282, 625, 670, 670,
token_arg (camlyacc non-terminal), 299, used:
                                                             711, 715, 716, 717, 717, 718, 719, 683,
        299, 299, 300, 300, 300
                                                             700, 17, 42, 285, 285, 286, 286, 287,
token_arg_pair (camlyacc non-terminal), 299,
                                                             287, 332, 333, 334, 334, 365, 55, 59,
                                                             used: ??, ??, 631, 631, 631, 631, 632, 633,
        used: 299
token_list (camlyacc non-terminal), 301, used:
                                                             671, 621, 711, 681, 681, 688, 217, 218,
                                                             290, 290, 291, 292, 292, 292, 293, 295,
        299, 299, 301, 301
token_list_arg (camlyacc non-terminal), 299,
                                                             295, 296, 306, 306, 308, 308, 310, 312,
        used: 300, 300, 300, 301, 301
                                                             312, 312, 313, 338, 339, 342, 344, 346,
token_list_opt_arg (camlyacc non-terminal),
                                                             351, 351, 67, 281, ??, ??, 603, 604, 605
        299, used: 300, 300, 301
                                                     to_string', 293, 342, used: 293
       ??, used: ??, ??, ??, ??, ??, ??, ??
                                                     to_string1, 587, used: 587, 588
TopHb, ??, used: ??, ??, ??, ??, ??, ??, ??
                                                     to_strings, 296, 288, 359, used: 304, 304
Topology (module), 18, ??, 16, ??, used: ??, ??,
                                                     translate_vertices, 65, used: 66
        ??, ??, ??, ??
                                                     transpose, 628, 636, 214, 216, 624, 214, 364,
Topology3 (module), ??, ??, used: 597
                                                             378, used: 636, 636, 219, 219
Topp, ??, used: ??, ??, ??, ??, ??, ??
                                                     transpose_array, 695, used: 695, 696
Toppb, ??, used: ??, ??, ??, ??, ??, ??, ??
                                                     tree (type), 589, used: 590
Tree (module), 645, 701, 644, 699, used: 636,
        ??, ??, ??, ??, ??, ??, ??, ??, used:
                                                             684, 707, 33, 38, ??, ??, 596, 598, 598,
        599, 599, 599, 600, 600, 600, 600, 600,
        600, 600, 600, 605, 606, 32, 124, ??, 596
        Tree2 (module), 711, 711, used: 33, 38, ??, ??,
        ??, ??, ??, ??, ??, 31, 123, ??
        trees, 589, 590, 589, used: 604
        ??, ??, ??, ??
                                                     trees (field), 590, used: 590, 591
top\_anom\_4f, ??, ??, ??, ??, ??, ??, ??, ??,
                                                           658, 661, used: 658, 658, 659, 659, 659,
        ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                                             661, 661, 661, 661, 661
        used: ??, ??, ??, ??, ??, ??, ??, ??.
                                                     Trie (module), 658, 656, used:
        triples, 691, ??, ??, 690, used: 11, 13, ??, ??
                                                     triplet_gauge2_higgs, ??, ??, used: ??, ??
        ??, ??, ??, ??
                                                     triplet_gauge_higgs, ??, ??, used: ??, ??
top_quartic, ??, ??, ??, used: ??, ??
                                                     triple_anom, ??, ??, ??, ??, ??, ??, ??, ??,
tower\_of\_dag, ??, used:
tower_to_dot, ??, ??, 124, ??, used: 604
                                                             ??, ??, ??, ??, ??, ??, ??, ??, ??,
to_channel, 711, 711, used: 622, 622, 712, ??, ??
                                                             used: ??, ??, ??, ??, ??
                                                     triple_gauge, ??, ??, ??, ??, ??, ??, ??, ??,
to_coupling, 339, 332, used: 339
                                                             to_feynmf_channel, 704, used: 705, 706
                                                             used: ??, ??, ??, ??, ??, ??, ??, ??,
to\_float, 715, used:
                                                             ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
to_fortran, 316, 320, used: 325
                                                     triple_gravitino, ??, used: ??
to\_gauss, 65, used: 67
to_int, 22, 23, 315, 315, 315, 323, 17, used: 27
                                                     triple_gravitino', ??, used: ??
                                                     triple_gravitino", ??, used: ??
to_integer, 715, used: 346
to_ints, 44, 45, 50, 41, used: 50, 54, 587, 591,
                                                     triple_to_nested, ??, used: ??
        62, 66, ??, ??, ??, ??
                                                     TRL, 180, used: ??, ??, ??, ??, ??, ??, ??,
                                                             ??, ??
to\_ints', 50, used: 50, 50
to_list, 7, 9, 11, 13, 15, 6, used: 630, 631, 637,
                                                     TRLM, 180, used: ??, ??, ??, ??, ??
        684, 685, 686, 686, 687, 687, 687, 709,
                                                     True, 55, 61, 55, used: 56, 61, 61, 61
        709, 325, ??, ??, ??, ??, ??, ??, ??, ??,
                                                     truncated\_arity, 14, used: 14
        ??, ??, 597, 602
                                                     try_add, 44, 46, 51, 42, used: 48, 52
to_lists, 671, 671, 670, used: 66, 282
                                                     try_add', 46, used: 46, 46
to_momentum, 53, 54, 54, 43, used: 54
                                                     try\_first, 621, used: 621
to\_on\_shell, 64, used: 67
                                                     try_fusion, 44, 49, 52, 42, used: 587
to_ratio, 715, used: 339
                                                     try\_of\_momenta, 588, used:
                                                     try_sub, 44, 47, 51, 42, used: 48, 51, 52, 587
to_selectors, 60, 67, 59, used: 603, 604
to\_select\_p, 64, used: 67
                                                     try\_sub', 46, used: 46, 47
```

```
try_thread_unfinished_decays, 588, used: 588
                                           u1_gauged, ??, ??, ??, ??, used: ??, ??, ??,
TTGG, ??, ??, ??, ??, used: ??, ??, ??, ??,
                                                 ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                 ??, ??, ??, ??, ??, ??, ??, ??, ??
TTWW, ??, ??, ??, ??, used: ??, ??, ??.
                                           Ubar, ??, used: ??, ??, ??, ??
      ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                           UED (module), ??, ??, used: ??
tt_threshold, ??, ??, ??, ??, ??, ??, ??, ??,
                                           UFO, 184, used: 65
      ??, ??, ??, used: ??, ??, ??
                                           UFO (module), 379, 377, used: ??, ??, ??
tt\_threshold\_ttA, ??, used: ??
                                           UFOx (module), 335, 332, used: 365, 365, 378
tt\_threshold\_ttZ, ??, used: ??
                                           UFOx_lexer (module), ??, ??, 330, used: 336
TUGG, ??, used: ??, ??
                                           UFOx\_parser \text{ (module)}, ??, ??, 331, used: ??,
                                                 336, ??, 330
Tuple (module), 7, 21, 27, 5, used: 19, 20, 20,
      21, 21, 22, 22, 27, 27, 27, 28, 28, 34, ??,
                                           UFOx\_syntax \text{ (module)}, \quad 329, 329, \text{ used: } ??, ??,
      ??, ??, ??, ??, ??, ??, ??, ??, ??, 606,
                                                 ??, 336, 336, 336, 337, 337, 343, 343, 344,
                                                 351, 356, ??, 334, 334, 331, 331, 332
      16, 18, 18, 30, 124, 124, 126, ??, ??, ??
                                           UFO_lexer (module), ??, ??, 360, used:
tuples, 690, 690, used: 690, 691, 15, 20, 27
tuples', 690, used: 690, 690
                                           UFO_Lorentz (module), 365, 364, used: 422,
tuple\_of\_list2, 20, used: 20
                                                 422, 378
TVA, 180, used: ??, ??, ??, ??, ??, ??, ??,
                                           UFO_parser (module), ??, ??, 361, used: ??, ??
      ??, ??, ??, ??, ??, ??, ??
                                           UFO_syntax (module), 359, 358, used: ??, ??,
TVAM, 180, used: ??, ??, ??, ??, ??, ??, ??,
                                                 ??, ??, 377, 378, 361, 361, 362
      UFO_targets (module), 423, 422, used:
                                           UFO_tools (module), ??, ??, used: ??, ??, ??,
      ??, ??, ??, ??, ??, ??, ??
    23, 217, used: 24, 24, 25, 25, 25, 26, 26, 26,
                                                 ??, ??, ??, 331
two.
                                           Unbounded_Nary (module), 15, 7, used:
      217
TwoHiggsDoublet (module), ??, ??, used: ??, ??,
                                           uncolored_colored, 600, used: 600
      ??, ??
                                           uncolored\_only, 600, used: 600
                                           uncolorize_wf, ??, used: ??
two_and_one, ??, ??, used: ??, ??, ??, ??
two_and_one', ??, ??, used: ??, ??, ??, ??two_of_each, 319, used: 319
                                           uncompress, 636, 635, used: 636
                                           uncompress2, 636, 635, used:
two\_unit, 217, used: 217
                                           uncons, 645, 647, 651, 644, used:
                                           uncons_opt, 645, 647, 651, 644, used:
t_channel (field), 586, used: 586, 586, 587, 588
U, 293, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                           Undefined (exn), 735, 736, 735, used:
      unfinished_decays (type), 586, used:
      used: 293, ??, ??, ??, ??, ??, ??, ??,
                                           unfinished_decays_of_momenta, 587, used: 587,
      588
      unfold, 591, used: 591
      unfold1, 591, used: 591
      unfold_tree, 591, used: 591
      Unhandled (exn), 378, used:
      uninitialized, ??, used: ??, ??, ??, ??, ??, ??,
                                                 ??, ??, ??, ??, ??, ??
      Uninitialized (exn), ??, used:
      union, 645, 649, 652, 671, 671, 644, 670, used:
      ??, ??, ??, ??, ??, ??, ??, ??, ??, ??.
                                                 649, 652, 630, 671, 672, 672, 330, 330,
      345, 345, 66, 281, 282, 603
      uniq,
                                                628, 636, 61, 598, 623, 635, used: 681,
      309, 309, 315, 342, 348, 61, 61, ??, 598,
      598, 602
      uniq (field), 635, used: 636, 636, 636, 636
      uniq', 628, used: 628, 628
      uniq2, 636, 635, used:
      uniq2 (field), 636, used: 636, 636, 636
      unique_final_state, 278, used:
      unique_flavors, 278, used: 278
      unit, 214, 215, 715, 716, 716, 717, 718, 213,
      ??, ??, ??, ??, ??, ??, ??, ??, ??
                                           ??, ??, ??, ??, ??, ??, used: ??, 615,
U (module), ??, 361, used: ??, 362, 362, 362,
                                                 ??, 600, ??, ??, ??, ??, ??, ??, ??, ??,
      362, 363, 363, 363
```

```
valid_tableau, 695, 693, used: 695
       ??, ??, ??, ??, ??, ??, ??
                                                 value (field), 621, 293, 286, used: 293, ??, 300
                                                 value (type), 620, 621, 620, 359, used: 621, 621,
Unitarity\_Gauge (sig), 212, used:
unit\_tension, 707, used:
                                                         620, 620, 359, 359
Unmatched, 345, used: 345
                                                 value (camlyacc non-terminal), 362, used: 362
Unordered (exn), 584, 585, 583, used:
                                                 Value (module), 337, 332, used: 349, 349, 350,
unpack, 341, 342, 333, used: 342, 342, 354
                                                         351, 335
unpack\_constant, 65, used: 66
                                                 Value (sig), 621, 620, used: 621, 620
unpack\_map, 591, used: 591
                                                 VAM, 180, used: ??, ??
unphysical_of_flavor, ??, used: ??
                                                 vanilla, 705, 700, used: 705
unphysical_of_flavors, ??, used: ??
                                                 Vanishing, 180, used: ??, ??, ??, ??, ??, ??, ??,
                                                         unphysical_of_flavors1, ??, used: ??
unphysical_of_lorentz, ??, used: ??
                                                         ??, ??, ??
unphysical_polarization, 601, used: 601, 603
                                                 Vanishing\_couplings (exn), ??, used:
                                                 vanishing_flavors, ??, ??, 125, ??, used:
unquote, ??, 57, used: ??, 57
Uodd, ??, used: ??, ??, ??, ??
                                                 vanishing_flavors (field), ??, used: ??, ??
update, 645, 648, 650, 644, used: 648, 648, 649,
                                                 variable, 330, 343, 350, 356, ??, ??, 598, 329,
       650, 650
                                                         334, used: ??, 345, ??, ??, ??, 605, 332
upper (camllex regexpr), 57, 297, 331, used: 57,
                                                 variable (type), 208, used: 208
                                                 Variable, 329, 338, 349, 349, 329, 335, used:
       297, 331
                                                        330, 336, 337, 340, 350, 352
uppercase, 642, 640, used:
                                                 variable', 598, used: 600, 600, 600
used (field), 343, used: 343, 343
use_channel, 617, used: 617, 618, 618
                                                 variables, 330, 337, 343, 345, ??, ??, ??, ??,
use_fudged_width, ??, ??, ??, ??, ??, ??, ??, ??,
                                                         329, 332, 334, 124, ??, used: 330, 337
       variables (field), 378, used:
       ??, used: ??, ??, ??, ??, ??, ??, ??,
                                                 variables\_of\_atoms, 345, used: 345
       variables\_of\_linear, 345, used: 345
       variables\_set, 345, used: 345
       ??, ??, ??, ??, ??, ??, ??, ??, ??
                                                 variable_array (type), 208, used: 208
                                                 VA\_ILC\_ttA, ??, used: ??
Using_Arrays (module), 685, 683, used: 687
                                                 VA\_ILC\_ttZ, ??, used: ??
Using\_Lists (module), 684, 683, used:
U_{-}K1_{-}L, ??, used: ??, ??, ??
                                                 vc (type), ??, ??, used: ??, ??, ??
U_K1_R, ??, used: ??, ??, ??
                                                 vector, 316, 316, 320, 320, 320, 322, used: 322,
U_K2_L, ??, used: ??, ??, ??
U_K2_R, ??, used: ??, ??, ??
                                                 vector (field), 365, used:
v (field), 24, used: 24, 24, 25, 25
                                                 vector (type), 316, 317, 348, 349, 335, used:
V, 316, 318, 353, 180, used: 318, 326, 326, 326,
                                                         316, 318, 349, 349, 349, 335, 335, 365
       326, 327, 327, 353, 354, 354, 354, ??, ??,
                                                 Vector,
                                                         294, 316, 317, 349, 349, 178, 286, 335,
       used: 317, 326, 326, 350, 350, 352, 355,
       ??, ??, 607
                                                 Vector4, 182, used: ??, ??, ??, ??, ??, ??, ??,
       ??, ??, 608
V \text{ (module)}, ??, 298, used: ??, 300, 300
                                                         ??, ??, ??, ??, ??, 609
V2, 180, used: ??
                                                 Vector_4 K_Matrix_cf_m\theta, 182, used: ??, ??
                                                 Vector 4 \_K \_Matrix \_cf \_m1, 182, used: ??, ??
V2LR, 180, used: ??, ??
v3 (field), ??, used: ??, ??, ??
                                                 Vector 4 \_K \_Matrix \_cf \_m7, 182, used: ??, ??
V3, 184, used: ??, ??
                                                 Vector_4 K_Matrix_cf_t0, 182, used: ??, ??
v4 (field), ??, used: ??, ??, ??
                                                 Vector4_K_Matrix_cf_t1, 182, used: ??, ??
                                                 Vector_4 K_Matrix_cf_t^2, 182, used: ??, ??
V4, 184, used: ??, ??
VA, 180, used: ??, ??, ??, ??, ??, ??, ??,
                                                 Vector \angle K_Matrix\_cf\_t\_rsi, 182, used: ??, ??
       Vector4_K_Matrix_jr, 182, used: ??, ??, ??,
       ??, ??
       ??, ??
                                                 Vector 4 \_K \_Matrix \_tho, 182, used:
VA2, 180, used: ??, ??, ??, ??, ??
                                                 Vectorspinor, 178, used: 355, ??, ??
VA3, 180, used:
                                                 vectorspinors, 355, used: 355
VA3M, 180, used:
                                                 vector\_contraction\_ok, 319, 319, used: 319
valid_diagram, 694, 692, used: 695
                                                 vector\_current\_color\_mismatch, 327, used: 327
valid_fortran_id, 642, 640, used: 642, 642, 643,
                                                 vector_current_conjspinor_misplaced, 326, used:
       643, 643, 643, 643
                                                         327
```

```
vertices4", ??, used: ??
vector\_current\_ok, 326, used: 327, 327
                                                vertices4", ??, used: ??
vector_current_out_of_bounds, 327, used: 327
                                                vertices_aaww, ??, used: ??
vector_current_spinor_misplaced, 326, used: 327
                                                vertices_all, ??, used: ??
vector_current_vector_misplaced, 326, used: 327
                                                vertices_aqq, ??, used: ??
vector_field, 326, used: 326, 326, 326, 326
                                                vertices_aww, ??, used: ??
vector\_index, 352, used: 352
                                                vertices_ggg, ??, used: ??
vector_indices, 319, used: 319, 319
                                                vertices_gggg, ??, used: ??
vector\_map, 322, used: 322
vector\_ok, 317, used: 318
                                                vertices_gqq, ??, used: ??
Vector_Scalar_Scalar, 180, used: ??, ??, ??, ??,
                                                vertices_nocache, ??, used: ??
       vertices_to_string, 63, used: 63
       vertices_wll, ??, used: ??, ??
                                                vertices_wll_diet, ??, used: ??
       609
                                                vertices_wqq, ??, used: ??
vector_to_string, 351, used: 351
                                                vertices\_wqq\_no\_ckm, ??, used: ??, ??
version, 614, used:
vertex (camlyacc non-terminal), 58, 301, used:
                                                vertices_wqq_no_ckm_diet, ??, used: ??
                                                vertices_wwww, ??, used: ??
       58, 299
Vertex, 295, 296, 287, 287, used: 296, ??, 299
                                                vertices_wwza, ??, used: ??
                                                vertices_wwzz, ??, used: ??
Vertex (module), 303, 303, used: 65, 184
VERTEX, ??, ??, used: ??, 297
                                                vertices_zll, ??, used: ??
                                                vertices_zqq, ??, used: ??
VERTEX (camlyacc token), 298, used: 301
vertex3 (type), 180, used: ??, ??, 184, 210, 211,
                                                vertices_zww, ??, used: ??
                                                ??, ??, ??, ??, ??, ??, used: ??, ??,
             182, used: ??, ??, 184, 210, 211,
vertex4 (type),
                                                       ??, ??, ??
       ??
                                                VHeavy, ??, ??, ??, used:
vertexn (type), 184, used: ??, ??, 184, 210, 211,
                                                vintage, ??, ??, 122, ??, used:
       ??
                                                VL, 180, used: ??, ??, ??, ??, ??, ??, ??,
vertex_indices_broken, 327, used: 328
                                                       vertex_indices_ok, 327, used: 328
                                                       Vertex_lexer (module), ??, ??, 296, used: 303,
                                                       304
Vertex_parser (module), ??, ??, 298, used: ??,
                                                       ??, ??, ??, ??, ??, ??
       303, 304, ??, 296
                                                VLR, 180, used: ??, ??, ??, ??, ??, ??, ??,
                                                       Vertex_syntax (module), 288, 284, used: ??, ??,
       ??, 303, 304, 304, 304, 306, 306, 306, 307,
                                                       ??, ??
       311, ??, 303, 298, 298, 299
                                                VLRM, 180, used: ??, ??, ??, ??, ??, ??,
vertex_table (type), ??, used:
                                                       ??, ??, ??
vertex_to_string, 63, used: 63
                                                VM (module), 448, used: ??, ??, ??, ??, ??, ??,
vertices, ??, ??, ??, ??, 306, ??, ??, ??, ??, ??,
                                                       ??, ??, ??, ??
       VMaj, 353, used: 353, 354, 354
       VMOM, 180, used:
       ??, 609, 210, used: ??, ??, ??, ??, 306,
                                                vn (field), ??, used: ??, ??, ??
       Vn, 184, used: ??
       VR, 180, used: ??, ??
       ??, ??, ??, ??, ??, ??, ??, 609
                                                VSet (module), ??, 597, used: ??, ??, 603
vertices (field), 61, 378, used: 61, 61, 66
                                                VSp, 353, used: 353, 354, 354
vertices (label), ??, ??, 211, used:
                                                vtx (type), 61, used: 61
vertices (type), ??, used: ??
                                                V_{-}\theta, ??, used:
vertices3, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                                V \_ CKM, ??, used:
       V_{-}P, ??, used:
       609, used: ??, ??, ??, ??, ??, ??, ??,
                                                w, ??, used: ??
       ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
       ??, ??, 609
                                                       ??, ??, ??, ??
vertices3'', ??, used: ??
vertices3'', ??, used: ??
                                                W \text{ (module)}, 597, used: 604
                                                was_successful, ??, used: ??, ??, ??
vertices4, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                                wavefunctions, ??, ??, used: ??, ??
       Weyl, 294, 286, used:
       609, used: ??, ??, ??, ??, ??, ??, ??,
                                                wf (field), 61, used: 61, 61, 61, 64, 67
                                                wf (type), 61, ??, ??, ??, ??, ??, ??, ??, ??,
       ??, ??, 609
                                                       ??, 122, 125, 126, ??, ??, ??, used: 61,
vertices4', ??, used: ??
```

```
??, ??, ??, ??, 122, 122, 122, 122, 123,
                             123, 123, 123, 124, 124, 124, 125, 125,
   126, 126, ??, ??, ??, ??, ??, ??, ??, ??,
                             ??, ??, ??, ??, ??, ??, ??
                             wf1, 597, used: 598
WFMap (module), ??, ??, used: ??, ??, ??, ??
                             WFMap2 (module), ??, used: ??
WFSet (module), ??, ??, used: ??, ??, ??
                             WFSet2 (module), ??, used: ??, ??
                             WFTSet (module), ??, used: ??
                             wf\_declaration, 325, used: 325
wf_tag, ??, ??, ??, 122, ??, used: ??
                             wf_tag (field), ??, ??, used: ??, ??, ??, ??, ??,
                             ??, ??, ??, ??, ??
                             wf\_tag\_raw, ??, ??, used:
                             wf_to_string, 62, ??, ??, ??, 126, ??, used:
                             62, 63, ??
wf\_to\_TeX, 600, used: 600, 600, 600
                             ??, ??, ??, ??, ??
white (camllex regexpr), 57, 297, 331, used: 57,
                            ??, used: ??, ??, ??, ??
   297.331
                         Wm1.
Whizard (module), 589, 589, used: 597
                         Wm2.
                            ??, used: ??, ??, ??
Whizard (sig), 53, 43, used: 589, 43, 589
                         word (camllex regexpr), 331, used: 331
                         whizard\_tree, 591, used:
whizard_tree_debug, 591, used:
                             ??, ??, ??, ??, ??, ??, ??, ??, used:
WHm, ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                             ??, ??, ??, ??
                             ??, ??, used: ??, ??, ??, ??, ??, ??, ??,
                             WHp,
   ??, ??, ??, ??
                             width.
   ??, ??, ??, ??, ??, ??, ??, ??, 608,
                             210, used: ??, ??, ??, ??, ??
                             width (field), 444, used: 445
width (label), ??, ??, 211, used: 447, 447, 447,
                             448
                             width (type), 180, used: 210, 211
                            292, 349, 349, ??, ??, ??, ??, ??, ??, ??,
   ??, ??, ??, ??, ??, ??, ??, ??, ??,
                             285, 335, used: ??, 349, 350, 352, 300
                             WIDTH, ??, ??, used: ??, 297
                             WIDTH (camlyacc token), 298, used: 300, 300
width_symbol, ??, ??, ??, ??, ??, ??, ??, ??,
                             611, 211, used: ??, ??, ??, ??, ??
                             width\_symbol (label), ??, ??, 211, used:
                             with_infinity, 703, used: 703
                             with_supremum (type), 702, used:
with_supremum_or_infinity (type), 703, used:
                             ??, ??, ??, ??, ??, ??
Wp1, ??, used: ??, ??, ??, ??
   ??, ??, ??, ??, ??, ??, ??, ??, used:
                         Wp2, ??, used: ??, ??, ??, ??
   wrap_list, 289, 285, used: 289, 312
   wrap_scripted, 288, 284, used: 312
   write, 621, 622, 589, 592, 620, 589, used:
   write\_dir, 621, 622, 620, used:
   write\_fusion, 325, used: 326
   write\_fusions, 326, used: 327
```

```
write_interface, 594, 589, used:
                                              yuk\_lgino\_ec\_su2,
                                                              ??, used: ??
                                              yuk\_lgino\_ec\_su2', ??, used: ??
write_interface_function, 593, used: 594
                                              yuk\_lqino\_nc\_sd1, ??, used: ??
write_interface_subroutine, 593, used: 594
                                              yuk\_lqino\_nc\_sd2, ??, used: ??
write_makefile, 594, 589, used:
                                              yuk\_lqino\_phiggs, ??, used: ??
write_makefile_processes, 595, 589, used:
write_other_declarations, 594, used: 594
                                              yuk_lqino_se_uc1, ??, used: ??
write_other_interface_functions, 593, used: 594
                                              yuk\_lqino\_se\_uc1', ??, used: ??
wwzz, 327, used: 327, 327, 327
                                              yuk\_lqino\_se\_uc2, ??, used: ??
                                              yuk\_lqino\_se\_uc2', ??, used: ??
WZW (module), ??, ??, used: ??
X (module), ??, 307, ??, 298, 331, used: ??,
                                              yuk_lqino_shiggs, ??, used: ??
                                              yuk\_lqino\_sn\_dc1, ??, used: ??
       309, 310, 310, ??, 300, 301, 332
                                              yuk\_lqino\_sn\_dc2, ??, used: ??
X0, ??, used: ??, ??, ??, ??, ??, ??, ??
Xdim \text{ (module)}, ??, ??, used: ??
                                              yuk\_lq\_ec\_uc, ??, used: ??
                                              yuk_lq_ec_uc', ??, used: ??
XiA, ??, used: ??
XiW, ??, used: ??
                                              yuk\_lq\_ec\_uc2, ??, used: ??
XiZ, ??, used: ??
                                              yuk\_lq\_ec\_uc2', ??, used: ??
Xm, ??, used: ??, ??, ??, ??, ??, ??, ??
                                              yuk\_lq\_nc\_dc, ??, used: ??
Xp, ??, used: ??, ??, ??, ??, ??, ??, ??
                                              yuk\_lq\_nc\_dc2, ??, used: ??
                                              yyact, ??, ??, ??, used: ??, ??, ??, ??
xy\_currents, ??, used: ??
                                              yycheck, ??, ??, ??, used: ??, ??, ??, ??
x-couplings (field), ??, used: ??
                                              yydefred, ??, ??, ??, used: ??, ??, ??, ??
X_{-}Flavor, 55, 55, used: 56
                                              yydgoto, ??, ??, ??, used: ??, ??, ??, ??
x-flavors (field), ??, used: ??
X\_Vertex, 55, 55, used: 56
                                              yygindex, ??, ??, ??, used: ??, ??, ??, ??
Y0, ??, used: ??, ??, ??, ??, ??
                                              yylen, ??, ??, ??, used: ??, ??, ??, ??
Young (module), 694, 692, used:
                                              yylhs, ??, ??, ??, used: ??, ??, ??, ??
yukawa, ??, ??, ??, ??, ??, ??, ??, ??, ??,
                                              yynames_block, ??, ??, ??, used: ??, ??, ??,
      ??, ??, ??, ??, used: ??, ??, ??, ??,
       ??, ??, ??, ??, ??, ??, ??, ??, ??, ??
                                              yynames_const, ??, ??, ??, ??, used: ??, ??, ??,
yukawa_add, ??, ??, ??, used: ??, ??, ??
yukawa\_add', ??, ??, used: ??, ??
                                              yyrindex, ??, ??, ??, used: ??, ??, ??, ??
yukawa_c, ??, ??, ??, used: ??, ??, ??
                                              yysindex, ??, ??, ??, used: ??, ??, ??, ??
yukawa_cq, ??, ??, ??, used: ??, ??, ??
                                              yytable, ??, ??, ??, used: ??, ??, ??, ??
yukawa\_cq', ??, ??, used: ??, ??, ??
                                              yytables, ??, ??, ??, used: ??, ??, ??, ??
yukawa\_cq'', ??, used: ??
                                              yytablesize, ??, ??, ??, used: ??, ??, ??, ??
yukawa_c_1, ??, used: ??
                                              yytransl_block, ??, ??, ??, used: ??, ??, ??,
yukawa_c_2, ??, ??, used: ??, ??, ??
                                                     ??
yukawa_c_3, ??, ??, ??, used: ??, ??, ??
                                              yytransl_const, ??, ??, ??, used: ??, ??, ??,
yukawa_goldstone, ??, used: ??
yukawa_goldstone_2, ??, used: ??
                                                ??, used:
yukawa\_goldstone\_2', ??, used: ??
                                                 316, 320, ??, ??, ??, ??, ??, ??, ??, ??,
yukawa\_goldstone\_2'', ??, used: ??
                                                     yukawa_goldstone_quark, ??, used: ??
                                                     ??, used: 321, 322, 322, ??, ??, ??, ??,
yukawa_higgs, ??, ??, ??, used: ??, ??, ??
                                                     yukawa_higgs_2, ??, ??, ??, used: ??, ??, ??
                                                     yukawa_higgs_2', ??, used: ??
yukawa_higgs_2", ??, used: ??
                                                     yukawa_higgs_FFP, ??, ??, used: ??, ??
                                                     yukawa_higgs_FFS, ??, ??, used: ??, ??
                                                     yukawa\_higgs\_NLC, ??, ??, used: ??, ??
                                                     yukawa_higgs_quark, ??, ??, ??, used: ??, ??, ??
                                                     yukawa_n, ??, ??, used: ??, ??, ??
                                                     yukawa_n_1, ??, used: ??
yukawa_n_2, ??, ??, ??, used: ??, ??, ??
                                                     yukawa_n_3, ??, ??, ??, used: ??, ??, ??
                                                     yukawa_n_4, ??, used: ??
                                                     yukawa_n_5, ??, ??, used: ??, ??, ??
                                                     yukawa\_phiggs\_2, ??, ??, used: ??, ??
                                                     yukawa_shiggs_2, ??, ??, used: ??, ??
                                                     yukawa_v, ??, ??, ??, used: ??, ??, ??
                                                     yuk\_lqino\_ec\_su1, ??, used: ??
yuk\_lqino\_ec\_su1', ??, used: ??
```

```
??, ??, ??, ??, ??, ??, ??, ??, ??
Z \text{ (module)}, 249, 248, used:
Z1, ??, used: ??, ??, ??, ??
Z2, ??, used: ??, ??, ??, ??
zero, 22, 23, 44, 45, 50, 214, 215, 215, 17, 42,
     213, used: 24, 24, 24, 24, 25, 25, 25, 26,
     26, 26, 26, 27, 215, 215, 215, 215, 216,
     216, 216, 216, 216, 216, 217, 219, 219,
     221, 221, 221, 221, 221, 60, ??, ??
Zero (exn), 24, used:
ZH, ??, ??, ??, ??, used: ??, ??, ??, ??, ??, ??,
     ??, ??, ??, ??, ??, ??
Zprime (module), ??, ??, used: ??, ??
ZZ (module), 249, 248, used: ??, ??, ??
__ocaml_lex_string1_rec, ??, used: ??, ??
__ocaml_lex_string2_rec, ??, used: ??, ??
\_-ocaml\_lex\_tables, ??, ??, ??, ??, used: ??, ??,
     ??, ??, ??, ??
\_-ocaml\_lex\_token\_rec, ??, ??, ??, ??, used: ??,
     ??, ??, ??, ??, ??, ??, ??
```