The CiME system Version 2.02

Evelyne Contejean, Claude Marché, Benjamin Monate, and Xavier

Urbain

LRI, CNRS UMR 8623

Bât. 490

Université Paris-Sud, Centre d'Orsay

91405 Orsay Cedex, France

Phone:+33 1 69 15 64 85

Fax: +33 1 69 41 65 86

Email: {contejea, marche, monate, urbain}@lri.fr

This research was supported in part by the EU project CCL, the HCM Network CONSOLE, and the "GDR de programmation du CNRS".

Contents

I	Use	er man	ual	9			
1	Tuto	Tutorial					
	1.1	The to	plevel language	11			
		1.1.1	Expressions	11			
		1.1.2	Definitions with let	11			
		1.1.3	Definitions of functions	12			
		1.1.4	Higher-order functions	13			
	1.2	More t	oplevel user interaction	13			
		1.2.1	Loading expressions from files	13			
		1.2.2	Interrupting evaluation	14			
		1.2.3	Time spent in computation	14			
		1.2.4	Quitting CiME	14			
	1.3	Playing	g with the Diophantine constraints solver	14			
	1.4	String	rewriting	16			
	1.5		eterized string rewriting system	16			
	1.6	Definit	ion of first-order signatures	16			
	1.7	Definition of terms					
	1.8	Rewriting					
	1.9	Checki	ing termination of rewrite systems	17			
		1.9.1	Search for incremental and modular termination proof	17			
		1.9.2	Examples of modular expert sessions	18			
2	Refe	erence n	nanual	29			
	2.1	CiME	toplevel core language	29			
		2.1.1	Formal syntax	29			
		2.1.2	A note on static binding	29			
		2.1.3	Directives	30			
	2.2	CiME	toplevel predefined functions	30			
		2.2.1	Diophantine constraints	30			
		2.2.2	Rewriting on words	30			
		2.2.3	Rewriting on first-order terms	31			
		2.2.4	Defining term orderings	32			
		2.2.5	Checking termination	33			
		2.2.6	Finding termination proofs	33			
Bi	bliogi	raphy		33			
In	dex			33			

II	Im	plementation manual	35
3	Libr	ary Compat	37
	3.1	Module Int_utils	37
	3.2	Module Arrayutils	37
	3.3	Module Listutils	37
	3.4	Module Balanced_trees	38
	3.5	Module Mapord	39
	3.6	Module Balanced_trees2	39
	3.7	Module Ordered_maps	40
	3.8	Module Bit_field	40
	3.9	Module Numbers	41
	3.10	Module Ordered_sets	43
	3.11	Module <i>Cache</i>	44
		3.11.1 Type of caches, and basic access functions	44
		3.11.2 Iterator	45
		3.11.3 Statistics	45
	3.12	Module Ordered_types	45
	3.13	Module Class_maps	45
	3.14	Module Polynomials	45
		3.14.1 Module for coefficient rings	46
		3.14.2 Module for polynomials	46
		3.14.3 The polynomial functor	48
	3.15	Module Classical_maps	49
	3.16	Module Labelled_graphs	49
		3.16.1 Vertices	49
		3.16.2 Type for graphs	49
		3.16.3 Constructor functions for graphs	49
		3.16.4 Basic access functions	50
		3.16.5 Iterator	50
		3.16.6 Extraction of subgraphs	50
		3.16.7 Printing	50
		3.16.8 Cycles, connected components and such	51
4	Libr	ary regexp	53
	4.1	Module Regular_expr	53
		4.1.1 The regexp datatype and its constructors	53
		4.1.2 Simple regexp operations	54
		4.1.3 Regexp matching by runtime interpretation of regexp	54
	4.2	Implementation of module Regular_expr	54
		4.2.1 extended regexp	55
		4.2.2 Regexp match by run-time interpretation of regexp	55
	4.3	Module Automata	56
	4.4	Implementation of module <i>Automata</i>	56
		4.4.1 The type of automata	57
5	Libr	ary Orderings	61
	5.1	Module Finite_orderings	61
	5.2	Module Orderings_generalities	61
6	Libr	ary dioph	65

7	Libr	brary Integer_solver 6				
	7.1	Module <i>I_solve</i>				
	7.2	Module <i>I_solve_modulo</i>				
	7.3	Module Abstract_constraint				
	7.4	Module Linear_constraints				
		7.4.1 Abstract data type for variables				
		7.4.2 Abstract data type for expressions				
		7.4.3 Abstract data type for formulas				
		7.4.4 Atomic formulas				
		7.4.5 Propositional connectors				
		7.4.6 Iterators for conjunction and disjunction				
		7.4.7 Quantifiers				
		7.4.8 Translation from abstract formulas				
		7.4.11 Renaming				
	7.5	7.4.12 Printing				
	7.5	Module Presburger				
	7.6	Module Finite_domains				
		7.6.1 Finite domain variables				
		7.6.2 The domains expressions				
		7.6.3 Finite domain constraints				
		7.6.4 Solving constraints				
	7.7	Module Non_linear_solving				
	7.8	Module <i>Poly_lexer</i>				
	7.9	Lexer <i>Poly_lexer</i>				
	7.10	Parser Poly_parser				
	7.11	Module <i>Poly_syntax</i>				
_						
8		ary words 8				
	8.1	Module String_rewriting				
		8.1.1 String rewriting systems				
		8.1.2 Efficient normalization				
		8.1.3 Printing SRSs				
	8.2	Module Local_confluence				
	8.3	Module String_signatures				
		8.3.1 The parameterized signature type				
	8.4	Module String_unification				
	8.5	Module <i>User_words</i>				
	8.6	Module Word_lexer				
	8.7	Lexer Word_lexer				
	8.8	Parser Word_parser				
	8.9	Module Word_orderings				
	8.10	Module Word_syntax				
		Module Srs_completion				
		Module <i>Words</i>				
		8.12.1 type for words				
		8.12.2 Printing words				
	8 13	Module Parameterized_signatures				
	0.13	8.13.1 The parameters type				
		8.13.2 The parameterized signature type				
	Q 1/					
	0.14					
	0 15	8.14.1 The user parameterized signature type				
	8.13	Parser Parameterized_signatures_parser				

	0.10	Lexer Parameterized_signatures_lexer	•	91
	8.17	Module Parameterized_signatures_syntax		91
_				<u>.</u>
9		ary matching		93
		Module Lazy_list		
	9.2	Module Lazy_controle		93
10		ary terms		95
		Module Signatures		
		10.1.1 The signature class type		
		10.1.2 Finite symbol sets and maps		
		Module Signature_syntax		
	10.3	Module Substitution		96
	10.4	Module Rewrite_rules		
	10.5	Module <i>Term_lexer</i>		97
	10.6	Module Signature_lexer		97
	10.7	Lexer Term_lexer		97
	10.8	Lexer Signature_lexer		98
	10.9	Module Term_syntax		99
		0Module Hierarchical_signatures		
		1Module <i>User_terms</i>		
		2Module Hierarchical_trs		
		3Module Sorted_signatures		
		4Module Variables		
		5Module Sorts		
	10.13	Simodule Botts	•	103
11	Libra	ary Term Orderings		105
		Module Poly_interp		105
		· ·		
	11.2	Module Term_orderings		107
	11.2 11.3	Module Term_orderings		107 107
	11.2 11.3 11.4	Module Term_orderings		107 107 107
	11.2 11.3 11.4	Module Term_orderings		107 107 107
12	11.2 11.3 11.4 11.5	Module Term_orderings Module Poly_ordering Module Rpo		107 107 107
12	11.2 11.3 11.4 11.5 Libra	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo ary theories	• •	107 107 107 108 109
12	11.2 11.3 11.4 11.5 Libra 12.1	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo ary theories Module Axioms		107 107 107 108 109
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo Tary theories Module Axioms Module Standard_matching		107 107 107 108 109 109
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo Module Acrpo Module Axioms Module Standard_matching Module Unif_index		107 107 107 108 109 109 110
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo Module Acrpo ary theories Module Axioms Module Standard_matching Module Unif_index Module Controle		107 107 107 108 109 110 110
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo ary theories Module Axioms Module Standard_matching Module Unif_index Module Controle Module Theory		107 107 107 108 109 110 110 111
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5 12.6	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo Module Acrpo ary theories Module Axioms Module Standard_matching Module Unif_index Module Controle Module Theory Module Variable_abstraction		107 107 108 109 109 110 111 111 111
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5 12.6 12.7	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo Module Acrpo Tary theories Module Axioms Module Standard_matching Module Unif_index Module Controle Module Theory Module Variable_abstraction Module Problems		107 107 108 109 109 110 111 111 112 113
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo Module Acrpo ary theories Module Axioms Module Standard_matching Module Unif_index Module Controle Module Theory Module Variable_abstraction Module Problems Module Theory_syntax		107 107 108 109 109 110 111 111 112 113 114
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.9	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo Module Axioms Module Standard_matching Module Unif_index Module Controle Module Theory Module Variable_abstraction Module Problems Module Theory_syntax Parser Theory_parser		107 107 108 109 109 110 110 111 111 112 113 114
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.9 12.10	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo Module Axioms Module Standard_matching Module Unif_index Module Controle Module Theory Module Variable_abstraction Module Problems Module Theory_syntax Parser Theory_parser OLexer Theory_lexer		107 107 108 109 109 110 111 111 112 113 114 114 116
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.9 12.10	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo Module Acrpo Tary theories Module Standard_matching Module Unif_index Module Unif_index Module Theory Module Variable_abstraction Module Problems Module Theory_syntax Parser Theory_parser OLexer Theory_lexer 1 Module User_theory		107 107 108 109 110 110 111 111 112 113 114 116 116
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.9 12.10 12.11	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo Module Acrpo Module Axioms Module Standard_matching Module Unif_index Module Controle Module Theory Module Variable_abstraction Module Problems Module Theory_syntax Parser Theory_parser 0Lexer Theory_lexer 1Module User_theory 2Module Oc		107 107 108 109 109 110 111 111 112 113 114 116 116
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.9 12.10 12.11 12.12	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo Module Acrpo Module Axioms Module Standard_matching Module Unif_index Module Controle Module Theory Module Variable_abstraction Module Problems Module Theory_syntax Parser Theory_parser 0Lexer Theory_lexer 1Module User_theory 2Module Oc 3Module Unif_free		107 107 108 109 110 110 111 111 112 113 114 116 116 116
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.9 12.10 12.11 12.12	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo Module Acrpo Module Axioms Module Standard_matching Module Unif_index Module Controle Module Theory Module Variable_abstraction Module Problems Module Theory_syntax Parser Theory_parser 0Lexer Theory_lexer 1Module User_theory 2Module Oc 3Module Unif_free 4Module Unif_commutative		107 107 108 109 109 110 110 111 111 112 113 114 116 116 117 118
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.9 12.10 12.11 12.12 12.13	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo Module Acrpo Module Axioms Module Standard_matching Module Unif_index Module Controle Module Theory Module Variable_abstraction Module Problems Module Theory_syntax Parser Theory_parser 0Lexer Theory_lexer 1Module User_theory 2Module Usif_free 4Module Unif_commutative 5Module Unif_commutative 5Module Unif_to_arith		107 107 108 109 109 110 111 111 112 113 114 116 116 116 117 118
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.10 12.11 12.12 12.13 12.14 12.15 12.15	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo ary theories Module Axioms Module Standard_matching Module Unif_index Module Controle Module Theory Module Variable_abstraction Module Problems Module Theory_syntax Parser Theory_syntax Parser Theory_lexer 1Module User_theory 2Module User_theory 2Module Oc 3Module Unif_free 4Module Unif_commutative 5Module Unif_commutative 5Module Unif_to_arith 6Module Hullot_bin_trees		107 107 108 109 110 110 111 112 113 114 116 116 116 117 118 118
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.10 12.11 12.12 12.13 12.14 12.15 12.15	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo Module Acrpo Module Axioms Module Standard_matching Module Unif_index Module Controle Module Theory Module Variable_abstraction Module Problems Module Theory_syntax Parser Theory_parser 0Lexer Theory_lexer 1Module User_theory 2Module Usif_free 4Module Unif_commutative 5Module Unif_commutative 5Module Unif_to_arith		107 107 108 109 110 110 111 112 113 114 116 116 117 118 118 120 120
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.9 12.10 12.11 12.12 12.13 12.14 12.15 12.16	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo ary theories Module Axioms Module Standard_matching Module Unif_index Module Controle Module Theory Module Variable_abstraction Module Problems Module Theory_syntax Parser Theory_syntax Parser Theory_lexer 1Module User_theory 2Module User_theory 2Module Oc 3Module Unif_free 4Module Unif_commutative 5Module Unif_commutative 5Module Unif_to_arith 6Module Hullot_bin_trees		107 107 108 109 110 110 111 111 112 113 114 116 116 117 118 120 120
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.10 12.11 12.12 12.13 12.14 12.15 12.16 12.17	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo ary theories Module Axioms Module Standard_matching Module Unif_index Module Controle Module Theory Module Variable_abstraction Module Problems Module Theory_syntax Parser Theory_syntax Parser Theory_lexer 1Module User_theory 2Module User_theory 2Module Usif_free 4Module Unif_commutative 5Module Unif_commutative 5Module Unif_to_arith 6Module Hullot_bin_trees 7Module Arith_to_unif		107 107 108 109 109 110 111 111 112 113 114 116 116 117 118 120 120 122
12	11.2 11.3 11.4 11.5 Libra 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.10 12.11 12.12 12.13 12.14 12.15 12.16 12.17 12.18 12.19 12.19	Module Term_orderings Module Poly_ordering Module Rpo Module Acrpo Tary theories Module Axioms Module Standard_matching Module Unif_index Module Unif_index Module Controle Module Variable_abstraction Module Problems Module Theory_syntax Parser Theory_parser Ol_exer Theory_lexer 1Module User_theory 2Module User_theory 2Module Unif_free 4Module Unif_commutative 5Module Unif_to_arith 6Module Hullot_bin_trees 7Module Arith_to_unif 8Module Unif_ac		107 107 108 109 109 110 111 111 112 113 114 116 116 117 118 120 120 122 122 123

	12.22Module <i>Unif_aci</i>	123
	12.23Module <i>Unif_ag_acun</i>	123
	12.24Module <i>Unif_bool</i>	124
	12.25Module <i>Mark</i>	124
	12.26Module <i>Cycle</i>	
	12.27Module <i>E_res</i>	
	12.28Module <i>Ege</i>	
	12.29Module Unification	
	12.30Module Ac_unification	
	12.50 Wiodule Ac_unmeation	12/
13	Library rewriting	129
	13.1 Module <i>Full_dnet</i>	
	13.2 Module Standard_innermost	
	13.3 Module Innermost	
	13.5 Module Inhermost	131
14	Library completion	133
17	14.1 Module Standard_critical_pairs	
	14.2 Module Ac_critical_pairs	
	<u>.</u>	
	14.3 Module Abstract_rewriting	
	14.4 Module Confluence	
	14.5 Module <i>Kb</i>	136
15	T. Sharaman and a control of the con	139
13	S Library eq_proof 15.1 Module Proof	
	15.1 Module <i>Proof</i>	139
16	Library coq_interface	141
10	16.1 Module <i>Trace</i>	
	16.2 Module Traced_rewriting	
	· · · · · · · · · · · · · · · · · · ·	
	16.3 Module <i>Coq_syntax</i>	142
17	Library Termination	143
1/	17.1 Module Basic_criterion	_
	17.1 Module <i>Basic_criterion</i>	
	17.2.1 Simple dependency pairs criterion	
	17.3 Module Generic_polynomials	
	17.4 Module Genpoly_lexer	
	17.5 Lexer Genpoly_lexer	
	17.6 Parser Genpoly_parser	
	17.7 Module Genpoly_syntax	
	17.8 Module Marked_dp_criteria	
	17.9 Module Old_basic_criterion	150
	17.10Lexer <i>Poly_lexer</i>	150
	17.11Module Termination_constraints	150
	17.11.1 Checking constraints	151
	17.12Module Minimal_split	151
	17.13Module Relative_dp	
	17.14Module Termination_expert	
	17.14.1 Global configuration	
	17.14.2 Computing dependency pairs	
	17.14.2 Computing dependency pairs	
	17.14.4 The main function	
	17.14.4 The main function	
	17.16Module <i>Specif</i>	
	17.17Module <i>Types</i>	158

18	Library toplevel	159
	18.1 Module Abstract_syntax	159
	18.2 Parser Toplevel_parser	159
	18.3 Module <i>Eval</i>	
	18.4 Module <i>Typing</i>	162
	18.5 Module <i>Predef</i>	164
	18.6 Module <i>Values</i>	164
	18.7 Module <i>Toplevel_lexer</i>	166
	18.8 Module Version	166
	18.9 Lexer Toplevel_lexer	166

Part I User manual

Chapter 1

Tutorial

This chapter is a small tutorial, to present the basic features in a progressive way.

1.1 The toplevel language

1.1.1 Expressions

The toplevel language is a simple language to allow the user to type-in expressions terminated by a semi-colon, and let CiME evaluate them. Simple expressions are made with integers, booleans and strings, and operations on them:

```
CiME> 12;
-: int = 12
CiME> "hello world";
- : string = "hello world"
CiME> true;
- : bool = true
CiME> 1+2;
-: int = 3
CiME> 3*4-1;
-: int = 11
CiME> -4*(3-2)+1;
-: int = -3
CiME> 4*4 < 5 or 6 <> 3+1;
- : bool = true
CiME> true and not false or true;
- : bool = true
```

Notice that for each expression, CiME first compute its type. In case of typing error, an error message is displayed and no evaluation is performed:

```
CiME> 1+true;
Typing error: bad type argument in application
```

1.1.2 Definitions with let

It is possible to give a name to the result of an evaluation, for further use in next expressions:

```
CiME> let x = 4*5;
x : int = 20
CiME> x*x-1;
- : int = 399
```

```
CiME> x > 10;
- : bool = true
```

An attempt to use an undefined name results in a typing error:

```
CiME> y+1;
Typing error: undefined identifier y
```

Notice that it is allowed to redefine a name:

```
CiME> let x=1;
x : int = 1
CiME> x+3;
- : int = 4
CiME> let x=2;
x : int = 2
CiME> x+3;
- : int = 5
```

notice that the second let does not behave like an assignment like in traditional imperative programming languages, but it is a new definition hiding to previous one, like in functional languages. (More precisely, it follows the static binding semantics, see Section 2.1.2.)

1.1.3 Definitions of functions

The core language allows the definition of functions, with a let fun construct. Application is denoted by a juxtaposition of the function and its arguments, as in LISP or other functional languages based on lambda-calculus.

```
CiME> let fun succ x = x+1;
succ : int -> int = <fun>
CiME> (succ 4);
- : int = 5
CiME> (succ (succ 6));
- : int = 8
```

Notice that putting parentheses around an application is recommended, by not mandatory:

```
CiME> succ 4;
- : int = 5
CiME> succ (succ 6);
- : int = 8
```

Functions with several arguments are defined analogously:

```
CiME> let fun norm x y = x*x+y*y;
norm : int -> int -> int = <fun>
CiME> norm 3 4;
- : int = 25
```

Functions may be defined recursively:

```
CiME> let fun fact n = if n <= 1 then 1 else n * (fact (n-1));
fact : int -> int = <fun>
CiME> fact 7;
- : int = 5040
CiME> fact 100;
- : int =
9332621544394415268169923885626670049071596826438162146859296389521759
9993229915608941463976156518286253697920827223758251185210916864000000
0000000000000000000000
```

Notice on that last example that integers may have arbitrary size.

1.1.4 Higher-order functions

The core language is fully higher-order polymorphic functional, so that you can use partial application, functions as arguments, polymorphic arguments:

```
CiME> let fun f x y = x*x+y*y;
f: int -> int -> int = <fun>
CiME> let g = f 2;
g: int -> int = <fun>
CiME> g 5;
-: int = 29
CiME> let fun eval_at_2 f = f 2;
eval_at_2: (int -> 'a) -> 'a = <fun>
CiME> eval_at_2 g;
-: int = 8
CiME> eval_at_2 (fun x -> x+1);
-: int = 3
CiME> let fun compose f g x = f (g x);
compose : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
```

1.2 More toplevel user interaction

1.2.1 Loading expressions from files

For easy use, it is possible to write expressions in a file and tell CiME to evaluate them as if they were typed directly. It is done by the directive #load followed by the file name.

Suppose the file test.cim contains

```
let fun fib n = if n <= 1 then 1 else (fib (n-1)) + (fib (n-2));
fib 3;
fib 4;
fib 15;
fib 25;
then one may do

CiME> #load "test.cim";
fib : int -> int = <fun>
- : int = 3
- : int = 5
```

or alternatively, you may run cime directly with file names as arguments: the toplevel will first load and execute the files, and then let the user type more expressions:

```
unix prompt> cime2 test.cim
fib : int -> int = <fun>
- : int = 3
- : int = 5
- : int = 987
- : int = 121393
CiME>
```

-: int = 987 -: int = 121393

1.2.2 Interrupting evaluation

If you try to performed a evaluation that takes a large amount of time, and then want to stop the evaluation, you may type Control-C and you return back to the toplevel loop:

```
CiME> fib 100;
^CCommand aborted.
CiME>
```

1.2.3 Time spent in computation

You may display the CPU time spent in evaluation of each expression by typing #time on;:

```
CiME> #time on;
time is now on
CiME> fib 15;
Execution time: 0.2 sec.
- : int = 987
```

To turn off this feature, type #time off;. #time; alone toggles between on and off.

1.2.4 Quitting CiME

You may quit CiME by typing #quit;, or simply hitting Control-D.

1.3 Playing with the Diophantine constraints solver

CiME provides a general Diophantine constraints solver. Such constraints are equalities or inequalities between polynomials over an arbitrary number of variables, and a CiME predefined function may be used to try to solve such constraints for variables belonging to an interval $[0, \dots, N]$ for a given N.

The following example shows how to solve the system

$$xy = 6$$
$$x + y = 5$$

you must first define your constraints, and then call the solver, for a given bound N, say 100 here:

Notice first the constraints displayed is not the same as the input, since CiME performed some internal transformations.

One may also give inequalities:

```
CiME> let constr = dioph_constraint 

"x^2 + y^2 = z^2; x > 0; y > 0; z > 0"; constr : dioph_constraint = { -1*z^2 + 1*y^2 + 1*x^2 = 0; 1*x + -1 >= 0;
```

```
1*y + -1 >= 0;
1*z + -1 >= 0;
\{ \text{(4 inequalitie(s) over 3 variable(s).)} \}
\text{CiME> dioph\_solve 100 constr;}
\text{Solution:}
x = 4
y = 3
z = 5
-: \text{unit = ()}
```

Notice that only the first solution found is displayed, even if there are others.

Here is a quite larger example, the famous problem SEND+MORE=MONEY. Notice that no negation is allowed, so in this problem the fact that two variables x and y must be distinct is encoded as $(x-y)^2 > 0$. This example is found in the bunch of examples available:

```
CiME> #load "examples/diophantine_solver/send+more=money.cim2";
c : dioph_constraint = \{-10*r1 + -1*y + 1*e + 1*d = 0;
                         -10*r2 + 1*r + 1*n + 1*r1 + -1*e = 0;
                         -10*r3 + 1*o + 1*r2 + -1*n + 1*e = 0;
                         -9*m + 1*s + 1*r3 + -1*o = 0;
                         1*s + -1 >= 0;
                         1*m + -1 >= 0;
                         1*s^2 + -2*e*s + 1*e^2 + -1 >= 0;
                         1*s^2 + -2*n*s + 1*n^2 + -1 >= 0;
                         1*s^2 + -2*d*s + 1*d^2 + -1 >= 0;
                         1*m^2 + -2*s*m + 1*s^2 + -1 >= 0;
                         1*s^2 + -2*o*s + 1*o^2 + -1 >= 0;
                         1*s^2 + -2*r*s + 1*r^2 + -1 >= 0;
                         1*s^2 + -2*y*s + 1*y^2 + -1 >= 0;
                         1*n^2 + -2*e*n + 1*e^2 + -1 >= 0;
                         1*e^2 + -2*d*e + 1*d^2 + -1 >= 0;
                         1*m^2 + -2*e*m + 1*e^2 + -1 >= 0;
                         1*o^2 + -2*e*o + 1*e^2 + -1 >= 0;
                         1*r^2 + -2*e*r + 1*e^2 + -1 >= 0;
                         1*v^2 + -2*e*v + 1*e^2 + -1 >= 0;
                         1*n^2 + -2*d*n + 1*d^2 + -1 >= 0;
                         1*m^2 + -2*n*m + 1*n^2 + -1 >= 0;
                         1*o^2 + -2*n*o + 1*n^2 + -1 >= 0;
                         1*r^2 + -2*n*r + 1*n^2 + -1 >= 0;
                         1*n^2 + -2*y*n + 1*y^2 + -1 >= 0;
                         1*m^2 + -2*d*m + 1*d^2 + -1 >= 0;
                         1*o^2 + -2*d*o + 1*d^2 + -1 >= 0;
                         1*r^2 + -2*d*r + 1*d^2 + -1 >= 0;
                         1*y^2 + -2*d*y + 1*d^2 + -1 >= 0;
                         1*m^2 + -2*o*m + 1*o^2 + -1 >= 0;
                         1*m^2 + -2*r*m + 1*r^2 + -1 >= 0;
                         1*m^2 + -2*y*m + 1*y^2 + -1 >= 0;
                         1*o^2 + -2*r*o + 1*r^2 + -1 >= 0;
                         1*o^2 + -2*y*o + 1*y^2 + -1 >= 0;
                         1*r^2 + -2*y*r + 1*y^2 + -1 >= 0;
(34 inequalitie(s) over 11 variable(s).)
```

```
time is now on
Solution :
d = 7
e = 5
y = 2
r1 = 1
n = 6
r = 8
r2 = 1
o = 0
r3 = 0
s = 9
m = 1
Execution time: 0.49 sec.
- : unit = ()
```

1.4 String rewriting

TODO

1.5 Parameterized string rewriting system

TODO

1.6 Definition of first-order signatures

```
CiME> let F_peano = signature "
0 : constant;
s : unary;
+,* : infix binary;
";
F_peano : signature = <signature>
CiME> let X = vars "x y z";
X : variable_set = <variable set>
```

1.7 Definition of terms

Once you have defined a signature and a set of variables, you may define terms, using the predefined function term which takes as arguments a signature, a set of variables, and a string denoting a term. Example:

```
CiME> let t = term F_peano X "s(s(s(0)))*(s(0)+s(s(0)))";
t : (F_peano,X) term = s(s(s(0)))*(s(0) + s(s(0)))
CiME> let u = term F_peano X "s(x)+0+x*s(0)";
u : (F_peano,X) term = ((s(x) + 0) + x) * s(0)
```

Beware! As you may see on this last example, CiME does not make any assumptions on the priorities between binary operators of your defined signature. By default, it always associate to the left. To enforce a priority, you have to use parentheses, like below.

```
CiME> let u = \text{term } F_peano \ X \ "s(x)+0+(x*s(0))";

u : (F_peano, X) \ \text{term} = (s(x) + 0) + (x * s(0))
```

1.8. REWRITING 17

1.8 Rewriting

One may define a rewrite system with the ternary function TRS, which takes a signature as first argument, and a set of variables as second argument, a finally a string that contains a natural definition of rules:

Given a term and a TRS, you can normalize the term with the TRS, by use of the predefined function normalize:

1.9 Checking termination of rewrite systems

```
CiME> termination R_peano;
Entering the termination expert. Verbose level = 0
[0] = 1;
[s](X0) = 1*X0 + 1;
[+](X0,X1) = 2*X1 + 1*X0;
[*](X0,X1) = 2*X0*X1 + 2*X1 + 1*X0;
-: unit = ()
```

1.9.1 Search for incremental and modular termination proof

Termination proofs may be found in a much more efficient way using hierarchies of rewriting modules (HTRS) instead of TRS.

Hierarchical TRS

A hierarchical term rewriting system R is defined as follow:

```
let R = HTRS {R0;...;Ri;...;Rk} Sig X " list of rules ";
```

where Ri are HTRS, Sig consists of some new symbols (possibly AC) and X is a set of variables. The given *list of rules* define some symbols in Sig such that the union of rules of R and all Ri is a TRS over the union of all Sig.

All symbols defined in *list of rules* must be in signature Sig.

Semantics of such a declaration is that R is an hierarchical extension of Ri with new symbols of Sig and rules from *list of rules*.

```
An extension of the empty HTRS is denoted:
```

```
let R = HTRS {} Sig X " list of rules";
```

Proving termination with HTRS

Termination proofs of HTRS are performed incrementally/modularly on modules constituting the extension. In particular C_E -termination of a Ri is proven only once, and never again in a termination proof of any extension of Ri.

The function for searching an incremental/modular termination proof of HTRS R is:

```
h_termination : ('A,'B) HTRS -> unit
```

it tries to find a termination proof of the rewrite system given as argument by providing for each sub-hierarchy (w.r.t. topological sorting) a suitable ordering.

The result of that search is displayed on standard output.

Note that, by definition, h_termination always makes use of dependency pairs of modules, even if termcrit "standard"; was selected.

Optimizations such as marks and graph may be used as they affect both CiME incremental/modular and classical termination experts.

In order to make the termination expert more efficient, it may prove useful to restrict to nonexpensive criteria on most of the proof. Such tuning may be done with help of function:

```
h_termination_with : ((string,int) set) -> ('A,'B) HTRS -> unit
```

which takes as first argument a list of pairs *criterion*, *bound*. They denote polynomials (as with polyinterpkind) and bounds (as with termpolybound) to be tried successively on a module when a search fails.

Finally, it is possible to check if a given Ri has been proven terminating with:

```
check_ce : ('A,'B) HTRS -> unit
```

which prints the result on standard output.

Splitting HTRS

A HTRS may be considered as a hierarchy of minimal modules by use of:

```
termcrit "minimal";
```

HTRS are then split up in minimal modules for termination proofs.

This is particularly useful when dealing with a huge bunch of rules in a single HTRS (see examples below).

1.9.2 Examples of modular expert sessions

Step by step

```
(* Binary arithmetics:
   Addition
   Multiplication
   Lists
*)

CiME> let X = vars "x y z l b";

(* Nonnegative rational integer in binary notation *)

CiME> let Fbin = signature "
   # : constant;
   0,1 : postfix unary;
   ".
```

```
CiME> let Mbin = HTRS {} Fbin X "
  (#) 0 -> #;
CiME> h_termination Mbin;
Entering the termination expert for modules. Verbose level = 1
checking each of the 0 strongly connected components :
Termination proof found.
Execution time: 0.000000 sec
-: unit = ()
CiME>
(* Addition *)
CiME> let Fadd = signature "
   + : infix binary
CiME> let Madd = HTRS {Mbin} Fadd X "
     # + x -> x;
     x + \# -> x;
     (x) 0 + (y) 0 \rightarrow (x+y) 0;
     (x) 0 + (y) 1 \rightarrow (x+y) 1;
     (x) 1 + (y) 0 \rightarrow (x+y) 1;
     (x)1 + (y)1 \rightarrow (x+y+(\#)1)0;
";
CiME> h_termination Madd;
Entering the termination expert for modules. Verbose level = 0
CE-termination of
\{ (#)0 \rightarrow # \} (1 rules)
is already proven.
checking each of the 1 strongly connected components :
checking component 1 (disjunction of 1 constraints)
[#] = 1;
[+](X0,X1) = X1*X0;
[0](X0) = X0 + 1;
[1](X0) = 2*X0 + 1;
['+'](X0,X1) = X1*X0;
Termination proof found.
Execution time: 0.170000 sec
-: unit = ()
CiME>
(* Multiplication *)
CiME> let Fmult = signature "
 * : infix binary
```

```
CiME> let Mmult = HTRS {Madd} Fmult X "
     # * x -> #;
     x * # -> #;
     (x) 0 * y -> (x*y) 0;
     (x) 1 * y -> (x*y) 0 + y;
CiME> h_termination Mmult;
Entering the termination expert for modules. Verbose level = 0
CE-termination of
\{ (\#) 0 \rightarrow \#,
  # + V_0 -> V_0,
  V_0 + \# -> V_0,
  (V_0) + (V_1) - (V_0 + V_1) 
  (V_0) + (V_1) - (V_0 + V_1)
  (V_0)1 + (V_1)0 \rightarrow (V_0 + V_1)1,
  (V_0)1 + (V_1)1 \rightarrow ((V_0 + V_1) + (\#)1)0 } (7 rules)
is already proven.
checking each of the 1 strongly connected components :
checking component 1 (disjunction of 1 constraints)
[#] = 0;
[+](X0,X1) = X1 + X0;
[*](X0,X1) = X1*X0 + X0;
[0](X0) = X0 + 1;
[1](X0) = X0 + 1;
['*'](X0,X1) = X0;
Termination proof found.
Execution time: 0.080000 sec
-: unit = ()
CiME>
(* Lists *)
CiME> let Flist = signature "
  nil : constant ;
  cons : binary";
CiME> let Mlist = HTRS {} Flist X "";
(* Sum in a liste *)
CiME> let Fsum = signature " sum : unary ";
CiME> let Msum = HTRS {Mlist; Madd}
                 Fsum X "
  sum(nil) \rightarrow (\#)0;
  sum(cons(x,1)) \rightarrow x + sum(1);
```

Suitable interpretation are often simpler when using the incremental/modular expert. It is often judicious to search firstly for linear interpretations.

CiME> h_termination Mbag;

```
CiME> polyinterpkind "linear";
Now searching only linear polynomial interpretations
Execution time: 0.000000 sec
-: unit = ()
CiME> h_termination Msum;
Entering the termination expert for modules. Verbose level = 0
checking each of the O strongly connected components :
Termination proof found.
CE-termination of
\{ (\#) 0 \rightarrow \#,
  # + V_0 -> V_0,
  V_0 + \# -> V_0,
  (V_0) + (V_1) - (V_0 + V_1)
  (V_0) + (V_1) - (V_0 + V_1) 
  (V_0)1 + (V_1)0 \rightarrow (V_0 + V_1)1,
  (V_0)1 + (V_1)1 \rightarrow ((V_0 + V_1) + (\#)1)0 } (7 rules)
is already proven.
checking each of the 1 strongly connected components :
checking component 1 (disjunction of 1 constraints)
[#] = 0;
[+](X0,X1) = X1 + X0;
[nil] = 0;
[sum](X0) = X0;
[0](X0) = 0;
[cons](X0,X1) = X1 + X0 + 1;
[1](X0) = 0;
['sum'](X0) = X0;
Termination proof found.
Execution time: 0.050000 sec
-: unit = ()
CiME>
   The incremental/modular expert of CiME is able to prove termination even if AC symbols are involved.
(* Addition, Multiplication
   and Multisets
(* Multisets with an AC union operator *)
CiME> let Fbag = signature "
  empty : constant ;
  singl : unary ;
  U : AC";
CiME> let Mbag = HTRS {} Fbag X "
  empty U b -> b;
```

```
Entering the termination expert for modules. Verbose level = 0
checking each of the O strongly connected components :
Termination proof found.
Execution time: 0.000000 sec
-: unit = ()
CiME>
(* Sum in a multiset *)
CiME> let Msumbag = HTRS {Mbag ; Madd}
                   Fsum X
  sum(empty) \rightarrow (#)0;
  sum(singl(x)) \rightarrow x;
 sum(x U y) \rightarrow sum(x) + sum(y);
CiME> h_termination Msumbag;
Entering the termination expert for modules. Verbose level = 0
CE-termination of
{ empty V_4 \rightarrow V_4 } (1 rules)
is already proven.
CE-termination of
\{ (\#) 0 \rightarrow \#,
  # + V_0 -> V_0,
  V_0 + \# -> V_0
  (V_0) + (V_1) - (V_0 + V_1),
  (V_0) + (V_1) - (V_0 + V_1) 
  (V_0)1 + (V_1)0 \rightarrow (V_0 + V_1)1,
  (V_0)1 + (V_1)1 \rightarrow ((V_0 + V_1) + (\#)1)0 } (7 rules)
is already proven.
checking each of the 1 strongly connected components :
checking component 1 (disjunction of 1 constraints)
[#] = 0;
[+](X0,X1) = X1 + X0;
[sum](X0) = X0;
[empty] = 0;
[0](X0) = 0;
[singl](X0) = X0;
[1](X0) = 0;
[U](X0,X1) = X1 + X0 + 1;
['sum'](X0) = X0;
Termination proof found.
Execution time: 0.050000 sec
-: unit = ()
CiME>
```

Proof on a hierarchy

A termination proof may be search for when all the relevant hierarchy has been defined.

```
CiME> let X = vars "x y z l b";
CiME> let Fbin = signature "
   # : constant;
    0,1 : postfix unary ;
CiME> let Mbin = HTRS {} Fbin X "
  (#) 0 -> #;
CiME> let Fadd = signature "
  + : infix binary";
CiME> let Madd = HTRS {Mbin} Fadd X "
     # + x -> x;
     x + \# -> x;
     (x) 0 + (y) 0 \rightarrow (x+y) 0;
     (x) 0 + (y) 1 \rightarrow (x+y) 1;
     (x) 1 + (y) 0 \rightarrow (x+y) 1;
     (x)1 + (y)1 \rightarrow (x+y+(\#)1)0;
۳;
CiME> let Fmult = signature "
  * : infix binary";
CiME> let Mmult = HTRS {Madd} Fmult X "
     # * x -> #;
     x * # -> #;
     (x) 0 * y -> (x*y) 0;
     (x) 1 * y -> (x*y) 0 + y;
CiME> let Flist = signature "
 nil : constant ;
  cons : binary";
CiME> let Mlist = HTRS {} Flist X "";
CiME> let Fsum = signature " sum : unary ";
CiME> let Msum = HTRS {Mlist ; Madd} Fsum X
  sum(nil) \rightarrow (\#)0;
 sum(cons(x,1)) \rightarrow x + sum(1);
CiME> let Fbag = signature "
  empty : constant ;
  singl : unary ;
  U : AC";
```

```
CiME> let Mbag = HTRS {} Fbag X "
  empty U b -> b;
CiME> let Msumbag = HTRS {Mbag ; Madd}
                    Fsum X "
  sum(empty) \rightarrow (#)0;
 sum(singl(x)) \rightarrow x;
 sum(x U y) \rightarrow sum(x) + sum(y);
CiME> h_termination Msumbag;
Entering the termination expert for modules. Verbose level = 0
checking each of the O strongly connected components :
Termination proof found.
checking each of the O strongly connected components :
Termination proof found.
checking each of the 1 strongly connected components :
checking component 1 (disjunction of 1 constraints)
[#] = 1;
[+](X0, X1) = X1*X0;
[0](X0) = X0 + 1;
[1](X0) = 2*X0 + 1;
['+'](X0,X1) = X1*X0;
Termination proof found.
checking each of the 1 strongly connected components :
checking component 1 (disjunction of 1 constraints)
[#] = 1;
[+](X0,X1) = X1*X0;
[sum](X0) = X0;
[empty] = 1;
[0](X0) = X0;
[singl](X0) = X0;
[1](X0) = 0;
[U](X0,X1) = 2*X1*X0 + 2*X1 + 2*X0 + 1;
['sum'](X0) = X0;
Termination proof found.
Execution time: 0.240000 sec
-: unit = ()
CiME>
```

Note that nothing is done about the irrelevant multiplication.

A termination proof search regarding the hierarchy headed by Mmult does not need to be performed using simple polynomials everywhere since linear polynomials suffice for Madd. In order to avoid overheads one may use h_termination_with. For instance, termination of binary arithmetics (Mbin, Madd and Mmult) may be proven with:

```
CiME> h_termination_with {("linear",1); ("simple",1)} Mmult;
```

Splitting up a hierarchy in minimal modules

Finally one can provide a single set of rules and ask CiME to perform a minimal splitting up of it. For instance for sum in multiset:

```
CiME> let X = vars "x y z l b";
CiME> let F = signature "
    # : constant;
    0,1 : postfix unary ;
    + : infix binary ;
    empty : constant ;
    singl : unary ;
    U : AC ;
    sum : unary";
CiME> let R = HTRS {} F X "
  (#) 0 -> #;
  # + x -> x;
  x + \# -> x;
  (x) 0 + (y) 0 -> (x+y) 0;
  (x) 0 + (y) 1 \rightarrow (x+y) 1;
  (x) 1 + (y) 0 \rightarrow (x+y) 1;
  (x)1 + (y)1 \rightarrow (x+y+(\#)1)0;
  empty U b -> b;
  sum(empty) -> (#)0;
  sum(singl(x)) \rightarrow x;
 sum(x U y) \rightarrow sum(x) + sum(y);
CiME> termcrit "minimal";
Termination now uses minimal decomposition
-: unit = ()
CiME> polyinterpkind "linear";
Now searching only linear polynomial interpretations
-: unit = ()
CiME> h_termination R;
Entering the termination expert for modules. Verbose level = 0
Checking module:
{ }
checking each of the O strongly connected components :
Termination proof found.
Checking module:
{ empty V_4 \rightarrow V_4 } (1 rules)
checking each of the O strongly connected components :
Termination proof found.
Checking module:
```

```
{}
checking each of the O strongly connected components :
Termination proof found.
Checking module:
{ }
checking each of the 0 strongly connected components :
Termination proof found.
Checking module:
checking each of the O strongly connected components :
Termination proof found.
Checking module:
\{ (#)0 \rightarrow # \} (1 rules)
checking each of the 0 strongly connected components :
Termination proof found.
Checking module:
\{ # + V_0 \rightarrow V_0,
 V_0 + # -> V_0,
  (V_0) + (V_1) - (V_0 + V_1),
  (V_0) + (V_1) - (V_0 + V_1),
  (V_0)1 + (V_1)0 \rightarrow (V_0 + V_1)1,
  (V_0)1 + (V_1)1 \rightarrow ((V_0 + V_1) + (\#)1)0 } (6 rules)
checking each of the 1 strongly connected components :
checking component 1 (disjunction of 1 constraints)
[\#] = 0;
[0](X0) = X0 + 1;
[1](X0) = X0 + 1;
[+](X0,X1) = X1 + X0;
['+'](X0,X1) = X1 + X0;
Termination proof found.
Checking module:
{ sum(V_0 U V_1) \rightarrow sum(V_0) + sum(V_1),
  sum(singl(V_0)) \rightarrow V_0,
  sum(empty) \rightarrow (\#)0 } (3 rules)
checking each of the 1 strongly connected components :
checking component 1 (disjunction of 1 constraints)
[\#] = 0;
[0](X0) = 0;
[1](X0) = 0;
[+](X0,X1) = X1 + X0;
[empty] = 0;
[singl](X0) = X0;
```

```
[U](X0,X1) = X1 + X0 + 1;

[sum](X0) = X0;

['sum'](X0) = X0;
```

Termination proof found.

```
- : unit = ()
```

Chapter 2

Reference manual

2.1 CiME toplevel core language

This section describes the toplevel core language. This language is a simple functional language that allows one to evaluate expressions, and possibly naming the results by a let construct.

2.1.1 Formal syntax

The whole language is described by the following grammar:

The identifiers are made with sequences of letters, digits, underline and quote characters; and should start by a letter. Beware that the interpreter makes a difference between uppercase and lowercase letters.

The priority between operators are, from lowest to highest: or, and, not, comparisons, + and -, *.

Comments may be put between (* and *).

Apart from these very general constructs, all the toplevel language is made of predefined functions. These are described in the next section.

2.1.2 A note on static binding

The let construct is not like assignment, there is no possibility to change the value of a variable. Several lets for the same name define several variable, following the static binding semantics, which is illustrated by the following example:

```
CiME> let x=1;
x : int = 1
CiME> let fun f y = x*y+1;
f : int -> int = <fun>
CiME> f 3;
- : int = 4
CiME> let x=2;
x : int = 2
CiME> f 3;
- : int = 4
```

The second definition of x hides the first one, but in the definition of function f, x still refers to the first definition of x.

2.1.3 Directives

Some directives controls the interpreter. These directives are prefixed by #, and optionally take an argument. The available directives are the following.

- #help. Prints the list of predefined functions available.
- #help *name*. Prints some help on the predefined function *name*.
- #load *file*. Read commands from *file* as if they were typed directly. Notice that such a file may also contain directives, including the #load directive.
- #time [on or off]. When on, the time spent in executing a command is shown for each command. When no argument, toggles between on and off.
- #quit. Exits the interpreter.
- #verbose [n]. Sets verbose level to n, default 1. The verbose level controls for some functions how much verbose are they working. See details on each such function.
- #quiet. Equivalent to #verbose 0.
- #mem or #memory. Display amount of memory allocated so far.
- #extern command. Execute the command in a sub-shell.

2.2 CiME toplevel predefined functions

This section described the predefined functions of the toplevel language. The list of each predefined function available may be obtain by directive #help, and a simple on-line help is obtained on a given function *name* by #help "name".

2.2.1 Diophantine constraints

```
dioph_constraint : string -> dioph_constraint
```

(dioph_constraint s) returns a Diophantine constraint as read syntactically from the string s. The syntax of constraint is

```
Constraints ::= (Constraint;)*

Constraint ::= Poly Op Poly

Op ::= = | > | < | > = |

Poly ::= Int | Var | Poly + Poly | Poly
```

```
dioph_solve : int -> dioph_constraint -> unit
```

(dioph_solve N c) solves the constraints c for variables between 0 and N. Only the first solution found is displayed.

2.2.2 Rewriting on words

This subsection describes functions for defining alphabets, words, word rewriting systems (SRSs¹), precedences on alphabet symbols, orderings on words; and functions for normalizing words, performing completion on SRSs.

¹we use the standard abbreviation SRS for *string rewriting system*, but we always use *word* instead of *string* to avoid confusion with character strings

Definition of alphabet, words and SRSs

```
word_signature : string -> word_signature,
word : ('A:word_signature) -> string -> 'A word,
SRS : ('A:word_signature) -> string -> 'A SRS,
```

Normalization

```
word_normalize : 'A SRS -> 'A word -> 'A word.
```

Orderings on words

word_precedence, length_lex, multi_lex, word_compare.

Knuth-Bendix completion

word_completion.

2.2.3 Rewriting on first-order terms

Definition of signatures

the signature function takes a string as argument and return a signature:

```
signature : string -> signature
```

The string must constain syntactic definitions of operator symbols together with their arity, and possibly declare them as commutative or AC. These definitions must follow the following syntax.

```
Decl ::= (ArityDecl;)^*

ArityDecl ::= OpList : Fix \ Arity

Arity ::= AC \mid commutative \mid constant \mid unary \mid binary \mid Int

Fix ::= infix \mid prefix \mid postfix \mid \varepsilon

OpList ::= Ident \ (, Ident)^*

Ident ::= [a-z, A-Z, 0-9, \_, ']^+

Ident ::= [^+, +, , _*, _*, -, /, ?, !, _0, _*, _\#, |, :, _*, _*, -, >]^+

Int ::= [0-9]^+
```

Arities of free symbols have to be given either by an number or by the abbreviations constant for 0, unary for 1 or binary for 2. Commutative or associative-commutative symbols are simply declared by the special keywords commutative and AC.

```
constant unary binary infix prefix postfix commutative AC : -> <> = <= >= <>
```

The optional fix status allows one to enforce the way a symbol will be syntactically used in the following: prefix for an operator placed before its arguments, infix for a binary operator placed between its arguments, and postfix for an operator placed after its arguments. By default, commutative and AC operators are infix and the others are prefix.

For example:

```
a,b,c,0,1 : constant ;
f,g : unary ;
! : postfix unary ;
. : infix binary ;
p : 3;
+ : AC;
eq : commutative ;
```

declares five constants a, b, c, 0 and 1; two unary functions f and g; a unary function! to be used in postfix notation (may be for the factorial function...); a binary function that will be used in infix notation even if not AC; a ternary function p, an AC symbol +; acommutative symbol eq and three variables x, y and z.

Notice that the last semicolon is optional.

Declaring variable names

the vars function takes a string as argument and return a set variable names:

```
vars : string -> variable set
```

Definition of terms

```
term : ('A:signature) -> ('B:variable_set) -> string -> ('A,'B) term
```

term F X s builds the term denoted by the string s, using signature F and variables of X. A error message is issued and no term is built of the string s is not well-formed with respect to the grammar below:

```
Term ::= Var | ConstantSymb | PrefixSymb Term | PrefixSymb ( Term (, Term)* )
Term ::= Term PostfixSymb | ( Term (, Term)* ) PostfixSymb | Term InfixSymb Term
```

Ambiguities are resolved as follows: prefix symbols have priority over postfix symbols, which themselves have priority over infix symbols. Infix symbols associate to the left. For example, for the signature

```
# : prefix unary ;
! : postfix unary ;
+,* : infix binary ;
```

#x! is parsed as (#x)!, #x+y is parsed as (#x)+y, x+y! is parsed as x+(y!), x+y*z is parsed as (x+y)*z. You must use parenthesis to enforce other priorities.

Definition of TRS

```
TRS: ('A:signature) -> ('B:variable_set) -> string -> ('A,'B) TRS
```

Reduction of terms

```
normalize : ('A,'B) TRS \rightarrow ('A,'B) term \rightarrow ('A,'B) term
```

normalize R t returns a normal form of the term t with respect to the rewrite system R. This function call may run forever if R is not terminating on t. The strategy is innermost.

Notice that the type of this function implies that the rewrite system and the term have to be defined on exactly the same signature.

2.2.4 Defining term orderings

```
precedence : ('A:signature) -> string -> 'A precedence
precedence S s builds the precedence on signature S given by string s.

MPO : 'A precedence -> 'A term_ordering

MPO p is the multiset path ordering on terms defined the precedence p.

ACRPO : 'A precedence -> 'A term_ordering

ACRPO p is the AC path ordering [?] on terms defined the precedence p.

term_compare : 'A term_ordering -> ('A,'B) term -> ('A,'B) term -> string

term_compare o t1 t2 compares the terms t1 and t2 w.r.t. the term ordering o.
```

2.2.5 Checking termination

Note: checking termination of SRSs is not yet available.

2.2.6 Finding termination proofs

The main function for searching for a termination proof is

```
termination : ('A,'B) TRS -> unit
```

it tries to find a termination proof of the rewrite system given as argument. The result of that search is displayed on standard output.

There are several way of controlling the search, explained below.

Search for polynomial interpretations

```
polyinterpkind : (string * int) set -> unit
```

Sets the kind of interpretations to search for, in order. Valid values are "linear", "simple", and simple-mixed" [?, ?]. Sets the search bound for polynomial coefficients. Valid values are any positive integer. Default is { ("linear", 2) ; ("simple-mixed", 6) }.

Linear interpretations are of the form

$$P_f(x_1,\ldots,x_n)=a_1x_1+\cdots+a_nx_n+c$$

simple interpretations are of the form

$$P_f(x_1,\ldots,x_n) = \sum_{i_1,\ldots,i_n \in \{0,1\}} a_{i_1,\ldots,i_n} x_1^{i_1} \cdots x_n^{i_n}$$

and simple-mixed are the same as simple except for unary symbols, which are of the form

$$P_f(x) = ax^2 + bx + c$$

Index

```
AC, 31 arity declaration, 31 binary, 31 commutative, 31 constant, 31 crit:minimal, 18 fix status, 31 fun:h_termination, 18 fun:h_termination_with, 18 fun:HTRS, 17 identifiers, 31 infix, 31 postfix, 31 prefix, 31 unary, 31
```

Part II Implementation manual

Chapter 3

Library Compat

This library provides complements OCaml standard library, and various all-purposes data-structures.

3.1 Module Int_utils

```
power, pgcd = greatest common divisor, ppcm = least common multiple. full\_pgcd \ x \ y returns d, k1, k2 such that d=gcd(x,y), x=k1*d and y=k2*d. val power: int \rightarrow int \rightarrow int val pgcd: int \rightarrow int \rightarrow int val full\_pgcd: int \rightarrow int \rightarrow int \times int \times int val ppcm: int \rightarrow int \rightarrow int
```

3.2 Module Arrayutils

```
val index : `a \rightarrow `a \text{ array} \rightarrow \text{ int}
val find : (`a \rightarrow bool) \rightarrow `a \text{ array} \rightarrow \text{ int}
val filter : (`a \rightarrow bool) \rightarrow `a \text{ array} \rightarrow `a \text{ list}
val fold_lefti : (`a \rightarrow \text{ int} \rightarrow `b \rightarrow `a) \rightarrow `a \rightarrow `b \text{ array} \rightarrow `a 
val fold_righti : (\text{int} \rightarrow `a \rightarrow `b \rightarrow `b) \rightarrow `a \text{ array} \rightarrow `b \rightarrow `b 
val filter_indices : (`a \rightarrow bool) \rightarrow `a \text{ array} \rightarrow \text{ int list}
```

3.3 Module Listutils

This module provides some basic functions on lists, mainly functions that used to exist in earlier version of CAML, but do not exist anymore.

Advice: the functions that operate on lists as if they were sets should not be used anymore, the module Set should be used instead.

```
val power: 'a list \rightarrow int \rightarrow 'a list
```

```
val intersect: 'a list \rightarrow 'a list \rightarrow 'a list
val add: 'a \rightarrow 'a list \rightarrow 'a list
val union : 'a list \rightarrow 'a list \rightarrow 'a list
val index : 'a \rightarrow 'a list \rightarrow int
val except : 'a \rightarrow 'a list \rightarrow 'a list
val remove : 'a \rightarrow (a \times b) list \rightarrow (a \times b) list
val flat_map : (a \rightarrow b \text{ list}) \rightarrow a \text{ list} \rightarrow b \text{ list}
val subtract: 'a list \rightarrow 'a list \rightarrow 'a list
val do_list_combine : (a \times b \rightarrow c) \rightarrow a \text{ list} \times b \text{ list} \rightarrow \text{ unit}
val map_filter : ('a \rightarrow 'b) \rightarrow ('a \rightarrow bool) \rightarrow 'a list \rightarrow 'b list
val map_with_exception : exn \rightarrow ('a \rightarrow 'b) \rightarrow 'a list \rightarrow 'b list
val split: string \rightarrow string list
val mapi : (int \rightarrow `a \rightarrow `b) \rightarrow `a list \rightarrow `b list
val flat_mapi : (int \rightarrow `a \rightarrow `b list) \rightarrow `a list \rightarrow `b list
val foldi : (int \rightarrow 'a \rightarrow 'b \rightarrow 'b \rightarrow 'a list \rightarrow 'b \rightarrow '
fold_right_env f env [11;..;1k] is equivalent to
              let env1,e1 = f env 11 in
             let env2, e2 = f env1 12 in
             let envk,ek = f env\{k-1\} lk in
              envk, [e1; ..., ek]
val fold_right_env:
              (a \rightarrow b \rightarrow a \times c) \rightarrow a \rightarrow b \text{ list} \rightarrow a \times c \text{ list}
val assoc\_right: 'a \rightarrow ('b \times 'a) \ list \rightarrow 'b
val flatten_left : ('a list \times 'b) list \rightarrow ('a \times 'b) list
```

3.4 Module Balanced_trees

data type of balanced tree with one info of type 'a

```
type 'a \ t = Empty \mid Node \ of 'a \ t \times 'a \times 'a \ t \times int val height : 'a \ t \rightarrow int
```

Creates a new node with left son l, value x and right son r. l and r must be balanced and | height | - height | - | - 2. Inline expansion of height for better speed.

```
val create : 'a\ t \rightarrow 'a \rightarrow 'a\ t \rightarrow 'a\ t
```

Same as create, but performs one step of rebalancing if necessary. Assumes 1 and r balanced. Inline expansion of create for better speed in the most frequent case where no rebalancing is required.

```
val bal: 'a t \rightarrow 'a \rightarrow 'a t \rightarrow 'a t
```

Same as bal, but repeat rebalancing until the final result is balanced.

```
val join: at \rightarrow at \rightarrow at \rightarrow at
```

Merge two trees 1 and r into one. All elements of 1 must precede the elements of r. Assumes | height 1 - height r | ≤ 2 .

```
val merge: at \rightarrow at \rightarrow at
```

Same as merge, but does not assume anything about 1 and r.

```
val concat : 'a t \rightarrow 'a t \rightarrow 'a t
```

3.5. MODULE MAPORD 39

3.5 Module Mapord

```
Id: mapord.mli, v1.21998/04/2813: 35: 00marcheExp
module type OrderedType =
   sig
      type t
      val compare : t \rightarrow t \rightarrow int
module type OrderedMap =
   sig
      type key
      type 'a t
      val empty: 'a t
      val add: key \rightarrow 'a \rightarrow 'a t \rightarrow 'a t
      val find : key \rightarrow 'a t \rightarrow 'a
      val remove : key \rightarrow 'a t \rightarrow 'a t
      val iter: (key \rightarrow 'a \rightarrow unit) \rightarrow 'a t \rightarrow unit
      val map: (a \rightarrow b) \rightarrow at \rightarrow bt
      val fold: (key \rightarrow 'a \rightarrow 'b \rightarrow 'b) \rightarrow 'a t \rightarrow 'b \rightarrow 'b
      val max : 'a t \rightarrow key \times 'a
      val min : 'a t \rightarrow kev \times 'a
      val elements_increasing_order: 'a t \rightarrow (key \times 'a) list
module\ Make(Ord:\ OrderedType):\ OrderedMap\ with\ type\ key=Ord.t
```

3.6 Module Balanced_trees2

```
type ('key, 'data) node_info =
         key: 'key;
         data: 'data;
         height: int;
         left: ('key, 'data) t;
         right: ('key, 'data) t
and ('key, 'data) t =
      Empty
  | Node of ('key, 'data) node_info
val empty: ('key, 'data) t
val height: ('key, 'data) t \rightarrow int
val create:
   ('key, 'data) t \rightarrow 'key \rightarrow 'data \rightarrow ('key, 'data) t \rightarrow ('key, 'data) t
   ('key, 'data) t \rightarrow 'key \rightarrow 'data \rightarrow ('key, 'data) t \rightarrow ('key, 'data) t
val merge: ('key, 'data) t \rightarrow ('key, 'data) t \rightarrow ('key, 'data) t
val iter : ('key \rightarrow 'data \rightarrow 'a) \rightarrow ('key, 'data) t \rightarrow unit
val map: ('data1 \rightarrow 'data2) \rightarrow ('key, 'data1) t \rightarrow ('key, 'data2) t
val mapi : ('key \rightarrow 'data1 \rightarrow 'data2) \rightarrow ('key, 'data1) t \rightarrow ('key, 'data2) t
val fold : ('key \rightarrow 'data \rightarrow 'a \rightarrow 'a) \rightarrow ('key, 'data) t \rightarrow 'a \rightarrow 'a
```

3.7 Module Ordered_maps

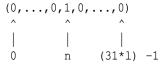
```
module type OrderedMap =
   sig
      type 'a key
      type ('a, 'b) t
      val empty: (a, b) t
      val is_empty: (a, b) t \rightarrow bool
      val add: 'a key \rightarrow 'b \rightarrow ('a, 'b) t \rightarrow ('a, 'b) t
      val find: 'a key \rightarrow ('a, 'b) t \rightarrow 'b
      val remove: 'a key \rightarrow ('a, 'b) t \rightarrow ('a, 'b) t
      val mem : 'a key \rightarrow ('a, 'b) t \rightarrow bool
      val iter : ('a key \rightarrow 'b \rightarrow unit) \rightarrow ('a, 'b) t \rightarrow unit
      val map: (b \rightarrow c) \rightarrow (a,b) t \rightarrow (a,c) t
      val mapi : ('a key \rightarrow 'b \rightarrow 'c) \rightarrow ('a, 'b) t \rightarrow ('a, 'c) t
      val fold: (a \text{ key } \rightarrow b \rightarrow c \rightarrow c) \rightarrow (a,b) \leftarrow c \rightarrow c
      val fold_from_min : ('a key \rightarrow 'b \rightarrow 'c \rightarrow 'c) \rightarrow ('a, 'b) t \rightarrow 'c \rightarrow 'c
      val max : ('a, 'b) t \rightarrow 'a \text{ key } \times 'b
      val min : (a, b) t \rightarrow a \text{ key } \times b
      val elements_increasing_order: ('a, 'b) t \rightarrow ('a key \times 'b) list
      val exists_one : ('a key \rightarrow 'b \rightarrow bool) \rightarrow ('a, 'b) t \rightarrow bool
      val find_one : ('a key \rightarrow 'b \rightarrow bool) \rightarrow ('a, 'b) t \rightarrow 'b
      val find_key : ('a key \rightarrow 'b \rightarrow bool) \rightarrow ('a, 'b) t \rightarrow 'a key
      val size : (a, b) t \rightarrow int
      val union : (a, b) t \rightarrow (a, b) t \rightarrow (a, b) t
module Make(Ord : Ordered_types.OrderedType) :
   OrderedMap with type 'a key = Ord.t
module MakePoly(Ord : Ordered_types.OrderedPolyType) :
   OrderedMap with type 'a key = 'a Ord.t
```

3.8 Module Bit_field

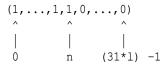
val $bit_not : t \rightarrow t$

```
Obsolete header:
   CiME Project - Démons research team - LRI - Université Paris XI
   Id: bit_field.mli, v1.42001/04/2013: 42: 09marcheExp
   **********************
module type S =
sig
  type t
     all_zero n returns a bit filed of size n filled with 0. all_one n returns a bit filed of size n filled with 1.
  val all_zero : int \rightarrow t
  val all_one : int \rightarrow t
       size of a bit field
  val bit\_length: t \rightarrow int
       boolean operations. bit fields have to be the same length.
  val bit\_and : t \rightarrow t \rightarrow t
  val bit\_or: t \rightarrow t \rightarrow t
```

bit_nth n 1 returns a bit_field encoding a vector of bits of length 31*1 where all the bits are equal to 0, except at position n which is a 1.



bit_nth_first n 1 returs a bit_field encoding a vector of bits of length 31*1 where the first nth bits are equal to 1, the others being equal to 0.



```
val bit\_nth : int \rightarrow int \rightarrow t val bit\_nth\_first : int \rightarrow int \rightarrow t
```

is_zero b returns true if the bit field b encodes the integer 0 in base 2. atmost_one_one b returns true if the bit field b encodes a power of 2 in base 2.

```
val is\_zero: t \rightarrow bool
val atmost\_one\_one: t \rightarrow bool
```

bit_field_to_vect_of_bits 1 bf returns the vector of bits of length 1 encoded by the bit_field bf vect_of_bits_to_bit_field v returns the bit_field encoding the vector of bits v

```
val bit\_field\_to\_vect\_of\_bits: int \to t \to int \ array val vect\_of\_bits\_to\_bit\_field: int \ array \to t pretty-prints a bit field in the current formating box
```

val print_bit_field : int $\rightarrow t \rightarrow unit$

end

```
module Small\_bit\_field: S with type t = int module Large\_bit\_field: S with type t = int array
```

3.9 Module Numbers

the type of numbers (integer or rational)

```
type t
```

```
val hash : t \rightarrow int
```

basic constants and conversion from machine ints

```
val zero : t
val one : t
val two : t
val minus_one : t
val of_int : int \rightarrow t
raises Invalid\_argument "Numbers.to_int" if too big
val to\_int : t \rightarrow int
```

basic operations overs rational numbers

```
val\ is\_zero: t \rightarrow bool
val denominator : t \rightarrow t
\mathsf{val}\; \mathit{add}\; :\; t\; \rightarrow\; t\; \rightarrow\; t
val sub: t \rightarrow t \rightarrow t
val minus : t \rightarrow t
val abs: t \rightarrow t
val mult: t \rightarrow t \rightarrow t
\mathsf{val}\; div\; :\; t\; \rightarrow\; t\; \rightarrow\; t
val power\_int : t \rightarrow int \rightarrow t
comparators
val compare : t \rightarrow t \rightarrow int
val\ eq:t 	o t 	o bool
val\ neq:t 	o t 	o bool
\mathsf{val} \ ge \ : \ t \ \to \ t \ \to \ bool
\mathsf{val}\; \mathsf{gt}\; :\; t\; \to\; t\; \to\; bool
\mathsf{val}\ le\ :\ t\ \to\ t\ \to\ bool
\mathsf{val}\ lt\ :\ t\ \to\ t\ \to\ bool
\mathsf{val}\; max\; :\; t\; \rightarrow\; t\; \rightarrow\; t
\mathsf{val}\; \mathit{min}\; :\; t\; \rightarrow\; t\; \rightarrow\; t
integer operations
val succ: t \rightarrow t
val pred: t \rightarrow t
val quo: t \rightarrow t \rightarrow t
val modulo: t \rightarrow t \rightarrow t
\mathsf{val}\,\,\mathsf{pgcd}\,:\,\mathsf{t}\,\to\,\mathsf{t}\,\to\,\mathsf{t}
\textit{val full\_pgcd} \ : \ t \ \rightarrow \ t \ \times \ t \ \times \ t
\mathsf{val}\; ppcm\; :\; t\; \to\; t\; \to\; t
\mathsf{val}\; div\_floor:\; t\;\rightarrow\; t\;\rightarrow\; t
val div_ceil: t \rightarrow t \rightarrow t
val\ sqrt\_floor: t \rightarrow t
val sqrt\_ceil : t \rightarrow t
val root\_floor: int \rightarrow t \rightarrow t
val root\_ceil : int \rightarrow t \rightarrow t
parsing and printing
val from_string : string \rightarrow t
```

val to_string : $t \rightarrow string$

3.10 Module Ordered_sets

```
module type OrderedSet =
  type 'a elt
  type 'a t val empty: 'a t
      (* The empty set. *)
val is\_empty: 'a t \rightarrow bool
   (* Test whether a set is empty or not. *)
val mem : 'a elt \rightarrow 'a t \rightarrow bool
   (* mem \ x \ s \text{ tests whether } x \text{ belongs to the set } s. *)
val add: 'a elt \rightarrow 'a t \rightarrow 'a t
   (* add x s returns a set containing all elements of s, plus x. If x was already in s, s is returned unchanged. *)
val singleton : 'a elt \rightarrow 'a t
   (* singleton x returns a set containing x. *)
val remove : 'a elt \rightarrow 'a t \rightarrow 'a t
   (* remove x s returns a set containing all elements of s, except x. If x was not in s, s is returned unchanged. *)
val get\_and\_remove\_min: 'a t \rightarrow 'a elt \times 'a t
   (* get_and_remove_min s returns a pair m, s' where m is the minimum element of s and s' is a set containing
all elements of s, except m. If s is empty, the exception Not_found is raised. *)
val union : a t \rightarrow a t \rightarrow a t
val inter: 'a t \rightarrow 'a t \rightarrow 'a t
val diff: 'a t \rightarrow 'a t \rightarrow 'a t
   (* Union, intersection and set difference. *)
val compare : 'a t \rightarrow 'a t \rightarrow int
   (* Total ordering between sets. Can be used as the ordering function for doing sets of sets. *)
val equal : 'a t \rightarrow 'a t \rightarrow bool
   (* equal s1 s2 tests whether the sets s1 and s2 are equal, that is, contain the same elements. *)
val subset: 'at \rightarrow 'at \rightarrow bool
   (* subset s1 s2 tests whether the set s1 is a subset of the set s2. *)
val iter: (a \text{ elt } \rightarrow \text{ unit}) \rightarrow a \text{ t } \rightarrow \text{ unit}
   (* iter f s applies f in turn to all elements of s. The order in which the elements of s are presented to f is
unspecified. *)
val fold : (a \text{ elt } \rightarrow b \rightarrow b) \rightarrow a \text{ t} \rightarrow b \rightarrow b
   (* fold f s a computes (f \times N \dots (f \times 2 (f \times 1 a))\dots), where x \times 1 \dots x \times N are the elements of s. The order in which
elements of s are presented to f is unspecified. *)
val fold_from_min : ('a elt \rightarrow 'b \rightarrow 'b) \rightarrow 'a t \rightarrow 'b \rightarrow 'b
   (* fold f s a computes (f \times N \dots (f \times 2 (f \times 1 a))\dots), where x \times 1 \dots x \times N are the elements of s. The order in which
elements of s are presented to f is increasing. *)
val filter: (a \text{ elt } \rightarrow \text{ bool}) \rightarrow a \text{ } t \rightarrow a \text{ } t
   (* filter p s returns the set of all elements in s that satisfy predicate p. \times)
val cardinal: 'a t \rightarrow int
   (* Return the number of elements of a set. *)
val elements : 'a t \rightarrow 'a elt list
   (* Return the list of all elements of the given set. The elements appear in the list in some unspecified order. *)
```

```
val choose : 'a t \rightarrow 'a elt
   (* Return one element of the given set, or raise Not_found if the set is empty. Which element is chosen is
unspecified, but equal elements will be chosen for equal sets. *)
val min\_elt: 'a t \rightarrow 'a elt
   (* Return the smallest element of the given set, or raise Not_found if the set is empty. *)
val find : (a \text{ elt } \rightarrow \text{ bool}) \rightarrow a \text{ t } \rightarrow a \text{ elt}
val find_and_apply : ('a elt \rightarrow 'c) \rightarrow 'a t \rightarrow 'c
   (* find_and_apply f set returns f e where e is an element of set such that f e does not raise Not_found. If there
is no such element, Not_found is raised. *)
val exists : (a \text{ elt } \rightarrow \text{ bool}) \rightarrow a \text{ } t \rightarrow \text{ bool}
val for_all : ('a elt \rightarrow bool) \rightarrow 'a t \rightarrow bool
val half\_set: 'a t \rightarrow 'a t
   (* half_set set returns a subset of set such that its elements are minimal and its cardinal is roughly the half of
the cardinal of set. *)
module Make(Ord : Ordered_types.OrderedType) :
   (OrderedSet \text{ with type '} a elt = Ord.t)
           (* Functor building an implementation of the set structure given a totally ordered type. *)
module MakePoly(Ord : Ordered_types.OrderedPolyType) :
   (OrderedSet with type 'a elt = 'a Ord.t)
           (* Functor building an implementation of the set structure given a totally ordered polymorphic type. *)
```

3.11 Module Cache

This module provides some kind of maps with efficient access to elements. The size is bounded so when the cache is full, a free place is obtained by removing the element with the oldest access time.

3.11.1 Type of caches, and basic access functions

val int_int_find : int \times int \rightarrow (int \times int, 'b) cache \rightarrow 'b

```
type ('a, 'b) cache
(create n) returns a new cache of size n. n should be at least 1!
val create: int → ('a, 'b) cache
(add x y c) adds the pair (x,y) in cache c. If there is no room left in c, the earliest accessed pair of c is removed. If there was already a pair (x,z) for some z in c, it becomes hidden. It will never accessible again since there is no remove function on cache. It will eventually be removed after enough calls to add.
val add: 'a → 'b → ('a, 'b) cache → unit
(find x c) returns y such that (x,y) belongs to c. This pair will be marked as the more recently accessed pair. find raises exception Not_found if no such pair exists.
val find: 'a → ('a, 'b) cache → 'b
int_int_find is a specialized version of find whenever the type of its first argument is int × int. In this case, this equality test should be more efficient.
```

3.11.2 Iterator

(iter f c) applies function f to each pair (x,y) of the cache c. the elements are processed from the newest accessed pair to the oldest

```
val iter : (a \rightarrow b \rightarrow unit) \rightarrow (a, b) cache \rightarrow unit
```

function $1 \, n \, f$ is a function that calls f with caching of arguments, in a cache of size n

```
val function 1: int \rightarrow ('a \rightarrow 'b) \rightarrow ('a \rightarrow 'b)
```

3.11.3 Statistics

(collision_rate c) returns a real value t which estimate the repartition of elements in the cache hash table. In fact, it is the average time to access an element in the cache.

```
(max\_hash\_length\ c) is the maximal amount of time to access to an element. (list\_of\_cache\ c) returns the contents of the cache, useful for debugging purposes.
```

```
val collision_rate : ('a, 'b) cache \rightarrow float
val max_hash_length : ('a, 'b) cache \rightarrow int
val list_of_cache : ('a, 'b) cache \rightarrow ('a \times 'b) list
```

3.12 Module Ordered_types

```
\begin{array}{ll} \text{module type } \textit{OrderedType} = \\ \text{sig} \\ \text{type } t \\ \text{val } \textit{compare} : t \rightarrow t \rightarrow \textit{int} \\ \text{end} \\ \\ \text{module type } \textit{OrderedPolyType} = \\ \text{sig} \\ \text{type '} \textit{a } t \\ \text{val } \textit{compare} : \textit{'a } t \rightarrow \textit{'a } t \rightarrow \textit{int} \\ \text{end} \\ \end{array}
```

3.13 Module Class_maps

```
class type ['a, 'b] map = object method \ add : 'a \rightarrow 'b \rightarrow unit method \ find : 'a \rightarrow 'b method \ iter : ('a \rightarrow 'b \rightarrow unit) \rightarrow unit end val empty\_map : ('a \rightarrow 'a \rightarrow int) \rightarrow ('a, 'b) \ map
```

3.14 Module *Polynomials*

This module allows to build polynomials over an arbitrary coefficient ring and an arbitrary set of variables.

3.14.1 Module for coefficient rings

Gives the type of coefficients, two constant 0 and 1, a test for nullity, and fonctions for addition, opposite, substraction and multiplication

```
module type RingType = sig

type coef

val zero : coef

val one : coef

val is\_null : coef \rightarrow bool

val add : coef \rightarrow coef \rightarrow coef

val minus : coef \rightarrow coef

val sub : coef \rightarrow coef \rightarrow coef

val mult : coef \rightarrow coef \rightarrow coef

end
```

3.14.2 Module for polynomials

```
module type PolynomialType =
sig
```

Coefficients and variables

Types for coefficients and variables, and modules for sets and maps of variables.

```
module Base_ring: RingType

type coef = Base_ring.coef

type variable

module Var_set: Ordered_sets.OrderedSet

module Var_map: Ordered_maps.OrderedMap
```

Monomials

 $(make_monomial\ c\ v)$ builds a monomial where c is the coefficient and v is a map giving to each variable its exponent (beware of giving only positive exponents and a non-zero coefficient)

```
(monomial\_coef\ m) and (monomials\_vars) are the corresponding acces functions. (monomial\_degree) returns the degree of m, that is the sum of its exponents. type monomial val make\_monomial: coef \rightarrow (variable,int)\ Var\_map.t \rightarrow monomial val monomial\_coef: monomial \rightarrow coef val monomial\_vars: monomial \rightarrow (variable,int)\ Var\_map.t val monomial\_degree: monomial \rightarrow int
```

Type for polynomials, and basic polynomials

```
zero is the polynomial 0, one is 1 (cte a) is the constant polynomial a
```

```
(var x) is the polynomial consisting in the variable x
type poly
val zero: poly
val one: poly
val cte: coef \rightarrow poly
val var: variable \rightarrow poly
val poly_of_monomial: monomial \rightarrow poly
```

Basic functions over polynomials

```
(is_null p) is true if p is the polynomial 0
  (constant_coef p) is the constant coefficient of p
  (total_degree p) is the degree of p that is the maximum degree of its monomials
  (partial_degre x p) is the partial degree of p in variable x, that is the maximum exponent of x in p
  (set_of_vars p) is the set of variables occurring in p

val is_null : poly → bool
 val constant_coef : poly → coef
 val total_degree : poly → int
 val partial_degree : variable → poly → int
 val set_of_vars : poly → variable Var_set.t
 for debugging purposes
 val size : poly → int
```

Printing function

($print_polynomial\ fc\ fv\ p$) prints the polynomial p on standard output, where fc and fv are two printing functions for coefficients and variables. They must use the Format module for printing.

```
val print_polynomial : (coef \rightarrow unit) \rightarrow (variable \rightarrow unit) \rightarrow poly \rightarrow unit val latex_print_polynomial : out\_channel \rightarrow (coef \rightarrow unit) \rightarrow (variable \rightarrow unit) \rightarrow poly \rightarrow unit (*
```

Polynomial operations

```
(add p1 p2) is [p1] + [p2]

(minus p) is -[p]

(sub p1 p2) is [p1] - [p2]

(mult p1 p2) is [p1] \times [p2]

(power p n) is [p]^{[n]}

*)

val add: poly \rightarrow poly \rightarrow poly val minus: poly \rightarrow poly val sub: poly \rightarrow poly \rightarrow poly val mult: poly \rightarrow poly \rightarrow poly val power: poly \rightarrow int \rightarrow poly
```

Polynomial evaluation

(eval p m) evaluates p with variable values given by m, which is a map from variables to coefficients

```
val eval : (variable \rightarrow coef) \rightarrow poly \rightarrow coef
```

Polynomial substitutions

(substitute m p) replaces in p each occurrence of variables by the corresponding polynomial in m, which is a map from variables to polynomials

```
val substitute : (variable, poly) Var_map.t \rightarrow poly \rightarrow poly
```

Higher-order functions over polynomials

 $(map\ f\ p)$ applies f to each monomials of p and builds the sum of the results. WARNING: this fonction assumes that the resulting set of monomials does have any two monomials with a common power product. If not the case, please use fold and add instead.

```
(iter f p) applies f to each monomials of p.
```

(fold f p a) computes (f m1 ... (f mN a)...) where mi is the ith monomial of p, given in the lexicographic order of power products.

(find f p) search for a monomial m in p such that (f m) does not raise Not_found. raises Not_found if no such monomial exists.

```
\begin{array}{l} \text{val } \textit{map} : (\textit{monomial} \rightarrow \textit{monomial}) \rightarrow \textit{poly} \rightarrow \textit{poly} \\ \text{val } \textit{iter} : (\textit{monomial} \rightarrow \textit{unit}) \rightarrow \textit{poly} \rightarrow \textit{unit} \\ \text{val } \textit{fold} : (\textit{monomial} \rightarrow \textit{`a} \rightarrow \textit{`a}) \rightarrow \textit{poly} \rightarrow \textit{`a} \rightarrow \textit{`a} \\ \text{val } \textit{find} : (\textit{monomial} \rightarrow \textit{`a}) \rightarrow \textit{poly} \rightarrow \textit{`a} \\ \end{array}
```

3.14.3 The polynomial functor

Functor for building polynomial modules from a coefficient ring, an ordered set of variables, and two modules for set and map of variables.

```
module Make

(Ring: RingType)

(Vars: Ordered_types.OrderedType)

(Var_set: Ordered_sets.OrderedSet with type 'a elt = Vars.t)

(Var_map: Ordered_maps.OrderedMap with type 'a key = Vars.t):

PolynomialType with

module Base_ring = Ring
and
type variable = Vars.t
and
module Var_set = Var_set
and
module Var_map = Var_map
```

3.15 Module Classical_maps

```
abstract data type of maps over some keys type ('key, 'data) t empty comp returns an empty map, comp is a comparison function that is used for comparing keys. val empty: ('key \rightarrow 'key \rightarrow int) \rightarrow ('key, 'data) t val add: 'key \rightarrow 'data \rightarrow ('key, 'data) t \rightarrow ('key, 'data) t val find: 'key \rightarrow ('key, 'data) t \rightarrow 'data val remove: 'key \rightarrow ('key, 'data) t \rightarrow ('key, 'data) t val iter: ('key \rightarrow 'data \rightarrow 'a) \rightarrow ('key, 'data) t \rightarrow unit val map: ('data \rightarrow 'data) \rightarrow ('key, 'data) t \rightarrow ('key, 'data) t val fold: ('key \rightarrow 'data \rightarrow 'a) \rightarrow ('key, 'data) t \rightarrow 'a \rightarrow 'a
```

3.16 Module Labelled_graphs

This module implements oriented graphs with labelled vertices (but no labels on arcs), in a functional way.

3.16.1 Vertices

abstract type for vertices and related modules for sets and maps of vertices

type vertex

```
module VertexSet: Set.S with type elt = vertex module VertexMap: Map.S with type key = vertex val print\_vertex\_set: Format.formatter 	o VertexSet.t 	o unit
```

3.16.2 Type for graphs

abstract type for graphs, parameterized by the type of vertex labels type 'vertex_labels graph

3.16.3 Constructor functions for graphs

```
empty_graph is the graph with no vertex
```

 $(add_vertex\ g\ l)$ returns a pair (v,g') where v is the new vertex having label l and g' is the graph obtained by adding v to g

(add_arc $g \ v1 \ v2$) returns the graph obtained by adding to g a new arc from vertex v1 to vertex v2. Raises Failure if the arc already exists.

```
val empty_graph : 'vertex_labels graph

val add_vertex :
    'vertex_labels graph → 'vertex_labels
    → vertex × 'vertex_labels graph

val add_arc :
    'vertex_labels graph → vertex → vertex → 'vertex_labels graph
```

3.16.4 Basic access functions

```
(get_vertex_label g v) returns the label of the vertex v in graph g
```

($get_successors\ g\ v$) returns the set of vertices reachables from v in graph g, i.e. the set of v' such that there is an arc from v to v'.

```
val get_vertex_label : 'vertex_labels graph \rightarrow vertex \rightarrow 'vertex_labels val get_successors : 'vertex_labels graph \rightarrow vertex \rightarrow VertexSet.t
```

3.16.5 Iterator

```
(fold_left_graph f a g) returns f (... (f (f a v1) v2) ...) vn where v1,...,vn is the set of vertices of g, in a non-specified order.

val fold_left_graph:

('a \rightarrow vertex \rightarrow 'a) \rightarrow 'vertex_labels graph \rightarrow 'a
```

3.16.6 Extraction of subgraphs

Removing one vertex

The next two functions provide two ways of removing a vertex from a graph. ($remove_vertex_and_its_arcs\ g\ v$) returns the graph obtained from g by removing the vertex v and all the arcs going to v or comming from v. ($remove_vertex_and_link_its_arcs\ g\ v$) returns the graph obtained from g by removing g and "linking" the arcs going to v with arcs coming from v: if (v1,v) and (v,v2) are arcs in g then (v1,v2) is an arc in the new graph returned.

```
val remove_vertex_and_its_arcs :
    'vertex_labels graph → vertex → 'vertex_labels graph
val remove_vertex_and_link_its_arcs :
    'vertex_labels graph → vertex → 'vertex_labels graph
```

Extraction of of a whole subgraph

 $(get_subgraph\ g\ e)$ returns the subgraph of g whose vertices are in e, and arcs are the arcs of g that come from a vertex in e and go to a vertex in e.

```
\begin{tabular}{ll} val $ get\_subgraph : \\ $'vertex\_labels $ graph $ \to $ VertexSet.t $ \to $'vertex\_labels $ graph $ \end{tabular}
```

3.16.7 Printing

(print g f) prints the graph g, using f as printing function for labels. Warning: this function f must use Format library!

```
(compact_print g f) does the same in a more compact format, using numbers to denote vertices. (dot_print g f) does the same in dot format, see http://www.research.att.com/sw/tools/graphviz/. val print: 'vertex_labels graph \rightarrow ('vertex_labels \rightarrow unit) \rightarrow unit val compact_print: 'vertex_labels graph \rightarrow ('vertex_labels \rightarrow unit) \rightarrow unit val dot_print: 'vertex_labels graph \rightarrow ('vertex_labels \rightarrow unit) \rightarrow unit
```

3.16.8 Cycles, connected components and such

Cycles

```
(vertex_has_a_1_cycle g v) returns true if there is an arc from v to v in gq.
    (vertices_on_cycle g) returns the set of vertices of g that belong to a cycle
    (vertices_not_on_cycle g) returns the set of vertices of g that do not belong to any cycle
    (split_cycles g) returns a pair (a,b) of sets of vertices, where a is the set of vertices that lay on a cycle of g, b
the set of vertices that do not belong to any cycle of g

val vertex_has_a_1_cycle :
    'vertex_labels graph → vertex → bool

val split_cycles :
    'vertex_labels graph → VertexSet.t × VertexSet.t

obsolete

val vertices_on_cycle :
    'vertex_labels graph -> VertexSet.t;;

val vertices_not_on_cycle :
    'vertex_labels graph -> VertexSet.t;;
```

Strongly connected components

(compute_strongly_connected_components even_if_no_cycles g) returns the set of strongly connected components of the directed graph g. These are defined as follows: let G = (V, E) be a directed graph, where V is the set of vertices and E the arcs, then let G' = (V, E') where E' is defined by $v_1 \leftrightarrow_{E'} v_2$ iff $v_1 \to_E^* v_2$ and $v_2 \to_E^* v_1$. Then $\leftrightarrow_{E'}$ is an equivalence relation, and the strongly connected components of G are the equivalence classes of it.

if even_if_no_cycles is set to false, then the components containing only one vertex which has no arc to itself are not returned. Example: for the graph $a \to a, a \to b, b \to c, c \to c$, the component b will not be returned.

```
val compute_strongly_connected_components: bool \rightarrow 'vertex_labels graph \rightarrow VertexSet.t list
```

Chapter 4

Library regexp

This is a regexp library, to be documented by Benjamin.

4.1 Module Regular_expr

This module defines the regular expressions, and provides some simple manipulation of them.

4.1.1 The regexp datatype and its constructors

The type of regular expressions is abstract. Regular expressions may be built from the following constructors:

- empty is the regexp that denotes no word at all.
- epsilon is the regexp that denotes the empty word.
- char c returns a regexp that denotes only the single-character word c.
- chars s returns a regexp that denotes any single-character word belonging to set of chars s.
- string str denotes the string str itself.
- star e where e is a regexp, denotes the Kleene iteration of e, that is all the words made of concatenation of zero, one or more words of e.
- alt e1 e2 returns a regexp for the union of languages of e1 and e2.
- seq e1 e2 returns a regexp for the concatenation of languages of e1 and e2.
- opt e returns a regexp for the set of words of e and the empty word.
- some e denotes all the words made of concatenation of one or more words of e.
- compl e denotes all the words not recognized by the regexp e.

```
type 'a regexp module CharSet : Inttagset.IntTagSetModule with type 'a elt = 'a val empty : 'a regexp val epsilon : 'a regexp val char : 'a \rightarrow 'a regexp val chars : 'a CharSet.t -> 'a regexp; val string : string -> regexp;
```

```
val star : 'a regexp \rightarrow 'a regexp
val alt : 'a regexp \rightarrow 'a regexp \rightarrow 'a regexp
val seq : 'a regexp \rightarrow 'a regexp \rightarrow 'a regexp
val opt : 'a regexp \rightarrow 'a regexp
val some : 'a regexp \rightarrow 'a regexp
```

4.1.2 Simple regexp operations

The following three functions provide some simple operations on regular expressions:

- nullable r returns true if regexp r accepts the empty word.
- residual r c returns the regexp r' denoting the language of words w such that cw is in the language of r.
- firstchars r returns the set of characters that may occur at the beginning of words in the language of e.

```
val nullable : 'a regexp \rightarrow bool
val residual : 'a regexp \rightarrow 'a \rightarrow 'a regexp
val firstchars : 'a regexp \rightarrow 'a CharSet.t
```

4.1.3 Regexp matching by runtime interpretation of regexp

 $match_string \ r \ s$ returns true if the string s is in the language denoted by r. This function is by no means efficient, but it is enough if you just need to match once a simple regexp against a string.

If you have a complicated regexp, or if you're going to match the same regexp repeatedly against several strings, we recommend to use compilation of regexp provided by module *Automata*.

4.2 Implementation of module Regular_expr

```
module CharSet =
  Inttagset.MakePoly
     (struct type 'a t = 'a let tag = Hashtbl.hash end)
type 'a regexp =
  | Empty
  | Epsilon
  Char of 'a CharSet.elt
                                                                                                  (* cardinal at least 2 *)
  Charset of 'a CharSet.t
  Star of 'a regexp
  Alt of 'a regexp_set
  | Seq of 'a regexp \times 'a regexp
and 'a regexp_set = 'a regexp list
                                                                                      (* naive implementation of sets *)
let add e 1 =
  if List.mem e 1 then 1 else e :: 1
let rec union 11 12 =
  match 11 with
     |[] \rightarrow 12
     |e::1 \rightarrow \text{ if } List.mem e 12 \text{ then } union 1 12 \text{ else } e:: union 1 12
```

```
let empty = Empty
let epsilon = Epsilon
let char c = Char c
let star e =
   match e with
      | Empty | Epsilon \rightarrow Epsilon
      | Star \_ \rightarrow e |
      | \_ \rightarrow Star e
let alt e1 e2 =
   match e1, e2 with
      |Empty, \_ \rightarrow e2|
      |\_,Empty \rightarrow e1
      |Alt(11),Alt(12)| \rightarrow Alt(union 11 12)
      |Alt(11), \_ \rightarrow Alt(add \ e2 \ 11)
      |\_,Alt(12)| \rightarrow Alt(add\ e1\ 12)
      | \_ \rightarrow \text{ if } e1 = e2 \text{ then } e1 \text{ else } Alt([e1; e2])
let seg e1 e2 =
   match e1, e2 with
      | Empty, \_ \rightarrow Empty
      | \_, Empty \rightarrow Empty
       | Epsilon, \_ \rightarrow e2
      \downarrow_, Epsilon \rightarrow e1
      | \bot \rightarrow Seq(e1, e2)
```

4.2.1 extended regexp

```
let some e = (seq \ e \ (star \ e))
let opt e = (alt \ e \ epsilon)
```

4.2.2 Regexp match by run-time interpretation of regexp

```
let rec nullable r = match r with | Empty \rightarrow false | Epsilon \rightarrow true | Char \_ \rightarrow false | Charset \_ \rightarrow false | Star e \rightarrow true | Alt(1) \rightarrow List.exists nullable 1 | Seq(e1, e2) \rightarrow nullable e1 \land nullable e2
```

```
let rec residual r c =
  match r with
      |Empty \rightarrow Empty|
       Epsilon \rightarrow Empty
       Char a \rightarrow \text{if } a = c \text{ then } Epsilon \text{ else } Empty
       Charset s \rightarrow if CharSet.mem c s then Epsilon else Empty
      Star e \rightarrow seq (residual e c) r
      |Alt(1)| \rightarrow
            List.fold_right
              (fun e accu \rightarrow alt (residual e c) accu)
              Empty
     |Seq(e1,e2)| \rightarrow
           if nullable(e1)
           then alt (seq (residual e1 c) e2) (residual e2 c)
           else seq (residual e1 c) e2
firstchars r returns the set of characters that may start a word in the language of r
 let rec firstchars r =
    match r with
        \mid Empty \rightarrow CharSet.empty
        Epsilon \rightarrow CharSet.empty
        Char a \rightarrow CharSet.singleton a
        Charset s \rightarrow s
        Star e \rightarrow first chars e
       |Alt(1)| \rightarrow
             List.fold_right
                (fun e accu \rightarrow
                    CharSet.union (firstchars e) accu)
                CharSet.empty
       |Seq(e1,e2)| \rightarrow
             if nullable e1
             then CharSet.union (firstchars e1) (firstchars e2)
             else firstchars e1
```

4.3 Module Automata

```
module CharSet: Ordered_sets.OrderedSet with type 'a elt = 'a
```

Raised when one tries to count the number of words in an infinite language.

```
exception Infinite_language
```

count_nf sigma left_members compiled_srs returns the number of words not reducible by the left_members of the compiled_srs.

```
val count_nf: 'a CharSet.t \rightarrow 'a Words.word list \rightarrow 'a String_rewriting.compiled_srs \rightarrow int
```

4.4 Implementation of module Automata

```
module CharSet =
Ordered_sets.MakePoly
(struct type 'a t = 'a let compare = compare end)
```

4.4.1 The type of automata

Automata considered here are deterministic.

The states of these automata are always represented as natural numbers, the initial state always being 0. An automaton is then made of a transition table containing at $i \times j$ the number of transitions from i to j.

```
type 'a full_automaton =
       full_auto_trans : int array array;
       full_auto_accept : bool array;
       full_auto_init : bool array;
```

Raised when one tries to count the number of words in an infinite language.

exception Infinite_language

```
let count\_words \{full\_auto\_accept = a;
                      full_auto\_trans = t;
                      full_auto_init = initials \} =
  let n = Array.length t in
  let reaches\_final = Array.create n false in
  let reaches_final_on_cycle = Array.create n false in
  let is\_open = Array.create n false in
  let rec traverse i =
     if is\_open.(i) \lor reaches\_final\_on\_cycle.(i) then begin
        reaches_final_on_cycle (i) \leftarrow true
     end else begin
        reaches\_final(i) \leftarrow true;
        is\_open.(i) \leftarrow true;
       for j = 0 to n - 1 do if t.(j).(i) > 0 then traverse j done;
        is\_open.(i) \leftarrow false
     end
  in
     for i = 0 to n - 1 do if a(i) then traverse i done;
  let count = Array.create n None in
  let rec count\_from i = match count.(i) with
     | Some k \rightarrow
     | None \rightarrow
          let k = ref 0 in
          for i = 0 to n - 1 do
             if t.(i).(j) > 0 \land reaches\_final.(j) then begin
               if reaches_final_on_cycle.(j) then raise Infinite_language;
                k := !k + t.(i).(j) + count\_from j
             end
          done:
          count.(i) \leftarrow Some !k;
  in
  let total = ref 1 in
  for i = 0 to n - 1 do if initials.(i) then total := !total + count\_from\ i done;
  !total
let rec get\_all\_strict\_sub \ w =
  if Array.length w \leq 1 then CharSet.empty else
       let result = ref (CharSet.empty) in
```

```
for i = 1 to (Array.length w - 1) do
            result := CharSet.add (Array.sub w i (Array.length w - i) !result
         done:
         !result
     end
let does_overlap 1 r n =
  assert (n \leq Array.length r);
  if n > Array.length 1 then false else
     let first_1 = Array.length 1 - n in
     let first_r = 0 in
       try
          for i = 0 to n - 1 do
             if 1.(first\_1 + i) \neq r.(first\_r + i) then raise Exit
          true
        with Exit \rightarrow false
exception Found of int
let longest_right_overlap left_array right =
  for n = Array.length \ right \ downto 1 \ do
     Array.iteri (fun i l \rightarrow if i = 0 then () else
                         if does_overlap 1 right n then raise (Found i))
        left_array
  done:
  assert false
with
     Found n \rightarrow n
let count_nf sigma word_list compiled_srs =
  let word_array = List.map Array.of_list word_list in
  let sigma\_list = CharSet.elements \ sigma in
  let states_from_constructors = CharSet.fold
                                               (\text{fun } x \ acc \rightarrow CharSet.add [|x|] \ acc)
                                              sigma CharSet.empty
  in
  let states_from_sub_terms =
     List.fold_left
        (\text{fun } acc \ x \rightarrow CharSet.union } acc \ (get\_all\_strict\_sub \ x))
        states_from_constructors word_array
  in
  let state_number = 1 + CharSet.cardinal states_from_sub_terms in
  let _ = Format.printf "Number_of_states_of_the_normal_forms_automaton:_%d\n" state_number in
  let states_array = Array.create state_number [||] in
  let counter = ref 1 in
  let _ = CharSet.iter
                (\text{fun } x \rightarrow \text{states\_array.}(!\text{counter}) \leftarrow x; \text{ incr counter})
               states_from_sub_terms
  in
  let automaton =
     { full_auto_trans = Array.make_matrix state_number state_number 0;
        full\_auto\_accept = Array.init state\_number (fun x \rightarrow true);
        full\_auto\_init = Array.init state\_number (function <math>x \rightarrow x = 0);
  in
```

Chapter 5

Library Orderings

This library is for general definitions of orderings.

5.1 Module Finite_orderings

This module allows one to define ordering on finite sets of objects. The construction of such an ordering on a set E is obtained by starting from the "smallest" ordering on E (that is the relation =) and adding comparison pairs x > y or x = y, performing the transitive closure at each step.

Formally speaking, this allows to build ordering on infinite sets, but only if they differ from identity on finitely many pairs.

```
exception Incompatible
```

```
type 'a finite_ordering
```

```
val compare : 'a finite_ordering \rightarrow 'a Orderings_generalities.ordering val add_equiv : 'a finite_ordering \rightarrow 'a \rightarrow 'a \rightarrow 'a finite_ordering val add_greater : 'a finite_ordering \rightarrow 'a \rightarrow 'a \rightarrow 'a finite_ordering
```

(identity_ordering comp) builds an identity ordering object on a given data type t. comp is any total comparison function on t: comp x y is 0 if x and y are equal, negative if x < y, and positive if x > y

```
val identity_ordering : (a \rightarrow a \rightarrow int) \rightarrow a \text{ finite\_ordering}
```

Building finite orderings from lists of comparisons.

(add_list o p) adds all comparison in list p to o. p is a concrete type that represents a list of comparison. For example x > y = z < t is represented as Gt(x, Eq(y, Lt(z, One(t)))). On that example, it will do add_greater x y, add_equiv y z, and add_greater t z, of course performing transitive closure at each step.

```
type 'a precedence =
One of 'a

| Gt of 'a × 'a precedence
| Lt of 'a × 'a precedence
| Eq of 'a × 'a precedence

val add_list: 'a finite_ordering \rightarrow 'a precedence \rightarrow 'a finite_ordering

val map_prec: ('a \rightarrow 'b) \rightarrow 'a precedence \rightarrow 'b precedence
```

5.2 Module Orderings_generalities

Comparison results

First, we define a type for given all different possible results of a comparison. Usually, a comparaison may result in either >, < or =; but here the situation is a more complicated.

First, we are interested in what is called pre-orderings, or quasi-orderings, that is two different objects may be *equivalent* w.r.t. an ordering. Moreover, we are also interested in partial orderings, that is two objects may be *uncomparable* w.r.t to an ordering (neither > nor <). That's why here an ordering function over a type t will be any function of type $t \to t \to result$ where *result* is either *equivalent*, *greater_than*, *less_than* or *uncomparable*. Such a ordering function defines in fact both a (partial) quasi-ordering \succeq and an associated (partial) strict ordering \succeq :

```
x ≥ y if order(x,y) = '> 'or' = ';
x > y if order(x,y) = '> '.
The ordering function order must be design in such a way that:
≥ is reflexive and transitive: for all x, x ≥ x; and for all x, y and z, x ≥ y and y ≥ z imply x ≥ z;
> is transitive: for all x, y and z, x > y and y > z imply x > z;
if x ≥ y then y ≤ x;
if x > y then y < x;</li>
```

• if $x \succ y$ then $x \succeq y$;

• ...

Moreover, we will be interested mainly an *term orderings*, that is orderings defined over sets of terms with variables. In that case, one is interested in having *stable* orderings: if $s \succeq t$ then $s\sigma \succeq t\sigma$ for all substitutions σ (and the same for \succ). In that situation, more complicated things may happen, for example let us assume that a is the smallest constant of a signature, then one would like to have $x \succeq a$ for any variable x, but in fact for the substitution $\sigma = \{x \mapsto a\}$ we would get $x\sigma \simeq a$ and for any other $x\sigma \succ a$. So what should return the function *order* on (x,a)? '> ' or ' = ' would be wrong so the only correct answer would be 'uncomparable'. These is not satisfactory at all, so we introduce two additional possible answers for an ordering function: *greater or equivalent* and *less or equivalent*.

Finally, the type of comparison is given by the following

```
type comparison_result =
  | Equivalent
   Greater_than
   | Less_than
   | Greater_or_equivalent
   Less_or_equivalent
  | Uncomparable
(string_of_comparison_result r) returns a string for displaying r
val string_of_comparison_result : comparison_result → string
type 'a ordering = 'a \rightarrow 'a \rightarrow comparison_result
val is_greater_or_equal : comparison_result → bool
val is_less_or_equal : comparison_result → bool
(greater_than_all o x 1) returns true if for all y in list 1, x is greater than y for o
val greater_than_all:
   'a ordering \rightarrow 'a \rightarrow 'a list \rightarrow bool
(exists_greater_or_equal o 1 y) returns true if there exists x in list 1 such that x is greater than or equal to y for o
val exists_greater_or_equal:
   'a ordering \rightarrow 'a list \rightarrow 'a \rightarrow bool
```

```
val forall_exists_greater:
'a ordering \rightarrow 'a list \rightarrow 'a list \rightarrow bool
val remove_equivalent_elements:
'a ordering \rightarrow 'a list \rightarrow 'a list \rightarrow ('a list \times 'a list)
```

General ordering functionals

Products of orderings

(lexicographic_extension o) where o is an ordering function over a type t, is the ordering function o' over lists of t by lexicographic use of o, that is $(x_1, \ldots, x_n) \succ' (y_1, \ldots, y_n)$ if there is a k s.t. $x_1 \succeq y_1, \ldots, x_{k-1} \succeq y_{k-1}$ and $x_k \succ y_k$. Raises Invalid_argument of the two lists do not have the same length.

(lexicographic_extension_of_orderings 1) where 1 is a list of ordering functions o_1, \ldots, o_n all over the same type t, is the ordering function over t given by $x \succ y$ if there is a k s.t. $x \succeq_1 y, \ldots x \succeq_{k-1} y$ and $x \succ_k y$

```
val lexicographic_extension : ('a ordering) \rightarrow ('a list ordering)
val lexicographic_extension_of_orderings : ('a ordering) list \rightarrow ('a ordering)
```

($multiset_extension\ o$) where o is an ordering function over a type t, is the ordering function o' over lists of t by multiset use of o.

```
val multiset_extension : ('a ordering) \rightarrow ('a list ordering)
```

Chapter 6

Library dioph

This is a dioph library, to be documented by Evelyne.

Chapter 7

Library Integer_solver

This library provides methods for solving integer constraints: decision of validity/satisfiability of arbitrary first-order linear formulas, solving arbitrary quantifer-free formulas over finite domains, etc.

7.1 Module *I_solve*

i_solve prend comme arguments

- un vecteur d'entiers [|v0; v0 + v1; ...; v0 + v1 + ... + vk|] où v0 représente le nombre de variables non instanciées, et vi, i > 0, le nombre de variables instanciées dans la theorie Ti,
- un systeme d'equations ligne par ligne ;
- une liste qui correspond a pe.edge

et retourne un couple de vecteurs de solutions tel que: le premier element contient les solutions homogenes, i.e. qui valent 0 sur les variables instanciees, le second les autres solutions, sachant que les composantes associees a des variables instanciees ne peuvent prendre que les valeurs 0 et 1, et que de plus, si une composante associee a une variable instanciee dans la theorie Ti vaut 1, alors toutes les composantes associees a des variables instanciees dans une theorie Tj , i <> j, valent 0, Ti et Tj etant toutes deux des theories REGULIERES, COLLAPSE-FREE.

On va essayer autant que faire se peut de ne pas instancier des variables par une valeur contenant une variable marquee interdite.

Si des identifications de variables instanciees sont necessaires, on aura $AVEC_IDENT$, sinon $SANS_IDENT$. i_solve_modulo fonctionne comme i_solve , avec un premier argument supplementaire, n, qui est l'entier modulo lequel on travaille. Au niveau de l'unification, ca correspond a l'unification ACUN(n), ou N(n) est la nilpotence d'ordre n: $x^n = 1$.

```
type identifications_for_unification =
    Without_identifications of (int array array) × (int array array)
    | With_identifications of (int array array) × (int array array)
    | No_sol
```

Careful extension of the functions / and mod, quotient and reminder of the integer division over the negative and positive integers in such a way that

```
\text{ocwbegindcode } p = (p \setminus p \neq p \neq p \neq q + (p \setminus p \neq q), \land p \neq 0 \leq (p \neq q) \leq abs(q).
```

```
\begin{array}{l} \text{val } \textit{zquo} : \textit{int} \rightarrow \textit{int} \rightarrow \textit{int} \\ \text{val } \textit{zmod} : \textit{int} \rightarrow \textit{int} \rightarrow \textit{int} \end{array}
```

(divides c n v) checks whether the integer c divides the nth first elements of the array of integers v.

```
val divides : int \rightarrow int \rightarrow int \ array \rightarrow bool
val i_solve :
int \ array \rightarrow int \ array \ array \rightarrow (int \times int) \ list \rightarrow identifications_for_unification
```

7.2 Module *I_solve_modulo*

open I_solve

i_solve_modulo prend comme arguments

- un entier n modulo lequel on travaille Au niveau de l'unification, ca correspond a l'unification ACUN(n), ou N(n) est la nilpotence d'ordre n: xⁿ = 1.
- un vecteur d'entiers [|v0;v0+v1;...;v0+v1+...+vk|] où v0 represente le nombre de variables non instanciees, et vi, i>0, le nombre de variables instanciees dans la theorie Ti,
- un systeme d'equations ligne par ligne
- une liste qui correspond a pe.edge

et retourne un couple de vecteurs de solutions tel que: le premier element contient les solutions homogenes, i.e. qui valent 0 sur les variables instanciees, le second les autres solutions, sachant que les composantes associees a des variables instanciees ne peuvent prendre que les valeurs 0 et 1, et que de plus, si une composante associee a une variable instanciee dans la theorie Ti vaut 1, alors toutes les composantes associees a des variables instanciees dans une theorie Tj , i <> j, valent 0, Ti et Tj etant toutes deux des theories REGULIERES, COLLAPSE-FREE.

On va essayer autant que faire se peut de ne pas instancier des variables par une valeur contenant une variable marquee interdite.

Si des identifications de variables instanciees sont necessaires, on aura AVEC_IDENT, sinon SANS_IDENT.

```
val i_solve_modulo :
    int → int array → int array array → (int × int) list →
    identifications_for_unification
```

7.3 Module Abstract_constraint

```
type expr =
   Cte of Numbers.t
   Var of string
   Plus of expr \times expr
   | Mult of expr \times expr
   | Sub of expr \times expr
   Minus of expr
   | Quotient of expr \times expr
val plus : expr \rightarrow expr \rightarrow expr
val mult : expr \rightarrow expr \rightarrow expr
val minus : expr \rightarrow expr
val power : expr \rightarrow int \rightarrow expr
type formula =
   True
   False
   Comp of expr \times string \times expr
                                                                                                   (* <, >, =, > =, < = or < > or | *)
   And of formula list
                                                                                                           (* length at least 2 *)
    Or of formula list
                                                                                                           (* length at least 2 *)
   Neg of formula
   | Implies of formula \times formula
   Equiv of formula \times formula
   Exists of string list × formula
                                                                                                           (* length at least 1 *)
   | Forall of string list \times formula
                                                                                                           (* length at least 1 *)
val divisible : expr \rightarrow expr \rightarrow formula
```

```
val neg: formula \rightarrow formula
val exists: string list \rightarrow formula \rightarrow formula
val forall : string list \rightarrow formula \rightarrow formula
val conj: formula \rightarrow formula \rightarrow formula
val conj_all: formula list \rightarrow formula
val disj : formula \rightarrow formula \rightarrow formula
val disj_all: formula list \rightarrow formula
val free_vars : formula → string list
val print_formula : formula \rightarrow unit
val print\_expr : expr \rightarrow unit
Rename all variables in a formula.
val rename_formula : (string \times string) list \rightarrow formula \rightarrow formula
Rename all variables in an expression.
val rename_expr : (string \times string) list \rightarrow expr \rightarrow expr
build_renaming I builds an association list from elements of I to fresh strings, that is never used before for renam-
ing strings.
val build_renaming : string list \rightarrow (string \times string) list
```

7.4 Module Linear_constraints

7.4.1 Abstract data type for variables

variables are abstract, may only be built with make_var.

type var_id

val make_var : string → var_id

Same as make_var but the generated name is prepended with unique number.

val $make_var_uniq_name : string \rightarrow var_id$

val var_name : var_id → string

7.4.2 Abstract data type for expressions

```
zero is 0, one is 1, (cte n) is the constant n, (var v) is the variable v, (add e1 e2) is e_1 + e_2, (sub e1 e2) is e_1 - e_2, (minus e) is -e, (times n e) is ne.
```

type expr

val zero : expr val one : expr val minus_one : expr val cte : Numbers.t \rightarrow expr val var : var_id \rightarrow expr val add_cte : expr \rightarrow Numbers.t \rightarrow expr

```
val add: expr \rightarrow expr \rightarrow expr
val sub: expr \rightarrow expr \rightarrow expr
val minus: expr \rightarrow expr
val times: Numbers.t \rightarrow expr \rightarrow expr
val div: expr \rightarrow Numbers.t \rightarrow expr
val is\_cte: expr \rightarrow bool
val val\_of\_cte: expr \rightarrow Numbers.t
val get\_coef: expr \rightarrow var\_id \rightarrow Numbers.t
val remove\_coef: expr \rightarrow var\_id \rightarrow expr
common\_denominator e returns the least common multiple of denominators in e.
val common\_denominator: expr \rightarrow Numbers.t
module VarMap: Inttagmap.IntTagMapModule
with type 'a key = var\_id
type substitution = (unit, expr) VarMap.t
```

7.4.3 Abstract data type for formulas

Formulas are hashconsed for efficiency. All formulas should be build using given constructor below. The internal structure is exported for pattern-matching purpose only.

```
type formula = formula_struct Hcons.hash_consed

and formula_struct =

| True
| False
| Null of expr
| PositiveOrNull of expr
| Divisible of expr × Numbers.t
| And of substitution × formula_struct Ptset.t
| Or of formula_struct Ptset.t
| Not of formula × formula
| Implies of formula × formula
| Equiv of formula × formula
| Exists of var_id list × formula
| Forall of var_id list × formula
```

7.4.4 Atomic formulas

```
(eq e1 e2) is e_1 = e_2, (ne e1 e2) is e_1 \neq e_2, (ge e1 e2) is e_1 \geq e_2, (gt e1 e2) is e_1 > e_2, (le e1 e2) is e_1 \leq e_2, (lt e1 e2) is e_1 < e_2. (divides n e) is n divides e, where n should be non null. val true_formula: formula val false_formula: formula val null: expr \rightarrow formula val positive: expr \rightarrow formula
```

val positive_or_null : expr \rightarrow formula

val negative : $expr \rightarrow formula$

 $val\ eq: expr
ightarrow expr
ightarrow formula$

val $ne : expr \rightarrow expr \rightarrow formula$

 $\mathsf{val}\; ge\; :\; expr\; \rightarrow\; expr\; \rightarrow\; formula$

 $\mathsf{val}\; \mathsf{gt}\; :\; \mathsf{expr}\; \rightarrow\; \mathsf{expr}\; \rightarrow\; \mathsf{formula}$

val $le: expr \rightarrow expr \rightarrow formula$

val $lt: expr \rightarrow expr \rightarrow formula$

val divides : Numbers.t \rightarrow expr \rightarrow formula

val is_atom: formula → bool

7.4.5 Propositional connectors

hash_disj s (resp. hash_conj) builds the disjunction (resp conjunction) of formulas in s, assuming they are already hash consed.

these functions take in account the basic cases where s is empty or has only one elements.

val hash_disj : formula_struct Ptset.t \rightarrow formula

val hash_conj : substitution \rightarrow formula_struct Ptset.t \rightarrow formula

neg f builds $\neg f$, conj f1 f2 builds $f1 \land f2$, disj f1 f2 builds $f1 \lor f2$. conj_list l builds the conjunct of all formulas in list l, disj_list l builds the disjunct of all formulas in list l. implies f1 f2 builds f1\rightarrow f2, equiv f1 f2 builds f1 \leftrightarrow f2.

val $neg : formula \rightarrow formula$

val conj: formula \rightarrow formula \rightarrow formula

val disj: formula \rightarrow formula \rightarrow formula

val conj_list : $formula\ list \rightarrow\ formula$

val $disj_list$: formula $list \rightarrow formula$

val implies : formula \rightarrow formula \rightarrow formula val equiv : formula \rightarrow formula \rightarrow formula

7.4.6 Iterators for conjunction and disjunction

map_conj_subst f sigma s computes

$$\bigwedge_{x=e \in [sigma]} f(x=e) \land \bigwedge_{h \in [s]} fs$$

map_conj_no_subst f sigma s computes

$$\bigwedge_{x=e \in [sigma]} x = e \land \bigwedge_{h \in [s]} fs$$

map_disj_set f s computes

$$\bigvee_{h \in [s]} fs$$

```
val map_conj_subst:  
    (formula \rightarrow formula) \rightarrow substitution \rightarrow formula_struct Ptset.t \rightarrow formula val map_conj_no_subst:  
    (formula \rightarrow formula) \rightarrow substitution \rightarrow formula_struct Ptset.t \rightarrow formula val map_disj_set:  
    (formula \rightarrow formula) \rightarrow formula_struct Ptset.t \rightarrow formula
```

7.4.7 Quantifiers

```
exists x f builds formula \exists x.f. forall x f builds formula \forall x.f.
val exists : var\_id \rightarrow formula \rightarrow formula
val forall : var\_id \rightarrow formula \rightarrow formula
val exists_s : var\_id list \rightarrow formula \rightarrow formula
val forall_s : var\_id list \rightarrow formula \rightarrow formula
```

7.4.8 Translation from abstract formulas

from_abstract_formula f builds the linear formula from the abstract formula f. returns also first the environment of free variables. Raises *Not_linear* of f is not linear.

```
exception Not\_linear type var\_env = (string \times var\_id) list val from\_abstract\_expr: var\_env \rightarrow Abstract\_constraint.expr \rightarrow var\_env \times expr val from\_abstract\_formula: var\_env \rightarrow Abstract\_constraint.formula \rightarrow var\_env \times formula
```

7.4.9 Free vars of formula

```
free_vars f returns the list of free variables of f
      free_vars_env s f returns the union of s and the list of free variables of f
val free_vars : formula → var_id list
val free_vars_env : var_id list → formula → var_id list
val var_occurs : var_id → formula → bool
```

7.4.10 Instanciation, substitution

```
    inst f x n returns the formula obtained by replace the variable x by number n in f.
    subst f x e returns the formula obtained by substituting the variable x by expression e in f.
    type instantiation = (unit,Numbers.t) VarMap.t
    val inst1: var_id → Numbers.t → formula → formula
    val subst1: var_id → expr → formula → formula
```

```
val inst: instantiation \rightarrow formula \rightarrow formula val subst: substitution \rightarrow formula \rightarrow formula
```

7.4.11 Renaming

```
Type of the renaming tables
```

type var_renaming = (unit, var_id) VarMap.t

Rename all variables in a formula.

val rename_formula : var_renaming \rightarrow formula \rightarrow formula

Rename all variables in an expression.

val rename_expr : var_renaming \rightarrow expr \rightarrow expr

build_renaming 1 builds a fresh renaming for the variables of 1.

val build_renaming : var_id list → var_renaming

7.4.12 Printing

(print f) prints the formula f on Format.std_formatter channel.

val fprint_expr : Format.formatter \rightarrow expr \rightarrow unit

val $print_expr : expr \rightarrow unit$

val fprint : Format.formatter \rightarrow formula \rightarrow unit

val print : formula \rightarrow unit

val remove_denominators : formula \rightarrow formula

7.5 Module Presburger

open Linear_constraints

simplify f returns a formula g equivalent to f if interpreted in integers, performing geometric deductions over linear constraints.

For example:

x > 0 and x < 0 simplifies to false.

x > 0 and x > 2 simplifies to x > 2.

 $x \le 0$ or $x \ge 0$ simplifies to true.

 $x \le 0$ or $x \ge 1$ simplifies to true (since x is an integer).

assumes no denominators

val simplify: formula -> formula;;

assumes no denominators

val eliminate_quantifiers : formula \rightarrow formula

is_satisfiable f returns true whenever f is satisfiable, i.e. is true for some integer values of its free variables. is_valid f returns true whenever f is valid, i.e. is true for each integer values of its free variables. has_finitely_many_solutions f is true whenever f is true for finitely many number of values of its free variables.

```
val is_satisfiable : formula → bool
val is_valid : formula → bool
```

```
val has_finitely_many_solutions: formula \rightarrow bool get\_all\_solutions f returns all solutions of f. Raises Infinite if there are infinitely many of them. exception Infinite val get\_all\_solutions: formula \rightarrow (var\_id \times Numbers.t) list list
```

7.6 Module Finite_domains

This module defines finite domains constraints over integers, and allows to solve such constraints for given intervals to search for variables.

7.6.1 Finite domain variables

```
val verbose : int ref

type fd_var_id

val string_of_fd_var_id : fd_var_id → string

val print_fd_var_id : fd_var_id → unit

val new_fd_var_id : unit → fd_var_id

val fd_var_id_of_string : string → fd_var_id

module Fd_var_ord :

sig

type t = fd_var_id

val compare : fd_var_id → fd_var_id → int

end

module Fd_var_map : Ordered_maps.OrderedMap with type 'a key = fd_var_id

module Fd_var_set : Ordered_sets.OrderedSet with type 'a elt = fd_var_id
```

7.6.2 The domains expressions

Domain expressions are made from classical arithmetical operations, and min(x) and max(x) for any domain variable x.

There are two version of division depending whether the result is rounded to the floor or the ceiling. The same for square root and arbitrary nth root.

```
type domain_operation =
    Add | Sub | Mult | Div_floor | Div_ceil

type domain_expression =
    Const of Numbers.t
| Min of fd_var_id
| Max of fd_var_id
| Oper of domain_expression × domain_operation × domain_expression
| Sqrt_floor of domain_expression
| Sqrt_ceil of domain_expression
| Power of domain_expression × int
| Root_floor of int × domain_expression
| Root_ceil of int × domain_expression
```

7.6.3 Finite domain constraints

7.6.4 Solving constraints

(finite_domain_solve M C) returns a solution of the constraints C in the domain 0..M(x) for each variable x. Raises exception Finite_domains.No_solution if no solution exists.

```
exception No\_solution
exception Time\_out
val time\_out: bool\ ref
val time\_out\_signal\_handler: `a 	o unit
val finite\_domain\_solve:
(fd\_var\_id, Numbers.t)\ Fd\_var\_map.t 	o fd\_constraint\ list \\ 	o (fd\_var\_id, Numbers.t)\ Fd\_var\_map.t
val print\_fd\_solution: (fd\_var\_id, Numbers.t)\ Fd\_var\_map.t 	o unit
```

7.7 Module Non_linear_solving

```
76
                                                              CHAPTER 7. LIBRARY INTEGER_SOLVER
val from_abstract_formula:
  Abstract_constraint.formula → non_linear_constraint list
(non_linear_solve M C) returns a solution of the constraints C in the domain 0..M for each variable. Raises
exception Not_found if no solution exists.
val time_limit : float ref
val non_linear_solve :
    Numbers.t \rightarrow non_linear_constraint list
       → (Finite_domains.fd_var_id, Numbers.t) Finite_domains.Fd_var_map.t
temporaire
val translate_constraints:
    (Finite_domains.fd_var_id, Numbers.t) Finite_domains.Fd_var_map.t \rightarrow
       non_linear_constraint list \rightarrow
         Finite_domains.fd_constraint list ×
         (Finite_domains.fd_var_id, Numbers.t) Finite_domains.Fd_var_map.t
7.8
       Module Poly_lexer
exception Invalid_char of string
val token: Lexing.lexbuf → Poly_parser.token
       Lexer Poly_lexer
7.9
  open Poly_parser
  exception Invalid_char of string
rule token = parse
  [[' ''\t''\n'] { token lexbuf }
  | "and" | "/\" \{ AND \}
   "or" | "\\/" { OR }
   "not" | "~" { NOT }
   "exists" { EXISTS }
   "forall" { FORALL }
   "true" { TRUE }
```

"false" { FALSE }

| ';' { SEMICOLON } |',' { COMMA } | '+' { PLUS } |'/' { DIV }

' ' ' { VERTICALBAR }

| "="|">"|">="|"<"|"<="|"<>"

|'-' { MINUS } '.' { MULT } | ' ^' { EXP }

"implies" | "->" { IMPLIES } "equiv" | "<->" { *EQUIV* }

['a'-'z''A'-'Z']['a'-'z''A'-'Z''0'-'9''_''\'''@''~']*

{ IDENT(Lexing.lexeme lexbuf) }

```
{ COMP(Lexing.lexeme lexbuf) }
| ['0'-'9']<sup>+</sup> { INT (Numbers.from_string (Lexing.lexeme lexbuf)) }
| '(' { PARGAUCHE }
| ')' { PARDROITE }
| _ { raise (Invalid_char (Lexing.lexeme lexbuf)) }
| eof { EOF }
```

7.10 Parser Poly_parser

Header

open Abstract_constraint

Token declarations

```
%token < string > IDENT
%token PARGAUCHE PARDROITE SEMICOLON COMMA EOF
%token TRUE FALSE AND OR NOT IMPLIES EQUIV EXISTS FORALL
%token PLUS MINUS EXP MULT DIV VERTICALBAR
%token < string > COMP
%token < Numbers.t > INT
%start constraint_entry
%type < Abstract_constraint.formula > constraint_entry
%start expr
%type < Abstract_constraint.expr> expr
%nonassoc EXISTS FORALL
%right IMPLIES EQUIV
%left OR
%left AND
%nonassoc NOT
%left PLUS MINUS
%left MULT DIV
%nonassoc UMINUS
%right EXP
```

Grammar rules

```
constraint_entry ::=
| formula EOF { $1 }
```

```
formula ::=
formula AND formula
    { conj $1 $3 }
formula OR formula
    { disj $1 $3 }
NOT formula
    \{ Neg(\$2) \}
formula IMPLIES formula
    { Implies($1,$3) }
formula EQUIV formula
    { Equiv($1,$3) }
EXISTS id_list COMMA formula
    { Exists ($2,$4) }
FORALL id_list COMMA formula
    { Forall($2,$4) }
expr COMP expr
    \{ Comp(\$1,\$2,\$3) \}
expr VERTICALBAR expr
    { Comp($1," | ",$3) }
expr COMP expr COMP expr
    { conj (Comp(\$1,\$2,\$3)) (Comp(\$3,\$4,\$5)) }
expr COMP expr COMP expr COMP expr
    { conj (conj (Comp(\$1,\$2,\$3)) (Comp(\$3,\$4,\$5))) (Comp(\$5,\$6,\$7)) }
PARGAUCHE formula PARDROITE
    { $2 }
TRUE
    { True }
FALSE
    { False }
id\_list ::=
IDENT
    { [$1] }
IDENT id_list
    { $1::$2 }
expr ::=
IDENT
    { Var($1) }
INT
    { Cte($1) }
PARGAUCHE expr PARDROITE
    { $2 }
expr PLUS expr
    { Plus($1,$3) }
expr MINUS expr
    { Sub($1,$3) }
MINUS expr %prec UMINUS
    { Minus($2) }
expr MULT expr
    { Mult($1,$3) }
expr DIV expr
    { Quotient($1,$3) }
expr EXP INT
```

```
{ try
    power $1 (Numbers.to_int $3)
    with
        Invalid_argument("Numbers.to_int") →
        failwith "Exponent_too_large"
}
```

7.11 Module *Poly_syntax*

exception Syntax_error of string

val constraint_of_string : string \rightarrow Abstract_constraint.formula

 $val\ expr_of_string\ :\ string\ \to\ Abstract_constraint.expr$

Chapter 8

Library words

This library provides all definitions and syntactic methods for string rewriting.

8.1 Module String_rewriting

This module defines functions for performing string rewriting of words.

8.1.1 String rewriting systems

A string rewriting system is defined by a set of pairs of words (left-hand side, right-hand side), represented by a list

```
type 'symbol rewrite_rule = ('symbol Words.word × 'symbol Words.word)
```

```
type 'symbol srs = ('symbol Words.word × 'symbol Words.word) list
```

(normalize S w) returns the normal form of w w.r.t the SRS S by the rightmost strategy. Warning! If the righmost reduction of w is infinite, this function loops.

```
val normalize:
```

```
'symbol srs \rightarrow 'symbol Words.word \rightarrow 'symbol Words.word
```

8.1.2 Efficient normalization

The complexity of the former normalization function increases linearly in the number of rules. The latter increases linearly in the maximal length of the left-hand sides of rules.

compiled_srs is an abstract data type that allows efficient matching (discrimination net)

```
type 'symbol compiled_srs
```

(compiled_normalize S w) returns the normal form of w w.r.t the SRS S by the rightmost strategy. Warning! If the righmost reduction of w is infinite, this function loops.

```
val compiled_normalize:
```

```
'symbol compiled_srs → 'symbol Words.word → 'symbol Words.word
```

(compile_srs S) returns a discrimination net equivalent to S, the be used in the previous function.

```
val compile_srs:
```

```
'symbol srs \rightarrow 'symbol compiled_srs
```

val is_nf_compiled:

```
'symbol compiled_srs \rightarrow 'symbol Words.word \rightarrow bool
```

8.1.3 Printing SRSs

(print sigma srs) prints the SRS srs, its letters being printed according to sigma#string_of_symbol. The letters are separated by spaces. Beware that the printing is done using printing functions of the module Format.

(pretty_print sigma srs) does the same, but factorizing consecutive occurences of the same letter.

```
val print:
'symbol #String_signatures.word_signature → 'symbol srs → unit
val pretty_print:
'symbol #String_signatures.word_signature → 'symbol srs → unit
```

8.2 Module Local_confluence

Tests if the given string rewriting system is locally confluent. Note that the result is not correct if the srs is not inter-reduced.

```
val is_localy_confluent:
'symbol String_rewriting.srs → bool
```

8.3 Module String_signatures

This module defines the class of signatures for words

8.3.1 The parameterized signature type

```
class type ['a] word_signature =
    object
    method string_of_symbol : 'a → string
    end

(from_list 1) builds the finite string signature made from symbols in 1

type symbol_id

class type user_word_signature =
    object
    inherit [symbol_id] word_signature
    method string_of_symbol : symbol_id → string
    (symbol_of_string s) returns the symbol represented by s, raises Not_found if no such symbol exists in the signature
    method symbol_of_string : string → symbol_id
    end

val from_list : string list → user_word_signature

val from_string : string → user_word_signature
```

8.4 Module String_unification

```
Computes the solutions of 11 w1 = w2 12 with w1 and w2 not empty val superpose:

('symbol Words.word × 'symbol Words.word)

→ ('symbol Words.word × 'symbol Words.word)

→ ('symbol Words.word × 'symbol Words.word) list
```

```
val unify: 
 'symbol Words.word \rightarrow 'symbol Words.word \rightarrow 
 (('symbol Words.word \times 'symbol Words.word) \times ('symbol Words.word \times 'symbol Words.word)) list
```

8.5 Module *User_words*

```
(word_of_string sigma s) returns the term over the signature sigma, read in the string s
val word_of_string :
    String_signatures.user_word_signature → string
    → String_signatures.symbol_id Words.word

(srs_of_string sigma s) returns the string rewriting system over the signature sigma, read in the string s
val srs_of_string :
    String_signatures.user_word_signature → string
    → String_signatures.symbol_id String_rewriting.srs

(word_prec_of_string sigma s) returns the precedence over the signature sigma, read in the string s
val word_prec_of_string :
    String_signatures.user_word_signature → string
    → String_signatures.user_word_signature → string
    → String_signatures.symbol_id Orderings_generalities.ordering
```

8.6 Module Word_lexer

val token : Lexing.lexbuf → Word_parser.token

8.7 Lexer Word_lexer

```
rule token = parse
   [' ''\t''\n'] { token lexbuf }
 " (*" { comment lexbuf; token lexbuf }
  ';' { SEMICOLON }
  |',' { COMMA }
  | "->" { ARROW }
  | ' ^ ' { POWER }
  (' (' { LPAR }
  |')' { RPAR }
  | ' <' { LT }
  |'>' \{ GT \}
  |'=' \{ EQ \}
 ['a'-'z''A'-'Z''0'-'9']['a'-'z''A'-'Z''0'-'9''_''\\'']*
 ['+''.''&''*''-''/''!''?''@''~''#']('_'(['a'-'z''A'-'Z''0'-'9']+))*
                      { ident(Lexing.lexeme lexbuf) }
 | eof { EOF }
and comment = parse
   "*)"{ ()}
  " (*" { comment lexbuf; comment lexbuf }
 _ { comment lexbuf }
 eof { raise (Syntax_error "unterminated_comment") }
```

8.8 Parser Word_parser

Header

```
open Words
open Word_syntax
open String_signatures
open Finite_orderings
let word_power = Listutils.power
```

Token declarations

```
%token <a href="mailto:signatures.symbol_id">String_signatures.symbol_id</a> IDENT
%token SEMICOLON ARROW LPAR RPAR COMMA
%token POWER EQ GT LT
%token <a href="mailto:sint">sint</a>> INT
%token EOF
%start word_eof
%start word_eof
%start rule_set_eof
%start rule_set_eof
%type <a href="mailto:string_signatures.symbol_id">String_rewriting.srs</a> rule_set_eof
%start precedence_eof
%start precedence_eof
%type <a href="mailto:string_signatures.symbol_id">String_rewriting.srs</a> rule_set_eof
```

Grammar rules

```
word\_eof ::=
  word EOF { $1 }
word ::=
  /* epsilon */
     { [] }
factor word
    { $1 @ $2 }
factor ::=
  IDENT
    { [$1] }
factor POWER expo
     { word_power $1 $3 }
LPAR word RPAR
     { $2 }
expo ::=
  INT { $1 }
IDENT
           int_of_string(!current_signature#string_of_symbol $1)
         with
                raise (Syntax_error "invalid_exponent")
       }
rule_set_eof ::= rule_set EOF { $1 }
rule\_set ::=
  /* epsilon */
     { [] }
rule
     { [$1] }
rule SEMICOLON rule_set
     { $1::$3 }
rule ::=
  word ARROW word { ($1,$3) }
precedence\_eof ::= precedence\ EOF
    let order = identity_ordering (Pervasives.compare : symbol_id → symbol_id → int)
    let order = List.fold_left add_list order $1
    in compare order
```

```
precedence ::=
    /* epsilon */ { [] }
| ordered_list { [$1] }
| ordered_list COMMA precedence { $1::$3 }

ordered_list ::=
    IDENT { One($1) }
| IDENT EQ ordered_list { Eq($1,$3) }
| IDENT GT ordered_list { Gt($1,$3) }
| IDENT LT ordered_list { Lt($1,$3) }
```

8.9 Module Word_orderings

open Orderings_generalities

```
val length_lex: 'symbol ordering \rightarrow 'symbol Words.word ordering val multi_lex: 'symbol ordering \rightarrow 'symbol Words.word ordering
```

8.10 Module Word_syntax

This module provides the environment for words and string rewrite rules syntactic analysis. exception raised when a syntax error occurs

```
exception Syntax_error of string

current_signature is the signature to be used for parsing symbols

val current_signature : String_signatures.user_word_signature ref
```

8.11 Module Srs_completion

Completion of string-rewriting systems

```
val verbose : int ref

open String_signatures

val complete_srs_std_strategy :
    'symbol #word_signature →
    'symbol Words.word Orderings_generalities.ordering →
    'symbol String_rewriting.srs →
    'symbol String_rewriting.srs

val complete_srs_alphabetic_order_std_strategy :
    symbol_id #word_signature →
    symbol_id String_rewriting.srs →
    symbol_id String_rewriting.srs
```

8.12 Module Words

This module defines the strings (or words) over a string signature

8.12.1 type for words

A word over an alphabet in explicitly represented by a list of elements of this alphabet.

```
type 'symbol word = 'symbol list val length : 'symbol word \rightarrow int
```

8.12.2 Printing words

(print sigma w) prints the word w, its letters being printed according to sigma#string_of_symbol. The letters are separated by spaces. Beware that the printing is done using printing functions of the module Format.

(pretty_print sigma w) does the same, but factorizing consecutive occurences of the same letter.

```
val print:
```

```
'symbol #String_signatures.word_signature → 'symbol word → unit val pretty_print :
'symbol #String_signatures.word_signature → 'symbol word → unit
```

8.13 Module Parameterized_signatures

This module defines the class of parameters.

8.13.1 The parameters type

```
val verbose: int ref
val debug: ?f: (string \rightarrow unit) \rightarrow int \rightarrow string \rightarrow unit
class type parameter\_c =
object
    method parameters: Linear\_constraints.var\_env
    method print: unit \rightarrow unit
end
```

This module defines the class of parameterized signatures for parameterized words.

8.13.2 The parameterized signature type

```
The type of elements in the signature. For example a_{i+n}|_{i}i \leq n is represented by \{sig\_symbol = "a"; sig\_index = ["i\_+\_n"]; sig\_constr = "i\_<=\_n"\} type element = \{sig\_symbol : string; sig\_index : Linear\_constraints.expr list; sig\_constr : Linear\_constraints.formula; sig\_env : Linear\_constraints.var\_env \} The general signature class. class type parameterized_signature = object val psig : \{string,element\} Hashtbl.t val parameters : parameter\_c
```

decoration_of_symbol f returns the index of the full description of the symbol f and its associated constraint.

```
method decoration_of_symbol : string \rightarrow element method to_list : unit \rightarrow element list method parameters : parameter_c method print : unit \rightarrow unit method print_element : element \rightarrow unit end
```

8.14 Module *User_parameterized_signatures*

This module defines the class of parameterized signatures for parameterized words

8.14.1 The user parameterized signature type

```
class type user_parameterized_signature =
  object
     inherit Parameterized_signatures.parameterized_signature
  end
Instantiates a parameterized signature into a word signature.
val instantiate_signature:
  user_parameterized_signature →
     (string \rightarrow Numbers.t) \rightarrow
       String_signatures.user_word_signature
from_string p s returns the signature with parameters p and definition s.
val from_string:
  Parameterized_signatures.parameter_c \rightarrow
     string \rightarrow
        user_parameterized_signature
val parameters_from_string:
  string \rightarrow
     Parameterized_signatures.parameter_c
```

8.15 Parser Parameterized_signatures_parser

Header

```
open Parameterized_signatures
open Parameterized_signatures_syntax
```

Token declarations

```
%token <string> FORMULA
%token SEMICOLON
%token PIPE
%token <string> IDENT
%token EOF
%token POWER FP LPAR RPAR ARROW
```

```
%start signature_eof
%type <Parameterized_signatures_syntax.signature> signature_eof
%start cword_eof
%type <Parameterized_signatures_syntax.constrained_word> cword_eof
%start rules_eof
%type <Parameterized_signatures_syntax.rules> rules_eof
```

Grammar rules

{ \$1::\$2 }

```
This part deals with signatures
signature\_eof ::=
  EOF {[]}
  signature EOF {$1}
signature ::=
  | elt {[$1]}
   elt SEMICOLON {[$1]}
  elt SEMICOLON signature {$1::$3}
elt ::=
  IDENT expr_l
       {$1,$2,Abstract_constraint.True}
  IDENT expr_l PIPE constr
       {$1,$2,$4}
This part deals with words.
cword\_eof ::=
  cword\ EOF\ \{\$1\}
cword ::=
  word
       { ($1, Abstract_constraint.True) }
  word PIPE constr
       { ($1,$3) }
word ::=
  /* epsilon */
    { [] }
  factor word
    { $1::$2 }
  simple_word word_no_simple
    { Simple($1) :: $2 }
word_no_simple ::=
  /* epsilon */
    { [] }
  factor word
```

```
factor ::=
  letter POWER expr
       \{ Exp([\$1],\$3) \}
  LPAR simple_word RPAR POWER expr
       \{ Exp(\$2,\$5) \}
  FP FORMULA expr expr LPAR simple_word RPAR
       { Product($2,$3,$4,$6) }
  FP FORMULA expr expr letter
       { Product($2,$3,$4,[$5]) }
simple\_word ::=
  letter
       { [$1] }
  simple_word letter
       { $1@[$2] }
letter ::=
  IDENT expr_l
       { ($1,$2) }
This part deals with rules.
rules\_eof ::=
  rules EOF
       { $1 }
rules ::=
  /* epsilon */
       { [] }
  rule
       { [$1] }
  rule SEMICOLON rules
       { $1::$3 }
rule ::=
  word ARROW word PIPE constr
       { ($1,$3,$5) }
  word ARROW word
       { ($1,$3, Abstract_constraint.True) }
This part deals with formulae and expressions.
expr_l ::=
  /* epsilon */
       { [] }
  expr expr_l
       { $1::$2 }
expr ::=
  | FORMULA {Poly_syntax.expr_of_string $1}
```

```
constr ::=
  | FORMULA {Poly_syntax.constraint_of_string $1}
```

8.16 Lexer Parameterized_signatures_lexer

```
open Parameterized_signatures_parser
  open Parameterized_signatures_syntax
  let current_formula = Buffer.create 17
rule token = parse
    [' ''\t''\n''\r'] { token lexbuf }
  "(*" { comment lexbuf; token lexbuf }
  | ';' { SEMICOLON }
  | ' | ' { PIPE }
  '^' { POWER }
  |"fp" { FP }
  | ' (' { LPAR }
  |')' { RPAR }
  | "->" { ARROW }
  ' {' { Buffer.clear current_formula;
                           FORMULA (formula lexbuf) }
  ['a'-'z''A'-'Z']['a'-'z''A'-'Z''0'-'9''_''\'''@''~']*
                         { IDENT(Lexing.lexeme lexbuf) }
  [ = { raise (Syntax_error ("invalid_char_'" ^ (Lexing.lexeme lexbuf) ^ "' ")) }
 | eof { EOF }
and comment = parse
    "*)" { () }
  "(*" { comment lexbuf; comment lexbuf }
  | _ { comment lexbuf }
  | eof { raise (Syntax_error "unterminated_comment") }
and formula = parse
  | " } " { Buffer.contents current_formula }
  |_ { Buffer.add_string
                             current_formula (Lexing.lexeme lexbuf);
                           formula lexbuf }
 | eof { raise (Syntax_error "formulae_must_end_with_}") }
```

8.17 Module Parameterized_signatures_syntax

```
exception raised when a syntax error occurs.

exception Syntax_error of string

type expr = Abstract_constraint.expr

type expr_l = expr list

type letter = string × expr_l

type signature_element = string × expr_l × Abstract_constraint.formula
```

```
type signature = signature_element list
type simple_word = letter list
type factor =
    | Simple of simple_word
    | Exp of simple_word × expr
    | Product of string × expr × expr × simple_word
type word = factor list
type constrained_word = word × Abstract_constraint.formula
type rule = word × word × Abstract_constraint.formula
type rules = rule list
```

Chapter 9

Library matching

This is a matching library, to be documented by Evelyne.

9.1 Module Lazy_list

```
type 'a frozen =
   | Freeze of (unit \rightarrow 'a)
   | Val of 'a
and 'a llist =
   Nil
   Cons of 'a cell
and 'a cell =
          head: 'a;
          mutable tail: 'a llist frozen;
val hdl: 'a llist \rightarrow 'a
val mapl: (a \rightarrow b) \rightarrow a llist \rightarrow b llist
val map : (a \rightarrow b) \rightarrow a  list \rightarrow b  llist
val map_without_repetition : (a \rightarrow b) \rightarrow a \text{ list } b \rightarrow b \text{ llist}
val map2_without_repetition : (a \rightarrow b \times b) \rightarrow a \text{ list } b + b \text{ llist}
val map12_without_repetition : (a \rightarrow b \times (b \text{ option})) \rightarrow a \text{ list } \rightarrow b \text{ llist}
val lazy_append : 'a llist \rightarrow 'a llist frozen \rightarrow 'a llist
val from_list : 'a list \rightarrow 'a llist
val to_list : 'a llist \rightarrow 'a list
```

9.2 Module Lazy_controle

```
exception No_solution exception Not_appliable type 'pb disjunction = 'pb Lazy_list.llist val orelse : (`pb \rightarrow `pb \ disjunction") \rightarrow (`pb \rightarrow `pb \ disjunction") \rightarrow `pb \rightarrow `pb \ disjunction" val repeat : (`pb \rightarrow `pb \ disjunction") \rightarrow `pb \rightarrow `pb \ disjunction"
```

Chapter 10

Library terms

This library provides all definitions and syntactic methods for symbols, signatures, terms and rewrite rules.

10.1 Module Signatures

This module defines the class of first-order signatures, allowing commutative or associative-commutative symbols.

10.1.1 The signature class type

class type ['a] parseable_signature =

inherit ['a] signature

method symbol_of_string : string \rightarrow 'a

object

end

The signature class type is a very general definition of a signature: it is a arbitrary set (even infinite) equipped with an arity function and some others. The set is modelized here by a type parameter.

```
(arity f) returns the arity of the symbol f.
    (is ac f) returns true if f is an associative-commutative symbol.
    (is_commutative f) returns true if f is a commutative (but not associative) symbol.
    (is_free f) returns true if f is neither commutative nor associative-commutative.
    contains_ac_symbols is true there is at least one AC symbol an that signature.
   contains_only_free_symbols is true there all symbols are free.
    (string_of_symbol f) must return a printable representation of the symbol f.
    (symbol_fix f) returns a concrete value that tells if f must printed, in a term, as a prefix symbol, an infix symbol
(like + in x + y, or a postfix symbol (like ! in n!). The default fix value is infix for AC symbols are prefix for others.
type symbol_fix = Prefix | Infix | Postfix | Default
class type ['a] signature =
       object
          method arity: 'a \rightarrow int
          method is_ac : 'a \rightarrow bool
          method is_commutative : 'a \rightarrow bool
          method is_free: 'a \rightarrow bool
          method contains_ac_symbols: bool
          method contains_only_free_symbols: bool
          method string_of_symbol : 'a \rightarrow string
          method symbol_fix : 'a \rightarrow symbol_fix
       end
(symbol_of_string s) returns the symbol whose name is s. Raises exception Not_found is no symbol corresponds.
```

For example, one may define the infinite signature made of the natural bers constant by saying : class nat_signature object inas herit [int] signature method arity f = 0 method is_ac f = f alse method is_commutative f = f alse method is_free f = true method contains_ac_symbols = false method contains_only_free_symbols = true method $string_of_symbol\ f = string_of_int\ f$ method $symbol_fix\ f = Prefix$ end

10.1.2 Finite symbol sets and maps

```
polymorphic set and map module for symbols, compared by the CAML polymorphic comparaison function. module SymbolOrd: Ordered\_types.OrderedPolyType with type 'a \ t = 'a module SymbolSet: Ordered\_sets.OrderedSet with type 'a \ elt = 'a module SymbolMap: Ordered\_maps.OrderedMap with type 'a \ key = 'a class ['a] default: ['a] parseable\_signature
```

10.2 Module Signature_syntax

```
type symbol\_theory = Free \mid Ac \mid Commutative exception Syntax\_error of string
```

val canonical_renaming: 'symbol term list \rightarrow 'symbol t

10.3 Module Substitution

```
open Signatures
open Variables
open Gen_terms
type 'symbol t = (unit, 'symbol term) VarMap.t
val print:
   'symbol #signature → Variables.user_variables
     \rightarrow 'symbol t \rightarrow unit
(apply_term h t \sigma) computes \tau and uses the hashconsing table of terms h
val apply_term : 'symbol term \rightarrow 'symbol term
val apply_sorted_term:
   'symbol #signature \rightarrow 'symbol term \rightarrow
     'symbol t \rightarrow 'symbol term
(merge_subst subst1 subst2) merges substitutions subst1 and subst2, that is put them together verifying that if
subst1 binds a variable x to a term t_1 and subst2 binds the same variable to t_2, then t_1 = t_2
    Raises Conflict if not.
   This is standard substitution merging: equality of t_1 and t_2 is performed assuming that all symbols are free.
exception Conflict
val merge\_substs: 'symbol t \rightarrow 'symbol t \rightarrow 'symbol t
val substitute : 'symbol \#signature \rightarrow 'symbol t \rightarrow 'symbol term \rightarrow 'symbol term
val apply_subst_to_eqs:
   'symbol \#signature \rightarrow 'symbol t \rightarrow
     ('symbol term \times 'symbol term) list \rightarrow ('symbol term \times 'symbol term) list
```

10.4 Module Rewrite_rules

```
open Signatures open Gen_terms
```

Rewrite rules are used mainly in two different contexts:

- in the termination tool, for finding an ordering which ensures the termination of a system of rewrite rules,
- and for rewriting terms.

t is the type of rewriting rules used in the termination tool, defined by the lefthand side, the righthand side, and optionally the lhs and the rhs of the AC-extension.

```
type 'symbol t =
    {
        Ihs : 'symbol term;
        rhs : 'symbol term;
        ext : ('symbol term × 'symbol term) option;
}
```

exception Rule_with_a_var_lhs

(make_rule sigma 1 r) builds a rewrite rule over signature sigma with lefthand side 1 and righthand side r. Determines whether the rule needs to be AC-extended; in this case the hashconsing table h is used to build the terms of the extension.

May raise the exception Rule_with_a_var_lhs when this is the case!

```
val make_rule :
    'symbol #signature → 'symbol term → 'symbol term → 'symbol t

val print_rewrite_rule :
    'symbol #Signatures.signature → Variables.user_variables → 'symbol t → unit

val print_rewrite_rule_set :
    'symbol #Signatures.signature → Variables.user_variables → 'symbol t list → unit

val latex_print_rewrite_rule_set :
    out_channel → 'symbol #Signatures.signature → Variables.user_variables → 'symbol t list → unit
```

10.5 Module Term_lexer

```
exception Invalid_char of string
exception Unterminated_comment
val token: Lexing.lexbuf → Term_parser.token
```

10.6 Module Signature_lexer

```
exception Invalid\_char of string val token: Lexing.lexbuf 	o Signature\_parser.token
```

10.7 Lexer Term_lexer

```
open Term_parser
open Term_syntax
open Signatures
```

```
exception Invalid_char of string
  exception Unterminated_comment
  let ident s =
    try
      match !parse_id s with
        | Postfix \rightarrow POSTFIX\_IDENT s
         Infix \rightarrow INFIX\_IDENT s
         Prefix \rightarrow PREFIX\_IDENT s
        | Default \rightarrow PREFIX\_IDENT s
    with
        Not_found →
          try
            let x = !current\_variables #var\_of\_string s
            in VAR_IDENT x
          with
               Not_found →
                 raise (Syntax_error ("undefined_identifier_"^s))
let letters = ['a'-'z''A'-'Z''0'-'9''\_''']^+
let symbols = ['^'+''.''&''*''-''/''!''?''@''~''#''|'':''%''$''<''=''>']+
rule token = parse
  [' ''\r''\t''\n'] { token lexbuf }
  |',' { COMMA }
  | "->" { ARROW }
  | ' (' { OPENPAR }
  ')' { CLOSEPAR }
  \ ' <' \ LT \
  | '>' { GT }
  '=' \{ EQ \}
  "<=" { LE }
  ">=" { GE }
  "<>" { NE }
  "mul" { MUL }
  | (letters | symbols) ('_' (letters | symbols))*
                          { ident(Lexing.lexeme lexbuf) }
  "(*" { comment lexbuf; token lexbuf }
 | { raise (Invalid_char (Lexing.lexeme lexbuf)) }
  eof { EOF }
and comment = parse
  | "*) " { () }
  " (*" { comment lexbuf; comment lexbuf }
  | _ { comment lexbuf }
  | eof { raise Unterminated_comment }
        Lexer Signature_lexer
10.8
{
```

```
open Signature_parser
```

exception Invalid_char of string

```
hash table of keywords any number is appropriate in place of 17, a prime number is better
```

```
let keywords\_table = let kt = ((Hashtbl.create 17) : (string,token) Hashtbl.t)
  in List.iter
        (function (s,k) \rightarrow Hashtbl.add kt s k)
          ("constant", KW_CONSTANT);
          ("unary", KW_UNARY);
          ("binary", KW_BINARY);
          ("infix", KW_INFIX);
          ("prefix", KW_PREFIX);
          ("postfix", KW_POSTFIX);
          ("commutative", KW_{-}C);
          ("AC", KW\_AC)
       ];
    kt
  let keyword\_or\_ident s =
    try (Hashtbl.find keywords_table s)
    with Not\_found \rightarrow IDENT s
}
let letters = ['a'-'z''A'-'z''0'-'9''\_''']^+
let symbols = ['^''+''.''&''*''-''/''!''?''@''~''#''|'':''&''$''\''-''>']+
rule token = parse
    [' ''\r''\t''\n'] { token lexbuf }
  |',' { COMMA }
  |':' { COLON }
  ';' { SEMICOLON }
  ['0'-'9']+ { INT(Lexing.lexeme lexbuf) }
  ['a'-'z''A'-'Z'] { keyword_or_ident(Lexing.lexeme lexbuf) }
  | (letters | symbols) ( '_' (letters | symbols))*
                           { IDENT(Lexing.lexeme lexbuf) }
  "(*" { comment lexbuf; token lexbuf }
  | _ { raise (Invalid_char (Lexing.lexeme lexbuf)) }
  | eof { EOF }
and comment = parse
  | "*) " { () }
  "(*" { comment lexbuf; comment lexbuf }
  [ comment lexbuf }
```

10.9 Module Term_syntax

exception Syntax_error of string

10.10 Module Hierarchical_signatures

→ 'a Rpo.status_function

```
type hsymbol
class type hierarchical_signature =
       object
          inherit [hsymbol] Signatures.parseable_signature
          method is_defined_here: hsymbol \rightarrow bool
        end
val extend_signature:
  hierarchical_signature list \rightarrow
     \#User\_signatures.user\_signature \rightarrow
     hierarchical_signature
            Module User_terms
10.11
(term_of_string sigma t) returns the term over the signature sigma, read in the string s
val term_of_string:
   'a #Signatures.parseable_signature \rightarrow Variables.user_variables \rightarrow
     string \rightarrow 'a Gen\_terms.term
val equation_of_string:
   'a #Signatures.parseable_signature \rightarrow Variables.user_variables \rightarrow
          string \rightarrow ('a Gen\_terms.term \times 'a Gen\_terms.term)
(equation_set_of_string sigma t) returns the set of equations over the signature sigma, read in the string s
val equation_set_of_string:
   'a #Signatures.parseable_signature \rightarrow Variables.user_variables \rightarrow
           string \rightarrow ('a \ Gen\_terms.term \times 'a \ Gen\_terms.term) \ list
(rule_set_of_string sigma t) returns the set of rewrite rules over the signature sigma, read in the string s
val rule_set_of_string:
   'a #Signatures.parseable_signature \rightarrow Variables.user_variables \rightarrow
           string \rightarrow ('a Gen\_terms.term \times 'a Gen\_terms.term) list
(prec_of_string sigma s) returns the precedence over the signature sigma, read in the string s
val prec_of_string:
   'a \#Signatures.parseable\_signature \rightarrow string
     → 'a Orderings_generalities.ordering
val order_constraint_of_string:
   'a #Signatures.parseable_signature \rightarrow Variables.user_variables \rightarrow
     string \rightarrow 'a \ Order\_constraints.formula
(status_of_string sigma s) returns the function status (for RPO-like term orderings) over the signature sigma, read
in the string s (raises the exception Invalid_status whenever an AC function symbol is not given a multiset status).
exception Invalid_status of string
val status_of_string :
   'a #Signatures.parseable_signature → string
```

10.12 Module Hierarchical_trs

extend imported sigma vars rules makes a new HTRS from the list imported of known HTRS, by adding the new signature sigma and the new rules syntactically given in the string rules, those being defined on sigma and the signatures of the imported htrs.

If a defined symbol in new rules rules is not defined in sigma then an exception Overriden is thrown.

```
val extend:

htrs list → #User_signatures.user_signature →

Variables.user_variables → string → htrs

val assume_ce:

htrs → unit

val clear_ce:

htrs → unit

exception Overriden of string
```

10.13 Module Sorted_signatures

```
open Sorts open User\_signatures open Signatures open Signatures Signature Signatu
```

10.14 Module Variables

This module provides an abstract data type for first-order variables. In an abstract point of view, a set of variables is any set equipped with total order (an equality is enough in theory, but we require a total order because we want to provide efficient implementation of finite sets of variables and finite maps indexed by variables. Finally a set of variables have to be infinite: it is always possible to find a variable that do not belong to a given finite set.

Variables have no readable representation by default. If you need a set of variables where some variables have a "name", for user interaction purpose, you can use the class *user_variables*. *var_id* is the abstract type for a variable.

string_of_var_id x displays a variable identifier in a raw way. The result is not supposed to be parse again. Mainly for debugging purpose. See user_variables class below for a better way of printing variable ids.

type var_id

val $string_of_var_id : var_id \rightarrow string$

val compare_var : $var_id \rightarrow var_id \rightarrow int$

val $leftify_var : var_id \rightarrow var_id$

val rightify_var : $var_id \rightarrow var_id$

VarOrd is a module that provides a total order on variables.

module VarOrd: $Ordered_types.OrderedType$ with type $t = var_id$

VarSet is a module that provides finite sets of variables.

module VarSet: Ordered_sets.OrderedSet with type 'a elt = var_id

VarMap is a module that provides finite maps indexed by variables.

module VarMap: Inttagmap.IntTagMapModule with type 'a key = var_id

(fresh_variables n) returns a list of n variables. Here, the word "fresh" does not means anything else that these are n distinct variables. Use the next function if you need to obtain a variable distinct from some others.

val fresh_variables : int \rightarrow var_id list

(var_outside_set s) returns a variable that do not belong to the set s.

val $var_outside_set$: unit $VarSet.t \rightarrow var_id$

val shift_variable : var_id → var_id → var_id

val $max_variable : var_id \rightarrow var_id \rightarrow var_id$

val max_var_of_set: unit VarSet.t → var_id

val min_var : var_id

(init_for_unif s) initializes the set of variables for unification. The intended meaning of s is the set of variables occurring in the initial unification problem.

```
val init\_for\_unif: unit VarSet.t \rightarrow unit
```

 $fresh_var_for_unif$ () generates a variable at each call. In a session starting by a call to $(init_for_unif\ s)$, these variables are pairwise distinct, and do not occur in s.

```
val fresh_var_for_unif : unit \rightarrow var_id
```

```
val print_unif_var : var_id \rightarrow unit
```

(user_variables 1) returns an object that provides two functions for converting a variable into a readable name and vice-versa. 1 is a list of strings that give a list of name that you would like to use. Example:

```
let my_vars = new Variables.from_string "x y z"
```

exception Syntax_error of string

(var_of_string s) returns the variable whose name is s. Raises exception Not_found is no variable corresponds.

10.15. MODULE SORTS 103

```
class type user_variables =
object
  method string_of_var : var_id → string
  method var_of_string : string → var_id
  end
val split : string → string list
val from_list : string list → user_variables
val from_string : string → user_variables
val default : user_variables
val hash : var_id → int
module Default :
  sig
  val string_of_var : var_id → string
  val var_of_string : string → var_id
  end
```

10.15 Module Sorts

```
type sort_id

type sort_table

class type sort =
    object
    val sort_table : sort_table
    method string_of_sort : sort_id → string
    end

new from_list 1 returns a class sort whose sort names are the strings in the list 1. Example :
new from_list ["nat";"int"]

class from_list : string list → sort

new from_string s returns a class sort obtained by the syntactical analysis of string s. Example :
new from_string "nat__int"

class from_string : string → sort
```

Chapter 11

Library Term Orderings

This library is for all various definitions of term orderings: polynomial interpretations, path orderings; and combination of orderings to build orderings from others: lexicographic combinations, recursive program schemes.

11.1 Module Poly_interp

```
open Orderings_generalities
open Signatures
open Variables
open Gen_terms
Polynomials used in polynomial orderings are multivariate polynomials with arbitrary size integers as coefficients
module\ NumRing\ :\ Polynomials.RingType\ with\ type\ coef\ =\ Numbers.t
module IntOrd:
  Ordered_types.OrderedType
  with
    type t = int
module IntSet:
  Ordered\_sets.OrderedSet with type 'a elt = int
module\ IntMap\ :\ Ordered\_maps.OrderedMap\ with\ type\ 'a\ key\ =\ int
module IntPolynomials:
  Polynomials.PolynomialType
    module Base_ring = NumRing
  and
    module Var\_set = IntSet
  and
    module Var\_map = IntMap
  and
    type variable = int
val current_signature : User_signatures.user_signature ref
current_signature is a reference onto the signature used for parsing the polynomial interpretations.
val current_poly_vars : string list ref
current_poly_vars is a reference onto the list of variables occurring in a polynomial interpretation (used in parsing).
type 'a t = (a, IntPolynomials.poly) SymbolMap.t
```

```
val print_int_polynomial : IntPolynomials.poly \rightarrow unit
val print : 'a #signature \rightarrow 'a t \rightarrow unit
val latex_print :
  out_channel \rightarrow 'a #Signatures.signature \rightarrow 'a t \rightarrow unit
(build_var_interp_of_term t) builds a one-to-one map from the (term) variables of t to integers
    (complete_var_interp_of_term old_interp t) builds a one-to-one map from the (term) variables of t to integers
variables by completing the old interpretations old_interp
val complete_var_interp_of_term :
  (var\_id,int) \ VarMap.t \rightarrow int \rightarrow `symbol \ term \rightarrow (var\_id,int) \ VarMap.t \times int
val build_var_interp_of_term : 'symbol term \rightarrow (var_id_int) VarMap.t \times int
(check_poly_interp sigma f p) checks whether the polynomial p is a valid interpretation for the symbol f. Raise
a Failure exception if not.
    Verifications are:
    • the dimension of p must be the arity of f;
    • p must be positive or null;
    • p must be increasing in each variable, that is each derivatives must be positive;
    • if f is commutative, p has to be symmetric;
    • if f is AC, f has to be of the form axy + b(x + y) + c where b^2 = b + ac.
val check_poly_interp:
   'symbol #signature \rightarrow 'symbol \rightarrow IntPolynomials.poly \rightarrow unit
(mu_translate mu P) returns the translation of P(X_1,...,X_n) by \mu that is P(X_1+\mu,...,X_n+\mu)-\mu.
val mu_translate : int \rightarrow IntPolynomials.poly \rightarrow IntPolynomials.poly
(poly_ord sigma P) is the ordering on sigma-term generated by P, that is:
ocwbegindcode s > t \setminus mbox\{ iff \} I(s) > I(t) \setminus where
ocwbegindcode I(f(t_{-1},...,t_{-n})) = P(f)(I(t_{-1}),...,I(t_{-n})) \setminus
val poly_ord:
   'symbol #signature
  \rightarrow ('symbol \rightarrow IntPolynomials.poly) \rightarrow 'symbol term ordering
(poly_times ()) returns the pair (user time, system time) spent in poly, in seconds
val poly_times : unit \rightarrow float \times float
module type S =
sig
  type symbol
  type term
  val o: term ordering
module Make (T: Term_algebra.TermAlgebra)
  (P: sig \ val \ symb\_interp: T.symbol \rightarrow IntPolynomials.poly \ end):
  (S \text{ with type } symbol = T.symbol
      and type term = T.term)
```

11.2 Module Term_orderings

```
open Orderings\_generalities open Orderings\_generalities open Orderings of Orderings ordering Orderings of Orderings ordering Orderings ordering
```

11.3 Module Poly_ordering

```
open Orderings
open Poly

ordre d'evaluation sur les polynomes
   (compare_poly P Q) retourne
   Equivalent si pour tt x1... xn P(x1..xn)=Q(x1..xn) Greater si pour tt x1... xn P(x1..xn)>Q(x1..xn) Less_than
si pour tt x1... xn P(x1..xn)<Q(x1..xn) Greater_or_equivalent si pour tt x1... xn P(x1..xn)>=Q(x1..xn)
Less_or_equivalent si pour tt x1... xn P(x1..xn)>=Q(x1..xn) Uncomparable sinon
```

bien sur c'est imcomplet : en fait la specification est :

si (compare_poly P Q) retourne

Equivalent alors pour tt x1.. xn P(x1..xn)=Q(x1..xn) Greater alors pour tt x1.. xn P(x1..xn)>Q(x1..xn) Less_than alors pour tt x1.. xn P(x1..xn)<Q(x1..xn) Greater_or_equivalent alors pour tt x1.. xn P(x1..xn)>=Q(x1..xn) Uncomparable alors on ne sait rien

val compare_poly : polynome \rightarrow polynome \rightarrow comparison_result

11.4 Module Rpo

```
open Orderings_generalities
open Gen_terms

module type S = sig
   type term
   val o : term ordering
end

type rpo_status = | Multiset
   | Lr_lexico
   | Rl_lexico
```

```
type 'symbol status_function = 'symbol \rightarrow rpo_status
```

The functor Make(T)(PS) builds a module S which contains the ordering o over the terms of the term algebra module T as the RPO built with the precedence prec and status stat provided by the module PS.

It is assumed that all the symbols of the module T. Signature are free. Otherwise, use Acrpo. Make instead.

Warning! It is assumed that the status are valid with respect to precedence, that is whenever f and g are equivalent w.r.t. prec then they have the same status, and if it is a lexicographic status, then f and g have the same arity. Generally speaking, instead of using RPO with equivalent symbols in the precedence, it is better to apply first a RPS to replace symbols that are supposed to be equivalent by a representant of them. Moreover this allows to replace a symbol by a symbol with smaller arity.

However, a partial check is made when this function is called by the toplevel.

```
module Make
```

```
(T: Term_algebra.TermAlgebra)
(PS:
    sig
    type symbol = T.symbol
    val prec: symbol ordering
    val status: symbol status_function
    end):
(S with type term = T.term)
```

11.5 Module Acrpo

```
open Orderings_generalities
open Rpo
val rpo_time : float ref
```

The functor $Make\ (T)\ (PS)$ builds a module S which contains the ordering o over the terms of the module T} as the ACRPO built with the precedence [prec] and status [stat] provided by the module [PS].

```
module Make
```

```
(T: Term_algebra.TermAlgebra)
(PS:
    sig
    type symbol = T.symbol
    val prec: symbol ordering
    val status: symbol status_function
    end):
(Rpo.S with type term = T.term)
```

Chapter 12

Library theories

This library provides ...

12.1 Module Axioms

```
module type S =
sig
  type symbol
  type term
  type side = Left | Right | Middle
  type axiom =
       Associativity of symbol
      Unit of side \times symbol \times symbol
      Inverse of side \times symbol \times symbol \times symbol
     Absorb of side \times symbol \times symbol
      Nilpotency of symbol \times symbol \times int \times int
      Idempotency of symbol \times int \times int
      Distributivity of side \times symbol \times symbol
      Add_multiply of symbol × symbol
      Pseudo_associativity of side × symbol
      Associator of symbol \times symbol \times symbol \times symbol
      Commutator4 of symbol \times symbol \times symbol \times symbol
      Commutator3 of symbol \times symbol \times symbol
      Ternary_associativity of symbol
      Ternary_projection of int \times int \times int \times symbol
      Ternary_inverse of side \times symbol \times symbol
      Ternary_single_axiom of symbol × symbol
      Pixley\_def of symbol \times symbol \times symbol \times symbol
      Pixley_1 of symbol \times symbol \times symbol
      Pixley_2 of symbol \times symbol \times symbol
      Pixley_3 of symbol \times symbol \times symbol
      BOO_11 of symbol \times symbol
      BOO_13 of symbol \times symbol
      BOO\_inverse of symbol \times symbol \times symbol
      Majority_1 of symbol × symbol
      Majority_2 of symbol × symbol
      Majority_3 of symbol \times symbol
      Self_dual_distributivity of symbol × symbol
      CL_def_axiom of symbol × symbol
```

```
|B_axiom\ of\ symbol\ 	imes\ symbol
      C_axiom of symbol \times symbol
      K_{axiom of symbol} \times symbol
      H_axiom of symbol \times symbol
      O_axiom of symbol \times symbol
      N_axiom of symbol \times symbol
      N1_axiom of symbol \times symbol
      S-axiom of symbol \times symbol
      Q-axiom of symbol \times symbol
      Q1_axiom of symbol \times symbol
      T_axiom of symbol \times symbol
      W_axiom of symbol \times symbol
      W1_axiom of symbol \times symbol
      W2_axiom of symbol \times symbol
      SB_property of symbol \times symbol \times symbol \times symbol
      Abstraction of symbol \times symbol \times symbol
      Wail of symbol × symbol
      Waj2 of symbol × symbol
      Waj3 of symbol
      Waj4 of symbol × symbol × symbol
      Robbins of symbol \times symbol
      Join_compl of symbol × symbol
      Meet\_compl of symbol \times symbol \times symbol
     |Cn\_axiom\ of\ int\ 	imes\ symbol
  val print_two_symbols : symbol \rightarrow symbol \rightarrow unit
  val print_three_symbols : symbol \rightarrow symbol \rightarrow symbol \rightarrow unit
  val print_four_symbols : symbol \rightarrow symbol \rightarrow symbol \rightarrow symbol \rightarrow unit
  val print_five_symbols : symbol \rightarrow symbol \rightarrow symbol \rightarrow symbol \rightarrow symbol \rightarrow unit
  val print_list_of_symbols : symbol list \rightarrow unit
  val print_axiom : axiom \rightarrow unit
  module AxiomSet: (Ordered_sets.OrderedSet with type 'a elt = axiom)
  val recognize_axiom : term \rightarrow term \rightarrow axiom
end
module Make:
  functor (T: Term\_algebra.TermAlgebra) \rightarrow
     (S \text{ with type } symbol = T.symbol
         and type term = T.term)
```

12.2 Module Standard_matching

```
open Gen_terms
```

(matching pattern subject) returns the most general filter of subject over pattern. Raises No_match if no match is found

This is standard matching: all symbols are assumed to be free.

```
exception No_match
```

```
val matching: 'symbol term \rightarrow 'symbol term \rightarrow 'symbol Substitution.t
```

12.3 Module Unif_index

```
open Term_algebra
```

```
module type S = \text{sig} type term type term type term type term val term compile_for_unification : term and type term term
```

12.4 Module Controle

```
exception No_solution exception No_solution exception Not_appliable type 'problem disjunction = 'problem list val orelse : 
('problem \rightarrow 'problem\_disjunction) \rightarrow ('problem \rightarrow 'problem\_disjunction) \\ \rightarrow 'problem \rightarrow 'problem\_disjunction
val repeat : 
('problem \rightarrow 'problem \ disjunction) \rightarrow 'problem \rightarrow 'problem \ disjunction
```

12.5 Module Theory

There 3 kinds of unifications in CiME:

- the PLAIN unification is the usual unification modulo;
- the AC_COMPLETE unification provides a representation of the unifiers modulo C and AC by a set of unifiers modulo all the current theories (see [?] for more details);
- the AC unification provides a complete set of unifiers modulo C and AC, the others axioms of the theory being ignored.

```
type unif_kind = PLAIN | AC_COMPLETE | AC_ONLY
type 'symbol elem_theory =
    | Empty of 'symbol option
    | C of 'symbol
    | AC of 'symbol
    | ACU of 'symbol × 'symbol
    | ACI of 'symbol × 'symbol
    | AG of 'symbol × 'symbol × 'symbol
    | AGUN of 'symbol × 'symbol × int
    | BR of 'symbol × 'symbol × 'symbol × 'symbol
    exception Syntax_error of string
module type S =
sig
    type symbol
val unif_type : unif_kind ref
```

```
type elem_theory =
    | Empty of symbol option
     C of symbol
     AC of symbol
     ACU of symbol \times symbol
     ACI of symbol
     |AG 	ext{ of } symbol 	imes symbol 	imes symbol
     ACUN of symbol \times symbol \times int
    |BR \text{ of } symbol \times symbol \times symbol \times symbol
  exception No_theory
  type unif_elem_theory = symbol option \times elem_theory
  module TheorySet:
    Ordered_sets.OrderedSet with type 'a elt = elem_theory
  module UnifTheorySet:
    Ordered_sets.OrderedSet with type 'a elt = unif_elem_theory
  val additive_symbol_of_theory : elem_theory → symbol
  val unit_symbol_of_theory : elem_theory → symbol
  val minus_symbol_of_theory : elem_theory → symbol
  val elem_theory_from_unif_elem_theory : unif_elem_theory → elem_theory
  val string_of_elem_theory : elem_theory → string
  val string_of_unif_elem_theory : unif_elem_theory → string
  val print_theory : 'a TheorySet.t \rightarrow unit
  val theory_check : elem_theory → elem_theory
end
module Make:
  functor (T: Term\_algebra.TermAlgebra) \rightarrow
    (S \text{ with type } symbol = T.symbol)
module String\_theory : (S \text{ with type } symbol = string)
```

12.6 Module Variable_abstraction

```
open Theory
module type S =
sig
  type symbol
  type unif_elem_theory
  type term
  val purify_list_of_equations:
     unif_kind \rightarrow (symbol \rightarrow unif_elem_theory) \rightarrow (term \times term) list
        \rightarrow (term \times term) list
end
module Make
  (T : Term_algebra.TermAlgebra)
  (Th : Theory.S \text{ with type } symbol = T.symbol) :
  (S \text{ with type } symbol = T.symbol
      and type unif_elem_theory = Th.unif_elem_theory
      and type term = T.term)
```

12.7 Module Problems

```
open Variables
open Theory
module type S =
sig
  type symbol
  type elem_theory
  type unif_elem_theory
  type term
  module UnifElemThMap:
     (Ordered_maps.OrderedMap with type 'a key = unif_elem_theory)
  type status = Unsolved | Merged | Marked | Solved
  type mark = No_mark | Erasable | Permanent of elem_theory
  type elem_pb =
       {
          key : unif_elem_theory;
          status: status;
          size: int option;
          elem_th : elem_theory;
          inst_variables : (unit, term) VarMap.t;
          marked_variables : (unit, mark) VarMap.t;
          edges : (var_id \times var_id) list;
          equations : (term \times term) list
       }
  type problem = \{
     unif_kind : unif_kind;
     global_status : status;
     find\_th : symbol \rightarrow unif\_elem\_theory;
     vars_for_eqe : unit VarSet.t;
     first_vars : unit VarSet.t;
     var_var : (unit, var_id) VarMap.t;
     elem_pbs : (unit, elem_pb) UnifElemThMap.t;
     solved\_part : (term \times term) list;
  val print_elem_pb : elem_pb \rightarrow unit
  val print\_problem : problem \rightarrow unit
  val replace_a_var_by_a_var_in_an_eq:
     var\_id \rightarrow term \rightarrow term \times term \rightarrow term \times term
  val add_an_equation_between_variables:
     problem \rightarrow var_id \rightarrow var_id \rightarrow problem
  Initialisation of the unification problem: (init th 1) builds the data structucture for solving 1 modulo the equa-
tional theory th, where I is a list of equations between terms.
     unif_kind \rightarrow (symbol \rightarrow unif_elem_theory) \rightarrow (term \times term) list \rightarrow problem
  val insert_solved_elem_pbs:
     problem \rightarrow unif_elem_theory \rightarrow (term \times term) list list \rightarrow problem list
  val existential_quantifiers_elimination : problem \rightarrow (term \times term) list
end
```

module type Solve =

```
type term
  type elem_pb
  solve elem_pb computes a unifier for elem_pb modulo an elementary unification theory, that is, solves the
equations of the problem and checks that the constraints of marks are satisfied.
  val solve:
     unif_kind \rightarrow unit VarSet.t \rightarrow elem_pb \rightarrow (term \times term) list list
end
module Make
  (T : Term\_algebra.TermAlgebra)
  (Th : Theory.S \text{ with type } symbol = T.symbol)
  (V : Variable\_abstraction.S \text{ with type } symbol = T.symbol
                                     and type unif_elem_theory = Th.unif_elem_theory
                                     and type term = T.term):
  (S \text{ with type } symbol = T.symbol
      and type elem_theory = Th.elem_theory
      and type unif_elem_theory = Th.unif_elem_theory
      and type term = T.term)
```

12.8 Module Theory_syntax

```
val is_defined : (string \rightarrow User_signatures.symbol_id) ref val current_signature : User_signatures.user_signature ref
```

12.9 Parser Theory_parser

Header

```
open User_signatures open Term_algebra
```

Token declarations

```
%token <User_signatures.symbol_id> IDENT
%token <int> INT
%token KW_ACU KW_ACI KW_AG KW_ACUN KW_BR
%token COMMA SEMICOLON OPENPAR CLOSEPAR
%token EOF
%start theory
%type <User_signatures.symbol_id Theory.elem_theory list> theory
```

Grammar rules

```
theory ::=
   EOF
     { [] }
  decl
       [$1]
  decl SEMICOLON theory
       $1::$3
decl ::=
 acu { $1 }
| aci { $1 }
ag { $1 }
acun { $1 }
| br { $1 }
acu ::=
KW_ACU OPENPAR IDENT COMMA IDENT CLOSEPAR
    Theory. ACU ($3,$5)
aci ::=
KW_ACI OPENPAR IDENT CLOSEPAR
    Theory.ACI($3)
KW_AG OPENPAR IDENT COMMA IDENT COMMA IDENT CLOSEPAR
    Theory. AG(\$3,\$5,\$7)
  }
acun ::=
KW_ACUN OPENPAR IDENT COMMA IDENT COMMA INT CLOSEPAR
    Theory. ACUN ($3, $5, $7)
  }
br ::=
KW_BR OPENPAR IDENT COMMA IDENT COMMA IDENT COMMA IDENT CLOSEPAR
    Theory.BR ($3,$5,$7,$9)
```

12.10 Lexer Theory_lexer

```
open Theory_parser
  open Theory_syntax
  open Signatures
  exception Invalid_char of string
  let ident s =
    try
      let f = !is\_defined s in IDENT f
        Not_found \rightarrow
           try
             INT(int_of_string s)
          with \rightarrow raise (Theory.Syntax\_error ("undefined_identifier_"^s))
}
rule token = parse
    [' '' \t'' \n'] \{ token lexbuf \}
  |',' { COMMA }
  | ';' { SEMICOLON }
  | ' (' { OPENPAR }
  ')' { CLOSEPAR }
  | "ACU" { KW_ACU }
  "ACI" { KW_ACI }
  "AG" { KW_AG }
   "ACUN" { KW_ACUN }
  | "BR" { KW_BR }
  ['a'-'z''A'-'Z''0'-'9']['a'-'z''A'-'Z''0'-'9''_''\'']*
  ['^''+''.''&''*''-''/''!''?''@''~''#']('_'(['a'-'z''A'-'Z''0'-'9']+))*
                           { ident(Lexing.lexeme lexbuf) }
  _ { raise (Invalid_char (Lexing.lexeme lexbuf)) }
  eof { EOF }
```

12.11 Module User_theory

```
val from_string: User\_signatures.user\_signature \rightarrow string \rightarrow User\_signatures.symbol\_id Theory.elem\_theory list
```

12.12 Module *Oc*

```
open Variables

module type S = sig

type term

type cycle = Cycle of var\_id list

|No\_cycle of var\_id list
```

(occur_check_without_var_var list_of_equations) check that there is no cycle in the occurrence graph generated by the list_of_equations, that is returns

- No_cycle list_of_vars when there is no cycle in the graph; in this case, the (total) ordering of the variables in the list_of_vars is compatible with the (partial) ordering induced by the graph,
- Cycle list_of_vars when there is a cycle in the graph going through the nodes list_of_vars.

It is assumed that *list_of_equations* does not contains any equation between variables.

```
val occur_check_without_var_var : (term \times term) list \rightarrow cycle
```

A call to (instanciate_when_no_cycle list_of_vars list_of_equations) assumes that

- list_of_equations is a list of pair of terms
- there is no cycle in the occur-check graph generated by the *list_of_equations*,
- and that *list_of_vars* provides a total ordering compatible with the occur-check graph.

This function takes *list_of_equations* as a DAG-solved form, in particular all the equations are of the form *variable* = *term*, and it returns an equivalent solved form.

```
val instanciate_when_no_cycle : var_id\ list \rightarrow (term \times term)\ list \rightarrow (term \times term)\ list
```

A call to (occur_check sign hct list_of_eqs_var_var list_of_equations) assumes that

- list_of_equations is a list of pair of terms built over the signature sign thanks to the hashconsing table hct,
- list_of_eqs_var_var contains only equations between variables,
- list_of_equations contains only equations between a variable and a non-variable term,
- and the **Coalesce** rule does not apply on the union of these two sets of equations.

It returns either a failure when there is a cycle in the occur-check graph or a list of equations which is a solved form for the union of *list_of_eqs_var_var* and *list_of_equations*.

```
\begin{array}{l} \text{val } \textit{occur\_check} \ : \\ \textit{(unit, var\_id)} \ \textit{VarMap.t} \ \rightarrow \ (\textit{term} \ \times \ \textit{term}) \ \textit{list} \ \rightarrow \\ \textit{(term} \ \times \ \textit{term}) \ \textit{list} \end{array} end \begin{array}{l} \text{module } \textit{Make} \ : \\ \textit{functor} \ (\textit{T} \ : \ \textit{Term\_algebra}.\textit{TermAlgebra}) \ \rightarrow \\ \textit{(S with type } \textit{term} \ = \ \textit{T.term}) \end{array}
```

12.13 Module *Unif_free*

```
open Variables  \begin{tabular}{ll} module type $S=$ sig \\ type term \\ type elem\_pb \\ type pure\_elem\_pb = \\ \{ & map\_of\_var\_var: (unit, var\_id) \ VarMap.t; \\ scanned\_equations: (term \times term) \ list; \\ other\_equations: (term \times term) \ list \\ \} \\ \end{tabular}
```

Deletion of trivial equations.

```
val delete: pure_elem_pb → pure_elem_pb

Coalesce (Replacement of a variable by a variable).

val coalesce: pure_elem_pb → pure_elem_pb

Merge for two equations with the same variable left-hand side.

val merge: pure_elem_pb → pure_elem_pb

solve_without_marks sign list_of_equations computes a unifier for list_of_equations modulo the empty equational theory.

val solve_without_marks: (term × term) list → (term × term) list

solve elem_pb computes a unifier for elem_pb modulo the empty equational theory, that is, solves the equations of the problem and checks that the constraints of marks are satisfied.

val solve: elem_pb → (term × term) list list
end

module Make
```

12.14 Module Unif commutative

(T: Term_algebra.TermAlgebra)

(S with type term = T.term)

(P: Problems.S with type term = T.term)(O: Oc.S with type term = T.term):

and type $elem_pb = P.elem_pb$)

```
\begin{array}{ll} \text{module type } S &= \\ \text{sig} \\ \text{type } term \\ \text{type } elem\_pb \end{array}
```

solve elem_pb computes a unifier for elem_pb modulo the commutativity, that is, solves the equations of the problem and checks that the constraints of marks are satisfied.

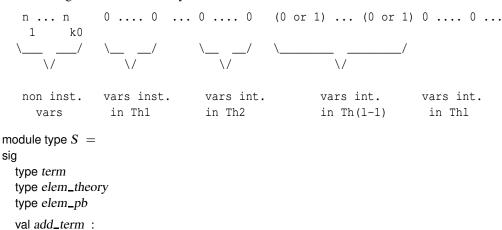
```
val solve : elem\_pb \rightarrow (term \times term) list list end module Make (T: Term\_algebra.TermAlgebra) (P: Problems.S with type term = T.term) (O: Oc.S with type term = T.term) (F: Unif\_free.S with type term = T.term and type elem\_pb = P.elem\_pb) : (S \text{ with type } term = T.term and type elem\_pb = P.elem\_pb)
```

12.15 Module Unif to arith

open Variables

This module provides some functions in order to translate a unification modulo AC (resp. ACU, resp. AG) into a system of linear equations over the non-negative integers (resp. non-negative integers, resp. integers). The variables of the problem are sorted according to the theory (other than the current one) where they are instanciated:

The integer solver returns only solutions of the form



elem_theory \rightarrow (unit, int) VarMap.t \rightarrow int array \rightarrow term \rightarrow unit

unif_to_arith_without_matrix elem_pb returns a pair made of

- a map giving the indices corresponding to the variables,
- an array of variables corresponding to the inverse map of the above map
- an array [|k1;k2;...|] of indices as described above.

Remark: this function should be called only on problems with AC-like theories, that is AC, ACU, ACI, ACUN, and AG.

```
val unif_to_arith_without_matrix:
  elem\_pb \rightarrow (unit, int) \ VarMap.t \times var\_id \ array \times int \ array
unif_to_arith elem_pb returns a quadruple made of
```

- a map giving the indices corresponding to the variables,
- an array of variables corresponding to the inverse map of the above map
- an array [|k1;k2;...|] of indices as described above.
- a matrix of non-negative integers corresponding to the translation of the equations of the elementary problem.

Remark: this function should be called only on problems with AC-like theories, that is AC, ACU, ACI, ACUN, and AG.

```
val unif_to_arith:
    elem_pb → (unit, int) VarMap.t × var_id array × int array × int array array
end
module Make
  (T : Term_algebra.TermAlgebra)
  (Th : Theory.S \text{ with type } symbol = T.symbol)
  (P: Problems.S \text{ with type } term = T.term
                    and type elem_theory = Th.elem_theory) :
  (S \text{ with type } term = T.term)
      and type elem_theory = Th.elem_theory
      and type elem\_pb = P.elem\_pb)
```

12.16 Module Hullot_bin_trees

```
module type BINARY\_TREE = sig
   type t
   val arbre\_binaire : int \rightarrow (t \rightarrow bool) \rightarrow (t \rightarrow bool) \rightarrow t \ list
   end

module Small\_binary\_tree :
   BINARY\_TREE with type t = int

module Large\_binary\_tree :
   BINARY\_TREE with type t = int array
```

12.17 Module Arith_to_unif

```
open Bit_field
open Variables
open Theory
module type S = sig

type elem\_theory
type term
type elem\_pb
```

cache est une fonction qui prend un vecteur de vecteurs d'entiers positifs (solutions diophantiennes d'un systeme), et qui retourne un vecteur de naturels codant des vecteurs de bits, tel que chaque 1 correspond a un entier non-nul de l'entree, et chaque 0 correspond a un 0, le tout transpose, pour avoir directement acces aux colonnes associees a chaque variable.

```
val pcache: int array array \rightarrow int array val psmall_enough: int \rightarrow int \rightarrow int array \rightarrow int \rightarrow bool val pgreat_enough: int \rightarrow int \rightarrow int array \rightarrow int \rightarrow bool val gcache: int array array \rightarrow Large_bit_field.t array val gsmall_enough: int \rightarrow int \rightarrow Large_bit_field.t array \rightarrow Large_bit_field.t \rightarrow bool val ggreat_enough: int \rightarrow int \rightarrow Large_bit_field.t array \rightarrow Large_bit_field.t \rightarrow bool
```

combinaison_lineaire + vect_var vect_sols vect_nouv_var vect_car retourne une liste d'equations de la forme var (de vect_var) = un terme de symbole de tete +, dont les variables sont celles de vect_nouv_var avec les coefficients apparaissant dans vect_sols.

```
val linear_combination:
unif_kind → elem_theory → unit VarSet.t →
(unit, int) VarMap.t → int array array → term array → int array →
(term × term) list
```

nettoyer pe_edge vect_var vect_sols enleve de vect_sols les solutions qui vont provoquer un echec pour OC, grace a pe_edge.

```
val clean_solutions:
    elem_pb → (unit, int) VarMap.t → int array array →
    int array array

(classify elem_pb array_of_vars map_var_int v_type vect_sols) returns a pair
(vect_homogeneous_sols, vect_heterogeneous_sols) where
```

- vect_homogeneous_sols contains the solutions from the array vect_sols which are equal to 0 over the marked variables,
- vect_heterogeneous_sols contains the other solutions of vect_sols

Remark: The solutions of vect_sols creating a cycle of size 2 are removed in sorted_vect_sols.

```
val classify:
elem_pb → var_id array → (unit, int) VarMap.t → int array →
int array array → (int array array) × (int array array)
```

(sum_of_columns matrix) returns an array containing the sum of the columns of the argument matrix.

```
val sum_of_columns : int array array → int array
```

(associated_var_with_sol sum_of_sols array_of_vars v_type sol) returns

- Some c when the value of the solution sol is equal to 1 for the component corresponding to the marked variable c (when there are several such marked variables, the function returns the first one encountered).
- Some x when the value of the solution sol is equal to 1 for the component corresponding to the variable x, and there is no other such variable.
- None otherwise.

```
val associated_var_with_sol :
    int array → var_id array → int array → int array → var_id option

(associated_marked_var_with_sol array_of_vars v_type sol) returns
```

- Some c when the value of the solution sol is equal to 1 for the component corresponding to the marked variable c (when there are several such marked variables, the function returns the first one encountered).
- None otherwise.

```
val associated_marked_var_with_sol : var_id \ array \rightarrow int \ array \rightarrow int \ array \rightarrow var_id \ option
```

(generate_vect_char_cst nb_var nb_true_var vect_sols_cst) generates the list of subsets of non-homogeneous Diophantine solutions (encoded as 0-1 vectors) which are usefull in order to build unifiers for "arithmetic" theories which possess a unit, that is ACU, ACUN and AG.

12.18 Module *Unif_ac*

12.19 Module *Unif_acu*

```
open Variables
open Theory
module type S =
  type term
  type elem_pb
  val generate_vect_char :
     unif_kind \rightarrow int \rightarrow int \rightarrow (unit, int) VarMap.t \rightarrow int array array \rightarrow
        int array array \rightarrow int array list
  val filter_non_unit_vars:
     unit VarSet.t \rightarrow int \rightarrow (unit, int) VarMap.t \rightarrow (unit, int) VarMap.t
  val solve:
     unif_kind \rightarrow unit VarSet.t \rightarrow elem_pb \rightarrow (term \times term) list list
  val is_ac_unifiable : unit VarSet.t \rightarrow elem\_pb \rightarrow bool
end
module Make:
  functor (T: Term\_algebra.TermAlgebra) \rightarrow
     functor (Th: Theory.S with type symbol = T.symbol) \rightarrow
        functor (P : Problems.S \text{ with type } term = T.term
                                          and type elem\_theory = Th.elem\_theory) \rightarrow
           functor (U: Unif\_to\_arith.S with type term = T.term
                                                    and type elem_theory = Th.elem_theory
                                                    and type elem\_pb = P.elem\_pb) \rightarrow
              functor (A: Arith\_to\_unif.S with type elem\_theory = Th.elem\_theory
                                                     and type term = T.term
                                                     and type elem\_pb = P.elem\_pb) \rightarrow
                 (S \text{ with type } term = T.term)
                     and type elem\_pb = P.elem\_pb)
```

12.20 Module Mark_acu

```
\begin{array}{lll} \mathsf{module} \ \mathit{Make} : \\ \mathsf{functor} \ (T : \mathit{Term\_algebra}. \mathit{TermAlgebra}) \ \to \\ \mathsf{functor} \ (\mathit{Th} : \mathit{Theory.S} \ \mathsf{with} \ \mathsf{type} \ \mathit{symbol} = \mathit{T.symbol}) \ \to \\ \mathsf{functor} \ (P : \mathit{Problems.S} \ \mathsf{with} \ \mathsf{type} \ \mathit{term} = \mathit{T.term} \\ & \mathsf{and} \ \mathsf{type} \ \mathit{elem\_theory} = \mathit{Th.elem\_theory}) \ \to \\ (\mathit{Problems.Solve} \ \mathsf{with} \ \mathsf{type} \ \mathit{term} = \mathit{T.term} \\ & \mathsf{and} \ \mathsf{type} \ \mathit{elem\_pb} = \mathit{P.elem\_pb}) \end{array}
```

12.21 Module Merge_acu

12.22 Module *Unif_aci*

12.23 Module Unif_ag_acun

```
and type elem_theory = Th.elem_theory
and type elem_pb = P.elem_pb)

(A: Arith_to_unif.S with type elem_theory = Th.elem_theory
and type term = T.term
and type elem_pb = P.elem_pb)

(Acu: Unif_acu.S with type term = T.term
and type elem_pb = P.elem_pb):

(Problems.Solve with type term = T.term
and type elem_pb = P.elem_pb)
```

12.24 Module Unif_bool

```
open Variables  \begin{tabular}{ll} \begin{tabular}{ll} module type $S=$ sig \\ type $term$ & type $elem$_pb \\ & val $solve: elem$_pb $\to (term \times term) list list end \\ \begin{tabular}{ll} module $Make$ & ($T: Term$_algebra.TermAlgebra)$ & ($Th: Theory.S$ with type $symbol = T.symbol)$ & ($P: Problems.S$ with type $term = T.term$ & and type $elem$_theory = Th.elem$_theory): & ($S$ with type $term = T.term$ & and type $elem$_pb = $P.elem$_pb) \\ \end{tabular}
```

12.25 Module Mark

end

```
open Variables

module type S = sig

type unif\_elem\_theory

type mark

type problem

val add\_a\_mark : var\_id \to mark \to unif\_elem\_theory \to problem \to problem

(mark\ pb) applies the Mark\ rule on the unification problem pb and returns

• either the exception Not\_appliable

• either the exception No\_solution

• or a list of marked unification problems.

val mark : problem \to problem\ list

val lazy\_mark : problem \to problem\ Lazy\_list.llist
```

12.26. MODULE CYCLE

```
\begin{array}{l} \operatorname{module} \ \mathit{Make} \ : \\ \operatorname{functor} \ (T : \mathit{Term\_algebra}. \mathit{TermAlgebra}) \ \to \\ \operatorname{functor} \ (\mathit{Th} : \mathit{Theory}. \mathit{S} \ \text{with type } \mathit{symbol} = \mathit{T.symbol}) \ \to \\ \operatorname{functor} \ (P : \mathit{Problems}. \mathit{S} \ \text{with type } \mathit{term} = \mathit{T.term} \\ \operatorname{and type } \mathit{elem\_theory} = \mathit{Th.elem\_theory} \\ \operatorname{and type } \mathit{unif\_elem\_theory} = \mathit{Th.unif\_elem\_theory}) \ \to \\ (\mathit{S} \ \text{with type } \mathit{unif\_elem\_theory} = \mathit{Th.unif\_elem\_theory} \\ \operatorname{and type } \mathit{mark} = \mathit{P.mark} \\ \operatorname{and type } \mathit{problem} = \mathit{P.problem}) \end{array}
```

12.26 Module Cycle

```
module type S =
sig
  type problem
  val cycle: problem \rightarrow problem list
  val lazy_cycle: problem \rightarrow problem Lazy_list.llist
end
module Make:
  functor (T : Term\_algebra.TermAlgebra) \rightarrow
     functor (Th: Theory.S with type symbol = T.symbol) \rightarrow
       functor (P : Problems.S \text{ with type } term = T.term
                                       and type elem_theory = Th.elem_theory
                                       and type unif\_elem\_theory = Th.unif\_elem\_theory) \rightarrow
          functor (M : Mark.S \text{ with type } unif\_elem\_theory = Th.unif\_elem\_theory)
                                    and type mark = P.mark
                                    and type problem = P.problem) \rightarrow
            functor (O: Oc.S with type term = T.term) \rightarrow
                (S \text{ with type } problem = P.problem)
```

12.27 Module E_{-} res

```
module type S =
sig
  type problem
  val general_E_resolution : problem \rightarrow problem list
  val lazy_general_E_resolution : problem \rightarrow problem Lazy_list.llist
module Make:
  functor (T : Term\_algebra. TermAlgebra) \rightarrow
     functor (Th: Theory.S \text{ with type } symbol = T.symbol) \rightarrow
       functor (P : Problems.S \text{ with type } term = T.term
                                       and type elem\_theory = Th.elem\_theory
                                       and type unif\_elem\_theory = Th.unif\_elem\_theory) \rightarrow
          functor (M: Mark.S with type unif_elem_theory = Th.unif_elem_theory
                                    and type mark = P.mark
                                    and type problem = P.problem) \rightarrow
            functor (O: Oc.S with type term = T.term) \rightarrow
               (S \text{ with type } problem = P.problem)
```

module Make

(T : Term_algebra.TermAlgebra) :
(S with type symbol = T.symbol
 and type term = T.term

and type substitution = T.substitution)

12.28 Module Eqe

12.29 Module Unification

```
open Variables
open Theory
val verbose: int ref
val unification_time : float ref
set_of_unifiers unif_k E term1 term2 returns a complete set of unifiers of term1 and term2 modulo the equational
theory E. E may be any combination of some elementary theories among
    • the free theory
    • C

    AC

    • ACU
    • ACI
    AG

    ACUN

    • BR
provided that they are pairwise signature-disjoint.
module type S =
  type symbol
  type term
  type substitution
  type elem_theory
  type unif_elem_theory
  module TheorySet:
     Ordered_sets.OrderedSet with type 'a elt = elem_theory
  val verbose: int ref
  val th_from_user_th:
     User\_signatures.user\_signature \rightarrow User\_signatures.symbol\_id Theory.elem\_theory \rightarrow
        elem_theory
  val set_of_solutions:
     unif_kind \rightarrow (symbol \rightarrow unif_elem_theory) \rightarrow term \rightarrow term \rightarrow substitution list
  val plain_set_of_solutions : unit TheorySet.t \rightarrow term \rightarrow term \rightarrow substitution list
  val display_plain_set_of_solutions : unit TheorySet.t \rightarrow term \rightarrow term \rightarrow unit
  val free_unification : term \rightarrow term \rightarrow substitution
  val ac_unification : term \rightarrow term \rightarrow substitution list
  val first_ac_solution : (term \times term) list \rightarrow (term \times term) list
  val has_an_ac_solution : (term \times term) list \rightarrow bool
  val free_solve_constraints : (term \times term) list \rightarrow (term \times term) list
end
```

12.30 Module Ac_unification

set_of_solutions ...

Chapter 13

Library rewriting

Functions to rewrite a term at top by a rule or a set of rules.

13.1 Module Full_dnet

```
module Data: (Inttagset.IntTagSetModule \ with \ type \ ('a, 'b) \ elt = int \times 'a) module type S= sig  type \ term \\ type \ term \\ type \ rule \\ type \ substitution \\ type \ 'a \ dnet \\ val \ all\_data: 'a \ dnet \ \rightarrow \ (int \times 'a) \ list \\ val \ compile: (term \times 'a) \ list \ \rightarrow \ 'a \ dnet \\ val \ compile\_rules: rule \ list \ \rightarrow \ rule \ dnet
```

add_a_data sign (term, data) dnet builds a new dnet corresponding to the old one dnet where one has added the data data corresponding with the term term.

```
val add_a_data : term \times 'a \rightarrow 'a dnet \rightarrow 'a dnet
```

remove_a_data sign (term, data) dnet builds a new dnet corresponding to the old one dnet where one has removed the data data corresponding with the term term. When data is not indexed by term in dnet, dnet is returned unchanged.

```
val remove_a_data : term \times 'a \rightarrow 'a dnet \rightarrow 'a dnet
```

replace_a_data sign old_data (term, new_data) dnet builds a new dnet corresponding to the old one dnet where one has replaced the data old_data corresponding with the term term by the data new_data. When old_data is not indexed by term in dnet, dnet is returned unchanged.

```
val replace_a_data : 'a \rightarrow term \times 'a \rightarrow 'a dnet \rightarrow 'a dnet type partial_match = | Partial of substitution | Total of substitution | val discriminate : 'a dnet \rightarrow term \rightarrow (partial\_match \times 'a, unit) Data.t val is_encompassed_by : 'a dnet \rightarrow term \rightarrow bool type position = int list
```

```
\label{eq:module PosData} \ : \ (Ordered\_maps.OrderedMap \ \ with type \ 'a \ key = int \times position)  \ type \ 'a \ full\_dnet  \ val \ full\_compile : \ (term \times \ 'data) \ list \to int \times \ 'data \ full\_dnet  \ val \ full\_compile\_rules : \ rule \ list \to int \times rule \ full\_dnet  \ val \ full\_discriminate : \ int \to \ 'data \ full\_dnet \to term \to \\ \ (int \times position, \ (partial\_match \times \ 'data, \ unit) \ Data.t) \ PosData.t  end \ module \ Make  \ (T : \ Term\_algebra.TermAlgebra) : \ (S \ with \ type \ term = \ T.term \\ \ and \ type \ rule = \ T.rule \\ \ and \ type \ substitution = \ T.substitution)
```

13.2 Module Standard_innermost

```
open Gen_terms
```

(innermost_normalize red t) returns the innermost normal form of t w.r.t to the reduction relation red.

red is a function which, given a term t, returns a pair (u, sigma) such that t rewrites to u sigma, or raises Irreducible if t is irreducible.

The substitution will be applied during the normalization process.

```
val innermost_normalize:
'symbol #Signatures.signature →
Variables.user_variables →
('symbol term → 'symbol term × 'symbol Substitution.t) →
'symbol term → 'symbol term
```

 $(force_innermost_normalize\ red\ t)$ does the same as $(innermost_normalize\ red\ t)$, but raises Irreducible if no reduction at all is possible.

exception Irreducible

```
val force_innermost_normalize :
    'symbol #Signatures.signature → Variables.user_variables →
    ('symbol term → 'symbol term × 'symbol Substitution.t) →
    'symbol term → 'symbol term
```

 $safe_innermost_normalize \ n \ red \ t$ does the same but where n is a bound for the number of rewrite steps to apply. If this bound is reached, then this function raises the exception $Unnormalized \ u$ where u is the reduct of t obtained so far

```
exception Unnormalized
```

13.3 Module Innermost

```
\begin{array}{ll} \text{module type } S &= \\ \text{sig} \\ \text{type } \textit{term} \\ \text{type } \textit{substitution} \end{array}
```

(innermost_normalize find memo red t) returns the innermost normal form of t w.r.t to the reduction relation red.

red is a function which, given a term t, returns a pair (u, sigma) such that t rewrites to u sigma, or raises Irreducible if t is irreducible.

The substitution will be applied during the normalization process. find and memo are memoization functions.

```
val innermost_normalize:
```

```
(term \rightarrow term) \rightarrow (term \rightarrow term \rightarrow unit) \rightarrow (term \rightarrow term \times substitution) \rightarrow term \rightarrow term
```

(force_innermost_normalize find memo red t) does the same as (innermost_normalize find memo red t), but raises Irreducible if no reduction at all is possible. find and memo are memoization functions.

exception Irreducible

```
val force_innermost_normalize : (term \rightarrow term) \rightarrow (term \rightarrow term \rightarrow unit) \rightarrow \\ (term \rightarrow term \times substitution) \rightarrow term \rightarrow term
```

safe_innermost_normalize find memo n red t does the same but where n is a bound for the number of rewrite steps to apply. If this bound is reached, then this function raises the exception $Unnormalized\ u$ where u is the reduct of t obtained so far

```
exception Unnormalized
```

```
val safe_innermost_normalize : (term \rightarrow term) \rightarrow (term \rightarrow term \rightarrow unit) \rightarrow int \rightarrow (term \rightarrow term \times substitution) \rightarrow term \rightarrow term val safe_force_innermost_normalize : (term \rightarrow term) \rightarrow (term \rightarrow term \rightarrow unit) \rightarrow int \rightarrow (term \rightarrow term \times substitution) \rightarrow term \rightarrow term end (term \rightarrow term \times substitution) \rightarrow term \rightarrow term end (term \rightarrow term \times substitution) \rightarrow term \rightarrow term and (term \rightarrow term \rightarrow term \rightarrow term \rightarrow term \rightarrow term \rightarrow term) end (term \rightarrow term \rightarrow term \rightarrow term \rightarrow term) \rightarrow term end (term \rightarrow term \rightarrow term \rightarrow term \rightarrow term) \rightarrow term end (term \rightarrow term \rightarrow term \rightarrow term \rightarrow term \rightarrow term) \rightarrow term \rightarrow term end (term \rightarrow term end (term \rightarrow term \rightarrow term
```

Chapter 14

Library completion

This is a completion library, to be documented.

14.1 Module Standard_critical_pairs

```
\begin{array}{l} \text{module type } S = \\ \text{sig} \\ \text{type } \textit{term} \\ \text{type } \textit{rule} \\ \text{val } \textit{critical\_pairs} : \\ (\textit{term} \rightarrow \textit{bool}) \rightarrow (\textit{int} \rightarrow \textit{term} \rightarrow \textit{bool}) \rightarrow \textit{rule} \rightarrow \textit{rule} \rightarrow (\textit{term} \times \textit{term}) \textit{list} \\ \text{val } \textit{self\_critical\_pairs} : \\ (\textit{term} \rightarrow \textit{bool}) \rightarrow (\textit{int} \rightarrow \textit{term} \rightarrow \textit{bool}) \rightarrow \textit{rule} \rightarrow (\textit{term} \times \textit{term}) \textit{list} \\ \text{end} \\ \text{module } \textit{Make} \ (T : \textit{Term\_algebra}.\textit{TermAlgebra}) : \\ (S \text{ with type } \textit{term} = \textit{T.term} \\ & \text{and type } \textit{rule} = \textit{T.rule}) \\ \end{array}
```

14.2 Module Ac_critical_pairs

```
\begin{array}{l} \text{module type } S = \\ \text{sig} \\ \text{type } \text{term} \\ \text{type } \text{substitution} \\ \text{type } \text{rule} \\ \\ \text{val } \text{critical\_pairs } : \\ (\text{term} \times \text{term} \rightarrow \text{substitution list}) \rightarrow (\text{term} \times \text{term} \rightarrow \text{substitution list} \rightarrow \text{unit}) \rightarrow \\ (\text{term} \rightarrow \text{bool}) \rightarrow (\text{int} \rightarrow \text{term} \rightarrow \text{bool}) \rightarrow \text{rule} \rightarrow \text{rule} \rightarrow (\text{term} \times \text{term}) \text{list} \\ \text{val } \text{self\_critical\_pairs } : \\ (\text{term} \times \text{term} \rightarrow \text{substitution list}) \rightarrow (\text{term} \times \text{term} \rightarrow \text{substitution list} \rightarrow \text{unit}) \rightarrow \\ (\text{term} \times \text{term} \rightarrow \text{bool}) \rightarrow (\text{int} \rightarrow \text{term} \rightarrow \text{bool}) \rightarrow \text{rule} \rightarrow (\text{term} \times \text{term}) \text{list} \\ \text{end} \\ \\ \\ \text{module } Make \ (T : Term\_algebra.TermAlgebra) : \\ (S \text{ with type } \text{term} = T.\text{term} \\ \\ \\ \text{and type } \text{substitution} = T.\text{substitution} \\ \\ \\ \text{and type } \text{rule} = T.\text{rule}) \\ \\ \end{array}
```

14.3 Module Abstract_rewriting

This module provides parameterized functions to complete a rewriting system. They are intended to apply as well on terms and on words.

```
well on terms and on words.
   CiME Project - Démons research team - LRI - Université Paris XI
   Id: abstract_rewriting.mli, v1.332003/09/1216:07:05contejeaExp
    *************************
open Variables
open Orderings_generalities
module type S =
  sig
     type t
     type rule
     type compiled_rules
     type compiled_pairs
     type 'a compiled_data
     type index_for_unif
     module Tset: (Ordered_sets.OrderedSet with type 'a elt = t)
     module Ttable : (Hashtbl.S \text{ with type } key = t)
     val my_o: t ordering
     val penalty_of_non_oriented_eq: int
     val smallest_constant : t option
     val a_variable_of_t : t \rightarrow t option
     val variables\_of\_t : t \rightarrow unit VarSet.t
     val equals: t \rightarrow t \rightarrow bool
     val make_rule : int \rightarrow t option \rightarrow t \rightarrow t \rightarrow rule
     val lhs\_of\_rule : rule \rightarrow t
     val rhs\_of\_rule : rule \rightarrow t
     val alt_rhs_of_rule : rule \rightarrow t option
     val is_oriented : rule → bool
     val is_encompassed_by_a_t : unit compiled_data \rightarrow t \rightarrow bool
     val compile : rule list \rightarrow compiled_rules
           (* update_compiled_rules sign compiled_rules to_add to_remove to_replace unchanged compiles the
rules corresponding to the list of rules to_add, unchanged, and the second components of to_replace, provided
that compiled_rules corresponds to the list of rules unchanged, to_remove and the first components of to_replace.
*)
     val update_compiled_rules:
        compiled_rules \rightarrow rule \rightarrow rule list \rightarrow (rule \times rule) list \rightarrow
          rule list \rightarrow compiled_rules
     val compile_pairs : rule list \rightarrow compiled_pairs
     val compile_data : (t \times 'a) list \rightarrow 'a compiled_data
     val compile_for_unification : rule list \rightarrow index_for_unif
             normalize find memo r t returns the normal form of t. force_normalize does the same but raises
Irreducible if already in normal form.
     val normalize:
        (t \rightarrow t) \rightarrow (t \rightarrow t \rightarrow unit) \rightarrow compiled\_rules \rightarrow t \rightarrow t
     exception Irreducible
     val force_normalize:
        (t \rightarrow t) \rightarrow (t \rightarrow t \rightarrow unit) \rightarrow compiled\_rules \rightarrow t \rightarrow t
```

self_critical_pairs find cr already_computed_cp r returns the list of critical pairs of r into itself. The

standard elimination criterion is used, the unifier of the left hand sides has to be irreducible wrt the rules *cr. find* is a function which tries to find the normal form of a term wrt *cr.*

```
val self_critical_pairs: (t \to t) \to \text{compiled\_rules} \to (\text{int} \to \text{int} \to \text{bool}) \to \text{rule} \to (t \times t) \text{ list}
```

critical_pairs find cr already_computed_cp r1 r2 returns the list of critical pairs between r1 and r2. r1 and r2 are supposed to be different, use self_critical_pairs for computing critical pairs of a rule into itself. The standard elimination criterion is used, the unifier of the left hand sides has to be irreducible wrt the rules cr. find is a function which tries to find the normal form of a term wrt cr.

```
val critical_pairs : (t \to t) \to compiled\_rules \to (int \to int \to bool) \to rule \to rule \to (t \times t) \ list val init_narrow : t \to t \to ((t \times t) \times (t \times t) \times (t \times t)) val is_a_nar_eq : t \to t \to bool val pair_size : t \to t \to int \times int val is_an_instance : (t \times t) \to (t \times t) \to bool
```

is_encompassed_by_a_pair p1 p2 t1 t2 returns true whenever there exists a context C[], and a substitution sigma such that $t1 = C[p1 \ sigma]$ and $t2 = C[p2 \ sigma]$.

```
val is_encompassed_by_a_pair : compiled_pairs \rightarrow t \rightarrow t \rightarrow bool
```

regular_pair c t1 t2 returns the term t2' such that all variables in t2 which do not occur in the term t1 have been substituted by the constant term c. It raises Not_found whenever t2' is identical to t2.

```
val regular_pair : t \rightarrow t \rightarrow t \rightarrow t
```

canonize_pairs 1 returns the list 1 where all pairs of terms have been put in a kind of canonical form with respect to name of variables. This is only for having a better printing of rules, it can be the identity function.

```
val canonize_pairs : (t \times t) list \rightarrow (t \times t) list
     val print_t : t \rightarrow unit
     val print_equation_set : (t \times t) list \rightarrow unit
     val print\_rewrite\_rule : rule \rightarrow unit
     val is_a_c_equality : t \rightarrow t \rightarrow bool
     val is_an_ac_equality : t \rightarrow t \rightarrow bool
     val nb\_shared\_symbols: t \rightarrow t \rightarrow int
     val print_stats : unit \rightarrow unit
module MakeWordRewriting:
  functor
     (SWords:
        sig
          val my_sign: String_signatures.symbol_id String_signatures.word_signature
          val my_o: String_signatures.symbol_id Words.word ordering
        (S \text{ with type } t = String\_signatures.symbol\_id Words.word)
            and type rule = String_signatures.symbol_id String_rewriting.rewrite_rule)
module MakePWordRewriting:
  functor
     (PWords:
       sig
          val my_sign: Parameterized_signatures.parameterized_signature
          val my_o: Parameterized_words.word ordering
        end) \rightarrow
        (S with type t = Parameterized\_words.word
            and type rule = Parameterized_rewriting.rewrite_rule)
```

14.4 Module Confluence

```
open User_signatures
open Variables
open Gen_terms
module type S =
sig
  type rule
  val is_confluent : rule list \rightarrow bool
  val print_all_critical_pairs : rule list \rightarrow unit
module StandardConfluence
  (T : Term_algebra.TermAlgebra) :
  (S \text{ with type } rule = T.rule)
module ACConfluence
  (T : Term_algebra.TermAlgebra) :
  (S \text{ with type } rule = T.rule)
val is_confluent:
  symbol_entry array \rightarrow (symbol_id term \times symbol_id term) list \rightarrow bool
val print_all_critical_pairs:
  symbol_entry array \rightarrow (symbol_id term \times symbol_id term) list \rightarrow unit
```

14.5 Module *Kb*

```
open Gen_terms
open User_signatures
open Orderings_generalities
val coeff_size : int ref
val coeff_age: int ref
val verbose: int ref
type builtin_th
module type KB =
  sig
     type t
     type simple_rule
     exception KB_th_detected of builtin_th \times t \times ((t \times t) list) \times ((t \times t) list)
     val verbose: int ref
     val o: t ordering
     val complete : (t \times t) list \rightarrow (t \times t) list \rightarrow simple_rule list
     val prove_conj_by_comp: (t \times t) list \rightarrow (t \times t) list \rightarrow (t \times t) list \rightarrow bool
  end
module Make (R: Abstract_rewriting.S) (Sco: Best_pair.Score):
  (KB \text{ with type } t = R.t)
       and type simple\_rule = R.rule)
module StandardKBCompletion
  (T : Term_algebra.TermAlgebra)
  (O: sig val my_o: T.term ordering end):
  (KB \text{ with type } t = T.term
         and type simple\_rule = T.rule)
```

14.5. MODULE KB 137

```
module ACCompletion
  (T : Term_algebra.TermAlgebra)
  (O: sig val my_o: T.term ordering end):
  (KB \text{ with type } t = T.term
        and type simple\_rule = T.rule)
module OrderedKBCompletion
  (T : Term_algebra.TermAlgebra)
  (O: sig val my_o: T.term ordering end)
  (Sco: Best_pair.Score):
  (KB \text{ with type } t = T.term
        and type simple\_rule = T.rule)
module OrderedACCompletion
  (T : Term_algebra.TermAlgebra)
  (O: sig val my_o: T.term ordering end)
  (Sco: Best_pair.Score):
  (KB \text{ with type } t = T.term
        and type simple\_rule = T.rule)
module WordCompletion
  (SWords:
    sig
       val my_sign: String_signatures.symbol_id String_signatures.word_signature
       val my_o : String_signatures.symbol_id Words.word ordering
  (KB \text{ with type } t = String\_signatures.symbol\_id Words.word)
       and type simple\_rule =
         String_signatures.symbol_id String_rewriting.rewrite_rule)
module PWordCompletion
  (PWords:
    sig
       val my_sign: Parameterized_signatures.parameterized_signature
       val my_o: Parameterized_words.word ordering
    end):
  (KB with type t = Parameterized\_words.word
       and type simple_rule = Parameterized_rewriting.rewrite_rule)
val complete:
  symbol_entry array \rightarrow symbol_id Term_orderings.t \rightarrow
    (symbol_id term × symbol_id term) list → (symbol_id term × symbol_id term) list
val prove_conjectures:
  symbol_entry array \rightarrow Variables.user_variables \rightarrow
    (symbol\_id\ term\ 	imes\ symbol\_id\ term)\ list\ 	o\ (symbol\_id\ term\ 	imes\ symbol\_id\ term)\ list\ 	o\ bool
```

Chapter 15

Library eq_proof

This library provides ...

15.1 Module Proof

```
\begin{array}{l} \text{module type } P = \\ \text{sig} \\ \text{type } t \\ \text{val } prove\_conj\_without\_strategy : (t \times t) \ list \rightarrow \ (t \times t) \ list \rightarrow \ bool \\ \text{end} \end{array}
```

Chapter 16

Library coq_interface

This library provides ...

Module Trace 16.1

This module defines the trace used when normalizing a term: it basically consists in remembering which and where rules are applied to reduce the term.

```
module type S =
  type rule
  class t: object
     method get: (int \times int \ list) \ list
     method add: rule \rightarrow Positions.position \rightarrow unit
     method to_string : string array → string
     method print : unit \rightarrow unit
  end
end
module Make (T: Term_algebra.TermAlgebra):
  (S \text{ with type } rule = T.rule)
```

Module Traced_rewriting

```
module type S =
sig
  type term
  type substitution
  type rule
  type dnet
  type trace
  (* new_compiled_rewrite_at_top is the same as Standard_rewriting.compiled_rewrite_at_top, except it returns
the whole rewrite rule applied for the reduction instead of just the right member. *)
  val new_compiled_rewrite_at_top : dnet \rightarrow term \rightarrow substitution \times rule
  traced_innermost_normalize sigma red t, with red returning the whole (numbered) rewrite rule, is the same as
```

Innermost_innermost_normalize red't, except that it returns a trace of the normalization as well.

```
val traced_innermost_normalize :
   (term \rightarrow substitution \times rule) \rightarrow term \rightarrow term \times trace
```

```
end
```

```
 \begin{array}{lll} \text{module } \textit{Make} \\ & (T: \textit{Term\_algebra}. \textit{TermAlgebra}) \\ & (D: \textit{Full\_dnet.S} \text{ with type } \textit{term} = \textit{T.term} \\ & & \text{and type } \textit{rule} = \textit{T.rule} \\ & & \text{and type } \textit{substitution} = \textit{T.substitution}) \\ & (\textit{Tr}: \textit{Trace.S} \text{ with type } \textit{rule} = \textit{T.rule}): \\ & (\textit{S} \text{ with type } \textit{term} = \textit{T.term} \\ & \text{and type } \textit{substitution} = \textit{T.substitution} \\ & \text{and type } \textit{dnet} = \textit{T.rule} \textit{D.dnet} \\ & \text{and type } \textit{trace} = \textit{Tr.t}) \\ \end{array}
```

16.3 Module Coq_syntax

Chapter 17

Library Termination

This library contains the needed stuff for either checking the termination of a rewrite system, or even trying to find a termination proof of a rewrite system.

17.1 Module Basic_criterion

(all_decreasing > R) returns true if all rules of R are strictly decreasing w.r.t to >]. If \neg , prints the first rule found that does \neg decrease and returns [false].

```
val all_decreasing :
   Terms.term Orderings.ordering -> Equations.regle list -> bool;;
```

 $(all_decreasing_constraints R)$ returns the list of ordering constraints that should be satisfied for that each rule of R is strictly decreasing.

```
val all_decreasing_constraints:
'symbol Rewrite_rules.t list →
'symbol Termination_constraints.conj_termination_constraints
```

17.2 Module Dependency_pairs_criteria

17.2.1 Simple dependency pairs criterion

 $(simple_dependency_pair_criterion\ R)$ returns the set of ordering existence constraints given by the simple dependency pairs termination criterion on R, which is the following:

- 1. let D be the set of defined symbols of R, that is the set of symbols occurring at root of a lefthand side of a rule of R;
- 2. a dependency pair is any pair of terms (s,t) such that s is the lefthand side of a rule $s \to r$ in R and t is any subterm of r whose root symbol is defined.
- 3. a condition for R terminating is that there is a weakly monotonic ordering \geq such that $l \geq r$ for all rules of R and s > t for all dependency pair of R;
- 4. \geq must be AC-compatible.

```
val compute_defined_symbols:
('symbol Rewrite_rules.t) list →
'symbol Signatures.SymbolSet.t
```

```
val all_rules_weakly_decreasing :
    ('symbol Rewrite_rules.t) list →
        'symbol Termination_constraints.AtomicTerminationConstraintSet.t

val all_dp_strictly_decreasing :
    ('symbol Gen_terms.term × 'symbol Gen_terms.term) list →
        'symbol Termination_constraints.AtomicTerminationConstraintSet.t

val compute_dependency_pairs :
    'symbol Signatures.SymbolSet.t →
        ('symbol Rewrite_rules.t) list →
        ('symbol Gen_terms.term × 'symbol Gen_terms.term) list

val simple_dependency_pair_criterion :
    'symbol #Signatures.signature →
    'symbol Rewrite_rules.t list →
        'symbol Termination_constraints.conj_termination_constraints
```

17.3 Module Generic_polynomials

```
type polyinterptype = Linear | Simple | Simple_Mixed | Quadratic

val use_interp : polyinterptype ref

module GenericPolynomial defines polynomials over the coefficients ring : polynomials of finite domain variables ; and variables indexed by integers.

module Fd_polynomials_ring : Polynomials.RingType

with

type coef = Non_linear_solving.Fd_polynomials.poly

module GenericPolynomials : Polynomials.PolynomialType

with

module Base_ring = Fd_polynomials_ring

and

type variable = int

(print_generic_polynomial p) outputs p to standard output

val print_generic_polynomial : GenericPolynomials.poly → unit
```

 $(print_interp\ i)$ outputs the polynomial interpretation i (that is a map which associates polynomials to symbols) to standard output

```
val print_interp:
'a #Signatures.signature

→ ('a, GenericPolynomials.poly) Signatures.SymbolMap.t → unit
```

(polynomial_constraints_of_termination_constraints sigma i c) returns a pair (interp, c1, c2, c3) where interp is a set of generic polynomial interpretations of symbols in c and c1, c2 and c3 are lists of diophantine constraints over variables of interp such that:

- if c1 then interp is AC-compatible
- if c2 then interp is strictly monotonic
- if c3 then c

sigma is the signature on which constraints are defined, and i is a 'predefined' interpretation for some symbols, that is the function search only for an interpretation interp that extends i. Of course i may be empty.

```
val polynomial_constraints_of_termination_constraints :
  'a #Signatures.signature
  → ('a, GenericPolynomials.poly) Signatures.SymbolMap.t
    → 'a Termination_constraints.conj_termination_constraints
         ('a, GenericPolynomials.poly) Signatures.SymbolMap.t
         × Non_linear_solving.non_linear_constraint list
         × Non_linear_solving.non_linear_constraint list
         × Non_linear_solving.non_linear_constraint list
(specialize_polynomial p s) returns the polynomial obtained by instantiation of finite domain variables in coeffi-
cients of p by their value given by the map s
val specialize_polynomial:
  GenericPolynomials.poly
  → (Finite_domains.fd_var_id, Numbers.t) Finite_domains.Fd_var_map.t
    → GenericPolynomials.poly
val instantiate_interp:
  ('symbol, GenericPolynomials.poly) Signatures.SymbolMap.t \rightarrow
    (Finite_domains.fd_var_id, Numbers.t) Finite_domains.Fd_var_map.t →
       ('symbol, Poly_interp.IntPolynomials.poly) Signatures.SymbolMap.t
(solve_constraint w sigma c) tries to solve the constraint c on signature sigma, using each kind of interpretations
in param_list in order. If w is true, searches for strictly monotonic ordering only.
   Returns the interpretation found, or raises No_proof_found.
param_list is the list of kinds of polynomial interpretation to look for, in order.
                                                                                                      Default
"linear",1;"linear",2;"simple",2
val param_list: (polyinterptype × int) list ref
val solve_constraint:
  bool \rightarrow 'a \# Signatures.signature
    → 'a Termination_constraints.conj_termination_constraints
       → 'a Poly_interp.t
```

17.4 Module Genpoly_lexer

val token : Lexing.lexbuf \rightarrow Genpoly_parser.token

17.5 Lexer Genpoly_lexer

```
{
    open Genpoly_syntax
    open Genpoly_parser
    open Marked_dp_criteria
    exception Syntax_error of string
```

```
let interp s =
    try
       let is_marked =
          String.get s \ 0 = ' \setminus '' \land String.get \ s \ (pred (String.length \ s)) = ' \setminus '
       in
       let s' =
          if is_marked
          then String.sub s 1 (pred (pred (String.length s)))
          else s
       let f = !current\_signature #symbol\_of\_string s
       in
       let g =
          if is_marked
          then Marked f
          else Original f
       in INTERP(g)
     with
          Not_found \rightarrow
            raise (Syntax_error ("undefined_identifier_"^s))
}
rule token = parse
     ['\ '' \ \ \ \ ]^+ \ \{\ \ token\ lexbuf\ \}
  ['a'-'z''A'-'Z']['0'-'9']*
                               { VAR(Lexing.lexeme lexbuf) }
  '[' { symbol lexbuf }
  ';' { SEMICOLON }
  | ' +' { PLUS }
  |'-' { MINUS }
  |'.' { MULT }
  | ' ^' { EXP }
  ['0'-'9'] { INT (Num.num_of_string (Lexing.lexeme lexbuf)) }
  | ' (' { PARGAUCHE }
  |')' { PARDROITE }
  eof { EOF }
and symbol = parse
     [' ''\t''\n']<sup>+</sup> { symbol lexbuf }
  \lceil \lceil \rceil \rceil' \rceil' \lceil \lceil \lceil \rceil \rceil \rceil  let f = interp (Lexing.lexeme lexbuf)
                                 in skip_bracket lexbuf; f }
and skip\_bracket = parse
    [' ''\t''\n']+ { skip_bracket lexbuf }
  |']'{ ()}
```

17.6 Parser Genpoly_parser

Header

```
open Signatures
open Generic_polynomials
```

```
open Genpoly_syntax
open Non_linear_solving
open Finite_domains
open Poly_interp
exception Syntax_error of string
let gen_var s =
  try
    let n = Listutils.index s !current_poly_vars
    in GenericPolynomials.var n
  with
       Not_found \rightarrow
          GenericPolynomials.cte
            (Fd_polynomials.var (fd_var_id_of_string s))
let var s =
  try
    let n = Listutils.index s !current_poly_vars
    in IntPolynomials.var n
  with
       Not\_found \rightarrow raise (Syntax\_error ("undefined\_variable\_"^s))
```

Token declarations

```
%token < user_signatures.symbol_id Marked_dp_criteria.dupl> INTERP
%token PARGAUCHE PARDROITE SEMICOLON EQUAL COMMA EOF
%token PLUS MINUS EXP MULT
%token < Num.num> INT
%start gen_poly_entry
%type < Generic_polynomials.GenericPolynomials.poly> gen_poly_entry
%start poly_interp_entry
%type < (User_signatures.symbol_id Marked_dp_criteria.dupl, Poly_interp.IntPolynomials.poly) Signatures.SymbolMap.t > poly_inter
%left PLUS MINUS
%left MULT
%nonassoc UMINUS
```

Grammar rules

%right EXP

```
gen_poly_entry ::=
  gen_poly EOF { $1 }
```

```
gen\_poly ::=
  VAR { gen_var $1 }
 INT { GenericPolynomials.cte (Fd_polynomials.cte $1) }
 PARGAUCHE gen_poly PARDROITE { $2 }
 gen_poly PLUS gen_poly { GenericPolynomials.add $1 $3 }
 gen_poly MINUS gen_poly { GenericPolynomials.sub $1 $3 }
 MINUS gen_poly %prec UMINUS { GenericPolynomials.minus $2 }
 gen_poly MULT gen_poly { GenericPolynomials.mult $1 $3 }
 gen_poly EXP INT
    { try
         GenericPolynomials.power $1 (Num.int_of_num $3)
      with
         Failure("int_of_big_int") →
           failwith "Exponent_too_large"
    }
poly_interp_entry ::=
  interp EOF { $1 }
interp ::=
  /* epsilon */ { SymbolMap.empty }
symbol_interp EQUAL poly SEMICOLON interp
                              { SymbolMap.add $1 $3 $5 }
symbol_interp ::=
  INTERP { current_poly_vars := []; $1 }
| INTERP PARGAUCHE vars { current_poly_vars := $3; $1 }
vars ::=
  VAR PARDROITE { [$1] }
| VAR COMMA vars { $1::$3 }
poly ::=
  VAR { var $1 }
 INT { IntPolynomials.cte $1 }
 PARGAUCHE poly PARDROITE { $2 }
 poly PLUS poly { IntPolynomials.add $1 $3 }
 poly MINUS poly { IntPolynomials.sub $1 $3 }
 MINUS poly %prec UMINUS { IntPolynomials.minus $2 }
 poly MULT poly { IntPolynomials.mult $1 $3 }
 poly EXP INT
    { try
         IntPolynomials.power $1 (Num.int_of_num $3)
         Failure("int_of_big_int") →
           failwith "Exponent_too_large"
```

17.7 Module Genpoly_syntax

current_signature is the signature to be used for parsing function symbols

val current_signature: User_signatures.user_signature ref
current_poly_vars is the current list of polynomial variables in parsing generic polynomials
val current_poly_vars: string list ref

17.8 Module Marked_dp_criteria

```
open Gen_terms
open Orderings_generalities
(var_multiplicities t) returns an association table from variables of t to their multiplicity, i.e., the number of their
occurrences at depth 1 in t.
val var_multiplicities:
   'symbol term \rightarrow (Variables.var_id,int) Variables.VarMap.t
type 'symbol dupl =
  Original of 'symbol
  Marked of 'symbol
val mark_root_symbol : 'symbol term \rightarrow ('symbol dupl) term
class ['a] marked_signature : ('a \#Signatures.signature) \rightarrow
  object
     method arity: 'a dupl \rightarrow int
     method is\_ac: 'a dupl \rightarrow bool
     method is_commutative : 'a dupl \rightarrow bool
     method is_free: 'a dupl \rightarrow bool
     method contains_ac_symbols: bool
     method contains_only_free_symbols : bool
     method string_of_symbol : 'a dupl \rightarrow string
     method symbol_fix : 'a dupl \rightarrow Signatures.symbol_fix
val signature_with_marks:
   'symbol #Signatures.signature → 'symbol marked_signature
val ac_rhs_rules:
     'a \#Signatures.signature \rightarrow
        'a dupl #Signatures.signature \rightarrow 'a Signatures.SymbolSet.t \rightarrow
           'a Rewrite_rules.t list → 'a dupl Rewrite_rules.t list
val copy_rule:
   'a Rewrite_rules.t \rightarrow 'a dupl Rewrite_rules.t
val remove_markings_rules:
   'a #Signatures.signature \rightarrow
     'a Signatures.SymbolSet.t → 'a dupl Rewrite_rules.t list
val compute_dependency_pairs_with_markings:
  bool \rightarrow
  ('a \# Signatures.signature) \rightarrow
        'a Signatures.SymbolSet.t \rightarrow
           'a Rewrite_rules.t list \rightarrow
             ('a dupl term \times 'a dupl term) list
val dependency_pair_criterion_with_markings:
  bool \rightarrow
  ('symbol #Signatures.signature) \rightarrow
        'symbol Rewrite_rules.t list \rightarrow
          ('symbol dupl) Termination_constraints.conj_termination_constraints
```

17.9 Module Old_basic_criterion

```
    (all_decreasing > R) returns true if all rules of R are strictly decreasing w.r.t to
    ⟩]. If ¬, prints the first rule found that does ¬ decrease and returns [false].
    val all_decreasing:
    Terms.term Orderings.ordering → Equations.regle list → bool
```

17.10 Lexer Poly_lexer

```
open Poly_parser
  open Poly_syntax
  open Generic_polynomials
  open Signatures
  exception Invalid_char of string
  let ident s =
    try
      let is_marked =
        String.get s \ 0 = ' \setminus '' \land String.get \ s \ (pred (String.length \ s)) = ' \setminus ''
      in
      let s' =
        if is_marked
        then String.sub s 1 (pred (pred (String.length s)))
      let f = !current\_signature #symbol\_of\_string s
        if is_marked
        then Marked f
        else Original f
    with
        Not\_found \rightarrow
          Errors.semantical_error ("undefined_identifier_"^s)
}
rule token = parse
    [' \ '' \ t'' \ n'] \{  token lexbuf \}
  |',' { COMMA }
  ';' { SEMICOLON }
  "->" { ARROW }
  ' (' { OPENPAR }
  |')' { CLOSEPAR }
  ['a'-'z''A'-'Z''0'-'9']['a'-'z''A'-'Z''0'-'9'' ''\'']*
  { ident(Lexing.lexeme lexbuf) }
 | _ { raise (Invalid_char (Lexing.lexeme lexbuf)) }
  eof { EOF }
```

17.11 Module Termination_constraints

An atomic termination constraint is either s = t, s > t or s > = t for two terms s and t

```
type constraint_op_type = Equal | Greater | Greater_or_equal
type 'symbol atomic_termination_constraint =
    {
       left: 'symbol term;
       right: 'symbol term;
       constraint_op : constraint_op_type;
a termination constraint is a disjunction of conjonction of atomic constraints.
module AtomicTerminationConstraintSet: Ordered_sets.OrderedSet
  with type 'symbol elt = 'symbol atomic_termination_constraint
type 'symbol conj_termination_constraints =
     'symbol AtomicTerminationConstraintSet.t
val print_atomic_termination_constraint :
  'symbol #Signatures.signature →
      Variables.user_variables → 'symbol atomic_termination_constraint
        \rightarrow unit
val print_conj_termination_constraints:
  'symbol #Signatures.signature →
      Variables.user_variables → 'symbol conj_termination_constraints
(set_of_symbols c) returns the set of symbols occurring in the list of constraints c
val set_of_symbols:
  'symbol conj_termination_constraints \rightarrow 'symbol SymbolSet.t
val conj_of_order_constraints:
  'symbol Order_constraints.formula →
     'symbol conj_termination_constraints
```

17.11.1 Checking constraints

```
val check_conj_constraints :
    'symbol #Signatures.signature →
        Variables.user_variables →
        'symbol conj_termination_constraints →
        'symbol Gen_terms.term Orderings_generalities.ordering → unit

val display_conj_termination_constraints :
    ('symbol #Signatures.signature) →
        Variables.user_variables →
        'symbol conj_termination_constraints
        → unit
```

17.12 Module Minimal_split

A module of graphs of symbols with a special function *scc* which returns strongly connected components of its argument (see module Graph).

```
module Symgraph:
   sig
     type 'symbol elt = 'symbol Signatures.SymbolOrd.t
     and 'symbol t = 'symbol Graph.Make(Signatures.SymbolOrd).t
     exception NodeNotInGraph
     exception NodeAlreadyInGraph
     val empty: 'symbol t
     val add_node : 'symbol t \rightarrow 'symbol elt \rightarrow 'symbol t
     val is_node: 'symbol t \rightarrow 'symbol elt \rightarrow bool
     val add_edge: 'symbol t \rightarrow 'symbol elt \rightarrow 'symbol elt \rightarrow 'symbol t
     val is_edge: 'symbol t \rightarrow 'symbol elt \rightarrow 'symbol elt \rightarrow bool
     val del\_node: 'symbol t \rightarrow 'symbol elt \rightarrow 'symbol t
     val del_edge : 'symbol t \rightarrow 'symbol elt \rightarrow 'symbol elt \rightarrow 'symbol t
     val node_list: 'symbol t \rightarrow 'symbol elt list
     val edge_list: 'symbol t \rightarrow ('symbol elt \times 'symbol elt) list
     val neighbours_list: 'symbol t \rightarrow 'symbol elt \rightarrow 'symbol elt list
     val iter_node : 'symbol t \rightarrow ('symbol elt \rightarrow unit) \rightarrow unit
     val iter_edge: 'symbol t \rightarrow ('symbol elt \rightarrow 'symbol elt \rightarrow unit) \rightarrow unit
     val fold : ('symbol elt \rightarrow 'b \rightarrow 'b) \rightarrow 'b \rightarrow 'symbol t \rightarrow 'b
     val transpose: 'symbol t \rightarrow 'symbol t
     val scc: 'symbol t \rightarrow 'symbol elt list list
   end
val graph_from_a_list:
   'symbol Rewrite_rules.t list →
      'symbol Symgraph.elt Symgraph.t
val packs_from_a_list:
   'symbol Rewrite_rules.t list →
      'symbol Symgraph.elt Symgraph.elt list list
val packs_and_rules_from_packs:
   'symbol Rewrite_rules.t list →
      'symbol list list \rightarrow ('symbol list \times 'symbol Rewrite_rules.t list) list
minimal_split sigma vars r returns a list of minimal modules in htrs r.
val minimal_split:
   'symbol #Signatures.signature → Hierarchical_trs.htrs → Hierarchical_trs.htrs list
```

17.13 Module Relative_dp

```
open Marked_dp_criteria

val mdp_of_rules:
    ('symbol Rewrite_rules.t) list \rightarrow
        ('symbol Gen_terms.term \times 'symbol Gen_terms.term) list

val mdp_of_rules_marks:
    bool \rightarrow
        'symbol #Signatures.signature \rightarrow ('symbol Rewrite_rules.t) list \rightarrow
        (('symbol dupl) Gen_terms.term \times ('symbol dupl) Gen_terms.term) list

val modular_dp_criterion:
    Hierarchical_trs.htrs \rightarrow
        (Hierarchical_signatures.hsymbol) Termination_constraints.conj_termination_constraints
```

```
val modular_dp_criterion_marks :
bool →
Hierarchical_trs.htrs →
Hierarchical_signatures.hsymbol Marked_dp_criteria.dupl #Signatures.signature →
Hierarchical_signatures.hsymbol dupl Termination_constraints.conj_termination_constraints
```

17.14 Module Termination_expert

This module is the termination expert. The main function looks for a termination proof for a given rewrite system.

17.14.1 Global configuration

the following global variables controls which criterion the termination expert uses.

When *standard_criterion* is true, the termination criterion is the standard one: each must decrease with respect to a monotonic well-founded ordering. When it is false, the dependency pair criteria are used. See module *Dependency_pair_criteria* 17.2. default true

val standard_criterion: bool ref

When dp_criterion_uses_markings is true, DP criteria with marks are used. default true

val dp_criterion_uses_markings: bool ref

When dp_criterion_uses_markings_ac is true, AC symbols are marked in DP criteria with marks. default false

val dp_criterion_uses_markings_ac : bool ref

When dp_criterion_uses_graph is true, DP criteria with graph are used. default true

val dp_criterion_uses_graph: bool ref

When enable_split is true, relative dp criteria are used. default false

val enable_split : bool ref

The verbose level.

1. the termination constraints to solve are displayed.

2. ...

Default is 0

val verbose: int ref

17.14.2 Computing dependency pairs

show_pairs F H MH X R computes the dependency pairs of the (F,X)-TRS R, and displays them. If the marked criterion is currently selected, then marked pairs are computed. If the graph criterion is currently selected, also displays the approximated dependency graph. H is an hashconsing table for standard terms and MH is an hashconsing table for marked terms.

```
val show_pairs : 

('symbol #Signatures.signature) \rightarrow 

Variables.user_variables \rightarrow 

('symbol Rewrite_rules.t) list \rightarrow unit
```

17.14.3 Computing termination constraints

```
val compute_termination_constraints_without_marks :

('symbol #Signatures.signature) →

Variables.user_variables →

('symbol Rewrite_rules.t) list →

'symbol Termination_constraints.conj_termination_constraints

val compute_termination_constraints_with_marks :

('symbol #Signatures.signature) → 'symbol Marked_dp_criteria.dupl #Signatures.signature →

Variables.user_variables →

('symbol Rewrite_rules.t) list →

'symbol Marked_dp_criteria.dupl

Termination_constraints.conj_termination_constraints
```

17.14.4 The main function

```
expert F \times R looks for a termination proof of the (F,X)-TRS R. Displays the results.
```

```
type 'a graph_termination_proof =
     (('a Gen_terms.term × 'a Gen_terms.term) Labelled_graphs.graph
      × 'a termination_proof_component) list
and 'a termination_proof_component =
  | Graph_simple_criterion of 'a Poly_interp.t
  | Graph_complex_criterion of
       'a Poly_interp.t × 'a graph_termination_proof
type 'a termination_proof =
  Failed
   Standard of 'a Signatures.signature × 'a Poly_interp.t
  | Dp_graph_nomark of 'a Signatures.signature × 'a graph_termination_proof
  | Dp_nograph_nomark of 'a Signatures.signature × 'a Poly_interp.t
  | Dp_graph_mark of
       ('a Marked_dp_criteria.dupl) Signatures.signature ×
       ('a Marked_dp_criteria.dupl) graph_termination_proof
  | Dp_nograph_mark of
       ('a Marked_dp_criteria.dupl) Signatures.signature ×
       ('a Marked_dp_criteria.dupl) Poly_interp.t
val print_proof : 'a termination_proof \rightarrow unit
val latex_print_proof : string \rightarrow 'a termination_proof \rightarrow unit
exception No_proof_found
val expert:
     ('symbol #Signatures.signature) \rightarrow
       Variables.user\_variables \rightarrow
          ('symbol Rewrite_rules.t) list \rightarrow
            'symbol termination_proof
```

```
type modular_termination_proof =
  | Modular_nograph_nomark of
       Hierarchical_signatures.hsymbol Signatures.signature ×
       Hierarchical_signatures.hsymbol Poly_interp.t
  | Modular_nograph_mark of
       (Hierarchical_signatures.hsymbol Marked_dp_criteria.dupl)
          Signatures.signature ×
       (Hierarchical_signatures.hsymbol Marked_dp_criteria.dupl) Poly_interp.t
  | Modular_graph_nomark of
       Hierarchical_signatures.hsymbol Signatures.signature ×
       Hierarchical_signatures.hsymbol graph_termination_proof
  | Modular_graph_mark of
       (Hierarchical_signatures.hsymbol Marked_dp_criteria.dupl)
          Signatures.signature ×
       (Hierarchical_signatures.hsymbol Marked_dp_criteria.dupl) graph_termination_proof
val print_module_termination_proof : modular_termination_proof \rightarrow unit
val print_modular_proof :
  (Hierarchical_trs.htrs \times modular_termination_proof) list \rightarrow unit
val latex_print_modular_proof:
  string \rightarrow (Hierarchical\_trs.htrs \times modular\_termination\_proof) list \rightarrow unit
val modular_expert:
       Variables.user_variables →
          Hierarchical_trs.htrs →
                      (Hierarchical_trs.htrs × modular_termination_proof) list
```

17.15 Module Automaton

Obsolete header:

Les automates ascendants d'arbres avec condition sur les sous-termes immediats pour des termes sans symboles AC. Les conditions sont de la forme : un entier = le numero de l'argument (de 0 a (arite f)-1 la comparaison a effectuer entre les sous-termes un entier = le numero de l'autre sous-terme a comparer ex : (2,=,3) verifie que les 2eme et 3eme arguments sont egaux

```
type t = int

open Types

type var\_id = int

exception VARIABLE of var\_id (*state*)

type ('a, 'b) u

type 'SYMBOL internal\_term =

Var of var\_id

| App of ('SYMBOL × ('SYMBOL internal\_term list))

type 'SYMBOL formula = OR of 'SYMBOL formula × 'SYMBOL formula

| AND of 'SYMBOL formula × 'SYMBOL formula

| EQ of ('SYMBOL internal\_term) list × ('SYMBOL internal\_term) list)

| EXIST of (('SYMBOL internal\_term) list × 'SYMBOL formula)

| TRUE \mid FALSE
```

```
type ('STATE) epsilon_rule = { initial_state : 'STATE;
                                      final_state : 'STATE; }
type ('SYMBOL, 'TERM, 'STATE) conditional_rewrite_rule = {tete : 'SYMBOL;
                                                                         etats_initiateurs : 'STATE list;
                                                                         conditions : ( int \times ( 'TERM \rightarrow
'TERM \rightarrow bool ) \times int ) list;
                                                                         etat_final : 'STATE }
type ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata = { alphabet : 'SYMBOL list;
                                                            typage : ('SYMBOL \rightarrow 'TYPE);
                                                            tete\_de\_terme : 'TERM \rightarrow 'SYMBOL';
                                                            liste_des_sous_termes : 'TERM \rightarrow ('TERM list);
                                                            etats: 'STATE list;
                                                            etats_finaux : 'STATE list ;
                                                            etat_poubelle : 'STATE ;
                                                            epsilon : ( 'STATE) epsilon_rule list ;
                                                            regles: (('SYMBOL, 'TERM, 'STATE) conditional_rewrite_rule) list;
                                                            internal_names : (('SYMBOL internal_term) list);
                                                            internal_formula : ( ( 'SYMBOL formula) list )
                                                                                               ('TERM
val make_automata
                              'SYMBOL list \rightarrow
                                                      ('SYMBOL
                                                                             TYPE

ightarrow 'STATE list 
ightarrow 'STATE list 
ightarrow
 "SYMBOL") \rightarrow ("TERM" \rightarrow "("TERM" list"))
                                                                                                 'STATE
  ( 'STATE) epsilon_rule list \rightarrow (('SYMBOL, 'TERM, 'STATE) conditional_rewrite_rule ) list \rightarrow
('SYMBOL, 'TERM, 'TYPE, 'STATE) automata
val make\_rule: 'SYMBOL \rightarrow 'STATE list \rightarrow ((int \times ( 'TERM \rightarrow 'TERM \rightarrow bool ) \times int ) list ) \rightarrow
'STATE → ('SYMBOL, 'TERM, 'STATE) conditional_rewrite_rule
val get_alphabet : ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata → 'SYMBOL list
val get\_states: ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata \rightarrow 'STATE list
val get_final_states : ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata → 'STATE list
                                  ('SYMBOL,
                                                   TERM.
                                                                TYPE,
                                                                             'STATE)
        get_rules
                                                                                          automata
((('SYMBOL, 'TERM, 'STATE) conditional_rewrite_rule ) list )
        add_rule
                                 ('SYMBOL,
                                                  'TERM,
                                                                TYPE,
                                                                             'STATE)
                                                                                          automata
('SYMBOL, 'TERM, 'STATE) conditional_rewrite_rule → ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata
val add_epsilon_rule : ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata → 'STATE epsilon_rule →
('SYMBOL, 'TERM, 'TYPE, 'STATE) automata
val get_epsilon_rules : ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata \rightarrow (('STATE) epsilon_rule list)
val get_head_of_rule : (('SYMBOL, 'TERM, 'STATE) conditional_rewrite_rule ) → 'SYMBOL
val get_constraints_of_rule: (('SYMBOL, 'TERM, 'STATE) conditional_rewrite_rule)
                                              \rightarrow ( (int \times ( 'TERM \rightarrow 'TERM \rightarrow bool ) \times int ) list)
val get_initiator_of_rule : (('SYMBOL, 'TERM, 'STATE) conditional_rewrite_rule )
                                              → 'STATE list
val get_final_state_of_rule: (('SYMBOL, 'TERM, 'STATE) conditional_rewrite_rule) \rightarrow 'STATE
Deterministe et sans epsilon-transitions termes clos
val run_automata : ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata → 'TERM → 'STATE
val test_acceptance : ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata \rightarrow 'TERM \rightarrow ('STATE \times bool)
Non Deterministe et avec epsilon-transitions termes clos
```

```
val nd_run_automata : ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata → 'TERM → 'STATE list
val nd_test_acceptance: ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata \rightarrow 'TERM \rightarrow ('STATE list \times bool)
nd et epsilon terme avec variable
val nd_test_var_acceptance : (var_id \times sorte_de_base) list \rightarrow ('SYMBOL, 'TERM, 'TYPE, Types.state) automata \rightarrow
TERM \rightarrow (Types.state\ list \times bool)
val nd\_test\_complement: ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata \rightarrow 'TERM \rightarrow ('STATE \ list \times bool)
      nd_test_intersect
                                    ('SYMBOL,
                                                    TERM
                                                                'TYPE,
                                                                           'STATE)
                                                                                       automata
('SYMBOL, 'TERM, 'TYPE, 'STATE1) automata \rightarrow 'TERM \rightarrow bool
Functions that should be done in the future
val nd_test_final_state : ('a, term, 'b, state) automata -> term ->
state list * bool ;;
val nd_test_final_state : ('a, 'b, 'c, 'd) automata -> 'b -> 'd list *
bool;;
val remove_epsilons : ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata ->
('SYMBOL, 'TERM, 'TYPE, 'STATE) automata ;;
val determinize_automata : ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata
 -> ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata ;;
      union_of_automata
                                     ('SYMBOL,
                                                     TERM,
                                                                 TYPE,
                                                                            'STATE)
                                                                                       automata
('SYMBOL, 'TERM, 'TYPE1, 'STATE1) automata → ('SYMBOL, 'TERM, 'TYPE, ('STATE1, 'STATE1) u) automata
Donne un etiquetage generique a l'automate sans oublier le premier etiquetage
      internalize_automata
                                      ('SYMBOL,
                                                      TERM
                                                                 TYPE,
                                                                            'STATE)
                                                                                        automata
('SYMBOL, 'TERM, 'TYPE, 'STATE) automata
Ne s'applique que si l'automate est internalize!!
      create_internal_formula
                                                       'TERM,
                                        ('SYMBOL,
                                                                  TYPE,
                                                                             'STATE)
                                                                                        automata
('SYMBOL, 'TERM, 'TYPE, 'STATE) automata
val triv_leads_to: ('SYMBOL, 'TERM, 'TYPE, 'STATE) automata → 'STATE → 'STATE → bool
val clean_automata : ('a, 'b, 'c, 'd) automata \times 'd epsilon_rule list \rightarrow ('a, 'b, 'c, 'd) automata \times
'd epsilon_rule list
          Module Specif
17.16
open Tree_automata
open Symbols
open Terms
open Equations
open Types
val make_signature_automata:
  unit \rightarrow (symbol_id, term, sorte list, state) automata
```

val create_current_env:

term → (Symbols.var_id × Symbols.sorte_de_base) list

```
val build_nf_automata :
    regle list → (symbol_id, term, sorte list, state) automata × state epsilon_rule list
val print_delta_automata :
    (symbol_id, term, sorte list, state) automata × state epsilon_rule list → unit
val print_clean_automata :
    (symbol_id, term, sorte list, state) automata × state epsilon_rule list → unit
val print_state : state → unit
```

17.17 Module Types

```
\label{eq:type_sorte_de_base} \begin{tabular}{ll} type & atomic\_state & = FS of (User\_signatures.symbol\_id \times (atomic\_state \ list)) \\ & | VS of sorte\_de\_base \\ \end{tabular} \begin{tabular}{ll} type & state & = State of (atomic\_state \ list) \\ \end{tabular}
```

Chapter 18

Library toplevel

This library defines the toplevel language and its interpreter.

18.1 Module Abstract_syntax

```
type abstract_expr =
  Integer of Numbers.t
   String of string
   | Bool of bool
   | Var of string
   | Fun of string × abstract_expr
   | Apply of abstract_expr × abstract_expr
   | If of abstract_expr × abstract_expr × abstract_expr
   | Tuple of abstract_expr list
   Set of abstract_expr list
type abstract_command =
   | Def of string × abstract_expr
   | Deftuple of string list × abstract_expr
   Deffun of string \times string list \times abstract_expr
   | Eval of abstract_expr
  | Directive of string \times string
val print\_expr: abstract\_expr \rightarrow unit
val print\_named\_expr: string \rightarrow abstract\_expr \rightarrow unit
```

18.2 Parser Toplevel_parser

Header

open Abstract_syntax

Token declarations

```
%token < string > IDENT
%token < Numbers.t > INTEGER
%token < string > STRING
%token LET FUN ARROW
%token IF THEN ELSE
%token AND OR NOT TRUE FALSE
%token PLUS STAR MINUS
%token CONCAT
%token GE GT LE LT NEQ
%token LEFTPAR RIGHTPAR SEMICOLON COMMA LEFTBRACE RIGHTBRACE
%token EQUAL
%token DIRECTIVE
%token EOF
%start command
%type <Abstract_syntax.abstract_command> command
%nonassoc ARROW IF
%left OR
%left AND
%left NOT
%left GE GT LE LT NEQ EQUAL
%left PLUS MINUS
%left STAR
%nonassoc UMINUS
%nonassoc IDENT INTEGER STRING FUN LEFTPAR LEFTBRACE CONCAT TRUE FALSE
%left APPLY
```

Grammar rules

```
command ::=

EOF { raise End_of_file }
| command_aux SEMICOLON { $1 }

command_aux SEMICOLON { $1 }

command_aux ::=
| LET IDENT EQUAL expr { Def($2,$4) }
| LET LEFTPAR identlist RIGHTPAR EQUAL expr { Deftuple($3,$6) }
| LET FUN IDENT args EQUAL expr { Deffun($3,$4,$6) }
| expr { Eval($1) }
| DIRECTIVE IDENT { Directive($2,"") }
| DIRECTIVE IDENT STRING { Directive($2,$3) }
| DIRECTIVE IDENT IDENT { Directive($2,$3) }
| DIRECTIVE IDENT INTEGER { Directive($2,Numbers.to_string $3) }

identlist ::=
| IDENT { [$1] }
| IDENT COMMA identlist { $1::$3 }
```

```
args ::=
  IDENT { [$1] }
| IDENT args { $1::$2 }
expr ::=
IF expr THEN expr ELSE expr %prec IF
    { If(\$2,\$4,\$6) }
expr simple_expr %prec APPLY
    \{ Apply(\$1,\$2) \}
FUN IDENT ARROW expr
    { Fun($2,$4) }
expr PLUS expr
    { Apply(Apply(Var("+"),$1),$3) }
expr MINUS expr
    { Apply(Apply(Var("-"),$1),$3) }
MINUS expr %prec UMINUS
    { Apply(Var("_minus"),$2) }
expr STAR expr
    { Apply(Apply(Var("*"),$1),$3) }
expr EQUAL expr
    { Apply(Apply(Var("="),$1),$3) }
expr NEQ expr { Apply(Apply(Var("<>"),$1),$3) }
 expr\ GE\ expr\ \{\ Apply(Apply(Var(">="),$1),$3) \}
 expr GT expr { Apply(Apply(Var(">"),$1),$3) }
 expr LE \ expr \{ Apply(Apply(Var("<="),$1),$3) \}
 expr LT expr { Apply(Apply(Var("<"),$1),$3) }</pre>
 expr AND expr { Apply(Apply(Var("and"),$1),$3) }
 expr OR expr { Apply(Apply(Var("or"),$1),$3) }
 expr CONCAT expr { Apply(Apply(Var("^"),$1),$3) }
NOT expr { Apply(Var("not"),$2) }
simple_expr
      { $1 }
simple\_expr ::=
IDENT
    { Var($1) }
INTEGER
    { Integer($1) }
TRUE
    { Bool(true) }
FALSE
    { Bool(false) }
STRING
    { String($1) }
LEFTPAR expr RIGHTPAR
    { $2 }
LEFTPAR expr COMMA commalist RIGHTPAR
      { Tuple ($2 :: $4) }
LEFTBRACE RIGHTBRACE
           { Set([]) }
LEFTBRACE semicolonlist RIGHTBRACE
           { Set ($2) }
```

18.3 Module Eval

```
open Abstract\_syntax
open Values
val eval: env\_type \rightarrow abstract\_expr \rightarrow values
```

18.4 Module *Typing*

```
open Abstract_syntax
```

exception raised in case of typing error, with an explicit message as argument.

exception Type_error of string

the type system in the toplevel

A type is either

- a base type : int, bool, string, etc ;
- a dependent type : (Σ, X) term, etc ;
- a type variable (without explicit name);
- an arrow type $t_1 \rightarrow t_2$;
- a dependent arrow type $(x:t_1)t_2$;

Base types are particular cases of dependent types where the list of dependent vars is empty.

An arrow type $t_1 \to t_2$ is a particular case of the dependent arrow type $(x:t_1)t_2$ when x does not occur in t_2 , which is made explicit by putting x as "anonymous".

and $typing_env_type = (string \times type_scheme)$ list val generalize : types \rightarrow type_scheme val $print_type : types \rightarrow unit$ val $unify_types: types \rightarrow types \rightarrow bool$ val $type_expr: typing_env_type \rightarrow typing_env_type \rightarrow abstract_expr \rightarrow types$ val unit_type : types val int_type : types val string_type : types val bool_type : types $val\ set_type:\ types \rightarrow\ types$ val $pair_type: types \rightarrow types \rightarrow types$ val dioph_constraint_type : types val signature_type : types val word_signature_type : types val theo_type : depend_name → types val variable_set_type : types val word_type : depend_name → types val $word_precedence_type : depend_name \rightarrow types$ val precedence_type : depend_name → types val status_type : depend_name → types val word_ordering_type : depend_name → types val $term_ordering_type : depend_name \rightarrow types$ val srs_type : $depend_name \rightarrow types$ val position_type : types val term_type : depend_name \rightarrow depend_name \rightarrow types val conjecture_type : depend_name \rightarrow depend_name \rightarrow types val equations_type : depend_name \rightarrow depend_name \rightarrow types val $trs_type: depend_name \rightarrow depend_name \rightarrow types$ val htrs_type : types val termination_constraint_type : depend_name \rightarrow depend_name \rightarrow types val order_constraint_type : depend_name → depend_name → types val parameter_set_type : types val pword_signature_type : types val $pword_type: depend_name \rightarrow types$ val psrs_type : depend_name → types val psubst_type : types val $arrow_type : types \rightarrow types \rightarrow types$ val bin_arith_type : type_scheme val bin_comp_type : type_scheme val bin_bool_type : type_scheme

18.5 Module *Predef*

```
open Values

val initial_env : unit \rightarrow env_type

val initial_env_type : unit \rightarrow Typing.typing_env_type

val get_code : string \rightarrow (int \times (values list \rightarrow values))

Returns the list of the predefined category names.

val get_predef_category_list : unit \rightarrow string list

Returns the list of the predefined command names in the given category.

val get_predef_command_list : string \rightarrow string list

Returns the category and the help string associated to a command name.

val get_help_of_command : string \rightarrow string \times string

Returns the type of the function associated to a command name.
```

18.6 Module Values

```
open Abstract_syntax
type named_sig =
  mutable sig_name : string ;
  sig_val : User_signatures.user_signature;
  sig_table : User_signatures.symbol_entry array;
type named_hsig =
  mutable hsig_name : string ;
  hsig_val: Hierarchical_signatures.hierarchical_signature
type named_word_sig =
  mutable word_sig_name : string ;
  word_sig_val : String_signatures.user_word_signature
type named_parameters =
  mutable parameters_name : string ;
  parameters_val : Parameterized_signatures.parameter_c ;
type named_pword_sig =
  mutable pword_sig_name : string ;
  pword_sig_val : User_parameterized_signatures.user_parameterized_signature
```

18.6. MODULE VALUES 165

```
type named_psubst =
  mutable psubst_name : string ;
  psubst\_val : string \rightarrow Numbers.t
type named_vars =
  mutable vars_name : string ;
  vars_val : Variables.user_variables;
  vars_table : string array;
type values =
  | Unit_value
  Int_value of Numbers.t
   Bool_value of bool
  String_value of string
  | Fun_value of string × env_type × abstract_expr
  | Predef of string \times values list
  | Set_value of values list
  | Prod_value of values list
  | Sig_value of named_sig
  | Word_sig_value of named_word_sig
  | Parameters_value of named_parameters
  | Pword_sig_value of named_pword_sig
  | Pword_value of
       named_pword_sig × Parameterized_words.word
  | Psrs_value of
       named_pword_sig × Parameterized_rewriting.srs
  | Psubst_value of named_psubst
  Theo_value of named_sig × User_signatures.symbol_id Theory.elem_theory list
  | Var_value of named_vars
  | Word_value of
       named_word_sig × String_signatures.symbol_id Words.word
  | Word_precedence_value of
       named_word_sig
       × String_signatures.symbol_id Orderings_generalities.ordering
  | Precedence_value of
       named_sig
       \times User_signatures.symbol_id Orderings_generalities.ordering
  | Status_value of
       named_sig
       × User_signatures.symbol_id Rpo.status_function
  | Word_ordering_value of
       named_word_sig
       × String_signatures.symbol_id Words.word Orderings_generalities.ordering
  Term_ordering_value of named_sig × User_signatures.symbol_id Term_orderings.t
  | SRS_value of
       named_word_sig ×
       String_signatures.symbol_id String_rewriting.srs ×
       (String_signatures.symbol_id String_rewriting.compiled_srs) Lazy.t
  | Position_value of Positions.position
```

```
Term_value of
      named_sig × named_vars ×
       User_signatures.symbol_id Gen_terms.term
  | Conjecture_value of
      named_sig × named_vars ×
       (User_signatures.symbol_id Gen_terms.term × User_signatures.symbol_id Gen_terms.term)
  | Equations_value of
      named_sig × named_vars ×
       (User_signatures.symbol_id Gen_terms.term × User_signatures.symbol_id Gen_terms.term) list
  | TRS_value of
      named_sig × named_vars ×
       (User_signatures.symbol_id Gen_terms.term × User_signatures.symbol_id Gen_terms.term) list ×
       (User_signatures.symbol_id Gen_terms.term → User_signatures.symbol_id Gen_terms.term)
  | HTRS_value of
       named_vars × Hierarchical_trs.htrs
  | Dioph_constraint of Abstract_constraint.formula
  | Linear_constraint of Linear_constraints.formula
  | Termination_constraint of
       named_sig × named_vars ×
       User_signatures.symbol_id Termination_constraints.conj_termination_constraints
  Order_constraint of
       named_sig × named_vars ×
       User_signatures.symbol_id Order_constraints.formula
and env_data =
       (* typ : types *)
      mutable value: values
and env\_type = (string \times env\_data) list
val print_value : values → unit
exception Eval_error of string
```

18.7 Module Toplevel_lexer

```
exception Lexical_error of string
```

val token : Lexing.lexbuf \rightarrow Toplevel_parser.token

18.8 Module Version

```
val version: string
val date: string
```

18.9 Lexer Toplevel_lexer

```
{
    open Toplevel_parser
    exception Lexical_error of string
```

```
rule token = parse
    [' '' \r'' \t'' \n'] \{ token lexbuf \}
  | "let" { LET }
  "fun" { FUN }
  "if" { IF }
  "then" { THEN }
  "else" { ELSE }
  "and" { AND }
  "or" { OR }
  | "not" { NOT }
  | "true" { TRUE }
  |"false" { FALSE }
  | "#" { DIRECTIVE }
  ['A'-'Z''a'-'z']['A'-'Z''a'-'z''0'-'9''_''\']*
                          { IDENT(Lexing.lexeme lexbuf) }
  |',' { COMMA }
  | ' +' { PLUS }
  |'-' { MINUS }
  | ' *' { STAR }
  | ' ^ ' { CONCAT }
  | "=" { EQUAL }
  | "->" { ARROW }
  ">" { GT }
  | ">= " { GE }
  | "<" { LT }
  | "<=" { LE }
  | "<>" { NEQ }
  | ['0'-'9'] | { INTEGER (Numbers.from_string (Lexing.lexeme lexbuf)) }
  \'\('\) \{ LEFTPAR \}
  | ' { ' { LEFTBRACE }
  | ' } ' { RIGHTBRACE }
  '"' { STRING(string lexbuf) }
  "(*" { comment lexbuf }
  | { raise (Lexical_error (Lexing.lexeme lexbuf)) }
  | eof { EOF }
and string = parse
    '"' { "" }
  | ' \setminus '  { let s = Lexing.lexeme\ lexbuf\ in\ s^(string\ lexbuf) }
  ['' \ '''']^+  { let s = Lexing.lexeme\ lexbuf\ in\ s^(string\ lexbuf) }
 eof { raise (Lexical_error "string_not_terminated") }
and comment = parse
    "*)" { token lexbuf }
  | [^' *'] | { comment lexbuf }
 _ { comment lexbuf }
 | eof { raise (Lexical_error "comment_not_terminated") }
```

Index of Identifiers

?f (label), 87	add_vertex, 49
abs, 42	ag (camlyacc non-terminal), 115, 115
Absorb, 109	AG, 111, 112, 115
Abstraction, 109	all_data, 129
abstract_command (type), 159, 160	all_decreasing, 150
Abstract_constraint (module), 68 , 72, 75, 77, 79,	all_decreasing_constraints, 143
89, 90, 91, 92, 165	all_dp_strictly_decreasing, 144
abstract_expr (type), 159 , 159, 162, 163, 165	all_one, 40
Abstract_rewriting (module), 134, 136	all_rules_weakly_decreasing, 143
Abstract_syntax (module), 159, 159, 160, 162, 164	all_zero, 40
Ac, 96	alphabet (field), 156
AC, 111, 112	alt, 54 , 55 , 55
ACCompletion (module), 137	Alt, 54 , 55
ACConfluence (module), 136	alt_rhs_of_rule, 134
aci (camlyacc non-terminal), 115, 115	And, 68 , 70
ACI, 111, 112, 115	<i>AND</i> , 155 , 76, 98, 167
Acrpo (module), 108	<i>AND</i> (camlyacc token), 77 , 160 , 77, 160, 161
acu (camlyacc non-terminal), 115, 115	Anonymous, 162
ACU, 111 , 112 , 115	App, 155
acun (camlyacc non-terminal), 115, 115	Apply, 159 , 161
ACUN, 111, 112, 115	apply_sorted_term, 96
AC_COMPLETE, 111	apply_subst_to_eqs, 96
Ac_critical_pairs (module), 133	apply_term, 96
AC-ONLY, 111	arbre_binaire, 120
ac_rhs_rules, 149	args (camlyacc non-terminal), 160 , 160
ac_unification, 126	Arith_to_unif (module), 120, 122, 123
add, 37, 39, 40, 42, 43, 44, 46, 47, 49, 54, 70, 55,	arity (method), 95, 149
57, 58, 99, 147, 148	Arrayutils (module), 37
add (method), 45, 141	ARROW (camlyacc token), 84 , 88 , 160 , 85, 90,
Add, 74	160, 161
additive_symbol_of_theory, 112	arrow_type, 163
add_an_equation_between_variables, 113	associated_marked_var_with_sol, 121
add_arc, 49	associated_var_with_sol, 121
add_a_data, 129	Associativity, 109
add_a_mark, 124	Associator, 109
add_cte, 69	assoc_right, 38
add_edge, 151	assume_ce, 101
add_epsilon_rule, 156	atmost_one_one, 41
add_equiv, 61	AtomicTerminationConstraintSet (module), 151,
add_greater, 61	143, 144, 151
add_list, 61, 85	atomic_state (type), 158 , 158
Add_multiply, 109	atomic_state (type), 136, 136 atomic_termination_constraint (type), 151, 151
add_node, 151	automata (type), 156 , 156, 157, 158
add_node, 151 add_rule, 156	Automata (type), 150, 150, 157, 158 Automata (module), 56
add_ture, 150 add_term, 119	
auu_teiiii, 119	Automaton (module), 155

100 110	.1
axiom (type), 109 , 110	clean_automata, 157
Axioms (module), 109	clean_solutions, 120
AxiomSet (module), 110	clear_ce, 101
a_variable_of_t, 134	CLOSEPAR (camlyacc token), 114, 115
bal, 38,39	CL_def_axiom, 109
Balanced_trees (module), 38	Cn_axiom, 109
Balanced_trees2 (module), 39	coalesce, 118
Base_ring (module), 46 , 46	coef (type), 46 , 46, 47, 48, 75, 105, 144
Base_type, 162	coeff_age, 136
Basic_criterion (module), 143	coeff_size, 136
BINARY_TREE (sig), 120, 120	collision_rate, 45
bin_arith_type, 163	COMMA (camlyacc token), 77, 84, 114, 147, 160,
bin_bool_type, 163	77, 85, 115, 148, 160, 161
bin_comp_type, 163	commalist (camlyacc non-terminal), 161 , 161
bit_and, 40	command (camlyacc non-terminal), 160 , 160
Bit_field (module), 40 , 120	command_aux (camlyacc non-terminal), 160, 160
bit_field_to_vect_of_bits, 41	comment (camllex parsing rule), 84, 91, 98, 99,
bit_length, 40	167
bit_not, 40	common_denominator, 70
bit_nth, 41	Commutative, 96
bit_nth_first, 41	Commutator3, 109
bit_or, 40	Commutator4, 109
Bool, 159 , 161	Comp, 68 , 77
bool_type, 163	COMP (camlyacc token), 77,77
Bool_value, 165	compact_print, 50
BOO_inverse, 109	compare, 39 , 42 , 43 , 45 , 56 , 61 , 74 , 56, 85
BOO_11, 109	compare_poly, 107
BOO_13, 109	compare_var, 102
br (camlyacc non-terminal), 115, 115	comparison_result (type), 62 , 62, 107
BR, 111, 112, 115	compile, 129, 134
build, 107	compiled_data (type), 134, 134
build_nf_automata, 158	compiled_normalize, 81
build_renaming, 69, 73	compiled_pairs (type), 134 , 134, 135
build_var_interp_of_term, 106	compiled_rules (type), 134 , 134, 135
builtin_th (type), 136 , 136	compiled_srs (type), 81 , 56, 81, 165
B_axiom, 109	compile_data, 134
C, 111, 112	compile_for_unification, 111, 134
cache (type), 44 , 44, 45	compile_pairs, 134
Cache (module), 44	compile_rules, 129
canonical_renaming, 96	compile_srs, 81
canonize_pairs, 135	complete, 136 , 137
cardinal, 43 , 58	complete_srs_alphabetic_order_std_strategy, 86
cell (type), 93 , 93	complete_srs_std_strategy, 86
char, 53,55	complete_var_interp_of_term, 106
Char, 54 , 55	compute_defined_symbols, 143
Charset, 54	compute_dependency_pairs, 144
CharSet (module), 53 , 54 , 56 , 54, 55, 56, 57, 58	compute_dependency_pairs_with_markings, 149
check_conj_constraints, 151	compute_strongly_connected_components, 51
check_make_term, 99	compute_termination_constraints_without_marks,
check_poly_interp, 106	154
choose, 43	compute_termination_constraints_with_marks,
Classical_maps (module), 49	154
classify, 121	concat, 38
Class_maps (module), 45	CONCAT (camlyacc token), 160 , 160, 161
Crass_maps (modute), To	2011211 (caminy acc to Kell), 100, 101

aand (field) 75	default 102
cond (field), 75	default, 103
conditional_rewrite_rule (type), 156 , 156	default (class), 96
conditions (field), 156	Default, 95
Conflict (exn), 96	Default (module), 103
Confluence (module), 136	Deffun, 159 , 160
conj, 69 , 71 , 77	Deftuple, 159 , 160
conjecture_type, 163	delete, 117
Conjecture_value, 165	del_edge, 151
conj_all, 69	del_node, 151
conj_list, 71	denominator, 42
conj_of_order_constraints, 151	depend (type), 162 , 162
conj_termination_constraints (type), 151, 143,	Dependency_pairs_criteria (module), 143
144, 145, 149, 151, 152, 154, 165	dependency_pair_criterion_with_markings, 149
Cons, 93	Depend_id, 162
Const, 74	depend_name (type), 162 , 162, 163
constant_coef, 47	Depend_var, 162
constr (camlyacc non-terminal), 90, 89, 90	Dep_arrow, 162
constrained_word (type), 92, 89	diff, 43
constraint_entry (camlyacc non-terminal), 77,77	Dioph_constraint, 165
constraint_of_string, 79 , 90	dioph_constraint_type, 163
constraint_op (field), 151	Directive, 159 , 160
constraint_op (type), 75,75	DIRECTIVE (camlyacc token), 160 , 160
constraint_op_type (type), 150, 151	discriminate, 129
contains_ac_symbols (method), 95, 149	disj, 69, 71, 77
contains_only_free_symbols (method), 95, 149	disjunction (type), 93 , 111 , 93, 111
Controle (module), 111	disj_all, 69
copy_rule, 149	disj_list, 71
count_nf, 56,58	display_conj_termination_constraints, 151
count_words, 57 , 58	display_plain_set_of_solutions, 126
create, 38 , 39 , 44 , 57, 58, 91, 99	Distributivity, 109
create_current_env, 157	div, 42,70
create_internal_formula, 157	DIV (camlyacc token), 77, 77, 78
critical_pairs, 133, 135	divides, 67 ,71
cte, 47, 69, 147, 148	divisible, 68
Cte, 68 , 78	Divisible, 70
current_formula, 91 , 91	div_ceil, 42
current_poly_vars, 105 , 149 , 147, 148	Div_ceil, 74
current_signature, 86 , 105 , 114 , 148 , 83, 85, 145,	div_floor, 42
150	Div_floor, 74
current_variables, 99, 98	dnet (type), 129 , 141 , 129, 141, 142
cword (camlyacc non-terminal), 89, 89	does_overlap, 58 , 58
cword_eof (camlyacc non-terminal), 89, 89	domain_expression (type), 74, 74, 75
cycle, 125	domain_operation (type), 74, 74
cycle (type), 116, 117	dot_print, 50
Cycle, 116	do_list_combine, 37
Cycle (module), 125	dp_criterion_uses_graph, 153
C_axiom, 109	dp_criterion_uses_markings, 153
data (field), 39	dp_criterion_uses_markings_ac, 153
Data (module), 129, 129, 130	Dp_graph_mark, 154
date, 166	Dp_graph_nomark, 154
debug, 87	Dp_nograph_mark, 154
decl (camlyacc non-terminal), 115, 115	Dp_nograph_nomark, 154 Dp_nograph_nomark, 154
decoration_of_symbol (method), 87	dupl (type), 149 , 147, 149, 152, 154
· · · · · · · · · · · · · · · · · · ·	± 1,4 ± 1
Def, 159 , 160	edges (field), 113

edge_list, 151	etat_final (field), 156
element (type), 87, 87, 88	etat_poubelle (field), 156
elements, 43 , 58	eval, 48, 162
elements_increasing_order, 39, 40	Eval, 159 , 160
elem_pb (type), 113, 114, 117, 118, 119, 120, 122,	Eval (module), 162
124 , 113, 114, 118, 119, 120, 121, 122,	Eval_error (exn), 166
123, 124	except, 37
elem_pbs (field), 113	EXIST, 155
elem_th (field), 113	existential_quantifiers_elimination, 113
elem_theory (type), 111, 112, 113, 119, 120, 126,	exists, 44, 69, 72, 55
112, 113, 114, 116, 119, 120, 121, 122,	Exists, 68 , 70 , 77
123, 124, 125, 126, 165	EXISTS (camlyacc token), 77, 77
elem_theory_from_unif_elem_theory, 112	exists_greater_or_equal, 62
eliminate_quantifiers, 73	exists_one, 40
ELSE (camlyacc token), 160, 161	exists_s, 72
elt (type), 43, 151, 43, 44, 48, 49, 53, 54, 56, 74,	Exp, 92 , 90
96, 102, 105, 110, 112, 126, 129, 134, 151,	<i>EXP</i> (camlyacc token), 77 , 147 , 77, 78, 147, 148
152	expert, 154
elt (camlyacc non-terminal), 89, 89	expo (camlyacc non-terminal), 85, 85
empty, 39 , 40 , 43 , 49 , 53 , 54 , 151 , 56, 57, 58, 148	expr (field), 75
Empty, 38 , 39 , 54 , 111 , 112 , 54, 55	expr (type), 68 , 69 , 91 , 68, 69, 70, 71, 72, 73, 77
empty_graph, 49	79, 87, 91, 92
empty_map, 45	expr (camlyacc non-terminal), 78, 90, 161, 77, 78
enable_split, 153	90, 160, 161, 162
env_data (type), 165 , 166	<i>expr_l</i> (type), 91 , 91
env_type (type), 165 , 162, 164, 165	expr_1 (camlyacc non-terminal), 90, 89, 90
EOF (camlyacc token), 77, 84, 88, 114, 147, 160,	expr_of_string, 79 , 90
77, 85, 89, 90, 115, 147, 148, 160	ext (field), 97
epsilon, 53 , 55 , 55	extend, 101
epsilon (field), 156	extend_signature, 100
Epsilon, 54 , 55	<i>E_res</i> (module), 125
epsilon_rule (type), 155 , 156, 157, 158	factor (type), 92 , 92
eq, 42,71	factor (camlyacc non-terminal), 85, 90, 85, 89
Eq, 61 , 85	Failed, 154
EQ, 155, 83, 98	False, 68 , 70 , 77
EQ (camlyacc token), 84, 85	FALSE, 155 , 76, 167
equal, 43	FALSE (camlyacc token), 77 , 160 , 77, 160, 161
Equal, 150	false_formula, 70
EQUAL (camlyacc token), 147, 160, 148, 160,	fd_constraint (type), 75 , 75, 76
161	Fd_polynomials (module), 75 , 75, 144, 147
equals, 134	Fd_polynomials_ring (module), 144 , 144
equations (field), 113	fd_var_id (type), 74 , 74, 75, 76, 145
equations_type, 163	fd_var_id_of_string, 74 , 147
Equations_value, 165	Fd_var_map (module), 74 , 75, 76, 145
equation_of_string, 100	Fd_var_ord (module), 74
equation_set_of_string, 100	Fd_var_set (module), 74
equiv, 71	filter, 37, 43
Equiv, 68, 70, 77	filter_indices, 37
EQUIV (camlyacc token), 77, 77	filter_non_unit_vars, 122
Equivalent, 62, 107	final_state (field), 155
Erasable, 113	find, 37, 39, 40, 44, 48, 49, 99
etats (field), 156	find (method), 45
etats_finaux (field), 156	find_and_apply, 44
etats_initiateurs (field). 156	find_kev. 40

find_one, 40	full_automaton (type), 57
find_th (field), 113	full_auto_accept (field), 57, 58
Finite_domains (module), 74 , 75, 76, 145, 147	full_auto_init (field), 57, 58
finite_domain_solve, 75	full_auto_trans (field), 57, 58
finite_ordering (type), 61 , 61	full_compile, 130
Finite_orderings (module), 61, 84	full_compile_rules, 130
firstchars, 54 , 56 , 56	full_discriminate, 130
first_ac_solution, 126	full_dnet (type), 130 , 130
first_vars (field), 113	Full_dnet (module), 129, 142
flatten_left, 38	full_pgcd, 37, 42
flat_map, 37	Fun, 159 , 161
flat_mapi, 38	FUN (camlyacc token), 160 , 160, 161
fold, 39, 40, 43, 48, 49, 151, 58	function 1, 45
foldi, 38	Fun_value, 165
fold_from_min, 40, 43	gcache, 120
fold_lefti, 37	ge, 42,71
fold_left_graph, 50	GE (camlyacc token), 160 , 160, 161
fold_righti, 37	generalize, 163
fold_right_env, 38	Generalized, 162
forall, 69 , 72	general_E_resolution, 125
Forall, 68, 70 , 77	generate_vect_char, 122
FORALL (camlyacc token), 77,77	generate_vect_char_cst, 121
forall_exists_greater, 62	Generic Polynomials (module), 144 , 144, 145, 147
forall_s, 72	Generic_polynomials (module), 144, 146, 147,
force_innermost_normalize, 130, 131	150
force_normalize, 134	Genpoly_lexer (module), 145
formula (type), 68, 70, 155 , 68, 69, 70, 71, 72, 73,	Genpoly_parser (module), 146, 145
74, 75, 77, 79, 87, 91, 92, 100, 151, 155,	Genpoly_syntax (module), 148, 145, 147
156, 165	gen_poly (camlyacc non-terminal), 147 , 147
formula (camllex parsing rule), 91	gen_poly_entry (camlyace non-terminal), 147,
formula (camlyacc non-terminal), 77 , 77	147
FORMULA (camlyacc token), 88, 90	gen_var, 147 , 147
formula_struct (type), 70 , 70, 71, 72	get (method), 141
for_all, 44	get_all_solutions, 74
Found (exn), 58	get_all_strict_sub, 57, 58
FP (camlyacc token), 88, 90	get_alphabet, 156
fprint, 73	get_and_remove_min, 43
fprint_expr, 73	get_code, 164
Free, 96	get_coef, 70
Freeze, 93	get_constraints_of_rule, 156
free_solve_constraints, 126	get_epsilon_rules, 156
free_unification, 126	get_final_states, 156
free_vars, 69,72	get_final_state_of_rule, 156
free_vars_env, 72	get_head_of_rule, 156
fresh_variables, 102	get_help_of_command, 164
fresh_var_for_unif, 102	get_initiator_of_rule, 156
from_abstract_expr, 72	get_predef_category_list, 164
from_abstract_formula, 72,75	get_predef_command_list, 164
from_list, 82, 93, 103	get_rules, 156
from_list (class), 103	get_states, 156
from_string, 42, 82, 88, 103, 116, 76, 167	get_subgraph, 50
from_string (class), 101, 103	get_successors, 50
frozen (type), 93, 93	get_type_of_command, 164
FS, 158	get_vertex_label, 50
,	O . =

ggreat_enough, 120	<i>IMPLIES</i> (camlyacc token), 77, 77
global_status (field), 113	Incompatible (exn), 61
graph (type), 49 , 49, 50, 51, 154	index, 37 , 147
* * · · · · · · · · · · · · · · · · · ·	
Graph_complex_criterion, 154	index_for_unif (type), 134 , 134
graph_from_a_list, 152	Infinite (exn), 74
Graph_simple_criterion, 154	Infinite_language (exn), 56, 57
graph_termination_proof (type), 154 , 154	Infix, 95
Greater, 150	init, 113, 58
Greater_or_equal, 150	initial_env, 164
Greater_or_equivalent, 62	initial_env_type, 164
Greater_than, 62	initial_state (field), 155
greater_than_all, 62	init_for_unif, 102
gsmall_enough, 120	init_narrow, 135
gt, 42,71	Innermost (module), 131
Gt, 61 , 85	innermost_normalize, 130, 131
GT (camlyacc token), 84 , 160 , 85, 160, 161	insert_solved_elem_pbs, 113
half_set, 44	inst, 73
hash, 41 , 103 , 54	inst1, 72
hash_conj, 71	instanciate_when_no_cycle, 117
hash_disj, 71	instantiate_interp, 145
has_an_ac_solution, 126	instantiate_signature, 88
has_finitely_many_solutions, 73	instantiation (type), 72, 73
hdl, 93	inst_variables (field), 113
head (field), 93	INT (camlyacc token), 77, 84, 114, 147, 78, 85,
height, 38, 39	115, 147, 148
height (field), 39	Integer, 159 , 161
hierarchical_signature (class), 100	<i>INTEGER</i> (camlyacc token), 160 , 160, 161
Hierarchical_signatures (module), 100, 101, 152,	inter, 43
154, 164	internalize_automata, 157
Hierarchical_trs (module), 101, 152, 155, 165	internal_formula (field), 156
hsig_name (field), 164	internal_names (field), 156
hsig_val (field), 164	internal_term (type), 155 , 155, 156
hsymbol (type), 100 , 100, 101, 152, 154	interp, 145 , 146
htrs (type), 101 , 101, 152, 155, 165	interp (camlyacc non-terminal), 148, 148
htrs_all_rules (field), 101	INTERP (camlyacc token), 147, 148
htrs_imported (field), 101	intersect, 37
htrs_new_rules (field), 101	IntMap (module), 105, 105
htrs_sig (field), 101	IntOrd (module), 105
htrs_type, 163	IntPolynomials (module), 105, 105, 106, 107, 145,
HTRS_value, 165	147, 148
Hullot_bin_trees (module), 120	IntSet (module), 105, 105
H_axiom, 109	int_int_find, 44
Idempotency, 109	int_type, 163
ident, 83, 98, 116, 150, 83, 98, 116, 150	Int_utils (module), 37
<i>IDENT</i> (camlyacc token), 77, 84, 88, 114, 160, 78,	Int_value, 165
85, 89, 90, 115, 160, 161	Invalid_char (exn), 76, 97, 98, 116, 150
identifications_for_unification (type), 67 , 67, 68	Invalid_status (exn), 100
identity_ordering, 61 , 85	Inverse, 109
identlist (camlyacc non-terminal), 160 , 160	Irreducible (exn), 130, 131, 134
id_list (camlyacc non-terminal), 78, 77, 78	is_ac (method), 95, 149
If, 159 , 161	is_ac_unifiable, 122
<i>IF</i> (camlyacc token), 160 , 160, 161	is_an_ac_equality, 135
implies, 71	is_an_instance, 135
Implies, 68 , 70 , 77	is_atom, 71
1	

is_a_c_equality, 135	latex_print_proof, 154
is_a_nar_eq, 135	latex_print_rewrite_rule_set, 97
is_commutative (method), 95, 149	lazy_append, 93
is_confluent, 136	Lazy_controle (module), 93
is_cte, 70	lazy_cycle, 125
is_defined, 114 , 116	lazy_general_E_resolution, 125
	• •
is_defined_here (method), 100	Lazy_list (module), 93 , 93, 124, 125
is_edge, 151	lazy_mark, 124
is_empty, 40, 43	le, 42,71
is_encompassed_by, 129	<i>LE</i> (camlyacc token), 160 , 160, 161
is_encompassed_by_a_pair, 135	left (field), 39, 151
is_encompassed_by_a_t, 134	Left, 109
is_free (method), 95, 149	<i>LEFTBRACE</i> (camlyacc token), 160 , 160, 161
is_greater_or_equal, 62	leftify_var, 102
is_less_or_equal, 62	LEFTPAR (camlyacc token), 160 , 160, 161
is_localy_confluent, 82	length, 87 , 57, 58, 145, 150
is_nf_compiled, 81,58	length_lex, 86
is_node, 151	Less_or_equivalent, 62
is_null, 46 , 47	Less_than, 62
is_oriented, 134	LET (camlyacc token), 160, 160
is_satisfiable, 73	<i>letter</i> (type), 91 , 92
is_valid, 73	letter (camlyacc non-terminal), 90, 90
is_zero, 41	<i>letters</i> (camllex regexpr), 98 , 99 , 98, 99
iter, 39 , 40 , 43 , 45 , 48 , 49 , 58, 99	LEX, 107
iter (method), 45	Lexical_error (exn), 166
iter_edge, 151	lexicographic_extension, 63
iter_node, 151	lexicographic_extension_of_orderings, 63
i_solve, 67	lhs (field), 97
<i>I_solve</i> (module), 67 , 68	lhs_of_rule, 134
i_solve_modulo, 68	Linear, 144
I_solve_modulo (module), 68	linear_combination, 120
join, 38	Linear_constraint, 165
Join_compl, 109	Linear_constraints (module), 69, 73, 87, 165
<i>Kb</i> (module), 136	liste_des_sous_termes (field), 156
KB (sig), 136 , 136, 137	<i>Listutils</i> (module), 37 , 84, 147
KBO, 107	list_of_cache, 45
KB_th_detected (exn), 136	<i>llist</i> (type), 93 , 93, 124, 125
key (field), 39, 113	Local_confluence (module), 82
key (type), 39 , 40 , 39, 40, 48, 49, 70, 74, 96, 102,	longest_right_overlap, 58, 58
105, 113, 129, 134	<i>LPAR</i> (camlyacc token), 84 , 88 , 85, 90
keywords_table, 99 , 99	Lr_lexico, 107
keyword_or_ident, 99,99	lt, 42 , 71
KW_ACI (camlyacc token), 114, 115	Lt, 61 , 85
<i>KW_ACU</i> (camlyacc token), 114 , 115	<i>LT</i> (camlyacc token), 84 , 160 , 85, 160, 161
KW_ACUN (camlyacc token), 114, 115	Majority_1, 109
<i>KW_AG</i> (camlyacc token), 114 , 115	<i>Majority</i> _ 2, 109
KW_BR (camlyacc token), 114, 115	<i>Majority_3</i> , 109
K_axiom, 109	Make (module), 39, 40, 44, 48, 106, 107, 108, 110,
Labelled_graphs (module), 49, 154	111, 112, 114, 117, 118, 119, 121, 122,
Large_binary_tree (module), 120	123, 124, 125, 126, 130, 131, 133, 136,
Large_bit_field (module), 41, 120	141 , 142 , 151
latex_print, 106	MakePoly (module), 40, 44, 54, 56
latex_print_modular_proof, 155	MakePWordRewriting (module), 135
latex_print_polynomial, 47	MakeWordRewriting (module), 135
± • • •	<i>z</i> ,

make_automata, 156	MINUS (camlyacc token), 77, 147, 160, 77, 78,
	147, 148, 160, 161
make_monomial, 46	minus_one, 41, 69
make_rule, 97, 134, 156	
make_signature_automata, 157	minus_symbol_of_theory, 112
make_var, 69	min_elt, 44
make_var_uniq_name, 69	min_var, 102
map, 39 , 40 , 48 , 49 , 93 , 58	modular_dp_criterion, 152
map (class), 45	modular_dp_criterion_marks, 152
map12_without_repetition, 93	modular_expert, 155
map2_without_repetition, 93	Modular_graph_mark, 154
mapi, 38, 39, 40	Modular_graph_nomark, 154
mapl, 93	Modular_nograph_mark, 154
Mapord (module), 39	Modular_nograph_nomark, 154
map_conj_no_subst, 72	modular_termination_proof (type), 154, 155
map_conj_subst, 71	modulo, 42
map_disj_set, 72	monomial (type), 46 , 46, 47, 48
map_filter, 37	monomial_coef, 46
map_of_var_var (field), 117	monomial_degree, 46
map_prec, 61	monomial_vars, 46
map_without_repetition, 93	mult, 42 , 46 , 47 , 68 , 147, 148
map_with_exception, 37	Mult, 68, 74 , 78
mark, 124	MULT (camlyacc token), 77 , 147 , 77, 78, 147, 148
mark (type), 113, 124, 113, 124, 125	Multiset, 107
Mark (module), 124, 125	multiset_extension, 63
Marked, 113, 149, 145, 150	multi_lex, 86
Marked_dp_criteria (module), 149, 145, 147, 152,	mu_translate, 106
154	my_o, 134, 135, 136, 137
marked_signature (class), 149	my_sign, 135, 137
marked_variables (field), 113	N1_axiom, 109
Mark_acu (module), 123	named_hsig (type), 164
mark_root_symbol, 149	named_parameters (type), 164, 165
matching, 110	named_psubst (type), 164, 165
max, 39, 40, 42	named_pword_sig (type), 164, 165
max (field), 75	named_sig (type), 164, 165
Max, 74	named_vars (type), 165 , 165
max_hash_length, 45	named_word_sig (type), 164, 165
max_variable, 102	nb_shared_symbols, 135
max_var_of_set, 102	nd_run_automata, 156
mdp_of_rules, 152	nd_test_acceptance, 157
mdp_of_rules_marks, 152	nd_test_complement, 157
Meet_compl, 109	nd_test_intersect, 157
mem, 40, 43, 54, 55	nd_test_var_acceptance, 157
merge, 38, 39, 118	ne, 71
Merged, 113	neg, 69 , 71
Merge_acu (module), 123	Neg, 68, 77
merge_substs, 96	negative, 71
Middle, 109	neighbours_list, 151
min, 39, 40, 42	neq, 42
min (field), 75	NEQ (camlyacc token), 160 , 160, 161
Min, 74	new_compiled_rewrite_at_top, 141
minimal_split, 152	new_fd_var_id, 74
Minimal_split (module), 151	Nil, 93
minus, 42 , 46 , 47 , 68 , 70 , 147, 148	Nilpotency, 109
Minus, 68 , 78	Node, 38, 39

NodeAlreadyInGraph (exn), 151 Ordered_sets (module), 43, 46, 48, 56, 74, 96, 102, NodeNotInGraph (exn), 151 105, 110, 112, 126, 134, 151 node_info (type), **39**, 39 Ordered_types (module), 45, 40, 44, 48, 96, 102, node_list. 151 ordering (type), **62**, 61, 62, 63, 83, 84, 86, 100, Non_generalized, 162 non_linear_constraint (type), **75**, 75, 76, 144 106, 107, 108, 134, 135, 136, 137, 150, non_linear_solve, 76 151, 165 Orderings_generalities (module), 61, 61, 83, 84, Non_linear_solving (module), 75, 144, 147 86, 100, 105, 107, 108, 134, 136, 149, 151, normalize, 81, 131, 134 Not, 70 NOT (camlyacc token), 77, 160, 77, 160, 161 Order_constraint, 165 order_constraint_of_string, 100 Not_appliable (exn), 93, 111 order_constraint_type, 163 Not_linear (exn), 72 orelse, 93, 111 No_cycle, 116 Original, 149, 145, 150 *No_mark*, **113** other_equations (field), 117 No_match (exn), 110 Overriden (exn), 101 No_proof_found (exn), 154 O_axiom, 109 No_sol, 67 P (sig), 139 No_solution (exn), 75, 93, 111 No_theory (exn), 112 packs_and_rules_from_packs, 152 packs_from_a_list, 152 Nul, 75 pair_size, 135 null, 70 pair_type, 163 Null, 70 nullable, 54, 55, 55, 56 parameterized_signature (class), 87, 88 Parameterized_signatures (module), 87, 88, 135, Numbers (module), 41, 68, 69, 70, 71, 72, 74, 75, 76, 77, 78, 88, 105, 145, 159, 160, 164, 137, 164 Parameterized_signatures_lexer (module), 91 165, 167 Parameterized_signatures_parser (module), 88, 91 NumRing (module), 75, 105, 75, 105 Parameterized_signatures_syntax (module), **91**, *N*_axiom, **109** 88, 89, 91 o, 106, 107, 136 parameters, 87 Oc (module), 116, 118, 125 occur_check, 117 parameters (method), 87, 88 occur_check_without_var_var, 117 parameters_from_string, 88 parameters_name (field), 164 of_int. 41 parameters_val (field), 164 Old_basic_criterion (module), 150 one, 41, 46, 47, 69 Parameters_value, 165 One, 61, 85 parameter_c (class), 87 OPENPAR (camlyacc token), 114, 115 parameter_set_type, 163 param_list, 145 *Oper*, 74 PARDROITE (camlyacc token), 77, 147, 77, 78, opt, 54, 55 Or, 68, 70 147, 148 PARGAUCHE (camlyacc token), 77, 147, 77, 78, OR, 155, 76, 98, 167 147, 148 OR (camlyacc token), 77, 160, 77, 160, 161 parseable_signature (class), 95, 96, 100 OrderedACCompletion (module), 137 OrderedKBCompletion (module), 137 parse_id, **99**, 98 OrderedMap (sig), 39, 40, 39, 40, 46, 48, 74, 96, Partial, 129 partial_degree, 47 105, 113, 129 partial_match (type), 129, 129, 130 OrderedPolyType (sig), 45, 40, 44, 96 OrderedSet (sig), 43, 44, 46, 48, 56, 74, 96, 102, pcache, 120 105, 110, 112, 126, 134, 151 penalty_of_non_oriented_eq, 134 Permanent, 113 OrderedType (sig), **39**, **45**, 39, 40, 44, 48, 102, 105 ordered_list (camlyacc non-terminal), 85, 85 pgcd, 37, 42 Ordered_maps (module), 40, 46, 48, 74, 96, 105, pgreat_enough, 120 PIPE (camlyacc token), 88, 89, 90 113, 129

Pixley_1, 109	Presburger (module), 73
Pixley_2, 109	pretty_print, 82, 87
Pixley_3, 109	<i>print</i> , 50 , 73 , 82 , 87 , 96 , 105
Pixley_def, 109	print (method), 87, 88, 141
PLAIN, 111	print_all_critical_pairs, 136
plain_set_of_solutions, 126	<pre>print_atomic_termination_constraint, 151</pre>
plus, 68	print_axiom, 110
Plus, 68 , 78	print_bit_field, 41
<i>PLUS</i> (camlyacc token), 77 , 147 , 160 , 77, 78, 147,	print_clean_automata, 158
148, 160, 161	print_conj_termination_constraints, 151
poly (type), 47, 47, 48, 75, 105, 106, 107, 144,	print_constraint, 75
145, 147	print_delta_automata, 158
poly (camlyacc non-terminal), 148 , 148	print_element (method), 88
POLY, 107	print_elem_pb, 113
polyinterptype (type), 144 , 144, 145	print_equation_set, 135
Polynomials (module), 45 , 75, 105, 144	print_expr, 69, 73, 159
PolynomialType (sig), 46 , 48, 75, 105, 144	print_fd_constraint, 75
polynomial_constraints_of_termination_constraints,	print_fd_solution, 75
144	print_fd_var_id, 74
Poly_interp (module), 105 , 107, 145, 147, 154	print_five_symbols, 110
poly_interp_entry (camlyacc non-terminal), 148,	print_formula, 69
147	print_four_symbols, 110
Poly_lexer (module), 76, 150	÷ •
poly_of_monomial, 47	print_generic_polynomial, 144
± 7	print_interp, 144
poly_ord, 106	print_int_polynomial, 105
Poly_ordering (module), 107	print_list_of_symbols, 110
Poly_parser (module), 77, 76, 150	print_modular_proof, 155
Poly_syntax (module), 79 , 90, 150	print_module_termination_proof, 155
poly_times, 106	print_named_expr, 159
PosData (module), 129, 130	print_non_linear_constraints, 75
position (type), 129 , 129, 130, 141, 165	print_polynomial, 47, 75
position_type, 163	print_problem, 113
Position_value, 165	print_proof, 154
positive, 70	print_rewrite_rule, 97, 135
PositiveOrNull, 70	print_rewrite_rule_set, 97
positive_or_null, 71	print_state, 158
Postfix, 95	print_stats, 135
Pos_or_nul, 75	print_t, 135
power, 37, 47, 68 , 78, 84, 147, 148	print_theory, 112
Power, 74	print_three_symbols, 110
<i>POWER</i> (camlyacc token), 84 , 88 , 85, 90	print_two_symbols, 110
power_int, 42	print_type, 163
ppcm, 37, 42	print_unif_var, 102
prec, 108	print_value, 166
precedence (type), 61, 61	print_vertex_set, 49
precedence (camlyacc non-terminal), 85, 85	problem (type), 113, 124, 125, 113, 124, 125
precedence_eof (camlyacc non-terminal), 85, 84	Problems (module), 113, 118, 119, 121, 122, 123,
precedence_type, 163	124, 125
Precedence_value, 165	Product, 92 , 90
prec_of_string, 100	Prod_value, 165
pred, 42 , 145, 150	profile_of_symbol (method), 101
Predef, 165	Proof (module), 139
Predef (module), 164	<pre>proved_c_e_terminating (field), 101</pre>
Prefix, 95	prove_conjectures, 137

individual continuity 109 Presendo_associativity 109 Presentative 104 Presendo_associativity 109 Presentative 104 Presendo_associativity 109 Presendo_associativity 109 Presentative 104 Presendo_associativity 109	prove_conj_by_comp, 136	RIGHTBRACE (camlyacc token), 160, 161
Pseudo_associativity, 109 RIGHTPAR (camlyace token), 160, 160, 161 RingType (sig), 46, 46, 48, 75, 105, 144 Rl_exico, 107 Robbins, 109 PSRS_Uple, 163 PSRS_Uple, 163 PSRS_Uple, 164 PSUBS_Uple, 165 PSUBS_Uple, 166 PSUBS_Uple, 165 PSUBS_Uple,		rightify_var, 102
psig. 87 psmall_enough, 120 psms_t_type, 163 Psrs_value, 165 psrs_t_type, 163 Psrs_value, 165 psubst_value, 165 psubst_value, 165 psubst_value, 165 pure_telm_pb (type), 117, 117, 118 purify_list_of_equations, 112 PWordCompletion (module), 137 pword_sig_name (field), 164 pword_sig_name (field), 164 pword_sig_name (field), 164 pword_sig_name (field), 165 pword_type, 163 pword_value, 165 QLaxiom, 109 Quadratic, 144 quo, 42 Quotient, 68, 78 Q_axiom, 109 quo, 42 Quotient, 68, 78 Q_axiom, 109 receording_axiom, 110 regexp_set (type), 53, 54, 53, 54 regles (field), 156 Regular_expr (module), 53, 54 regles (field), 156 Regular_expr (module), 53, 54 regles (field), 156 Regular_expr (module), 152 remove_a_adata, 129 remove_a_adata, 129 remove_denominators, 73 remove_denominators, 74 RPAR (camlyace token), 74 RPAR (camlyace token), 74 RPAR (camlyace token), 74 RPAR (camlyace token), 70 rpo_status (type), 107, 100, 107, 108, 165 RPO, 107 rpo_status (type), 107, 107 rpo_status (type), 107, 107 rpo_status (type), 108 rpo_telm_pb (telm_pb (telm	- · · · · · · · · · · · · · · · · · · ·	
psmall_enough, 120 psrs_type, 163 psubst_type, 163 pword_sig_name (field), 164 pword_sig_name (field), 164 pword_sig_name (field), 164 pword_sig_value, 165 pword_type, 163 pword_type, 163 pword_type, 163 pword_type, 163 pword_type, 165 pword_type, 166 pword_type, 165 pword_type, 166 pword_type, 165 pword_type, 166 pword_type, 163 pword_type, 165 pword_type, 165 pword_type, 165 pword_type, 165 pword_type, 166 pword_type, 166 pword_type, 166 pword_type, 163 pword_type, 165 pword_type, 166 pword_type, 166 pword_type, 163 pword_type, 163 pword_type, 164 pword_type, 165 pword_type, 166 pword_type, 166 pword_type, 167 pro_time, 108 rule (type), 92, 111, 129, 133, 134, 136, 141, 92, 131, 141, 142 pull, 129, 130, 133, 134, 135, 136, 137, 141, 142 pull, 143 pull, 144 pull, 142 pull, 143 pull, 144 pull, 142 pull, 142 pull, 142 pull, 142 pull, 142 p	· · · · · · · · · · · · · · · · · · ·	•
prst_type		
Post-value, 165 psubst.name (field), 164 psubst.val (field), 164 proved.sip.name (field), 164 pword.signature.type, 163 pword.sig.name (field), 164 pword.sig.val (field), 164 pword.sig.val (field), 165 pword.value, 165 Ql_axiom, 109 quo, 42 Quotient, 68, 78 Qaxiom, 109 recognize_axiom, 110 recgex (type), 53, 54, 53, 54 regexp.xet (type), 54, 54 regexp.xet (type), 53, 54, 53, 54 regexp.xet (type), 54, 54 regexp.xet (type), 53, 54, 53, 54 regexp.xet (type), 54, 54 regexp.xet (type), 53, 54, 53, 54 regexp.xet (type), 54, 54 regexp.xet (type), 54, 54 regexp.xet (type), 53, 54, 53, 54 regexp.xet (type), 54, 54 regexp.xet (type), 53, 54, 53, 54 regexp.xet (type), 54, 54 regexp.xet (type), 54, 54 regexp.xet (type), 53, 54, 54 regexp.xet (type), 54, 54 regexp.xet (type), 53, 54, 54 regexp.xet (type), 54, 54 regexp.xet (type), 53, 54, 54 regexp.xet (type), 53, 54, 54 regexp.xet (type), 54, 54 regexp.xet (type), 53, 54, 54 regexp.xet (type), 54, 54 regexp.xet (type), 53, 54 regexp.xet (type), 54, 54 regexp.xet (type), 54, 54 regexp.xet (type), 53, 54 regexp.xet (type), 54, 54 regexp.xet (type), 53, 54 regexp.xet (type), 54, 54 regexp.xet	-	
psubst_name (field), 164 psubst_type, 163 psubst_value, 165 psubst_value, 165 pure_elem_pb (type), 117, 117, 118 purity_list_of_equations, 112 PWordCompletion (module), 137 pword_signature_type, 163 pword_sig_name (field), 164 pword_sig_name (field), 164 pword_sig_value, 165 pword_type, 163 pword_value, 165 QLaxiom, 109 Quadratic, 144 quo, 42 Quotient, 68, 78 Q_axiom, 109 quo, 42 Quotient, 68, 78 Q_axiom, 109 recognize_axiom, 110 regexp (type), 53, 54, 53, 54 regexp_set (type), 54, 54 reglat_pair, 135 Relative_dp (module), 152 remove_denominators, 73 repear, 93, 111 replace_a_data, 129 remove_denominators, 73 repear, 93, 111 replace_a_data, 129 remove_cenformula, 69, 73 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 residual, 54,55,55 Seq. 58,55,55 Seq. 58,55 Seq. 58,55,55 Seq. 58,55,5		
psubst_type, 163 psubst_val (field), 164 psubst_value, 165 pure_elem_pb (type), 117, 117, 118 purify_list_of_equations, 112 PWordCompletion (module), 137 pword_signature_type, 163 pword_signature_type, 163 pword_sig_value, 165 pword_sig_value, 165 pword_value, 165 QL_axiom, 109 Quadratic, 144 quo, 42 Quotient, 68, 78 Q_axiom, 109 recognize_axiom, 110 regexp (type), 53, 54, 53, 54 regexp_set (type), 53, 54, 53, 54 regexp_set (type), 53, 54, 53, 54 regexp_remodule), 152 remove_a_data, 129 remove_a_data, 129 remove_equivalent_elements, 63 remove_vertex_and_dis_arcs, 50 remove_vertex_and_dis_arcs, 50 remove_vertex_and_dis_arcs, 50 remove_vertex_and_dis_arcs, 50 remove_vertex_and_dis_arcs, 50 remove_vertex_and_dis_arcs, 50 remove_markings_rules, 149 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 remove_trive, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 risplic (field), 97 rhs_ofield), 97 rhs_ofield), 97 rhs_ofield), 97 rhs_ofield, 97 rhs_ofield), 97 rhs_ofield, 98		
psubst_val (field). 164 Psubst_value. 165 pure_elem_pb (type). 117, 117, 118 purify_list_of_equations, 112 PWordCompletion (module). 137 pword_signature_type. 163 pword_sig_name (field), 164 Pword_sig_value. 165 pword_type, 163 Pword_value. 165 QL_axiom, 109 Quadratic, 144 quo, 42 Quotient, 68, 78 Q_axiom, 109 recognize_axiom, 110 regexp (type), 53, 54, 53, 54 regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 152 remove_adata, 129 remove_adata, 129 remove_markings_rules, 149 remove_wertex_and_lis_ares, 50 remowe_wertex_and_is_ares, 50 remome_wertenormala, 69, 73 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 retieve, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 ths (field), 97 this field, 97 the field, 68 RPO, 107 Rpo (module), 107, 100, 107, 108, 165 RPO, 107 Rpo (module), 107, 100, 107, 108, 165 RPO, 107 Rpo (module), 107, 100, 107, 108, 165 RPO, 107 rpo_status (type), 107, 101, 108, 144, 19, arlle (type), 92, 189 rule_ste (camlyace non-terminal), 85, 96, 98, 99, 99 rule_set (camlyace non-terminal), 85, 90, 85, 90 rules (camlyace non-terminal), 90, 99 rule_set (camlyace non-terminal), 85, 85 rule_ste_ard (camlyace non-terminal), 111, 112, 113, 116, 117, 118, 119, 120, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 134, 135, 134, 135, 134, 135, 134, 135, 134, 135, 134, 135, 134, 135, 134, 135, 134, 135, 1		
Psubst_value, 165 pure_elem_b (type), 117, 117, 118 pure_elem_be (type), 117, 117, 118 pure_elem_be (type), 117, 117, 118 purefly_list_of_equations, 112 PWordCompletion (module), 137 pword_signature_type, 163 pword_sig_value, 165 pword_sig_value, 165 pword_value, 165 pvord_value, 165 pvolue, 168 pro_value, 169 pvolue, 168 pro_velvalue, 169 pvolue, 168		
pure_clem_pb (type), 117, 117, 118 purify_list_of_equations, 112 PWordCompletion (module), 137 pword_signature_type, 163 pword_sig_name (field), 164 Pword_sig_value, 165 pword_type, 163 Pword_sig_value, 165 Pword_value, 165 Pword_value, 165 Ploanting that the properties of the pure time of the pure time to the pure time time time time time time time tim	•	
purify_list_of_equations, 112 PWordCompletion (module), 137 pword_signature_type, 163 pword_signature_type, 163 pword_signature_type, 164 pword_sig_value, 165 pword_type, 163 Pword_value, 165 pword_type, 163 Pword_value, 165 Ql_axiom, 109 Quadratic, 144 quo, 42 Quotient, 68, 78 Q_axiom, 109 recognize_axiom, 110 regexp (type), 53, 54, 53, 54 regexp_set (type), 53, 54, 53, 54 regexp_set (type), 54, 54 regls (field), 156 Refative_dp (module), 152 remove_a_data, 129 remove_denominators, 73 remove_equivalent_elements, 63 remove_vertex_and_link_its_arcs, 50 remove_vertex_and_link_arcs, 50 remove_vertex_an	pure_elem_pb (type), 117 , 117, 118	Rpo (module), 107, 100, 107, 108, 165
PWordCompletion (module). 137 pword_signature_type. 163 pword_signature_type. 163 pword_sig_name (field). 164 pword_sig_val (field). 164 pword_sig_value. 165 pword_value. 165 Pword_value. 165 Ql_axiom. 109 Quadratic. 144 quo, 42 Quotient, 68,78 Q_axiom. 109 recognize_axiom, 110 regexp (type). 53, 54, 53, 54 regexp_set (type). 53, 54, 55, 55 retrieve_tll: replace_a_atata, 129 replace_a_atata, 129 replace_a_atata, 129 replace_a_atata, 129 residual, 54, 55, 55 retrieve, 111 rewrite_rules (module), 97, 101, 143, 144, 149, 152, 152, 153, 154 right (field), 97 rhs. of_rule, 134 right (field), 97 rho_time, 108 rpo_time, 108 rule_time, 108 rule (type), 92, 111, 129, 133, 134, 136, 141, 12, 91 111, 129, 130, 133, 134, 135, 136, 137, 141, 14, 92 111, 129, 130, 133, 134, 135, 134, 135, 136, 137, 141, 14, 92 111, 129, 130, 133, 134, 135, 134, 135, 136, 137, 141, 14, 92 rule (camlyace non-terminal), 90, 89 rules (type), 92, 89 rule		
pword_sig_nature_type, 163 pword_sig_name (field), 164 pword_sig_value, 165 pword_type, 163 Pword_value, 165 Ql_axiom, 109 Quadratic, 144 quo, 42 Quotient, 68, 78 Q_axiom, 109 recognize_axiom, 110 regexp (type), 53, 54, 53, 54 regexp_set (type), 54, 54 reglar_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove_adata, 129 remove_denominators, 73 remove_aditis_arcs, 50 remove_wartex_and_link_its_arcs, 50 remove_wartex_and_link_its_arcs, 50 remove_wartex_and_link_its_arcs, 50 rename_cxprr, 69, 73 remove_adita, 129 replace_a_avar_by_a_var_in_an_eq, 113 replace_a_data, 129 replace_a_avar_by_a_var_in_an_eq, 113 residual, 54, 45, 55 retrieve, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 165 shift_variable, 102 rpo_time, 108 rule (type), 92, 111, 129, 133, 134, 136, 141, 9, 20, 131, 133, 134, 136, 131, 135, 136, 141, 14, 9, 20 rule, (type), 92, 89 rules (camlyace non-terminal), 85, 90, 85, 90 rules (camlyace non-terminal), 90, 90 rules (camlyace non-terminal), 85, 84 rule (scemlyace non-terminal), 85, 84 rule (camlyace non-terminal), 90, 90 rules (camlyace non-terminal), 85, 84 rule (cam		rpo_status (type), 107 , 107
pword_sig_name (field), 164 pword_sig_val (field), 164 pword_sig_val (field), 165 pword_type, 163 pword_type, 163 Pword_value, 165 pword_value, 165 pword_value, 165 pword_type, 163 Pword_value, 165 Pword_value, 1		
Pword_sig_value, 165 pword_type, 163 pword_type, 163 Pword_value, 165 Q1_axiom, 109 Quadratic, 144 quo, 42 Quotient, 68,78 Q_axiom, 109 recognize_axiom, 110 regexp (type), 53,54,53,54 regexp_set (type), 54,54 regular_expr (module), 53,54 remove_adia, 129 remove_denominators, 73 remove_denominators, 73 remove_wortex_and_tits_arcs, 50 remove_wortex_and_tits_arcs, 50 remove_wortex_and_tits_arcs, 50 remove_wortex_and_tits_arcs, 50 remove_cadial, 129 replace_a_a.var_by_a_var_in_an_eq, 113 residual, 54,55,55 retrieve, 111 rewrite_rules (module), 134 right (field), 97 rish (field), 97 rish (scanlyace non-terminal), 90, 89 rules (camlyace non-terminal), 162, 161, 117 regexe (camlyace non-terminal), 162, 161, 17 rule-set (camlyace non-terminal), 162, 161, 17 rule-set (camlyace non-terminal), 162, 161, 161 regexe (type), 53, 54, 53, 54 rule-set (camlyace non-terminal), 162, 161, 17 rule-set (camlyace non-terminal), 162, 161, 17 rule-set (camlyace non-terminal), 162, 161, 17 regexe (type), 53, 54, 53, 54 rule-set (camlyace non-terminal), 162, 161, 17 remove_averta, and_tits_arcs, 50 remove_averta, and_tits_arcs, 50 remove_averta, and_tits_arcs, 50 remove_averta, and_tits_arcs, 50 r	pword_sig_name (field), 164	•
pword_type, 163 Pword_value, 165 Pword_value, 165 Q_l_axiom, 109 Quadratic, 144 quo, 42 Quotient, 68, 78 Q_uaxiom, 109 recognize_axiom, 110 regexp_set (type), 53, 54, 53, 54 regexp_set (type), 54, 54 reglar_expr (module), 53, 54 remove_a_data, 129 remove_denominators, 73 remove_warkings_rules, 149 remove_vertex_and_link_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 remove_vertex_and_tink_its_arcs, 50 remove_normalia, 69, 73 residual, 54, 55, 55 retrieve, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97 total (type), 92, 89 rules (camlyacc non-terminal), 90, 90 rules_cof (camlyacc non-terminal), 90, 89 rule_set_eof (camlyacc non-terminal), 85, 85 rule_set_eof (camlyacc non-terminal), 90, 89 rule_set_eof (camlyacc non-terminal), 85, 85 rule_set_eof (camlyacc non-terminal), 90, 89 rule_set_eof (camlyacc non-terminal), 85, 85 rule_set_eof (camlyacc non-terminal), 85, 84 rule_set_eof (camlyacc non-terminal), 85, 84 rule_set_eof (camlyacc non-terminal), 85, 85 rule_set_eof (camlyacc non-terminal), 85, 85 rule_set_eof (camlyacc non-terminal), 85, 84 rule_set_eof (camlyacc non-terminal), 85, 85 rule_set_eof (camlyacc non-terminal), 85, 85 rule_set_eof (camlyacc non-terminal), 85, 86 rule_set_eof (camlyacc non-terminal),	pword_sig_val (field), 164	111, 129, 130, 133, 134, 135, 136, 137,
Pword_value, 165 rules (type), 92, 89 QL_axiom, 109 rules (camlyace non-terminal), 90, 90 Quadratic, 144 rule_set (camlyace non-terminal), 90, 89 quo, 42 rule_set (camlyace non-terminal), 85, 85 Quatient, 68, 78 rule_set_cof_string, 100 Q_axiom, 109 Rule_with_a_var_lbs (exn), 97 recognize_axiom, 110 Rule_with_a_var_lbs (exn), 97 regexp (type), 53, 54, 53, 54 regex (type), 54, 54 regles (field), 156 S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 14, 49, 106, 108, regular_pair, 135 110, 111, 112, 114, 117, 118, 119, 121, 122, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 remove, 37, 39, 40, 43, 49 remove_denominators, 73 remove_denominators, 73 sef_force_innermost_normalize, 130, 131 remove_denominators, 73 semove_equivalent_elements, 63 self_critical_pairs, 133, 135 remove_vertex_and_its_arcs, 50 self_critical_pairs, 133, 135 remove_vertex_and_its_arcs, 50 self_critical_pairs, 133, 135 remove_aten, 69, 73 self_critical_pairs, 133, 135 remove_aten, 69, 73 self_critical_pairs, 133, 135 repate, 93, 111 seg, 54, 55, 55 <t< td=""><td>Pword_sig_value, 165</td><td>141, 142</td></t<>	Pword_sig_value, 165	141, 142
Pword_value, 165 rules (type), 92, 89 QL_axiom, 109 rules (camlyace non-terminal), 90, 90 Quadratic, 144 rule_set (camlyace non-terminal), 90, 89 quo, 42 rule_set (camlyace non-terminal), 85, 85 Quatient, 68, 78 rule_set_cof_string, 100 Q_axiom, 109 Rule_with_a_var_lbs (exn), 97 recognize_axiom, 110 Rule_with_a_var_lbs (exn), 97 regexp (type), 53, 54, 53, 54 regex (type), 54, 54 regles (field), 156 S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 14, 49, 106, 108, regular_pair, 135 110, 111, 112, 114, 117, 118, 119, 121, 122, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 remove, 37, 39, 40, 43, 49 remove_denominators, 73 remove_denominators, 73 sef_force_innermost_normalize, 130, 131 remove_denominators, 73 semove_equivalent_elements, 63 self_critical_pairs, 133, 135 remove_vertex_and_its_arcs, 50 self_critical_pairs, 133, 135 remove_vertex_and_its_arcs, 50 self_critical_pairs, 133, 135 remove_aten, 69, 73 self_critical_pairs, 133, 135 remove_aten, 69, 73 self_critical_pairs, 133, 135 repate, 93, 111 seg, 54, 55, 55 <t< td=""><td>pword_type, 163</td><td>rule (camlyacc non-terminal), 85, 90, 85, 90</td></t<>	pword_type, 163	rule (camlyacc non-terminal), 85, 90, 85, 90
Quadratic, 144 rules_eof (camlyace non-terminal), 90, 89 quo, 42 rule_set (camlyace non-terminal), 85, 85 Quation, 109 rule_set_of_string, 100 recognize_axiom, 110 Rule_with_a_var_lhs (exn), 97 regexp_set (type), 53, 54, 53, 54 rule_set_of_string, 100 regexp_set (type), 53, 54, 54 S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, reglas (field), 156 S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, Regular_expr (module), 53, 54 118, 119, 120, 122, 124, 125, 126, 129, Relative_dp (module), 152 131, 133, 134, 136, 141, 41, 49, 106, 108, remove_a data, 129 remove_denominators, 73 remove_denominators, 73 remove_denominators, 73 remove_equivalent_elements, 63 safe_innermost_normalize, 130, 131 remove_vertex_and_link_its_arcs, 50 self_cutil_distributivity, 109 rename_expr, 69, 73 self_cutil_distributivity, 109 repata, 93, 111 semove_narkings_rules, 149 replace_a_ata, 129 semove_landink_its_arcs, 50 remove_actrex_and_link_its_arcs, 50 self_cutil_distributivity, 109 remove_landink_its_arcs, 50 self_cutil_distributivity, 109 remove_landink_its_arcs, 50 <td>Pword_value, 165</td> <td></td>	Pword_value, 165	
quo, 42 rule_set_cof (camlyace non-terminal), 85, 85 Quotient, 68, 78 rule_set_eof (camlyace non-terminal), 85, 84 Q_axiom, 109 rule_set_eof (camlyace non-terminal), 85, 84 recognize_axiom, 110 rule_set_eof (camlyace non-terminal), 85, 84 regexp_set (type), 53, 54, 53, 54 rule_set_eof (camlyace non-terminal), 85, 84 regexp_set (type), 53, 54, 53, 54 rule_set_eof (camlyace non-terminal), 85, 84 regexp_set (type), 53, 54, 53, 54 rule_set_eof (camlyace non-terminal), 85, 84 regexp_set (type), 53, 54, 53, 54 rule_set_eof (camlyace non-terminal), 87 regexp_set (type), 53, 54, 53, 54 rule_set_eof (camlyace non-terminal), 87 reges (type), 53, 54, 53, 54 rule_set_eof (camlyace non-terminal), 87 reges (type), 53, 54, 53, 54 rule_set_eof (camlyace non-terminal), 80, 117 reglar_exp (type), 53, 54, 53, 54 rule_set_eof.exp, 190, 111, 112, 113, 116, 117, 118, 119, 121, 121, 113, 116, 117, 118, 119, 121, 121, 131, 116, 117, 118, 119, 121, 121, 131, 133, 134, 136, 141, 14, 149, 106, 108, 131, 133, 134, 136, 141, 14, 149, 106, 108, 131, 133, 134, 136, 141, 141, 49, 106, 108, 131, 133, 134, 136, 141, 142, 142, 125, 126, 120, 130, 131, 133, 134, 136, 141, 142, 142, 125, 126, 120, 130, 131, 133, 134, 136, 141, 142, 142, 125, 126, 130, 131, 133, 134, 136, 141, 142, 142, 142, 125, 126, 130, 131, 133, 134, 136, 141, 142, 142, 142, 125, 126, 130, 131, 133, 134, 136, 141, 142, 142, 142, 125, 126, 130, 131, 133, 134, 136, 141	Q1_axiom, 109	rules (camlyacc non-terminal), 90, 90
Quotient, 68, 78 rule_set_eof (camlyace non-terminal), 85, 84 Q_axiom, 109 rule_set_of_string, 100 recognize_axiom, 110 Rule_with_a_var_lhs (exn), 97 regexp (type), 53, 54, 53, 54 regexp (type), 54, 54 reges (field), 156 S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, regular_pair, 135 Relative_dp (module), 152 110, 111, 112, 114, 117, 118, 119, 121, 122, 122, 123, 124, 125, 126, 130, 131, 133, remove_adata, 129 safe_innermost_normalize, 130, 131 remove_adata, 129 safe_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 remove_equivalent_elements, 63 semove_vertex_and_link_its_arcs, 50 self_critical_pairs, 133, 135 remove_vertex_and_link_its_arcs, 50 self_critical_pairs, 133, 135 self_critical_pairs, 133, 135 self_cortical_pairs, 133, 135 self_cortical_pairs, 133, 135 self_critical_pairs, 133, 135 remove_vertex_and_link_its_arcs, 50 self_critical_pairs, 133, 135 self_critical_pairs, 133, 135 repalace_a_var_by_a_var_in_an_eq, 113 seq, 54, 55, 55 seq, 54, 55, 55 replace_a_var_by_a_var_in_an_eq, 113 seq, 54, 55, 55 reget/ical_ical_ical_ical_ical_ical_ical_ical_	Quadratic, 144	rules_eof (camlyacc non-terminal), 90, 89
Q_axiom, 109 rule_set_of_string, 100 Rule_with_a_var_lhs (exn), 97 regexp (type), 53, 54, 53, 54 run_automata, 156 regexp_set (type), 54, 54 S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, reglas (field), 156 118, 119, 120, 122, 124, 125, 126, 129, Regular_expr (module), 53, 54 110, 111, 112, 114, 117, 118, 119, 121, regular_pair, 135 110, 111, 112, 114, 117, 118, 119, 121, Relative_dp (module), 152 122, 123, 124, 125, 126, 130, 131, 133, remove_adata, 129 134, 135, 136, 141, 142 remove_denominators, 73 remove_enominators, 73 remove_equivalent_elements, 63 safe_innermost_normalize, 130, 131 remove_wertex_and_lits_arcs, 50 scanned_equations (field), 117 rename_expr, 69, 73 self_critical_pairs, 133, 135 rename_formula, 69, 73 SEMICOLON (camlyacc token), 77, 84, 88, 114, replace_a_var_by_a_var_in_an_eq, 113 162 residual, 54, 55, 55 Seq, 54, 55 set_of_sublos, 151 set_of_sublos, 151 retrieve_ulle (type), 81, 135, 137 set_of_sublos, 151 rewrite_rule (type), 81, 135, 137 set_of_sublos, 151 retrieve_lidel, 97 set_of_sublos	quo, 42	rule_set (camlyacc non-terminal), 85, 85
recognize_axiom, 110 regexp (type), 53, 54, 53, 54 regexp_set (type), 54, 54 regexp_set (type), 53, 54 regexp_set (type), 54, 54 regels (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove_adata, 129 remove_denominators, 73 remove_denominators, 73 remove_wertex_and_link_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 remame_expr, 69, 73 replace_a_data, 129 replace_a_data, 129 replace_a_data, 129 replace_a_data, 129 replace_a_data, 129 replace_a_data, 129 replace_a_lata, 129 replace_a_lata, 129 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 retrieve, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 rhs (field), 97 rhs_of_rule, 134 right (field), 39, 151 Rule_with_a_var_lhs (exn), 97 run_automata, 156 S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 121, 118, 119, 121, 121, 121, 121, 121, 121, 121		rule_set_eof (camlyacc non-terminal), 85, 84
regexp (type), 53, 54, 53, 54 regexp_set (type), 54, 54 regexp_set (type), 54, 54 regexp_set (type), 54, 54 regesp_set (type), 54, 54 regels (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 110, 111, 112, 114, 117, 118, 119, 121, Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_denominators, 73 remove_denominators, 73 remove_denominators, 73 remove_denominators, 73 remove_vertex_and_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 remo		<u> </u>
regexp_set (type), 54,54 S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, regles (field), 156 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, regular_pair, 135 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, remove, 37, 39, 40, 43, 49 122, 123, 124, 125, 126, 130, 131, 133, 134, 136, 141, 142 remove_adata, 129 safe_force_innermost_normalize, 130, 131 remove_denominators, 73 safe_innermost_normalize, 130, 131 remove_markings_rules, 149 scc, 151 remove_vertex_and_link_its_arcs, 50 self_critical_pairs, 133, 135 remove_vertex_and_link_its_arcs, 50 self_cdual_distributivity, 109 rename_expr, 69, 73 SEMICOLON (camlyacc token), 77, 84, 88, 114, 160, 162 repeat, 93, 111 semicolonlist (camlyacc non-terminal), 162, 161, 162 replace_a_ata, 129 semicolonlist (camlyace non-terminal), 162, 161, 162 replace_a_avar_by_a_var_in_an_eq, 113 seq, 54, 55, 55 retrieve, 111 seq, 54, 55, 55 retrieve, 111 set_of_solutions, 126 rbs, (field), 97 set_of_solutions, 126 rhs, (field), 97 set_of_solutions, 126 rhs, (field), 97 set_of_solutions, 126 rhs_of_rule, 134 set_of_solutions, 165		Rule_with_a_var_lhs (exn), 97
regles (field), 156 Regular_expr (module), 53, 54 Regular_pair, 135 Relative_dp (module), 152 Relative_dp (module), 152 Remove_a_data, 129 Remove_a_data, 129 Remove_denominators, 73 Remove_equivalent_elements, 63 Remove_warkings_rules, 149 Remove_vertex_and_lits_arcs, 50 Remove_vertex_and_lits_its_arcs, 50 Remane_formula, 69, 73 Remane_formula, 69, 73 Remane_formula, 69, 73 Residual, 54, 55, 55 Retrieve, 111 Remove_rules (module), 97, 101, 143, 144, 149, 162 Residual, 134 Residual, 134 Residual, 135, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 121, 122, 123, 124, 125, 126, 130, 131, 133, 135, 136, 141, 142 Remove_a_data, 129 Remove_denominators, 73 Remove_equivalent_elements, 63 Remove_vertex_and_lits_arcs, 50 Remove_vertex_and_lits_its_arcs, 50 Remove_vertex_and_lits_arcs, 50 Remove_vertex_and_lits_its_arcs, 50 Remove_vertex_and_lits_arcs, 50 Remove_vertex_and_lits_arcs, 50 Remove_vertex_and_lits_arcs, 50 Remove_vertex_and_lits	regexp (type), 53, 54, 53, 54	run_automata, 156
Regular_expr (module), 53, 54 131, 133, 134, 136, 141, 41, 49, 106, 108, regular_pair, 135 Relative_dp (module), 152 122, 123, 124, 125, 126, 130, 131, 133, remove, 37, 39, 40, 43, 49 remove_adata, 129 safe_force_innermost_normalize, 130, 131 remove_denominators, 73 safe_innermost_normalize, 130, 131 remove_denominators, 73 scanned_equations (field), 117 remove_markings_rules, 149 scc, 151 remove_wovertex_and_its_arcs, 50 self_critical_pairs, 133, 135 remove_vertex_and_link_its_arcs, 50 self_critical_pairs, 133, 135 rename_expr, 69, 73 SEMICOLON (camlyacc token), 77, 84, 88, 114, rename_formula, 69, 73 147, 160, 85, 89, 90, 115, 148, 160, 162 repeat, 93, 111 semicolonlist (camlyacc non-terminal), 162, 161, replace_a_data, 129 seq, 54, 55, 55 residual, 54, 55, 55 Seq, 54, 55 retrieve, 111 set_of_solutions, 126 rewrite_rule (type), 81, 135, 137 set_of_solutions, 126 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 set_of_solutions, 151 rhs (field), 97 set_of_solutions, 151 rhs_of_rule, 134 set_of_vars, 47 rhs (field), 39, 151 shift_variable, 102		
regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_a_data, 129 remove_denominators, 73 remove_denominators, 73 remove_markings_rules, 149 remove_vertex_and_lits_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 replace_a_data, 129 replace_a_data, 129 replace_a_data, 129 remove_vertex_and_link_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 retrive, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 rhs (field), 97 rhs_of_rule, 134 right (field), 39, 151 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 135 rade_innermost_normalize, 130, 131 safe_innermost_normalize, 130,	regexp_set (type), 54 , 54	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117,
Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70 remove_denominators, 73 remove_equivalent_elements, 63 remove_vertex_and_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 replace_a_data, 129 replace_a_data, 129 remove_vertex_and_its_arcs, 50 residual, 54, 55, 55 retrieve, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 right (field), 39, 151 122, 123, 124, 125, 126, 130, 131, 133, 134, 134, 134, 135, 136, 141, 142 safe_innermost_normalize, 130, 131 safe_inle index_inle i	regexp_set (type), 54, 54 regles (field), 156	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129,
remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_lits_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 retrieve, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 residual, 39, 151 residual, 39, 151 residual, 97 rhs_of_rule, 134 right (field), 39, 151 remove_coef, 70 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 safe_inermost_normalize, 120 scanned_equations (field), 117 scent_clude, 150 safe_cital_pairs, 133, 135 safe_cital_pairs, 133, 135 safe_cital_pairs, 130, 131 safe_cital_pairs, 130, 131 safe_cital_pairs, 132, 135	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108,
remove_a_data, 129 safe_force_innermost_normalize, 130, 131 remove_coef, 70 safe_innermost_normalize, 130, 131 remove_denominators, 73 SB_property, 109 remove_equivalent_elements, 63 scanned_equations (field), 117 remove_markings_rules, 149 scc, 151 remove_vertex_and_its_arcs, 50 self_critical_pairs, 133, 135 remove_vertex_and_link_its_arcs, 50 self_critical_pairs, 133, 135 remove_vertex_and_link_its_arcs, 50 self_culal_distributivity, 109 rename_expr, 69, 73 SEMICOLON (camlyacc token), 77, 84, 88, 114, rename_formula, 69, 73 147, 160, 85, 89, 90, 115, 148, 160, 162 repeat, 93, 111 semicolonlist (camlyacc non-terminal), 162, 161, replace_a_data, 129 162 replace_a_var_by_a_var_in_an_eq, 113 seq, 54, 55, 55 residual, 54, 55, 55 Seq, 54, 55 retrieve, 111 set_of_solutions, 126 rewrite_rule (type), 81, 135, 137 set_of_symbols, 151 Rewrite_rules (module), 97, 101, 143, 144, 149, set_of_symbols, 151 set_of_symbols, 151 152, 153, 154 set_of_vars, 47 rhs (field), 97 set_of_vars, 47 rhs_of_rule, 134 set_value, 165 right (field), 39, 151 shift_variable, 102	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121,
remove_coef, 70 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 remove_coef, 70 safe_innermost_normalize, 130, 131 sB_property, 109 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135 self_critical_pairs, 134, 135 self_critical_pairs, 133, 135 self_critical_pairs, 134, 135, 135 self_critical_pairs, 133, 135 self_critical_pairs, 134, 149 self_critical_pairs, 134, 149 self_critical_pairs, 134, 149 self_critical_pairs, 132, 135 self_critical_pairs, 126 self_critical_pairs, 132, 135 self_critical_pairs, 126 self_critical_p	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133,
remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 rhs (field), 97 rhs_of_rule, 134 right (field), 39, 151 Sec, 151 sec, 151 sec, 151 self_critical_pairs, 133, 135 self_critical_pairs, 134, 135 self_critical_pairs, 132, 133 self_critical_pairs, 134, 149 self_critical_pairs, 132, 135 self_critical_pairs, 132, 135 self_critical_pairs, 134, 149 self_critical_pairs, 132, 135 self_critical_pairs, 126 self_criti	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142
remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 retrieve, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 rhs_of_rule, 134 right (field), 39, 151 scanned_equations (field), 117 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135 self_critical_pairs, 134, 149 self_critical_pairs, 134, 149 self_critical_pairs, 134, 145 self_critical_pairs, 134, 145 self_critical_pairs, 134, 145 self_critical_pairs, 132, 135 self_critical_pairs, 132, 135 self_critical_pairs, 132, 135 self_critical_pairs, 134, 145 self_critical_pairs, 134, 145 self_critical_pairs, 134, 145 self_critical_pairs, 132, 135 self_clual_distributivity, 109 self_critical_pairs, 132, 135 self_critical_pairs, 132, 135 self_critical_pairs, 132, 135 self_clual_distributivity, 109 self_crutical_pairs, 126 self_crutical_pairs self_crutical_pairs self_cruti	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131
remove_markings_rules, 149 remove_vertex_and_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 restricted_pairs, 133, 135 Self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114, 147, 160, 85, 89, 90, 115, 148, 160, 162 semicolonlist (camlyacc non-terminal), 162, 161, 162 replace_a_var_by_a_var_in_an_eq, 113 seq, 54, 55, 55 restricve, 111 Set, 159, 161 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 rhs (field), 97 rhs (field), 97 rhs_of_rule, 134 right (field), 39, 151 scc, 151 self_critical_pairs, 133, 135 self_critical_pairs, 134, 149 self_critical_pairs, 134, 145 self_critical_pairs, 143, 145	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131
remove_vertex_and_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 rhs (field), 97 remove_vertex_and_link_its_arcs, 50 self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114, 149 SEMICOLON (camlyacc non-terminal), 162, 161, 162 semicolonlist (camlyacc non-terminal), 162, 161, 162 seq, 54, 55, 55 Seq, 54, 55 Seq, 54, 55 Set, 159, 161 set_of_solutions, 126 set_of_symbols, 151 set_of_vars, 47 set_type, 163 rhs_of_rule, 134 Set_value, 165 right (field), 39, 151 self_critical_pairs, 133, 135 self_critical_pairs, 133, 135 self_critical_pairs, 133, 135 self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114, 149, 162 semicolonlist (camlyacc non-terminal), 162, 161, 162 semicolonlist (camlyacc non-t	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70 remove_denominators, 73	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109
remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 rbs_of_rule, 134 right (field), 97 rename_expr, 69, 73 Self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114, 149, 147, 160, 85, 89, 90, 115, 148, 160, 162 semicolonlist (camlyacc non-terminal), 162, 161, 162 seq, 54, 55, 55 Seq, 54, 55 Set, 159, 161 set_of_solutions, 126 set_of_symbols, 151 set_of_vars, 47 set_type, 163 Set_value, 165 shift_variable, 102	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70 remove_denominators, 73 remove_equivalent_elements, 63	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117
rename_expr, 69, 73 SEMICOLON (camlyacc token), 77, 84, 88, 114, rename_formula, 69, 73 147, 160, 85, 89, 90, 115, 148, 160, 162 repeat, 93, 111 semicolonlist (camlyacc non-terminal), 162, 161, replace_a_data, 129 162 replace_a_var_by_a_var_in_an_eq, 113 seq, 54, 55, 55 residual, 54, 55, 55 Seq, 54, 55 retrieve, 111 Set, 159, 161 rewrite_rule (type), 81, 135, 137 set_of_solutions, 126 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 set_of_symbols, 151 rhs (field), 97 set_type, 163 rhs_of_rule, 134 Set_value, 165 right (field), 39, 151 shift_variable, 102	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117 scc, 151
rename_formula, 69, 73 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 retrieve, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 rhs (field), 97 rhs (field), 97 right (field), 39, 151 147, 160, 85, 89, 90, 115, 148, 160, 162 semicolonlist (camlyacc non-terminal), 162, 161, 162 seq, 54, 55, 55 Seq, 54, 55 Set, 159, 161 set_of_solutions, 126 set_of_symbols, 151 set_of_vars, 47 set_type, 163 Set_value, 165 shift_variable, 102	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70 remove_denominators, 73 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_its_arcs, 50	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135
repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 retrieve, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 rhs (field), 97 rhs (field), 97 rhs (field), 39, 151 semicolonlist (camlyacc non-terminal), 162, 161, 162 reglace_a_data, 129 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 seq, 54, 55, 55 Seq, 54, 55 set_of_solutions, 126 set_of_symbols, 151 set_of_vars, 47 rhs (field), 97 set_type, 163 Set_value, 165 right (field), 39, 151 shift_variable, 102	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70 remove_denominators, 73 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_its_arcs, 50 remove_vertex_and_link_its_arcs, 50	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135 Self_dual_distributivity, 109
replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 residual, 54, 55, 55 retrieve, 111 Set, 159, 161 rewrite_rules (module), 97, 101, 143, 144, 149, set_of_solutions, 126 Rewrite_rules (module), 97, 101, 143, 144, 149, set_of_symbols, 151 residual, 54, 55, 55 Seq, 54, 55 Set, 159, 161 rewrite_rules (module), 97, 101, 143, 144, 149, set_of_solutions, 126 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 Seq, 54, 55 Set, 159, 161 rewrite_rules (module), 97, 101, 143, 144, 149, set_of_symbols, 151 residual, 54, 55, 55 retrieve, 111 Set_of_solutions, 126 residual, 54, 55 Set, 159, 161 rewrite_rules (module), 97, 101, 143, 144, 149, set_of_symbols, 151 residual, 54, 55 Seq, 54, 55 Seq, 54, 55 Seq, 54, 55 Set_of_solutions, 126 Set_of_symbols, 151 set_of_vars, 47 rhs (field), 97 rhs (field), 97 rhs (field), 39, 151 Set_of_symbols, 151 set_of_symbols, 151 set_of_vars, 47 rhs (field), 97 set_type, 163 set_type, 165 shift_variable, 102	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135 Self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114,
replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 retrieve, 111 set_of_solutions, 126 Rewrite_rules (module), 97, 101, 143, 144, 149, set_of_symbols, 151 152, 153, 154 rhs (field), 97 rhs (field), 97 rhs_of_rule, 134 right (field), 39, 151 seq, 54, 55, 55 Seq, 54, 55 Set, 159, 161 set_of_solutions, 126 set_of_symbols, 151 set_of_vars, 47 set_type, 163 Set_value, 165 shift_variable, 102	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_list_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135 Self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114, 147, 160, 85, 89, 90, 115, 148, 160, 162
residual, 54, 55, 55 retrieve, 111 Set, 159, 161 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, set_of_solutions, 126 rbs (field), 97 rhs (field), 97 rhs_of_rule, 134 right (field), 39, 151 Seq, 54, 55 Set, 159, 161 set_of_solutions, 126 set_of_symbols, 151 set_of_vars, 47 set_type, 163 Set_value, 165 shift_variable, 102	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_denominators, 73 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 repeat, 93, 111	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135 Self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114, 147, 160, 85, 89, 90, 115, 148, 160, 162 semicolonlist (camlyacc non-terminal), 162, 161,
retrieve, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, set_of_symbols, 151 152, 153, 154 rhs (field), 97 rhs_of_rule, 134 right (field), 39, 151 Set, 159, 161 set_of_solutions, 126 set_of_symbols, 151 set_of_vars, 47 set_type, 163 Set_value, 165 shift_variable, 102	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_denominators, 73 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 repeat, 93, 111 replace_a_data, 129	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135 Self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114, 147, 160, 85, 89, 90, 115, 148, 160, 162 semicolonlist (camlyacc non-terminal), 162, 161, 162
rewrite_rule (type), 81, 135, 137 set_of_solutions, 126 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 set_of_symbols, 151 rhs (field), 97 set_type, 163 rhs_of_rule, 134 Set_value, 165 right (field), 39, 151 shift_variable, 102	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135 Self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114, 147, 160, 85, 89, 90, 115, 148, 160, 162 semicolonlist (camlyacc non-terminal), 162, 161, 162 seq, 54, 55, 55
Rewrite_rules (module), 97, 101, 143, 144, 149, set_of_symbols, 151 152, 153, 154 set_of_vars, 47 rhs (field), 97 set_type, 163 rhs_of_rule, 134 Set_value, 165 right (field), 39, 151 shift_variable, 102	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135 Self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114, 147, 160, 85, 89, 90, 115, 148, 160, 162 semicolonlist (camlyacc non-terminal), 162, 161, 162 seq, 54, 55, 55 Seq, 54, 55
152, 153, 154 set_of_vars, 47 rhs (field), 97 set_type, 163 rhs_of_rule, 134 Set_value, 165 right (field), 39, 151 shift_variable, 102	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 retrieve, 111	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135 Self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114, 147, 160, 85, 89, 90, 115, 148, 160, 162 semicolonlist (camlyacc non-terminal), 162, 161, 162 seq, 54, 55, 55 Seq, 54, 55 Set, 159, 161
rhs (field), 97 set_type, 163 rhs_of_rule, 134 Set_value, 165 right (field), 39, 151 shift_variable, 102	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_denominators, 73 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_lits_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 retrieve, 111 rewrite_rule (type), 81, 135, 137	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135 Self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114, 147, 160, 85, 89, 90, 115, 148, 160, 162 semicolonlist (camlyacc non-terminal), 162, 161, 162 seq, 54, 55, 55 Seq, 54, 55 Set, 159, 161 set_of_solutions, 126
rhs_of_rule, 134 Set_value, 165 right (field), 39, 151 shift_variable, 102	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_its_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 retrieve, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149,	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135 Self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114, 147, 160, 85, 89, 90, 115, 148, 160, 162 semicolonlist (camlyacc non-terminal), 162, 161, 162 seq, 54, 55, 55 Seq, 54, 55 Set, 159, 161 set_of_solutions, 126 set_of_symbols, 151
right (field), 39, 151 shift_variable, 102	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_denominators, 73 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_lits_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 retrieve, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135 Self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114, 147, 160, 85, 89, 90, 115, 148, 160, 162 semicolonlist (camlyacc non-terminal), 162, 161, 162 seq, 54, 55, 55 Seq, 54, 55 Set, 159, 161 set_of_solutions, 126 set_of_symbols, 151 set_of_vars, 47
	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_lits_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 retrieve, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149, 152, 153, 154 rhs (field), 97	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135 Self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114, 147, 160, 85, 89, 90, 115, 148, 160, 162 semicolonlist (camlyacc non-terminal), 162, 161, 162 seq, 54, 55, 55 Seq, 54, 55 Set, 159, 161 set_of_solutions, 126 set_of_symbols, 151 set_of_vars, 47 set_type, 163
1 ,	regexp_set (type), 54, 54 regles (field), 156 Regular_expr (module), 53, 54 regular_pair, 135 Relative_dp (module), 152 remove, 37, 39, 40, 43, 49 remove_a_data, 129 remove_coef, 70 remove_denominators, 73 remove_equivalent_elements, 63 remove_markings_rules, 149 remove_vertex_and_lits_arcs, 50 remove_vertex_and_link_its_arcs, 50 rename_expr, 69, 73 rename_formula, 69, 73 repeat, 93, 111 replace_a_data, 129 replace_a_var_by_a_var_in_an_eq, 113 residual, 54, 55, 55 retrieve, 111 rewrite_rule (type), 81, 135, 137 Rewrite_rules (module), 97, 101, 143, 144, 149,	S (sig), 40, 106, 107, 109, 111, 112, 113, 116, 117, 118, 119, 120, 122, 124, 125, 126, 129, 131, 133, 134, 136, 141, 41, 49, 106, 108, 110, 111, 112, 114, 117, 118, 119, 121, 122, 123, 124, 125, 126, 130, 131, 133, 134, 135, 136, 141, 142 safe_force_innermost_normalize, 130, 131 safe_innermost_normalize, 130, 131 SB_property, 109 scanned_equations (field), 117 scc, 151 self_critical_pairs, 133, 135 Self_dual_distributivity, 109 SEMICOLON (camlyacc token), 77, 84, 88, 114, 147, 160, 85, 89, 90, 115, 148, 160, 162 semicolonlist (camlyacc non-terminal), 162, 161, 162 seq, 54, 55, 55 Seq, 54, 55 Set, 159, 161 set_of_solutions, 126 set_of_symbols, 151 set_of_vars, 47 set_type, 163 Set_value, 165

side (type), 109 , 109	split_cycles, 51
signature (type), 91 , 89, 154	sqrt_ceil, 42
signature (class), 95 , 95, 96, 97, 101, 105, 106,	Sqrt_ceil, 74
130, 144, 145, 149, 151, 152, 153, 154	sqrt_floor, 42
signature (camlyacc non-terminal), 89 , 89	Sqrt_floor, 74
Signatures (module), 95 , 96, 97, 99, 100, 101, 105,	srs (type), 81 , 81, 82, 83, 84, 86, 165
106, 116, 130, 143, 144, 145, 146, 147,	Srs_completion (module), 86
149, 150, 151, 152, 153, 154	srs_of_string, 83
signature_element (type), 91, 91	srs_type, 163
signature_eof (camlyacc non-terminal), 89, 89	SRS_value, 165
Signature_lexer (module), 97, 98	Standard, 154
Signature_syntax (module), 96	StandardConfluence (module), 136
signature_type, 163	StandardKBCompletion (module), 136
signature_with_marks, 149	standard_criterion, 153
sig_constr (field), 87	Standard_critical_pairs (module), 133
sig_env (field), 87	Standard_innermost (module), 130
sig_index (field), 87	Standard_matching (module), 110
sig_name (field), 164	star, 53 , 55 , 55
sig_symbol (field), 87	Star, 54 , 55
sig_table (field), 164	STAR (camlyacc token), 160 , 160, 161
sig_val (field), 164	state (type), 158 , 157, 158
Sig_value, 165	State, 158
Simple, 92, 144, 89	status, 108
simple_dependency_pair_criterion, 144	status (field), 113
simple_expr (camlyacc non-terminal), 161 , 161	status (type), 113, 113
Simple_Mixed, 144	status_function (type), 107, 100, 107, 108, 165
simple_rule (type), 136 , 136, 137	status_of_string, 100
simple_word (type), 130 , 130, 137 simple_word (type), 92 , 92	
simple_word (type), 92, 92 simple_word (camlyacc non-terminal), 90, 89, 90	status_type, 163
	Status_value, 165
singleton, 43 , 56	string (camllex parsing rule), 167
size, 40, 47	String, 159 , 161
size (field), 113	STRING (camlyacc token), 160 , 160, 161
skip_bracket (camllex parsing rule), 146	string_of_comparison_result, 62
smallest_constant, 134	string_of_elem_theory, 112
Small_binary_tree (module), 120	string_of_fd_var_id, 74
Small_bit_field (module), 41	string_of_sort (method), 103
solve, 114, 118, 122, 124	string_of_symbol (method), 82 , 95 , 149 , 85
Solve (sig), 114, 122, 123	string_of_unif_elem_theory, 112
Solved, 113	string_of_var, 103
solved_part (field), 113	string_of_var (method), 102
solve_constraint, 145	string_of_var_id, 102
solve_without_marks, 118	String_rewriting (module), 81 , 56, 58, 82, 83, 84,
some, 54 , 55	86, 135, 137, 165
sort (class), 103 , 101, 103	String_signatures (module), 82 , 82, 83, 84, 86, 87,
sorted_signature (class), 101, 101	88, 135, 137, 164, 165
Sorted_signatures (module), 101	String_theory (module), 112
sorte_de_base (type), 158 , 157, 158	string_type, 163
Sorts (module), 103, 101	String_unification (module), 82
sort_id (type), 103 , 101, 103	String_value, 165
sort_table, 103	sub, 42 , 46 , 47 , 70 , 57, 145, 147, 148, 150
sort_table (type), 103 , 103	Sub, 68, 74 , 78
specialize_polynomial, 145	subset, 43
Specif (module), 157	subst, 73
split, 37, 103	subst1, 72
•	

substitute, 48,96	145, 149, 152, 154, 165
substitution (type), 70, 126, 129, 131, 133, 141,	termination_constraint_type, 163
70, 71, 72, 73, 126, 129, 130, 131, 133,	Termination_expert (module), 153
141, 142	termination_proof (type), 154 , 154
Substitution (module), 96 , 110, 130	termination_proof_component (type), 154 , 154
subtract, 37	term_compare, 107
succ, 42	Term_lexer (module), 97
sum_of_columns, 121	term_of_string, 100
superpose, 82	<i>Term_orderings</i> (module), 107 , 137, 165
symbol (type), 106, 108, 109, 111, 112, 113, 126,	term_ordering_type, 163
106, 107, 108, 109, 110, 112, 113, 114,	Term_ordering_value, 165
119, 121, 122, 123, 124, 125, 126	<i>Term_syntax</i> (module), 99 , 97
symbol (camllex parsing rule), 146	term_type, 163
SymbolMap (module), 96 , 105, 144, 145, 147, 148	Term_value, 165
SymbolOrd (module), 96 , 151	Ternary_associativity, 109
symbols (camllex regexpr), 98 , 99 , 98, 99	Ternary_inverse, 109
SymbolSet (module), 96 , 143, 144, 149, 151	Ternary_projection, 109
symbol_fix (type), 95 , 95, 99, 149	Ternary_single_axiom, 109
symbol_fix (method), 95, 149	test_acceptance, 156 tete (field), 156
symbol_id (type), 82 , 82, 83, 84, 85, 86, 101, 107, 114, 116, 126, 131, 135, 136, 137, 147,	tete_de_terme (field), 156
157, 158, 165	THEN (camlyacc token), 160 , 161
symbol_interp (camlyacc non-terminal), 148 , 148	theory (camhyace toxen), 100, 101 theory (camhyace non-terminal), 115, 114, 115
symbol_of_string (method), 82 , 95 , 83, 145, 150	Theory (module), 111, 112, 113, 114, 115, 116,
symbol_theory (type), 96	119, 120, 121, 122, 123, 124, 125, 126,
symb_interp, 106	165
Symgraph (module), 151 , 152	TheorySet (module), 112, 126, 112, 126
Syntax_error (exn), 79, 86, 91, 96, 99, 102, 111,	theory_check, 112
145, 147	Theory_lexer (module), 116
<i>S_axiom</i> , 109	Theory_parser (module), 114, 116
t (type), 38, 39, 40, 41, 43, 45, 49, 54, 56, 74, 77,	Theory_syntax (module), 114, 116
84, 88, 89, 96, 97, 105, 107, 111, 114, 120,	theo_type, 163
134 , 136 , 139 , 147 , 151 , 155 , 160 , 38, 39,	Theo_value, 165
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,	th_from_user_th, 126
51, 54, 56, 68, 69, 70, 71, 72, 73, 74, 75,	times, 70
76, 77, 87, 88, 96, 97, 99, 101, 102, 105,	time_limit, 76 time_out, 75
106, 107, 110, 111, 112, 113, 114, 117,	Time_out (exn), 75
119, 120, 121, 122, 126, 129, 130, 134, 135, 136, 137, 139, 142, 143, 144, 145,	time_out_signal_handler, 75
147, 149, 151, 152, 153, 154, 159, 160,	token, 76, 83, 97, 145, 166, 76, 83, 91, 98, 99, 116,
164, 165	146, 150, 167
t (class), 141	token (camllex parsing rule), 76, 83, 91, 98, 99,
tag, 54	116, 146, 150, 167
tail (field), 93	Toplevel_lexer (module), 166
term (type), 106, 107, 109, 111, 112, 113, 114,	Toplevel_parser (module), 159, 166
116, 117, 118, 119, 120, 122, 124, 126,	Total, 129
129 , 131 , 133 , 141 , 96, 97, 99, 100, 106,	total_degree, 47
107, 108, 110, 111, 112, 113, 114, 117,	to_int, 41,78
118, 119, 120, 121, 122, 123, 124, 125,	to_list, 93, 58
126, 129, 130, 131, 133, 136, 137, 141,	to_list (method), 88
142, 144, 149, 150, 151, 152, 154, 157,	
150 165	to_string, 42 , 160
158, 165	to_string (method), 141
158, 165 Termination_constraint, 165 Termination_constraints (module), 150, 143, 144,	•

traced_innermost_normalize, 141	unit_type, 163
Traced_rewriting (module), 141	Unit_value, 165
translate_constraints, 76	Unnormalized (exn), 130, 131
transpose, 151	Unsolved, 113
triv_leads_to, 157	Unterminated_comment (exn), 97, 98
trs_type, 163	update_compiled_rules, 134
TRS_value, 165	user_parameterized_signature (class), 88
True, 68 , 70 , 77, 89, 90	User_parameterized_signatures (module), 88, 164
TRUE, 155, 76, 167	user_sorted_signature (class), 101 , 101
TRUE (camlyacc token), 77 , 160 , 77, 160, 161	User_terms (module), 100
true_formula, 70	User_theory (module), 116
Tset (module), 134	user_variables (class), 102
Ttable (module), 134	User_words (module), 83
Tuple, 159 , 161	user_word_signature (class), 82
two, 41	use_interp, 144
typage (field), 156	Val, 93
types (type), 162 , 162, 163	value (field), 166
Types (module), 158, 155, 157	values (type), 165 , 162, 164, 165, 166
Type_error (exn), 162	Values (module), 164 , 162, 164
type_expr, 163	val_of_cte, 70
type_scheme (type), 162 , 162, 163, 164	var, 47 , 69 , 147 , 147, 148
Type_variable, 162	var (field), 75
Typing (module), 162 , 164	Var, 68 , 155 , 159 , 78, 161
typing_env_type (type), 162 , 163, 164	<i>VAR</i> (camlyacc token), 147 , 147, 148
T_axiom, 109	variable (type), 46 , 46, 47, 48, 75, 105, 144
<i>u</i> (type), 155 , 157	VARIABLE (exn), 155
Uncomparable, 62	Variables (module), 101, 96, 97, 99, 100, 101, 105,
UnifElemThMap (module), 113, 113	113, 116, 117, 118, 120, 122, 124, 126,
Unification (module), 126	130, 134, 136, 137, 149, 151, 153, 154,
unification_time, 126	155, 165
UnifTheorySet (module), 112	variables_of_t, 134
unify, 83	Variable_abstraction (module), 112, 114
unify_types, 163	variable_set_type, 163
Unif_ac (module), 122	VarMap (module), 70, 102, 70, 72, 73, 96, 106,
Unif_aci (module), 123	113, 117, 119, 120, 121, 122, 149
<i>Unif_acu</i> (module), 122 , 123	VarOrd (module), 102
Unif_ag_acun (module), 123	vars (camlyacc non-terminal), 148, 148
Unif_bool (module), 124	VarSet (module), 102, 102, 113, 114, 120, 122,
Unif_commutative (module), 118	134
unif_elem_theory (type), 112, 113, 124, 126, 112,	vars_for_eqe (field), 113
113, 114, 124, 125, 126	vars_name (field), 165
<i>Unif_free</i> (module), 117 , 118	vars_table (field), 165
Unif_index (module), 110	vars_val (field), 165
unif_kind (field), 113	<i>var_env</i> (type), 72 , 72, 87
unif_kind (type), 111, 111, 112, 113, 114, 120,	var_id (type), 69 , 102 , 155 , 69, 70, 72, 73, 74, 102,
122, 126	103, 106, 113, 116, 117, 119, 121, 124,
unif_to_arith, 119	149, 155, 157
Unif_to_arith (module), 118, 122, 123	<i>Var_map</i> (module), 46 , 46, 48
unif_to_arith_without_matrix, 119	var_multiplicities, 149
unif_type, 111	var_name, 69
union, 37 , 40 , 43 , 54 , 55, 56, 58	var_occurs, 72
union_of_automata, 157	var_of_string, 103
Unit, 109	var_of_string (method), 102 , 98
unit_symbol_of_theory. 112	var_outside_set. 102

var_renaming (type), 73, 73 *Var_set* (module), **46**, 47 Var_value, 165 var_var (field), 113 vect_of_bits_to_bit_field, 41 verbose, 74, 75, 86, 87, 126, 136, 153 version, 166 Version (module), 166 vertex (type), 49, 49, 50, 51 VertexMap (module), 49 VertexSet (module), 49, 49, 50, 51 vertex_has_a_1_cycle, 51 VERTICALBAR (camlyacc token), 77,77 VS, 158 W1_axiom, 109 W2_axiom, 109 Waj1, 109 Waj2, 109 Waj3, 109 Waj4, 109 Without_identifications, 67 With_identifications, 67 word (type), 87, 92, 56, 81, 82, 83, 84, 86, 87, 92, 135, 137, 165 word (camlyacc non-terminal), 85, 89, 85, 89, 90 WordCompletion (module), 137 Words (module), 86, 56, 81, 82, 83, 84, 86, 135, 137, 165 word_eof (camlyacc non-terminal), 85, 84 Word_lexer (module), 83 word_no_simple (camlyacc non-terminal), 89, 89 word_of_string, 83 Word_orderings (module), 86 word_ordering_type, 163 Word_ordering_value, 165 Word_parser (module), 84, 83 word_power, **84**, 85 word_precedence_type, 163 Word_precedence_value, 165 word_prec_of_string, 83 word_signature (class), 82, 82, 86, 87 word_signature_type, 163 word_sig_name (field), 164 word_sig_val (field), 164 Word_sig_value, 165 *Word_syntax* (module), **86**, 83, 84 word_type, 163 Word_value, 165 W_axiom, 109 zero, 41, 46, 47, 69 zmod, **67**

zquo, **67**