Albert Ludwigs Universität Freiburg

Technische Fakultät

PicoC-Compiler

Übersetzung einer Untermenge von C in den Befehlssatz der RETI-CPU

Bachelorarbeit

Abgabedatum: 28th April 2022

Author:
Jürgen Mattheis

Gutachter:
Prof. Dr. Scholl

Betreung:
M.Sc. Seufert

Eine Bachelorarbeit am Lehrstuhl für
Betriebssysteme

ERKLÄRUNG

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Inhaltsverzeichnis

1 Implementierung **10**
1.1 Lexikalische Analyse . . . 11
1.1.1 Konkrette Syntax für Lexer erstellen . . . 11
1.1.2 Basic Lexer . . . 12
1.2 Syntaktische Analyse . . . 12
1.2.1 Konkrette Syntax für Parser erstellen . . . 12
1.2.2 Umsetzung von Präzidenz . . . 13
1.2.3 Derivation Tree Generierung . . . 14
1.2.3.1 Early Parser . . . 14
1.2.3.2 Codebeispiel . . . 14
1.2.4 Derivation Tree Vereinfachung . . . 15
1.2.4.1 Visitor . . . 15
1.2.4.2 Codebeispiel . . . 15
1.2.5 Abstrakt Syntax Tree Generierung . . . 16
1.2.5.1 PicoC Nodes . . . 16
1.2.5.2 RETI Nodes . . . 16
1.2.5.3 Abstrakte Syntax . . . 16
1.2.5.4 Transformer . . . 18
1.2.5.5 Codebeispiel . . . 18
1.3 Code Generierung . . . 19
1.3.1 Übersicht . . . 19
1.3.2 Passes . . . 20
1.3.2.1 Kompositionen mit besonderer Bedeutung . . . 20
1.3.2.2 PicoC-Shrink Pass . . . 20
1.3.2.2.1 Codebeispiel . . . 20
1.3.2.3 PicoC-Blocks Pass . . . 23
1.3.2.3.1 Abstrakte Syntax . . . 23
1.3.2.3.2 Codebeispiel . . . 23
1.3.2.4 PicoC-Mon Pass . . . 25
1.3.2.4.1 Abstrakte Syntax . . . 25
1.3.2.4.2 Codebeispiel . . . 25
1.3.2.5 RETI-Blocks Pass . . . 27
1.3.2.5.1 Abstrakte Syntax . . . 27
1.3.2.5.2 Codebeispiel . . . 28
1.3.2.6 RETI-Patch Pass . . . 30
1.3.2.6.1 Abstrakte Syntax . . . 30
1.3.2.6.2 Codebeispiel . . . 30
1.3.2.7 RETI Pass . . . 33
1.3.2.7.1 Konkrette und Abstrakte Syntax . . . 33
1.3.2.7.2 Codebeispiel . . . 34
1.3.3 Umsetzung von Pointern . . . 37
1.3.3.1 Referenzierung . . . 37
1.3.3.2 Pointer Dereferenzierung durch Zugriff auf Arrayindex ersetzen . . . 39
1.3.4 Umsetzung von Arrays . . . 40
1.3.4.1 Initialisierung von Arrays . . . 40
1.3.4.2 Zugriff auf Arrayindex . . . 42

1.3.4.3 Zuweisung an Arrayindex . . . 45
1.3.5 Umsetzung von Structs . . . 47
1.3.5.1 Deklaration von Structs . . . 47
1.3.5.2 Initialisierung von Structs . . . 48
1.3.5.3 Zugriff auf Structattribut . . . 52
1.3.5.4 Zuweisung an Structattribut . . . 55
1.3.6 Umsetzung der Derived Datatypes im Zusammenspiel . . . 58
1.3.6.1 Einleitungsteil für Globale Statische Daten und Stackframe . . . 58
1.3.6.2 Mittelteil für die verschiedenen Derived Datatypes . . . 61
1.3.6.3 Schlussteil für die verschiedenen Derived Datatypes . . . 64
1.3.7 Umsetzung von Funktionen . . . 71
1.3.7.1 Funktionen auflösen zu RETI Code . . . 71
1.3.7.1.1 Sprung zur Main Funktion . . . 74
1.3.7.2 Funktionsdeklaration und -definition . . . 76
1.3.7.3 Funktionsaufruf . . . 77
1.3.7.3.1 Ohne Rückgabewert . . . 77
1.3.7.3.2 Mit Rückgabewert . . . 79
1.3.7.3.3 Umsetzung von Call by Sharing für Arrays . . . 82
1.3.7.3.4 Umsetzung von Call by Value für Structs . . . 85
1.3.8 Umsetzung kleinerer Details . . . 87
1.4 Fehlermeldungen . . . 87
1.4.1 Error Handler . . . 87
1.4.2 Arten von Fehlermeldungen . . . 87
1.4.2.1 Syntaxfehler . . . 87
1.4.2.2 Laufzeitfehler . . . 87

Abbildungsverzeichnis

1.1 Cross-Compiler Kompiliervorgang ausgeschrieben 19
1.2 Cross-Compiler Kompiliervorgang Kurzform 19
1.3 Architektur mit allen Passes ausgeschrieben 20

Codeverzeichnis

1.1 PicoC Code für Derivation Tree Generierung . . . 14
1.2 Derivation Tree nach Derivation Tree Generierung . . . 15
1.3 Derivation Tree nach Derivation Tree Vereinfachung . . . 16
1.4 Abstract Syntax Tree aus vereinfachtem Derivarion Tree generiert . . . 18
1.5 PicoC Code für Codebespiel . . . 21
1.6 Abstract Syntax Tree für Codebespiel . . . 22
1.7 PicoC Shrink Pass für Codebespiel . . . 23
1.8 PicoC-Blocks Pass für Codebespiel . . . 24
1.9 PicoC-Mon Pass für Codebespiel . . . 27
1.10 RETI-Blocks Pass für Codebespiel . . . 30
1.11 RETI-Patch Pass für Codebespiel . . . 33
1.12 RETI Pass für Codebespiel . . . 36
1.13 PicoC Code für Pointer Referenzierung . . . 37
1.14 Abstract Syntax Tree für Pointer Referenzierung . . . 37
1.15 PicoC Mon Pass für Pointer Referenzierung . . . 38
1.16 RETI Blocks Pass für Pointer Referenzierung . . . 39
1.17 PicoC Code für Pointer Dereferenzierung . . . 39
1.18 Abstract Syntax Tree für Pointer Dereferenzierung . . . 39
1.19 PicoC Shrink Pass für Pointer Dereferenzierung . . . 40
1.20 PicoC Code für Array Initialisierung . . . 40
1.21 Abstract Syntax Tree für Array Initialisierung . . . 41
1.22 Symboltabelle für Array Initialisierung . . . 41
1.23 PicoC Mon Pass für Array Initialisierung . . . 42
1.24 RETI Blocks Pass für Array Initialisierung . . . 42
1.25 PicoC Code für Zugriff auf Arrayindex . . . 43
1.26 Abstract Syntax Tree für Zugriff auf Arrayindex . . . 43
1.27 PicoC Mon Pass für Zugriff auf Arrayindex . . . 44
1.28 RETI Blocks Pass für Zugriff auf Arrayindex . . . 45
1.29 PicoC Code für Zuweisung an Arrayindex . . . 45
1.30 Abstract Syntax Tree für Zuweisung an Arrayindex . . . 46
1.31 PicoC Mon Pass für Zuweisung an Arrayindex . . . 46
1.32 RETI Blocks Pass für Zuweisung an Arrayindex . . . 47
1.33 PicoC Code für Deklaration von Structs . . . 47
1.34 Symboltabelle für Deklaration von Structs . . . 48
1.35 PicoC Code für Initialisierung von Structs . . . 49
1.36 Abstract Syntax Tree für Initialisierung von Structs . . . 50
1.37 Symboltabelle für Initialisierung von Structs . . . 51
1.38 PicoC Mon Pass für Initialisierung von Structs . . . 52
1.39 RETI Blocks Pass für Initialisierung von Structs . . . 52
1.40 PicoC Code für Zugriff auf Structattribut . . . 52
1.41 Abstract Syntax Tree für Zugriff auf Structattribut . . . 53
1.42 PicoC Mon Pass für Zugriff auf Structattribut . . . 54
1.43 RETI Blocks Pass für Zugriff auf Structattribut . . . 55
1.44 PicoC Code für Zuweisung an Structattribut . . . 55
1.45 Abstract Syntax Tree für Zuweisung an Structattribut . . . 56
1.46 PicoC Mon Pass für Zuweisung an Structattribut . . . 57
1.47 RETI Blocks Pass für Zuweisung an Structattribut . . . 57

1.48 PicoC Code für den Einleitungsteil . . . 58
1.49 Abstract Syntax Tree für den Einleitungsteil . . . 59
1.50 PicoC Mon Pass für den Einleitungsteil . . . 60
1.51 RETI Blocks Pass für den Einleitungsteil . . . 60
1.52 PicoC Code für den Mittelteil . . . 61
1.53 Abstract Syntax Tree für den Mittelteil . . . 62
1.54 PicoC Mon Pass für den Mittelteil . . . 63
1.55 RETI Blocks Pass für den Mittelteil . . . 64
1.56 PicoC Code für den Schlussteil . . . 65
1.57 Abstract Syntax Tree für den Schlussteil . . . 66
1.58 PicoC Mon Pass für den Schlussteil . . . 68
1.59 RETI Blocks Pass für den Schlussteil . . . 71
1.60 PicoC Code für 3 Funktionen . . . 71
1.61 Abstract Syntax Tree für 3 Funktionen . . . 72
1.62 PicoC Blocks Pass für 3 Funktionen . . . 72
1.63 PicoC Mon Pass für 3 Funktionen . . . 73
1.64 RETI Blocks Pass für 3 Funktionen . . . 73
1.65 PicoC Code für Funktionen, wobei die main Funktion nicht die erste Funktion ist . . . 74
1.66 PicoC Mon Pass für Funktionen, wobei die main Funktion nicht die erste Funktion ist . . . 74
1.67 PicoC Blocks Pass für Funktionen, wobei die main Funktion nicht die erste Funktion ist . . . 75
1.68 PicoC Patch Pass für Funktionen, wobei die main Funktion nicht die erste Funktion ist . . . 76
1.69 PicoC Code für Funktionen, wobei eine Funktion vorher deklariert werden muss . . . 76
1.70 Symboltabelle für Funktionen, wobei eine Funktion vorher deklariert werden muss . . . 77
1.71 PicoC Code für Funktionsaufruf ohne Rückgabewert . . . 77
1.72 PicoC Mon Pass für Funktionsaufruf ohne Rückgabewert . . . 78
1.73 RETI Blocks Pass für Funktionsaufruf ohne Rückgabewert . . . 79
1.74 RETI Pass für Funktionsaufruf ohne Rückgabewert . . . 79
1.75 PicoC Code für Funktionsaufruf mit Rückgabewert . . . 80
1.76 PicoC Mon Pass für Funktionsaufruf mit Rückgabewert . . . 80
1.77 RETI Blocks Pass für Funktionsaufruf mit Rückgabewert . . . 81
1.78 RETI Pass für Funktionsaufruf mit Rückgabewert . . . 82
1.79 PicoC Code für Call by Sharing für Arrays . . . 82
1.80 PicoC Mon Pass für Call by Sharing für Arrays . . . 83
1.81 Symboltabelle für Call by Sharing für Arrays . . . 84
1.82 RETI Block Pass für Call by Sharing für Arrays . . . 85
1.83 PicoC Code für Call by Value für Structs . . . 85
1.84 PicoC Mon Pass für Call by Value für Structs . . . 86
1.85 RETI Block Pass für Call by Value für Structs . . . 87

Tabellenverzeichnis

1.1 Präzidenzregeln von PicoC 13
1.2 Kompsotionenen von PicoC-Nodes mit besonderer Bedeutung 20

Definitionsverzeichnis

1.1 Symboltabelle 25

Grammatikverzeichnis

1.1.1 Konkrette Syntax des Lexers in EBNF 11
1.2.1 Konkrette Syntax des Parsers in EBNF, Teil 1 12
1.2.2 Konkrette Syntax des Parsers in EBNF, Teil 2 13
1.2.3 Abstrakte Syntax für L_{PiocC} 17
1.3.1 Abstrakte Syntax für L_{PicoC_Blocks} 23
1.3.2 Abstrakte Syntax für L_{PicoC_Mon} 25
1.3.3 Abstrakte Syntax für L_{RETI_Blocks} 28
1.3.4 Abstrakte Syntax für L_{RETI_Patch} 30
1.3.5 Konkrette Syntax für L_{RETI_Lex} 33
1.3.6 Konkrette Syntax für L_{RETI_Parse} 33
1.3.7 Abstrakte Syntax für L_{RETI} 34

1 Implementierung

1.1 Lexikalische Analyse

1.1.1 Konkrette Syntax für Lexer erstellen

COMMENT	::=	"//" /[^\n]*/ | "/*" /(. | \n)*?/ "*/"	*L_Comment*
RETI_COMMENT.2	::=	"//""␣"?"#"/[^\n]*/	
DIG_NO_0	::=	"1" | "2" | "3" | "4" | "5"	*L_Arith*
	|	"6" | "7" | "8" | "9"	
DIG_WITH_0	::=	"0" | *DIG_NO_0*	
NUM	::=	"0" | *DIG_NO_0DIG_WITH_0*∗	
ASCII_CHAR	::=	"␣".."~"	
CHAR	::=	"'"*ASCII_CHAR*"'"	
FILENAME	::=	*ASCII_CHAR* + ".*picoc*"	
LETTER	::=	"*a*".."*z*" | "*A*".."*Z*"	
NAME	::=	(*LETTER* | "_") (LETTER — DIG_WITH_0 — "_")*	
name	::=	*NAME* | *INT_NAME* | *CHAR_NAME*	
	|	*VOID_NAME*	
NOT	::=	"~"	
REF_AND	::=	"&"	
un_op	::=	*SUB_MINUS* | *LOGIC_NOT* | *NOT*	
	|	*MUL_DEREF_PNTR* | *REF_AND*	
MUL_DEREF_PNTR	::=	"∗"	
DIV	::=	"/"	
MOD	::=	"%"	
*prec*1*_op*	::=	*MUL_DEREF_PNTR* | *DIV* | *MOD*	
ADD	::=	"+"	
SUB_MINUS	::=	"−"	
*prec*2*_op*	::=	*ADD* | *SUB_MINUS*	
LT	::=	"<"	*L_Logic*
LTE	::=	"<="	
GT	::=	">"	
GTE	::=	">="	
rel_op	::=	*LT* | *LTE* | *GT* | *GTE*	
EQ	::=	"=="	
NEQ	::=	"!="	
eq_op	::=	*EQ* | *NEQ*	
LOGIC_NOT	::=	"!"	
INT_DT.2	::=	"*int*"	
INT_NAME.3	::=	"*int*" (*LETTER* | *DIG_WITH_0* | "_")+	*L_Assign_Alloc*
CHAR_DT.2	::=	"*char*"	
CHAR_NAME.3	::=	"*char*" (*LETTER* | *DIG_WITH_0* | "_")+	
VOID_DT.2	::=	"*void*"	
VOID_NAME.3	::=	"*void*" (*LETTER* | *DIG_WITH_0* | "_")+	
prim_dt	::=	*INT_DT* | *CHAR_DT* | *VOID_DT*	

Grammar 1.1.1: Konkrette Syntax des Lexers in EBNF

1.1.2 Basic Lexer

1.2 Syntaktische Analyse

1.2.1 Konkrette Syntax für Parser erstellen

In 1.2.1

prim_exp	::=	*name* | *NUM* | *CHAR* | "("*logic_or*")"	*L_Arith +*
post_exp	::=	*array_subscr* | *struct_attr* | *fun_call*	*L_Array +*
		| *input_exp* | *print_exp* | *prim_exp*	*L_Pntr +*
un_exp	::=	*un_opun_exp* | *post_exp*	*L_Struct + L_Fun*
input_exp	::=	"*input*""("")"	*L_Arith*
print_exp	::=	"*print*""("*logic_or*")"	
*arith_prec*1	::=	*arith_prec*1 *prec*1*_op un_exp* | *un_exp*	
*arith_prec*2	::=	*arith_prec*2 *prec*2*_op arith_prec*1 | *arith_prec*1	
arith_and	::=	*arith_and* "&" *arith_prec*2 | *arith_prec*2	
arith_oplus	::=	*arith_oplus* "∧" *arith_and* | *arith_and*	
arith_or	::=	*arith_or* "|" *arith_oplus* | *arith_oplus*	
rel_exp	::=	*rel_exp rel_op arith_or* | *arith_or*	*L_Logic*
eq_exp	::=	*eq_exp eq_oprel_exp* | *rel_exp*	
logic_and	::=	*logic_and* "&&" *eq_exp* | *eq_exp*	
logic_or	::=	*logic_or* "||" *logic_and* | *logic_and*	
type_spec	::=	*prim_dt* | *struct_spec*	*L_Assign_Alloc*
alloc	::=	*type_spec pntr_decl*	
assign_stmt	::=	*un_exp* "=" *logic_or*";"	
initializer	::=	*logic_or* | *array_init* | *struct_init*	
init_stmt	::=	*alloc* "=" *initializer*";"	
const_init_stmt	::=	"*const*" *type_spec name* "=" *NUM*";"	
pntr_deg	::=	"∗"∗	*L_Pntr*
pntr_decl	::=	*pntr_deg array_decl* | *array_decl*	
array_dims	::=	("["*NUM*"]")∗	*L_Array*
array_decl	::=	*name array_dims* | "("*pntr_decl*")"*array_dims*	
array_init	::=	"{"*initializer*("," *initializer*) ∗ "}"	
array_subscr	::=	*post_exp*"["*logic_or*"]"	
struct_spec	::=	"*struct*" *name*	*L_Struct*
struct_params	::=	(*alloc*";")+	
struct_decl	::=	"*struct*" *name* "{"*struct_params*"}"	
struct_init	::=	"{""."*name*"="*initializer* ("," "."name"="initializer)*"}"	
struct_attr	::=	*post_exp*"."*name*	
if_stmt	::=	"*if*""("*logic_or*")" *exec_part*	*L_If_Else*
if_else_stmt	::=	"*if*""("*logic_or*")" *exec_part* "*else*" *exec_part*	
while_stmt	::=	"*while*""("*logic_or*")" *exec_part*	*L_Loop*
do_while_stmt	::=	"*do*" *exec_part* "*while*""("*logic_or*")"";"	

Grammar 1.2.1: Konkrette Syntax des Parsers in EBNF, Teil 1

decl_exp_stmt	::=	*alloc*";"	*L_Stmt*
decl_direct_stmt	::=	*assign_stmt* | *init_stmt* | *const_init_stmt*	
decl_part	::=	*decl_exp_stmt* | *decl_direct_stmt* | *RETI_COMMENT*	
compound_stmt	::=	"{" *exec_part* ∗ "}"	
exec_exp_stmt	::=	*logic_or*";"	
exec_direct_stmt	::=	*if_stmt* | *if_else_stmt* | *while_stmt* | *do_while_stmt*	
	|	*assign_stmt* | *fun_return_stmt*	
exec_part	::=	*compound_stmt* | *exec_exp_stmt* | *exec_direct_stmt*	
	|	*RETI_COMMENT*	
decl_exec_stmts	::=	*decl_part* ∗ *exec_part*∗	
fun_args	::=	[*logic_or*("," *logic_or*)∗]	*L_Fun*
fun_call	::=	*name*"(" *fun_args*")"	
fun_return_stmt	::=	"*return*" [*logic_or*]";"	
fun_params	::=	[*alloc*("," *alloc*)∗]	
fun_decl	::=	*type_spec pntr_deg name*"(" *fun_params*")"	
fun_def	::=	*type_spec pntr_deg name*"(" *fun_params*")" "{" *decl_exec_stmts*"}"	
decl_def	::=	(*struct_decl* | *fun_decl*)";" | *fun_def*	*L_File*
decls_defs	::=	*decl_def*∗	
file	::=	*FILENAME decls_defs*	

Grammar 1.2.2: Konkrette Syntax des Parsers in EBNF, Teil 2

1.2.2 Umsetzung von Präzidenz

Die **PicoC** Programmiersprache hat dieselben **Präzidenzregeln** implementiert, wie die Programmiersprache **C**[1]. Die **Präzidenzregeln** von **PicoC** sind in Tabelle 1.2 aufgelistet.

Präzidenz	Operator	Beschreibung	Assoziativität
1	`a()`	Funktionsaufruf	Links, dann rechts →
	`a[]`	Indexzugriff	
	`a.b`	Attributzugriff	
2	`-a`	Unäres Minus	Rechts, dann links ←
	`!a ~a`	Logisches NOT und Bitweise NOT	
	`*a &a`	Dereferenz und Referenz, auch Adresse-von	
3	`a*b a/b a%b`	Multiplikation, Division und Modulo	Links, dann rechts →
4	`a+b a-b`	Addition und Subtraktion	
5	`a<b a<=b a>b a>=b`	Kleiner, Kleiner Gleich, Größer, Größer gleich	
6	`a==b a!=b`	Gleichheit und Ungleichheit	
7	`a&b`	Bitweise UND	
8	`a^b`	Bitweise XOR (exclusive or)	
9	`a|b`	Bitweise ODER (inclusive or)	
10	`a&&b`	Logiches UND	
11	$a \| \| b$	Logisches ODER	
12	`a=b`	Zuweisung	Rechts, dann links ←
13	`a,b`	Komma	Links, dann rechts →

Tabelle 1.1: Präzidenzregeln von PicoC

[1] *C Operator Precedence - cppreference.com.*

1.2.3 Derivation Tree Generierung

1.2.3.1 Early Parser

1.2.3.2 Codebeispiel

```
1 struct st {int *(*attr)[5][6];};
2
3 void main() {
4   struct st *(*var)[3][2];
5 }
```

Code 1.1: PicoC Code für Derivation Tree Generierung

```
1  file
2    ./example_dt_simple_ast_gen_array_decl_and_alloc.dt
3    decls_defs
4      decl_def
5        struct_decl
6          name  st
7          struct_params
8            alloc
9              type_spec
10               prim_dt  int
11             pntr_decl
12               pntr_deg  *
13               array_decl
14                 pntr_decl
15                   pntr_deg  *
16                   array_decl
17                     name  attr
18                     array_dims
19                 array_dims
20                   5
21                   6
22     decl_def
23       fun_def
24         type_spec
25           prim_dt  void
26         pntr_deg
27         name  main
28         fun_params
29         decl_exec_stmts
30           decl_part
31             decl_exp_stmt
32               alloc
33                 type_spec
34                   struct_spec
35                     name  st
36                 pntr_decl
37                   pntr_deg  *
38                   array_decl
39                     pntr_decl
40                       pntr_deg  *
```

```
41                      array_decl
42                        name  var
43                        array_dims
44                    array_dims
45                      3
46                      2
```

Code 1.2: Derivation Tree nach Derivation Tree Generierung

1.2.4 Derivation Tree Vereinfachung

1.2.4.1 Visitor

1.2.4.2 Codebeispiel

Beispiel aus Subkapitel 1.2.3.2 wird fortgeführt.

```
1 file
2   ./example_dt_simple_ast_gen_array_decl_and_alloc.dt_simple
3   decls_defs
4     decl_def
5       struct_decl
6         name  st
7         struct_params
8           alloc
9             pntr_decl
10              pntr_deg  *
11              array_decl
12                array_dims
13                  5
14                  6
15                pntr_decl
16                  pntr_deg  *
17                  array_decl
18                    array_dims
19                    type_spec
20                      prim_dt  int
21            name  attr
22    decl_def
23      fun_def
24        type_spec
25          prim_dt  void
26        pntr_deg
27        name  main
28        fun_params
29        decl_exec_stmts
30          decl_part
31            decl_exp_stmt
32              alloc
33                pntr_decl
34                  pntr_deg  *
35                  array_decl
36                    array_dims
```

```
37                      3
38                      2
39                    pntr_decl
40                      pntr_deg  *
41                      array_decl
42                        array_dims
43                        type_spec
44                          struct_spec
45                            name  st
46                name  var
```

Code 1.3: Derivation Tree nach Derivation Tree Vereinfachung

1.2.5 Abstrakt Syntax Tree Generierung

1.2.5.1 PicoC Nodes

1.2.5.2 RETI Nodes

1.2.5.3 Abstrakte Syntax

un_op	$::=$	$Minus() \mid Not()$	*L_Arith*
bin_op	$::=$	$Add() \mid Sub() \mid Mul() \mid Div() \mid Mod()$	
	$\mid$	$Oplus() \mid And() \mid Or()$	
exp	$::=$	$Name(str) \mid Num(str) \mid Char(str)$	
	$\mid$	$BinOp(\langle exp\rangle, \langle bin\_op\rangle, \langle exp\rangle)$	
	$\mid$	$UnOp(\langle un\_op\rangle, \langle exp\rangle) \mid Call(Name('input'), None)$	
exp_stmts	$::=$	$Alloc(\langle type\_qual\rangle, \langle dataype\rangle, Name(str))$	
	$\mid$	$Call(Name('print'), \langle exp\rangle)$	
un_op	$::=$	$LogicNot()$	*L_Logic*
rel	$::=$	$Eq() \mid NEq() \mid Lt() \mid LtE() \mid Gt() \mid GtE()$	
bin_op	$::=$	$LogicAnd() \mid LogicOr()$	
exp	$::=$	$Atom(\langle exp\rangle, \langle rel\rangle, \langle exp\rangle)$	
	$\mid$	$ToBool(\langle exp\rangle)$	
$type\_qual$	$::=$	$Const() \mid Writeable()$	*L_Assign_Alloc*
$datatype$	$::=$	$IntType() \mid CharType() \mid VoidType()$	
$assign\_lhs$	$::=$	$Alloc(\langle type\_qual\rangle, \langle dataype\rangle, Name(str)) \mid \langle rel\_loc\rangle$	
exp_stmts	$::=$	$Alloc(\langle type\_qual\rangle, \langle dataype\rangle, Name(str))$	
$stmt$	$::=$	$Assign(\langle assign\_lhs\rangle, \langle exp\rangle)$	
	$\mid$	$Exp(\langle exp\_stmts\rangle)$	
$datatype$	$::=$	$PntrDecl(Num(str), \langle datatype\rangle)$	*L_Pntr*
$deref\_loc$	$::=$	$Ref(\langle ref\_loc\rangle) \mid \langle ref\_loc\rangle$	
ref_loc	$::=$	$Name(str)$	
	$\mid$	$Deref(\langle deref\_loc\rangle, \langle exp\rangle)$	
	$\mid$	$Subscr(\langle deref\_loc\rangle, \langle exp\rangle)$	
	$\mid$	$Attr(\langle ref\_loc\rangle, Name(str))$	
exp	$::=$	$Deref(\langle deref\_loc\rangle, \langle exp\rangle)$	
	$\mid$	$Ref(\langle ref\_loc\rangle)$	
$datatype$	$::=$	$ArrayDecl(Num(str)+, \langle datatype\rangle)$	*L_Array*
exp	$::=$	$Subscr(\langle deref\_loc\rangle, \langle exp\rangle) \mid Array(\langle exp\rangle+)$	
$datatype$	$::=$	$StructSpec(Name(str))$	*L_Struct*
exp	$::=$	$Attr(\langle ref\_loc\rangle, Name(str))$	
	$\mid$	$Struct(Assign(Name(str), \langle exp\rangle)+)$	
$decl\_def$	$::=$	$StructDecl(Name(str),$	
		$\quad Alloc(Writeable(), \langle datatype\rangle, Name(str))+)$	
$stmt$	$::=$	$If(\langle exp\rangle, \langle stmt\rangle*)$	*L_If_Else*
	$\mid$	$IfElse(\langle exp\rangle, \langle stmt\rangle*, \langle stmt\rangle*)$	
$stmt$	$::=$	$While(\langle exp\rangle, \langle stmt\rangle*)$	*L_Loop*
	$\mid$	$DoWhile(\langle exp\rangle, \langle stmt\rangle*)$	
exp	$::=$	$Call(Name(str), \langle exp\rangle*)$	*L_Fun*
exp_stmts	$::=$	$Call(Name(str), \langle exp\rangle*)$	
$stmt$	$::=$	$Return(\langle exp\rangle)$	
$decl\_def$	$::=$	$FunDecl(\langle datatype\rangle, Name(str),$	
		$\quad Alloc(Writeable(), \langle datatype\rangle, Name(str))*)$	
	$\mid$	$FunDef(\langle datatype\rangle, Name(str),$	
		$\quad Alloc(Writeable(), \langle datatype\rangle, Name(str))*, \langle stmt\rangle*)$	
$file$	$::=$	$File(Name(str), \langle decl\_def\rangle*)$	*L_File*

Grammar 1.2.3: Abstrakte Syntax für L_{PiocC}

1.2.5.4 Transformer

1.2.5.5 Codebeispiel

Beispiel welches in Subkapitel 1.2.3.2 angefangen wurde, wird hier fortgeführt.

```
File
  Name './example_dt_simple_ast_gen_array_decl_and_alloc.ast',
  [
    StructDecl
      Name 'st',
      [
        Alloc
          Writeable,
          PntrDecl
            Num '1',
            ArrayDecl
              [
                Num '5',
                Num '6'
              ],
              PntrDecl
                Num '1',
                IntType 'int',
          Name 'attr'
      ],
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Exp
          Alloc
            Writeable,
            PntrDecl
              Num '1',
              ArrayDecl
                [
                  Num '3',
                  Num '2'
                ],
                PntrDecl
                  Num '1',
                  StructSpec
                    Name 'st',
            Name 'var'
      ]
  ]
```

Code 1.4: Abstract Syntax Tree aus vereinfachtem Derivarion Tree generiert

1.3 Code Generierung

1.3.1 Übersicht

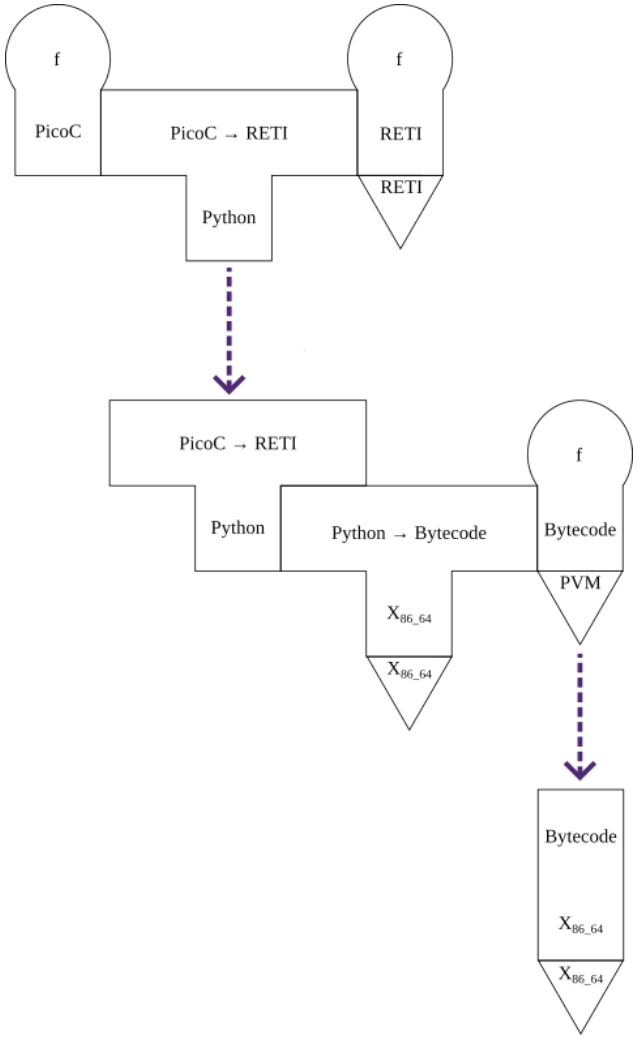

Abbildung 1.1: Cross-Compiler Kompiliervorgang ausgeschrieben

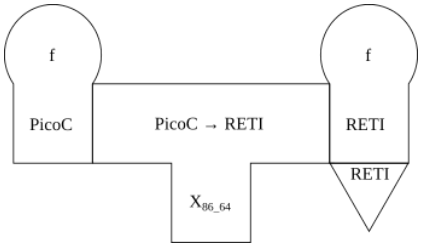

Abbildung 1.2: Cross-Compiler Kompiliervorgang Kurzform

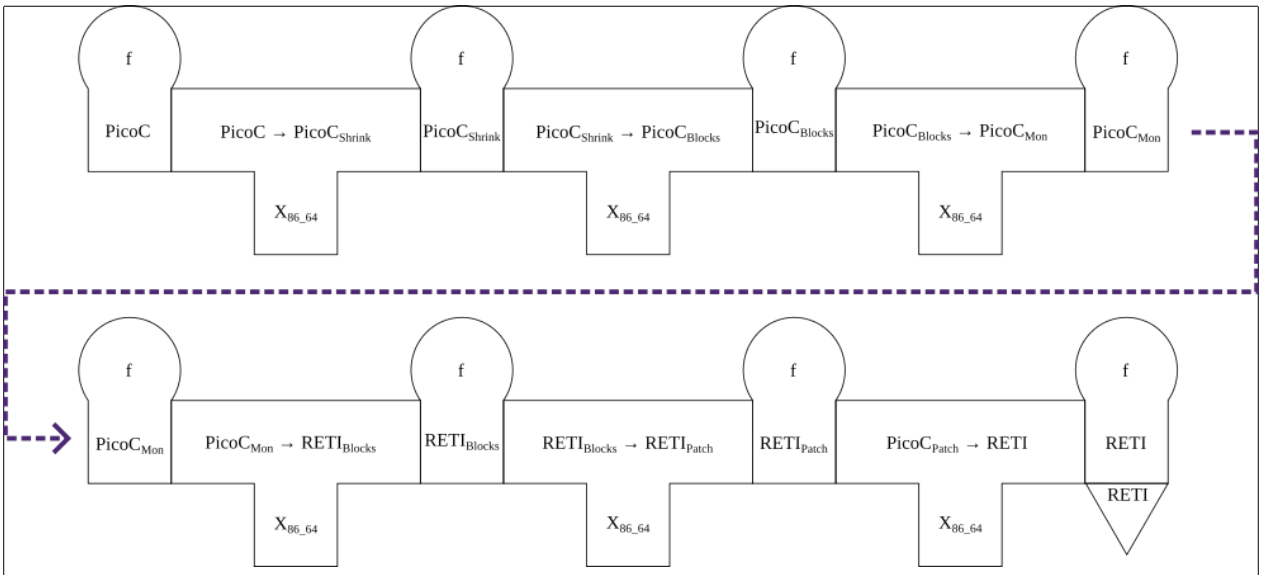

Abbildung 1.3: Architektur mit allen Passes ausgeschrieben

1.3.2 Passes

1.3.2.1 Kompositionen mit besonderer Bedeutung

Präzidenz	Operator	Beschreibung	Assoziativität
1	`a()`	Funktionsaufruf	Links, dann rechts →
	`a[]`	Indexzugriff	
	`a.b`	Attributzugriff	
2	`-a`	Unäres Minus	Rechts, dann links ←
	`!a ~a`	Logisches NOT und Bitweise NOT	
	`*a &a`	Dereferenz und Referenz, auch Adresse-von	
3	`a*b a/b a%b`	Multiplikation, Division und Modulo	Links, dann rechts →
4	`a+b a-b`	Addition und Subtraktion	
5	`a<b a<=b a>b a>=b`	Kleiner, Kleiner Gleich, Größer, Größer gleich	
6	`a==b a!=b`	Gleichheit und Ungleichheit	
7	`a&b`	Bitweise UND	
8	`a^b`	Bitweise XOR (exclusive or)	
9	`a|b`	Bitweise ODER (inclusive or)	
10	`a&&b`	Logiches UND	
11	$a \| \| b$	Logisches ODER	
12	`a=b`	Zuweisung	Rechts, dann links ←
13	`a,b`	Komma	Links, dann rechts →

Tabelle 1.2: Kompsotionenen von PicoC-Nodes mit besonderer Bedeutung

1.3.2.2 PicoC-Shrink Pass

1.3.2.2.1 Codebeispiel

```
// Author: Christoph Scholl, from the Operating Systems Lecture

void main() {
  int n = 4;
  int res = 1;
  while (1) {
    if (n == 1) {
      return;
    }
    res = n * res;
    n = n - 1;
  }
}
```

Code 1.5: PicoC Code für Codebespiel

```
File
  Name './example_faculty_it.ast',
  [
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Assign
          Alloc
            Writeable,
            IntType 'int',
            Name 'n',
          Num '4',
        Assign
          Alloc
            Writeable,
            IntType 'int',
            Name 'res',
          Num '1',
        While
          Num '1',
          [
            If
              Atom
                Name 'n',
                Eq '==',
                Num '1',
              [
                Return
                  Empty
              ],
            Assign
              Name 'res',
              BinOp
                Name 'n',
                Mul '*',
                Name 'res',
```

```
39            Assign
40              Name 'n',
41              BinOp
42                Name 'n',
43                Sub '-',
44                Num '1'
45            ]
46        ]
47    ]
```

Code 1.6: Abstract Syntax Tree für Codebespiel

```
1 File
2   Name './example_faculty_it.picoc_shrink',
3   [
4     FunDef
5       VoidType 'void',
6       Name 'main',
7       [],
8       [
9         Assign
10          Alloc
11            Writeable,
12            IntType 'int',
13            Name 'n',
14          Num '4',
15        Assign
16          Alloc
17            Writeable,
18            IntType 'int',
19            Name 'res',
20          Num '1',
21        While
22          Num '1',
23          [
24            If
25              Atom
26                Name 'n',
27                Eq '==',
28                Num '1',
29              [
30                Return
31                  Empty
32              ],
33            Assign
34              Name 'res',
35              BinOp
36                Name 'n',
37                Mul '*',
38                Name 'res',
39            Assign
40              Name 'n',
41              BinOp
42                Name 'n',
```

```
43                Sub '-',
44                Num '1'
45            ]
46        ]
47    ]
```

Code 1.7: PicoC Shrink Pass für Codebespiel

1.3.2.3 PicoC-Blocks Pass

1.3.2.3.1 Abstrakte Syntax

$decl\_def$	::=	$FunDef(\langle datatype\rangle, Name(str),$ $Alloc(Writeable(), \langle datatype\rangle, Name(str))*, \langle block\rangle*)$	*L_Fun*
$block$	::=	$Block(Name(str), \langle stmt\rangle*)$	*L_Blocks*
$stmt$	::=	$Goto(Name(str)) \mid NewStackframe(Name(), Goto(str))$	
	|	$RemoveStackframe() \mid SetScope(Name(str))$	
	|	$SingleLineComment(str, str)$	

Grammar 1.3.1: Abstrakte Syntax für L_{PicoC_Blocks}

1.3.2.3.2 Codebeispiel

```
1  File
2    Name './example_faculty_it.picoc_blocks',
3    [
4      FunDef
5        VoidType 'void',
6        Name 'main',
7        [],
8        [
9          Block
10           Name 'main.5',
11           [
12             Assign
13               Alloc
14                 Writeable,
15                 IntType 'int',
16                 Name 'n',
17               Num '4',
18             Assign
19               Alloc
20                 Writeable,
21                 IntType 'int',
22                 Name 'res',
23               Num '1',
24             // While(Num('1'), []),
25             GoTo
26               Name 'condition_check.4'
27           ],
28         Block
29           Name 'condition_check.4',
```

```
30          [
31            IfElse
32              Num '1',
33              GoTo
34                Name 'while_branch.3',
35              GoTo
36                Name 'while_after.0'
37          ],
38        Block
39          Name 'while_branch.3',
40          [
41            // If(Atom(Name('n'), Eq('=='), Num('1')), []),
42            IfElse
43              Atom
44                Name 'n',
45                Eq '==',
46                Num '1',
47              GoTo
48                Name 'if.2',
49              GoTo
50                Name 'if_else_after.1'
51          ],
52        Block
53          Name 'if.2',
54          [
55            Return
56              Empty
57          ],
58        Block
59          Name 'if_else_after.1',
60          [
61            Assign
62              Name 'res',
63              BinOp
64                Name 'n',
65                Mul '*',
66                Name 'res',
67            Assign
68              Name 'n',
69              BinOp
70                Name 'n',
71                Sub '-',
72                Num '1',
73            GoTo
74              Name 'condition_check.4'
75          ],
76        Block
77          Name 'while_after.0',
78          []
79      ]
80  ]
```

Code 1.8: PicoC-Blocks Pass für Codebespiel

1.3.2.4 PicoC-Mon Pass

1.3.2.4.1 Abstrakte Syntax

ref_loc	$::=$	$Tmp(Num(str)) \mid StackRead(Num(str))$	*L_Assign_Alloc*
	$\mid$	$StackWrite(Num(str)) \mid GlobalRead(Num(str))$	
	$\mid$	$GlobalWrite(Num(str))$	
$error\_data$	$::=$	$\langle exp \rangle \mid Pos(Num(str), Num(str))$	
exp	$::=$	$Stack(Num(str)) \mid Ref(\langle ref_loc \rangle, \langle datatype \rangle, \langle error_data \rangle)$	
$stmt$	$::=$	$Exp(\langle exp \rangle)$	
	$\mid$	$Assign(Alloc(Writeable(), StructSpec(Name(str)), Name(str)),$ $Struct(Assign(Name(str), \langle exp \rangle)+, \langle datatype \rangle))$	
	$\mid$	$Assign(Alloc(Writeable(), ArrayDecl(Num(str)+, \langle datatype \rangle)),$ $Name(str)), Array(\langle exp \rangle+, \langle datatype \rangle))$	
$symbol\_table$	$::=$	$SymbolTable(\langle symbol \rangle)$	*L_Symbol_Table*
$symbol$	$::=$	$Symbol(\langle type_qual \rangle, \langle datatype \rangle, \langle name \rangle, \langle val \rangle, \langle pos \rangle, \langle size \rangle)$	
$type\_qual$	$::=$	$Empty()$	
$datatype$	$::=$	$BuiltIn() \mid SelfDefined()$	
$name$	$::=$	$Name(str)$	
val	$::=$	$Num(str) \mid Empty()$	
pos	$::=$	$Pos(Num(str), Num(str)) \mid Empty()$	
$size$	$::=$	$Num(str) \mid Empty()$	

Grammar 1.3.2: Abstrakte Syntax für L_{PicoC_Mon}

Definition 1.1: Symboltabelle

1.3.2.4.2 Codebeispiel

```
File
  Name './example_faculty_it.picoc_mon',
  [
    Block
      Name 'main.5',
      [
        // Assign(Name('n'), Num('4')),
        Exp
          Num '4',
        Assign
          GlobalWrite
            Num '0',
          Tmp
            Num '1',
        // Assign(Name('res'), Num('1')),
        Exp
          Num '1',
        Assign
          GlobalWrite
            Num '1',
          Tmp
            Num '1',
```

```
23        // While(Num('1'), []),
24        Exp
25          GoTo
26            Name 'condition_check.4'
27      ],
28    Block
29      Name 'condition_check.4',
30      [
31        // IfElse(Num('1'), GoTo(Name('while_branch.3')), GoTo(Name('while_after.0'))),
32        Exp
33          Num '1',
34        IfElse
35          Tmp
36            Num '1',
37          GoTo
38            Name 'while_branch.3',
39          GoTo
40            Name 'while_after.0'
41      ],
42    Block
43      Name 'while_branch.3',
44      [
45        // If(Atom(Name('n'), Eq('=='), Num('1')), []),
46        // IfElse(Atom(Name('n'), Eq('=='), Num('1')), GoTo(Name('if.2')),
          ↪  GoTo(Name('if_else_after.1'))),
47        Exp
48          GlobalRead
49            Num '0',
50        Exp
51          Num '1',
52        Exp
53          Atom
54            Tmp
55              Num '2',
56            Eq '==',
57            Tmp
58              Num '1',
59        IfElse
60          Tmp
61            Num '1',
62          GoTo
63            Name 'if.2',
64          GoTo
65            Name 'if_else_after.1'
66      ],
67    Block
68      Name 'if.2',
69      [
70        Return
71          Empty
72      ],
73    Block
74      Name 'if_else_after.1',
75      [
76        // Assign(Name('res'), BinOp(Name('n'), Mul('*'), Name('res'))),
77        Exp
78          GlobalRead
```

```
79            Num '0',
80          Exp
81            GlobalRead
82              Num '1',
83          Exp
84            BinOp
85              Tmp
86                Num '2',
87              Mul '*',
88              Tmp
89                Num '1',
90          Assign
91            GlobalWrite
92              Num '1',
93            Tmp
94              Num '1',
95          // Assign(Name('n'), BinOp(Name('n'), Sub('-'), Num('1'))),
96          Exp
97            GlobalRead
98              Num '0',
99          Exp
100           Num '1',
101         Exp
102           BinOp
103             Tmp
104               Num '2',
105             Sub '-',
106             Tmp
107               Num '1',
108         Assign
109           GlobalWrite
110             Num '0',
111           Tmp
112             Num '1',
113         Exp
114           GoTo
115             Name 'condition_check.4'
116       ],
117     Block
118       Name 'while_after.0',
119       [
120         Return
121           Empty
122       ]
123   ]
```

Code 1.9: PicoC-Mon Pass für Codebespiel

1.3.2.5 RETI-Blocks Pass

1.3.2.5.1 Abstrakte Syntax

program	::=	$Program(Name(str), \langle block \rangle *)$	*L_Program*
exp_stmts	::=	$Goto(str)$	*L_Blocks*
instrs_before	::=	$Num(str)$	
num_instrs	::=	$Num(str)$	
block	::=	$Block(Name(str), \langle instr \rangle *, \langle instrs\_before \rangle, \langle num\_instrs \rangle)$	
instr	::=	$Goto(Name(str))$	

Grammar 1.3.3: Abstrakte Syntax für L_{RETI_Blocks}

1.3.2.5.2 Codebeispiel

```
File
  Name './example_faculty_it.reti_blocks',
  [
    Block
      Name 'main.5',
      [
        # // Assign(Name('n'), Num('4')),
        # Exp(Num('4')),
        SUBI SP 1,
        LOADI ACC 4,
        STOREIN SP ACC 1,
        # Assign(GlobalWrite(Num('0')), Tmp(Num('1'))),
        LOADIN SP ACC 1,
        STOREIN DS ACC 0,
        ADDI SP 1,
        # // Assign(Name('res'), Num('1')),
        # Exp(Num('1')),
        SUBI SP 1,
        LOADI ACC 1,
        STOREIN SP ACC 1,
        # Assign(GlobalWrite(Num('1')), Tmp(Num('1'))),
        LOADIN SP ACC 1,
        STOREIN DS ACC 1,
        ADDI SP 1,
        # // While(Num('1'), []),
        Exp
          GoTo
            Name 'condition_check.4'
      ],
    Block
      Name 'condition_check.4',
      [
        # // IfElse(Num('1'), GoTo(Name('while_branch.3')), GoTo(Name('while_after.0'))),
        # Exp(Num('1')),
        SUBI SP 1,
        LOADI ACC 1,
        STOREIN SP ACC 1,
        # IfElse(Tmp(Num('1')), GoTo(Name('while_branch.3')), GoTo(Name('while_after.0'))),
        LOADIN SP ACC 1,
        ADDI SP 1,
        JUMP== GoTo
                 Name 'while_after.0';,
        Exp
```

```
44          GoTo
45            Name 'while_branch.3'
46      ],
47    Block
48      Name 'while_branch.3',
49      [
50        # // If(Atom(Name('n'), Eq('=='), Num('1')), []),
51        # // IfElse(Atom(Name('n'), Eq('=='), Num('1')), GoTo(Name('if.2')),
          ↪  GoTo(Name('if_else_after.1'))),
52        # Exp(GlobalRead(Num('0'))),
53        SUBI SP 1,
54        LOADIN DS ACC 0,
55        STOREIN SP ACC 1,
56        # Exp(Num('1')),
57        SUBI SP 1,
58        LOADI ACC 1,
59        STOREIN SP ACC 1,
60        LOADIN SP ACC 2,
61        LOADIN SP IN2 1,
62        SUB ACC IN2,
63        JUMP== 3;,
64        LOADI ACC 0,
65        JUMP 2;,
66        LOADI ACC 1,
67        STOREIN SP ACC 2,
68        ADDI SP 1,
69        # IfElse(Tmp(Num('1')), GoTo(Name('if.2')), GoTo(Name('if_else_after.1'))),
70        LOADIN SP ACC 1,
71        ADDI SP 1,
72        JUMP== GoTo
73                 Name 'if_else_after.1';,
74        Exp
75          GoTo
76            Name 'if.2'
77      ],
78    Block
79      Name 'if.2',
80      [
81        # Return(Empty()),
82        LOADIN BAF PC -1
83      ],
84    Block
85      Name 'if_else_after.1',
86      [
87        # // Assign(Name('res'), BinOp(Name('n'), Mul('*'), Name('res'))),
88        # Exp(GlobalRead(Num('0'))),
89        SUBI SP 1,
90        LOADIN DS ACC 0,
91        STOREIN SP ACC 1,
92        # Exp(GlobalRead(Num('1'))),
93        SUBI SP 1,
94        LOADIN DS ACC 1,
95        STOREIN SP ACC 1,
96        LOADIN SP ACC 2,
97        LOADIN SP IN2 1,
98        MULT ACC IN2,
99        STOREIN SP ACC 2,
```

```
100         ADDI SP 1,
101         # Assign(GlobalWrite(Num('1')), Tmp(Num('1'))),
102         LOADIN SP ACC 1,
103         STOREIN DS ACC 1,
104         ADDI SP 1,
105         # // Assign(Name('n'), BinOp(Name('n'), Sub('-'), Num('1'))),
106         # Exp(GlobalRead(Num('0'))),
107         SUBI SP 1,
108         LOADIN DS ACC 0,
109         STOREIN SP ACC 1,
110         # Exp(Num('1')),
111         SUBI SP 1,
112         LOADI ACC 1,
113         STOREIN SP ACC 1,
114         LOADIN SP ACC 2,
115         LOADIN SP IN2 1,
116         SUB ACC IN2,
117         STOREIN SP ACC 2,
118         ADDI SP 1,
119         # Assign(GlobalWrite(Num('0')), Tmp(Num('1'))),
120         LOADIN SP ACC 1,
121         STOREIN DS ACC 0,
122         ADDI SP 1,
123         Exp
124           GoTo
125             Name 'condition_check.4'
126       ],
127     Block
128       Name 'while_after.0',
129       [
130         # Return(Empty()),
131         LOADIN BAF PC -1
132       ]
133   ]
```

Code 1.10: RETI-Blocks Pass für Codebespiel

1.3.2.6 RETI-Patch Pass

1.3.2.6.1 Abstrakte Syntax

$stmt$	$::=$	$Exit(Num(str))$

Grammar 1.3.4: Abstrakte Syntax für L_{RETI_Patch}

1.3.2.6.2 Codebeispiel

```
1 File
2   Name './example_faculty_it.reti_patch',
3   [
4     Block
5       Name 'start.6',
6       [],
```

```
7     Block
8       Name 'main.5',
9       [
10        # // Assign(Name('n'), Num('4')),
11        # Exp(Num('4')),
12        SUBI SP 1,
13        LOADI ACC 4,
14        STOREIN SP ACC 1,
15        # Assign(GlobalWrite(Num('0')), Tmp(Num('1'))),
16        LOADIN SP ACC 1,
17        STOREIN DS ACC 0,
18        ADDI SP 1,
19        # // Assign(Name('res'), Num('1')),
20        # Exp(Num('1')),
21        SUBI SP 1,
22        LOADI ACC 1,
23        STOREIN SP ACC 1,
24        # Assign(GlobalWrite(Num('1')), Tmp(Num('1'))),
25        LOADIN SP ACC 1,
26        STOREIN DS ACC 1,
27        ADDI SP 1,
28        # // While(Num('1'), [])
29      ],
30    Block
31      Name 'condition_check.4',
32      [
33        # // IfElse(Num('1'), GoTo(Name('while_branch.3')), GoTo(Name('while_after.0'))),
34        # Exp(Num('1')),
35        SUBI SP 1,
36        LOADI ACC 1,
37        STOREIN SP ACC 1,
38        # IfElse(Tmp(Num('1')), GoTo(Name('while_branch.3')), GoTo(Name('while_after.0'))),
39        LOADIN SP ACC 1,
40        ADDI SP 1,
41        JUMP== GoTo
42                 Name 'while_after.0';
43      ],
44    Block
45      Name 'while_branch.3',
46      [
47        # // If(Atom(Name('n'), Eq('==')), Num('1')), []),
48        # // IfElse(Atom(Name('n'), Eq('==')), Num('1')), GoTo(Name('if.2')),
          ↪  GoTo(Name('if_else_after.1'))),
49        # Exp(GlobalRead(Num('0'))),
50        SUBI SP 1,
51        LOADIN DS ACC 0,
52        STOREIN SP ACC 1,
53        # Exp(Num('1')),
54        SUBI SP 1,
55        LOADI ACC 1,
56        STOREIN SP ACC 1,
57        LOADIN SP ACC 2,
58        LOADIN SP IN2 1,
59        SUB ACC IN2,
60        JUMP== 3;,
61        LOADI ACC 0,
62        JUMP 2;,
```

```
63        LOADI ACC 1,
64        STOREIN SP ACC 2,
65        ADDI SP 1,
66        # IfElse(Tmp(Num('1')), GoTo(Name('if.2')), GoTo(Name('if_else_after.1'))),
67        LOADIN SP ACC 1,
68        ADDI SP 1,
69        JUMP== GoTo
70                 Name 'if_else_after.1';
71      ],
72    Block
73      Name 'if.2',
74      [
75        # Return(Empty()),
76        LOADIN BAF PC -1
77      ],
78    Block
79      Name 'if_else_after.1',
80      [
81        # // Assign(Name('res'), BinOp(Name('n'), Mul('*'), Name('res'))),
82        # Exp(GlobalRead(Num('0'))),
83        SUBI SP 1,
84        LOADIN DS ACC 0,
85        STOREIN SP ACC 1,
86        # Exp(GlobalRead(Num('1'))),
87        SUBI SP 1,
88        LOADIN DS ACC 1,
89        STOREIN SP ACC 1,
90        LOADIN SP ACC 2,
91        LOADIN SP IN2 1,
92        MULT ACC IN2,
93        STOREIN SP ACC 2,
94        ADDI SP 1,
95        # Assign(GlobalWrite(Num('1')), Tmp(Num('1'))),
96        LOADIN SP ACC 1,
97        STOREIN DS ACC 1,
98        ADDI SP 1,
99        # // Assign(Name('n'), BinOp(Name('n'), Sub('-'), Num('1'))),
100       # Exp(GlobalRead(Num('0'))),
101       SUBI SP 1,
102       LOADIN DS ACC 0,
103       STOREIN SP ACC 1,
104       # Exp(Num('1')),
105       SUBI SP 1,
106       LOADI ACC 1,
107       STOREIN SP ACC 1,
108       LOADIN SP ACC 2,
109       LOADIN SP IN2 1,
110       SUB ACC IN2,
111       STOREIN SP ACC 2,
112       ADDI SP 1,
113       # Assign(GlobalWrite(Num('0')), Tmp(Num('1'))),
114       LOADIN SP ACC 1,
115       STOREIN DS ACC 0,
116       ADDI SP 1,
117       Exp
118         GoTo
119           Name 'condition_check.4'
```

```
120       ],
121     Block
122       Name 'while_after.0',
123       [
124         # Return(Empty()),
125         LOADIN BAF PC -1
126       ]
127   ]
```

Code 1.11: RETI-Patch Pass für Codebespiel

1.3.2.7 RETI Pass

1.3.2.7.1 Konkrette und Abstrakte Syntax

dig_no_0	::=	"1" | "2" | "3" | "4" | "5" | "6"	*L_Program*
	|	"7" | "8" | "9"	
dig_with_0	::=	"0" | *dig_no_0*	
num	::=	"0" | *dig_no_0dig_with_0* $*$ | "$-$"*dig_with_0*$*$	
letter	::=	"*a*"..."*Z*"	
name	::=	*letter*(*letter* | *dig_with_0* | _)$*$	
reg	::=	"*ACC*" | "*IN*1" | "*IN*2" | "*PC*" | "*SP*"	
	|	"*BAF*" | "*CS*" | "*DS*"	
arg	::=	*reg* | *num*	
rel	::=	"==" | "!=" | "<" | "<=" | ">"	
	|	">=" | "*_NOP*"	

Grammar 1.3.5: Konkrette Syntax für L_{RETI_Lex}

instr	::=	"*ADD*" *reg arg* | "*ADDI*" *reg num* | "*SUB*" *reg arg*	*L_Program*
	|	"*SUBI*" *reg num* | "*MULT*" *reg arg* | "*MULTI*" *reg num*	
	|	"*DIV*" *reg arg* | "*DIVI*" *reg num* | "*MOD*" *reg arg*	
	|	"*MODI*" *reg num* | "*OPLUS*" *reg arg* | "*OPLUSI*" *reg num*	
	|	"*OR*" *reg arg* | "*ORI*" *reg num*	
	|	"*AND*" *reg arg* | "*ANDI*" *reg num*	
	|	"*LOAD*" *reg num* | "*LOADIN*" *arg arg num*	
	|	"*LOADI*" *reg num*	
	|	"*STORE*" *reg num* | "*STOREIN*" *arg argnum*	
	|	"*MOVE*" *reg reg*	
	|	"*JUMP*" *rel num* | *INT num* | *RTI*	
	|	"*CALL*" "*INPUT*" *reg* | "*CALL*" "*PRINT*" *reg*	
program	::=	*name* (*instr*";")$*$	

Grammar 1.3.6: Konkrette Syntax für L_{RETI_Parse}

reg	$::=$	$ACC() \mid IN1() \mid IN2() \mid PC() \mid SP() \mid BAF()$	*L_Program*
	$\mid$	$CS() \mid DS()$	
arg	$::=$	$Reg(\langle reg \rangle) \mid Num(str)$	
rel	$::=$	$Eq() \mid NEq() \mid Lt() \mid LtE() \mid Gt() \mid GtE()$	
	$\mid$	$Always() \mid NOp()$	
op	$::=$	$Add() \mid Addi() \mid Sub() \mid Subi() \mid Mult()$	
	$\mid$	$Multi() \mid Div() \mid Divi()$	
	$\mid$	$Mod() \mid Modi() \mid Oplus() \mid Oplusi() \mid Or()$	
	$\mid$	$Ori() \mid And() \mid Andi()$	
	$\mid$	$Load() \mid Loadin() \mid Loadi()$	
	$\mid$	$Store() \mid Storein() \mid Move()$	
$instr$	$::=$	$Instr(\langle op \rangle, \langle arg \rangle+) \mid Jump(\langle rel \rangle, Num(str)) \mid Int(Num(str))$	
	$\mid$	$RTI() \mid Call(Name('print'), \langle reg \rangle) \mid Call(Name('input'), \langle reg \rangle)$	
	$\mid$	$SingleLineComment(str, str)$	
$program$	$::=$	$Program(Name(str), \langle instr \rangle*)$	

Grammar 1.3.7: Abstrakte Syntax für L_{RETI}

1.3.2.7.2 Codebeispiel

```
# // Assign(Name('n'), Num('4'))
# Exp(Num('4'))
SUBI SP 1;
LOADI ACC 4;
STOREIN SP ACC 1;
# Assign(GlobalWrite(Num('0')), Tmp(Num('1')))
LOADIN SP ACC 1;
STOREIN DS ACC 0;
ADDI SP 1;
# // Assign(Name('res'), Num('1'))
# Exp(Num('1'))
SUBI SP 1;
LOADI ACC 1;
STOREIN SP ACC 1;
# Assign(GlobalWrite(Num('1')), Tmp(Num('1')))
LOADIN SP ACC 1;
STOREIN DS ACC 1;
ADDI SP 1;
# // While(Num('1'), [])
# // IfElse(Num('1'), GoTo(Name('while_branch.3')), GoTo(Name('while_after.0')))
# Exp(Num('1'))
SUBI SP 1;
LOADI ACC 1;
STOREIN SP ACC 1;
# IfElse(Tmp(Num('1')), GoTo(Name('while_branch.3')), GoTo(Name('while_after.0')))
LOADIN SP ACC 1;
ADDI SP 1;
JUMP== 49;
# // If(Atom(Name('n'), Eq('=='), Num('1')), [])
# // IfElse(Atom(Name('n'), Eq('=='), Num('1')), GoTo(Name('if.2')),
↪  GoTo(Name('if_else_after.1')))
# Exp(GlobalRead(Num('0')))
SUBI SP 1;
LOADIN DS ACC 0;
```

```
34 STOREIN SP ACC 1;
35 # Exp(Num('1'))
36 SUBI SP 1;
37 LOADI ACC 1;
38 STOREIN SP ACC 1;
39 LOADIN SP ACC 2;
40 LOADIN SP IN2 1;
41 SUB ACC IN2;
42 JUMP== 3;
43 LOADI ACC 0;
44 JUMP 2;
45 LOADI ACC 1;
46 STOREIN SP ACC 2;
47 ADDI SP 1;
48 # IfElse(Tmp(Num('1')), GoTo(Name('if.2')), GoTo(Name('if_else_after.1')))
49 LOADIN SP ACC 1;
50 ADDI SP 1;
51 JUMP== 2;
52 # Return(Empty())
53 LOADIN BAF PC -1;
54 # // Assign(Name('res'), BinOp(Name('n'), Mul('*'), Name('res')))
55 # Exp(GlobalRead(Num('0')))
56 SUBI SP 1;
57 LOADIN DS ACC 0;
58 STOREIN SP ACC 1;
59 # Exp(GlobalRead(Num('1')))
60 SUBI SP 1;
61 LOADIN DS ACC 1;
62 STOREIN SP ACC 1;
63 LOADIN SP ACC 2;
64 LOADIN SP IN2 1;
65 MULT ACC IN2;
66 STOREIN SP ACC 2;
67 ADDI SP 1;
68 # Assign(GlobalWrite(Num('1')), Tmp(Num('1')))
69 LOADIN SP ACC 1;
70 STOREIN DS ACC 1;
71 ADDI SP 1;
72 # // Assign(Name('n'), BinOp(Name('n'), Sub('-'), Num('1')))
73 # Exp(GlobalRead(Num('0')))
74 SUBI SP 1;
75 LOADIN DS ACC 0;
76 STOREIN SP ACC 1;
77 # Exp(Num('1'))
78 SUBI SP 1;
79 LOADI ACC 1;
80 STOREIN SP ACC 1;
81 LOADIN SP ACC 2;
82 LOADIN SP IN2 1;
83 SUB ACC IN2;
84 STOREIN SP ACC 2;
85 ADDI SP 1;
86 # Assign(GlobalWrite(Num('0')), Tmp(Num('1')))
87 LOADIN SP ACC 1;
88 STOREIN DS ACC 0;
89 ADDI SP 1;
90 JUMP -53;
```

```
91 # Return(Empty())
92 LOADIN BAF PC -1;
```

Code 1.12: RETI Pass für Codebespiel

1.3.3 Umsetzung von Pointern

1.3.3.1 Referenzierung

Die **Referenzierung** `&var` wird im Folgenden anhand des Beispiels Code 1.13 erklärt.

```
1 void main() {
2   int var = 42;
3   int *pntr = &var;
4 }
```

Code 1.13: PicoC Code für Pointer Referenzierung

Der Node `Ref(Name('var'))` repräsentiert in der **Abstrakten Syntax** in Code 1.14 eine **Referenzierung** `&var`.

```
1  File
2    Name './example_pntr_ref.ast',
3    [
4      FunDef
5        VoidType 'void',
6        Name 'main',
7        [],
8        [
9          Assign
10           Alloc
11             Writeable,
12             IntType 'int',
13             Name 'var',
14           Num '42',
15         Assign
16           Alloc
17             Writeable,
18             PntrDecl
19               Num '1',
20               IntType 'int',
21             Name 'pntr',
22           Ref
23             Name 'var'
24       ]
25   ]
```

Code 1.14: Abstract Syntax Tree für Pointer Referenzierung

Im **PicoC-Mon Pass** in Code 1.15 wird der Node `Ref(Name('var'))` durch die Nodes `Ref(GlobalRead(Num('0')))` und `Assign(GlobalWrite(Num('1')), Tmp(Num('1')))` ersetzt. Im Fall, dass in `Ref(exp))` das `exp` vielleicht nicht direkt ein `Name('var')` enthält und `exp` vielleicht ein `Subscr(Attr(Name('var')))` ist, sind noch weitere Anweisungen zwischen Zeile `14` und `15` nötig, die sich in diesem Beispiel um das Übersetzen von `Subscr(exp)` und `Attr(exp)` kümmern. Die Vorgehen hierfür ist in Subkapitel 1.3.6.2 erklärt.

```
File
  Name './example_pntr_ref.picoc_mon',
  [
    Block
      Name 'main.0',
      [
        // Assign(Name('var'), Num('42')),
        Exp
          Num '42',
        Assign
          GlobalWrite
            Num '0',
          Tmp
            Num '1',
        // Assign(Name('pntr'), Ref(Name('var'))),
        Ref
          GlobalRead
            Num '0',
        Assign
          GlobalWrite
            Num '1',
          Tmp
            Num '1',
        Return
          Empty
      ]
  ]
```

Code 1.15: PicoC Mon Pass für Pointer Referenzierung

Im **PicoC-Blocks Pass** in Code 1.16 wird der die Nodes

```
File
  Name './example_pntr_ref.reti_blocks',
  [
    Block
      Name 'main.0',
      [
        # // Assign(Name('var'), Num('42')),
        # Exp(Num('42')),
        SUBI SP 1,
        LOADI ACC 42,
        STOREIN SP ACC 1,
        # Assign(GlobalWrite(Num('0')), Tmp(Num('1'))),
        LOADIN SP ACC 1,
        STOREIN DS ACC 0,
        ADDI SP 1,
        # // Assign(Name('pntr'), Ref(Name('var'))),
        # Ref(GlobalRead(Num('0'))),
        SUBI SP 1,
        LOADI IN1 0,
        ADD IN1 DS,
        STOREIN SP IN1 1,
        # Assign(GlobalWrite(Num('1')), Tmp(Num('1'))),
```

```
23          LOADIN SP ACC 1,
24          STOREIN DS ACC 1,
25          ADDI SP 1,
26          # Return(Empty()),
27          LOADIN BAF PC -1
28        ]
29    ]
```

Code 1.16: RETI Blocks Pass für Pointer Referenzierung

1.3.3.2 Pointer Dereferenzierung durch Zugriff auf Arrayindex ersetzen

```
1 void main() {
2   int var = 42;
3   int *pntr = &var;
4   *pntr;
5 }
```

Code 1.17: PicoC Code für Pointer Dereferenzierung

```
1  File
2    Name './example_pntr_deref.ast',
3    [
4      FunDef
5        VoidType 'void',
6        Name 'main',
7        [],
8        [
9          Assign
10           Alloc
11             Writeable,
12             IntType 'int',
13             Name 'var',
14           Num '42',
15         Assign
16           Alloc
17             Writeable,
18             PntrDecl
19               Num '1',
20               IntType 'int',
21             Name 'pntr',
22           Ref
23             Name 'var',
24         Exp
25           Deref
26             Name 'pntr',
27             Num '0'
28       ]
29   ]
```

Code 1.18: Abstract Syntax Tree für Pointer Dereferenzierung

```
File
  Name './example_pntr_deref.picoc_shrink',
  [
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Assign
          Alloc
            Writeable,
            IntType 'int',
            Name 'var',
          Num '42',
        Assign
          Alloc
            Writeable,
            PntrDecl
              Num '1',
              IntType 'int',
            Name 'pntr',
          Ref
            Name 'var',
        Exp
          Subscr
            Name 'pntr',
            Num '0'
      ]
  ]
```

Code 1.19: PicoC Shrink Pass für Pointer Dereferenzierung

1.3.4 Umsetzung von Arrays

1.3.4.1 Initialisierung von Arrays

```
void main() {
  int ar[2][1] = {{4}, {2}};
}
```

Code 1.20: PicoC Code für Array Initialisierung

```
File
  Name './example_array_init.ast',
  [
    FunDef
      VoidType 'void',
      Name 'main',
      [],
```

```
8       [
9         Assign
10          Alloc
11            Writeable,
12            ArrayDecl
13              [
14                Num '2',
15                Num '1'
16              ],
17              IntType 'int',
18            Name 'ar',
19          Array
20            [
21              Array
22                [
23                  Num '4'
24                ],
25              Array
26                [
27                  Num '2'
28                ]
29            ]
30      ]
31  ]
```

Code 1.21: Abstract Syntax Tree für Array Initialisierung

```
1 SymbolTable
2   [
3     Symbol(
4       {
5         type qualifier:         Empty()
6         datatype:               FunDecl(VoidType('void'), Name('main'), [])
7         name:                   Name('main')
8         value or address:       Empty()
9         position:               Pos(Num('1'), Num('5'))
10        size:                   Empty()
11      },
12    Symbol(
13      {
14        type qualifier:         Writeable()
15        datatype:               ArrayDecl([Num('2'), Num('1')], IntType('int'))
16        name:                   Name('ar@main')
17        value or address:       Num('0')
18        position:               Pos(Num('2'), Num('6'))
19        size:                   Num('2')
20      }
21  ]
```

Code 1.22: Symboltabelle für Array Initialisierung

```
File
  Name './example_array_init.picoc_mon',
  [
    Block
      Name 'main.0',
      [
        // Assign(Name('ar'), Array([Array([Num('4')]), Array([Num('2')])])),
        Exp
          Num '4',
        Exp
          Num '2',
        Assign
          GlobalWrite
            Num '0',
          Tmp
            Num '2',
        Return
          Empty
      ]
  ]
```

Code 1.23: PicoC Mon Pass für Array Initialisierung

```
File
  Name './example_array_init.reti_blocks',
  [
    Block
      Name 'main.0',
      [
        # // Assign(Name('ar'), Array([Array([Num('4')]), Array([Num('2')])])),
        # Exp(Num('4')),
        SUBI SP 1,
        LOADI ACC 4,
        STOREIN SP ACC 1,
        # Exp(Num('2')),
        SUBI SP 1,
        LOADI ACC 2,
        STOREIN SP ACC 1,
        # Assign(GlobalWrite(Num('0')), Tmp(Num('2'))),
        LOADIN SP ACC 1,
        STOREIN DS ACC 1,
        LOADIN SP ACC 2,
        STOREIN DS ACC 0,
        ADDI SP 2,
        # Return(Empty()),
        LOADIN BAF PC -1
      ]
  ]
```

Code 1.24: RETI Blocks Pass für Array Initialisierung

1.3.4.2 Zugriff auf Arrayindex

Der Zugriff auf einen bestimmten Index eines Arrays ist wie folgt umgesetzt:

```
void main() {
  int ar[2] = {1, 2};
  ar[2];
}
```

Code 1.25: PicoC Code für Zugriff auf Arrayindex

```
File
  Name './example_array_access.ast',
  [
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Assign
          Alloc
            Writeable,
            ArrayDecl
              [
                Num '2'
              ],
              IntType 'int',
            Name 'ar',
          Array
            [
              Num '1',
              Num '2'
            ],
        Exp
          Subscr
            Name 'ar',
            Num '2'
      ]
  ]
```

Code 1.26: Abstract Syntax Tree für Zugriff auf Arrayindex

```
File
  Name './example_array_access.picoc_mon',
  [
    Block
      Name 'main.0',
      [
        // Assign(Name('ar'), Array([Num('1'), Num('2')])),
        Exp
          Num '1',
        Exp
          Num '2',
```

```
        Assign
          GlobalWrite
            Num '0',
          Tmp
            Num '2',
        // Exp(Subscr(Name('ar'), Num('2'))),
        Ref
          GlobalRead
            Num '0',
        Exp
          Num '2',
        Ref
          Subscr
            Tmp
              Num '2',
            Tmp
              Num '1',
        Exp
          Subscr
            Tmp
              Num '1',
            Num '0',
        Return
          Empty
      ]
  ]
```

Code 1.27: PicoC Mon Pass für Zugriff auf Arrayindex

```
File
  Name './example_array_access.reti_blocks',
  [
    Block
      Name 'main.0',
      [
        # // Assign(Name('ar'), Array([Num('1'), Num('2')])),
        # Exp(Num('1')),
        SUBI SP 1,
        LOADI ACC 1,
        STOREIN SP ACC 1,
        # Exp(Num('2')),
        SUBI SP 1,
        LOADI ACC 2,
        STOREIN SP ACC 1,
        # Assign(GlobalWrite(Num('0')), Tmp(Num('2'))),
        LOADIN SP ACC 1,
        STOREIN DS ACC 1,
        LOADIN SP ACC 2,
        STOREIN DS ACC 0,
        ADDI SP 2,
        # // Exp(Subscr(Name('ar'), Num('2'))),
        # Ref(GlobalRead(Num('0'))),
        SUBI SP 1,
        LOADI IN1 0,
```

```
        ADD IN1 DS,
        STOREIN SP IN1 1,
        # Exp(Num('2')),
        SUBI SP 1,
        LOADI ACC 2,
        STOREIN SP ACC 1,
        # Ref(Subscr(Tmp(Num('2')), Tmp(Num('1')))),
        LOADIN SP IN1 2,
        LOADIN SP IN2 1,
        MULTI IN2 1,
        ADD IN1 IN2,
        ADDI SP 1,
        STOREIN SP IN1 1,
        # Exp(Subscr(Tmp(Num('1')), Num('0'))),
        LOADIN SP IN1 1,
        LOADIN IN1 ACC 0,
        STOREIN SP ACC 1,
        # Return(Empty()),
        LOADIN BAF PC -1
      ]
  ]
```

Code 1.28: RETI Blocks Pass für Zugriff auf Arrayindex

1.3.4.3 Zuweisung an Arrayindex

```
void main() {
  int ar[2];
  ar[2] = 42;
}
```

Code 1.29: PicoC Code für Zuweisung an Arrayindex

```
File
  Name './example_array_assignment.ast',
  [
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Exp
          Alloc
            Writeable,
            ArrayDecl
              [
                Num '2'
              ],
              IntType 'int',
            Name 'ar',
        Assign
          Subscr
```

```
            Name 'ar',
            Num '2',
          Num '42'
      ]
  ]
```

Code 1.30: Abstract Syntax Tree für Zuweisung an Arrayindex

```
File
  Name './example_array_assignment.picoc_mon',
  [
    Block
      Name 'main.0',
      [
        // Assign(Subscr(Name('ar'), Num('2')), Num('42')),
        Exp
          Num '42',
        Ref
          GlobalRead
            Num '0',
        Exp
          Num '2',
        Ref
          Subscr
            Tmp
              Num '2',
            Tmp
              Num '1',
        Assign
          Subscr
            Tmp
              Num '1',
            Num '0',
          Tmp
            Num '2',
        Return
          Empty
      ]
  ]
```

Code 1.31: PicoC Mon Pass für Zuweisung an Arrayindex

```
File
  Name './example_array_assignment.reti_blocks',
  [
    Block
      Name 'main.0',
      [
        # // Assign(Subscr(Name('ar'), Num('2')), Num('42')),
        # Exp(Num('42')),
        SUBI SP 1,
```

```
10        LOADI ACC 42,
11        STOREIN SP ACC 1,
12        # Ref(GlobalRead(Num('0'))),
13        SUBI SP 1,
14        LOADI IN1 0,
15        ADD IN1 DS,
16        STOREIN SP IN1 1,
17        # Exp(Num('2')),
18        SUBI SP 1,
19        LOADI ACC 2,
20        STOREIN SP ACC 1,
21        # Ref(Subscr(Tmp(Num('2')), Tmp(Num('1')))),
22        LOADIN SP IN1 2,
23        LOADIN SP IN2 1,
24        MULTI IN2 1,
25        ADD IN1 IN2,
26        ADDI SP 1,
27        STOREIN SP IN1 1,
28        # Assign(Subscr(Tmp(Num('1')), Num('0')), Tmp(Num('2'))),
29        LOADIN SP IN1 1,
30        LOADIN SP ACC 2,
31        ADDI SP 2,
32        STOREIN IN1 ACC 0,
33        # Return(Empty()),
34        LOADIN BAF PC -1
35      ]
36  ]
```

Code 1.32: RETI Blocks Pass für Zuweisung an Arrayindex

1.3.5 Umsetzung von Structs

1.3.5.1 Deklaration von Structs

```
1 struct st1 {int *ar[3];};
2
3 struct st2 {struct st1 st;};
4
5 void main() {
6 }
```

Code 1.33: PicoC Code für Deklaration von Structs

```
1 SymbolTable
2   [
3     Symbol(
4       {
5         type qualifier:         Empty()
6         datatype:               ArrayDecl([Num('3')], PntrDecl(Num('1'), IntType('int')))
7         name:                   Name('ar@st1')
8         value or address:       Empty()
```

```
 9         position:               Pos(Num('1'), Num('17'))
10         size:                   Num('3')
11       },
12     Symbol(
13       {
14         type qualifier:         Empty()
15         datatype:               StructDecl(Name('st1'), [Alloc(Writeable(),
           ↪  ArrayDecl([Num('3')], PntrDecl(Num('1'), IntType('int'))), Name('ar'))])
16         name:                   Name('st1')
17         value or address:       [Name('ar@st1')]
18         position:               Pos(Num('1'), Num('7'))
19         size:                   Num('3')
20       },
21     Symbol(
22       {
23         type qualifier:         Empty()
24         datatype:               StructSpec(Name('st1'))
25         name:                   Name('st@st2')
26         value or address:       Empty()
27         position:               Pos(Num('3'), Num('23'))
28         size:                   Num('3')
29       },
30     Symbol(
31       {
32         type qualifier:         Empty()
33         datatype:               StructDecl(Name('st2'), [Alloc(Writeable(),
           ↪  StructSpec(Name('st1')), Name('st'))])
34         name:                   Name('st2')
35         value or address:       [Name('st@st2')]
36         position:               Pos(Num('3'), Num('7'))
37         size:                   Num('3')
38       },
39     Symbol(
40       {
41         type qualifier:         Empty()
42         datatype:               FunDecl(VoidType('void'), Name('main'), [])
43         name:                   Name('main')
44         value or address:       Empty()
45         position:               Pos(Num('5'), Num('5'))
46         size:                   Empty()
47       }
48   ]
```

Code 1.34: Symboltabelle für Deklaration von Structs

1.3.5.2 Initialisierung von Structs

```
1 struct st1 {int *pntr[1];};
2
3 struct st2 {struct st1 st;};
4
5 void main() {
6   int var = 42;
7   struct st1 st = {.st={.pntr={{&var}}}};
8 }
```

Code 1.35: PicoC Code für Initialisierung von Structs

```
File
  Name './example_struct_init.ast',
  [
    StructDecl
      Name 'st1',
      [
        Alloc
          Writeable,
          ArrayDecl
            [
              Num '1'
            ],
            PntrDecl
              Num '1',
              IntType 'int',
          Name 'pntr'
      ],
    StructDecl
      Name 'st2',
      [
        Alloc
          Writeable,
          StructSpec
            Name 'st1',
          Name 'st'
      ],
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Assign
          Alloc
            Writeable,
            IntType 'int',
            Name 'var',
          Num '42',
        Assign
          Alloc
            Writeable,
            StructSpec
              Name 'st1',
            Name 'st',
          Struct
            [
              Assign
                Name 'st',
                Struct
                  [
                    Assign
                      Name 'pntr',
                      Array
```

```
                    [
                      Array
                        [
                          Ref
                            Name 'var'
                        ]
                    ]
                ]
            ]
        ]
    ]
```

Code 1.36: Abstract Syntax Tree für Initialisierung von Structs

```
SymbolTable
  [
    Symbol(
      {
        type qualifier:         Empty()
        datatype:               ArrayDecl([Num('1')], PntrDecl(Num('1'), IntType('int')))
        name:                   Name('pntr@st1')
        value or address:       Empty()
        position:               Pos(Num('1'), Num('17'))
        size:                   Num('1')
      },
    Symbol(
      {
        type qualifier:         Empty()
        datatype:               StructDecl(Name('st1'), [Alloc(Writeable(),
        ↪  ArrayDecl([Num('1')], PntrDecl(Num('1'), IntType('int'))), Name('pntr'))])
        name:                   Name('st1')
        value or address:       [Name('pntr@st1')]
        position:               Pos(Num('1'), Num('7'))
        size:                   Num('1')
      },
    Symbol(
      {
        type qualifier:         Empty()
        datatype:               StructSpec(Name('st1'))
        name:                   Name('st@st2')
        value or address:       Empty()
        position:               Pos(Num('3'), Num('23'))
        size:                   Num('1')
      },
    Symbol(
      {
        type qualifier:         Empty()
        datatype:               StructDecl(Name('st2'), [Alloc(Writeable(),
        ↪  StructSpec(Name('st1')), Name('st'))])
        name:                   Name('st2')
        value or address:       [Name('st@st2')]
        position:               Pos(Num('3'), Num('7'))
        size:                   Num('1')
      },
```

```
    Symbol(
      {
        type qualifier:         Empty()
        datatype:               FunDecl(VoidType('void'), Name('main'), [])
        name:                   Name('main')
        value or address:       Empty()
        position:               Pos(Num('5'), Num('5'))
        size:                   Empty()
      },
    Symbol(
      {
        type qualifier:         Writeable()
        datatype:               IntType('int')
        name:                   Name('var@main')
        value or address:       Num('0')
        position:               Pos(Num('6'), Num('6'))
        size:                   Num('1')
      },
    Symbol(
      {
        type qualifier:         Writeable()
        datatype:               StructSpec(Name('st1'))
        name:                   Name('st@main')
        value or address:       Num('1')
        position:               Pos(Num('7'), Num('13'))
        size:                   Num('1')
      }
  ]
```

Code 1.37: Symboltabelle für Initialisierung von Structs

```
File
  Name './example_struct_init.picoc_mon',
  [
    Block
      Name 'main.0',
      [
        // Assign(Name('var'), Num('42')),
        Exp
          Num '42',
        Assign
          GlobalWrite
            Num '0',
          Tmp
            Num '1',
        // Assign(Name('st'), Struct([Assign(Name('st'), Struct([Assign(Name('pntr'),
        ↪  Array([Array([Ref(Name('var'))])]))]))])),
        Ref
          GlobalRead
            Num '0',
        Assign
          GlobalWrite
            Num '1',
          Tmp
```

```
23            Num '1',
24          Return
25            Empty
26        ]
27    ]
```

Code 1.38: PicoC Mon Pass für Initialisierung von Structs

```
1  File
2    Name './example_struct_init.reti_blocks',
3    [
4      Block
5        Name 'main.0',
6        [
7          # // Assign(Name('var'), Num('42')),
8          # Exp(Num('42')),
9          SUBI SP 1,
10         LOADI ACC 42,
11         STOREIN SP ACC 1,
12         # Assign(GlobalWrite(Num('0')), Tmp(Num('1'))),
13         LOADIN SP ACC 1,
14         STOREIN DS ACC 0,
15         ADDI SP 1,
16         # // Assign(Name('st'), Struct([Assign(Name('st'), Struct([Assign(Name('pntr'),
           ↪  Array([Array([Ref(Name('var'))])]))]))])),
17         # Ref(GlobalRead(Num('0'))),
18         SUBI SP 1,
19         LOADI IN1 0,
20         ADD IN1 DS,
21         STOREIN SP IN1 1,
22         # Assign(GlobalWrite(Num('1')), Tmp(Num('1'))),
23         LOADIN SP ACC 1,
24         STOREIN DS ACC 1,
25         ADDI SP 1,
26         # Return(Empty()),
27         LOADIN BAF PC -1
28       ]
29   ]
```

Code 1.39: RETI Blocks Pass für Initialisierung von Structs

1.3.5.3 Zugriff auf Structattribut

```
1 struct pos {int x; int y;};
2
3 void main() {
4   struct pos st = {.x=4, .y=2};
5   st.y;
6 }
```

Code 1.40: PicoC Code für Zugriff auf Structattribut

```
File
  Name './example_struct_attr_access.ast',
  [
    StructDecl
      Name 'pos',
      [
        Alloc
          Writeable,
          IntType 'int',
          Name 'x',
        Alloc
          Writeable,
          IntType 'int',
          Name 'y'
      ],
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Assign
          Alloc
            Writeable,
            StructSpec
              Name 'pos',
            Name 'st',
          Struct
            [
              Assign
                Name 'x',
                Num '4',
              Assign
                Name 'y',
                Num '2'
            ],
        Exp
          Attr
            Name 'st',
            Name 'y'
      ]
  ]
```

Code 1.41: Abstract Syntax Tree für Zugriff auf Structattribut

```
File
  Name './example_struct_attr_access.picoc_mon',
  [
    Block
      Name 'main.0',
      [
        // Assign(Name('st'), Struct([Assign(Name('x'), Num('4')), Assign(Name('y'),
        ↪  Num('2'))])),
        Exp
```

```
          Num '4',
        Exp
          Num '2',
        Assign
          GlobalWrite
            Num '0',
          Tmp
            Num '2',
        Ref
          GlobalRead
            Num '0',
        Ref
          Attr
            Tmp
              Num '1',
            Name 'y',
        Exp
          Subscr
            Tmp
              Num '1',
            Num '0',
        Return
          Empty
      ]
  ]
```

Code 1.42: PicoC Mon Pass für Zugriff auf Structattribut

```
File
  Name './example_struct_attr_access.reti_blocks',
  [
    Block
      Name 'main.0',
      [
        # // Assign(Name('st'), Struct([Assign(Name('x'), Num('4')), Assign(Name('y'),
        ↪  Num('2'))])),
        # Exp(Num('4')),
        SUBI SP 1,
        LOADI ACC 4,
        STOREIN SP ACC 1,
        # Exp(Num('2')),
        SUBI SP 1,
        LOADI ACC 2,
        STOREIN SP ACC 1,
        # Assign(GlobalWrite(Num('0')), Tmp(Num('2'))),
        LOADIN SP ACC 1,
        STOREIN DS ACC 1,
        LOADIN SP ACC 2,
        STOREIN DS ACC 0,
        ADDI SP 2,
        # Ref(GlobalRead(Num('0'))),
        SUBI SP 1,
        LOADI IN1 0,
        ADD IN1 DS,
```

```
26          STOREIN SP IN1 1,
27          # Ref(Attr(Tmp(Num('1')), Name('y'))),
28          LOADIN SP IN1 1,
29          ADDI IN1 1,
30          STOREIN SP IN1 1,
31          # Exp(Subscr(Tmp(Num('1')), Num('0'))),
32          LOADIN SP IN1 1,
33          LOADIN IN1 ACC 0,
34          STOREIN SP ACC 1,
35          # Return(Empty()),
36          LOADIN BAF PC -1
37        ]
38    ]
```

Code 1.43: RETI Blocks Pass für Zugriff auf Structattribut

1.3.5.4 Zuweisung an Structattribut

```
1 struct pos {int x; int y;};
2
3 void main() {
4   struct pos st = {.x=4, .y=2};
5   st.y = 42;
6 }
```

Code 1.44: PicoC Code für Zuweisung an Structattribut

```
1 File
2   Name './example_struct_attr_assignment.ast',
3   [
4     StructDecl
5       Name 'pos',
6       [
7         Alloc
8           Writeable,
9           IntType 'int',
10          Name 'x',
11        Alloc
12          Writeable,
13          IntType 'int',
14          Name 'y'
15      ],
16    FunDef
17      VoidType 'void',
18      Name 'main',
19      [],
20      [
21        Assign
22          Alloc
23            Writeable,
24            StructSpec
25              Name 'pos',
```

```
26            Name 'st',
27          Struct
28            [
29              Assign
30                Name 'x',
31                Num '4',
32              Assign
33                Name 'y',
34                Num '2'
35            ],
36        Assign
37          Attr
38            Name 'st',
39            Name 'y',
40          Num '42'
41      ]
42  ]
```

Code 1.45: Abstract Syntax Tree für Zuweisung an Structattribut

```
1 File
2   Name './example_struct_attr_assignment.picoc_mon',
3   [
4     Block
5       Name 'main.0',
6       [
7         // Assign(Name('st'), Struct([Assign(Name('x'), Num('4')), Assign(Name('y'),
          ↪  Num('2'))])),
8         Exp
9           Num '4',
10        Exp
11          Num '2',
12        Assign
13          GlobalWrite
14            Num '0',
15          Tmp
16            Num '2',
17        // Assign(Attr(Name('st'), Name('y')), Num('42')),
18        Exp
19          Num '42',
20        Ref
21          GlobalRead
22            Num '0',
23        Ref
24          Attr
25            Tmp
26              Num '1',
27            Name 'y',
28        Assign
29          Subscr
30            Tmp
31              Num '1',
32            Num '0',
33          Tmp
```

```
34            Num '2',
35          Return
36            Empty
37        ]
38    ]
```

Code 1.46: PicoC Mon Pass für Zuweisung an Structattribut

```
1 File
2   Name './example_struct_attr_assignment.reti_blocks',
3   [
4     Block
5       Name 'main.0',
6       [
7         # // Assign(Name('st'), Struct([Assign(Name('x'), Num('4')), Assign(Name('y'),
          ↪  Num('2'))])),
8         # Exp(Num('4')),
9         SUBI SP 1,
10        LOADI ACC 4,
11        STOREIN SP ACC 1,
12        # Exp(Num('2')),
13        SUBI SP 1,
14        LOADI ACC 2,
15        STOREIN SP ACC 1,
16        # Assign(GlobalWrite(Num('0')), Tmp(Num('2'))),
17        LOADIN SP ACC 1,
18        STOREIN DS ACC 1,
19        LOADIN SP ACC 2,
20        STOREIN DS ACC 0,
21        ADDI SP 2,
22        # // Assign(Attr(Name('st'), Name('y')), Num('42')),
23        # Exp(Num('42')),
24        SUBI SP 1,
25        LOADI ACC 42,
26        STOREIN SP ACC 1,
27        # Ref(GlobalRead(Num('0'))),
28        SUBI SP 1,
29        LOADI IN1 0,
30        ADD IN1 DS,
31        STOREIN SP IN1 1,
32        # Ref(Attr(Tmp(Num('1')), Name('y'))),
33        LOADIN SP IN1 1,
34        ADDI IN1 1,
35        STOREIN SP IN1 1,
36        # Assign(Subscr(Tmp(Num('1')), Num('0')), Tmp(Num('2'))),
37        LOADIN SP IN1 1,
38        LOADIN SP ACC 2,
39        ADDI SP 2,
40        STOREIN IN1 ACC 0,
41        # Return(Empty()),
42        LOADIN BAF PC -1
43      ]
44  ]
```

Code 1.47: RETI Blocks Pass für Zuweisung an Structattribut

1.3.6 Umsetzung der Derived Datatypes im Zusammenspiel

1.3.6.1 Einleitungsteil für Globale Statische Daten und Stackframe

```
1  struct ar_with_len {int len; int ar[2];};
2
3  void main() {
4    struct ar_with_len st_ar[3];
5    int *(*pntr2)[3];
6    pntr2;
7  }
8
9  void fun() {
10   struct ar_with_len st_ar[3];
11   int (*pntr1)[3];
12   pntr1;
13 }
```

Code 1.48: PicoC Code für den Einleitungsteil

```
1  File
2    Name './example_derived_dts_introduction_part.ast',
3    [
4      StructDecl
5        Name 'ar_with_len',
6        [
7          Alloc
8            Writeable,
9            IntType 'int',
10           Name 'len',
11         Alloc
12           Writeable,
13           ArrayDecl
14             [
15               Num '2'
16             ],
17             IntType 'int',
18           Name 'ar'
19       ],
20     FunDef
21       VoidType 'void',
22       Name 'main',
23       [],
24       [
25         Exp
26           Alloc
27             Writeable,
28             ArrayDecl
29               [
30                 Num '3'
31               ],
```

```
32              StructSpec
33                Name 'ar_with_len',
34            Name 'st_ar',
35        Exp
36          Alloc
37            Writeable,
38            PntrDecl
39              Num '1',
40              ArrayDecl
41                [
42                  Num '3'
43                ],
44                PntrDecl
45                  Num '1',
46                  IntType 'int',
47            Name 'pntr2',
48        Exp
49          Name 'pntr2'
50      ],
51    FunDef
52      VoidType 'void',
53      Name 'fun',
54      [],
55      [
56        Exp
57          Alloc
58            Writeable,
59            ArrayDecl
60              [
61                Num '3'
62              ],
63              StructSpec
64                Name 'ar_with_len',
65            Name 'st_ar',
66        Exp
67          Alloc
68            Writeable,
69            PntrDecl
70              Num '1',
71              ArrayDecl
72                [
73                  Num '3'
74                ],
75                IntType 'int',
76            Name 'pntr1',
77        Exp
78          Name 'pntr1'
79      ]
80  ]
```

Code 1.49: Abstract Syntax Tree für den Einleitungsteil

```
1 File
2   Name './example_derived_dts_introduction_part.picoc_mon',
```

```
  [
    Block
      Name 'main.1',
      [
        Exp
          GlobalRead
            Num '9',
        Return
          Empty
      ],
    Block
      Name 'fun.0',
      [
        Exp
          StackRead
            Num '9',
        Return
          Empty
      ]
  ]
```

Code 1.50: PicoC Mon Pass für den Einleitungsteil

```
File
  Name './example_derived_dts_introduction_part.reti_blocks',
  [
    Block
      Name 'main.1',
      [
        # Exp(GlobalRead(Num('9'))),
        SUBI SP 1,
        LOADIN DS ACC 9,
        STOREIN SP ACC 1,
        # Return(Empty()),
        LOADIN BAF PC -1
      ],
    Block
      Name 'fun.0',
      [
        # Exp(StackRead(Num('9'))),
        SUBI SP 1,
        LOADIN BAF ACC -11,
        STOREIN SP ACC 1,
        # Return(Empty()),
        LOADIN BAF PC -1
      ]
  ]
```

Code 1.51: RETI Blocks Pass für den Einleitungsteil

1.3.6.2 Mittelteil für die verschiedenen Derived Datatypes

```
1 struct st1 {int (*ar)[1];};
2
3 void main() {
4   int var[1] = {42};
5   struct st1 st_first = {.ar=&var};
6   (*st_first.ar)[0];
7 }
```

Code 1.52: PicoC Code für den Mittelteil

```
1 File
2   Name './example_derived_dts_main_part.ast',
3   [
4     StructDecl
5       Name 'st1',
6       [
7         Alloc
8           Writeable,
9           PntrDecl
10            Num '1',
11            ArrayDecl
12              [
13                Num '1'
14              ],
15              IntType 'int',
16          Name 'ar'
17      ],
18    FunDef
19      VoidType 'void',
20      Name 'main',
21      [],
22      [
23        Assign
24          Alloc
25            Writeable,
26            ArrayDecl
27              [
28                Num '1'
29              ],
30              IntType 'int',
31            Name 'var',
32          Array
33            [
34              Num '42'
35            ],
36        Assign
37          Alloc
38            Writeable,
39            StructSpec
40              Name 'st1',
41            Name 'st_first',
42          Struct
```

```
43            [
44              Assign
45                Name 'ar',
46                Ref
47                  Name 'var'
48            ],
49          Exp
50            Subscr
51              Deref
52                Attr
53                  Name 'st_first',
54                  Name 'ar',
55                Num '0',
56              Num '0'
57      ]
58  ]
```

Code 1.53: Abstract Syntax Tree für den Mittelteil

```
1 File
2   Name './example_derived_dts_main_part.picoc_mon',
3   [
4     Block
5       Name 'main.0',
6       [
7         // Assign(Name('var'), Array([Num('42')])),
8         Exp
9           Num '42',
10        Assign
11          GlobalWrite
12            Num '0',
13          Tmp
14            Num '1',
15        // Assign(Name('st_first'), Struct([Assign(Name('ar'), Ref(Name('var')))])),
16        Ref
17          GlobalRead
18            Num '0',
19        Assign
20          GlobalWrite
21            Num '1',
22          Tmp
23            Num '1',
24        // Exp(Subscr(Subscr(Attr(Name('st_first'), Name('ar')), Num('0')), Num('0'))),
25        Ref
26          GlobalRead
27            Num '1',
28        Ref
29          Attr
30            Tmp
31              Num '1',
32            Name 'ar',
33        Exp
34          Num '0',
35        Ref
```

```
          Subscr
            Tmp
              Num '2',
            Tmp
              Num '1',
        Exp
          Num '0',
        Ref
          Subscr
            Tmp
              Num '2',
            Tmp
              Num '1',
        Exp
          Subscr
            Tmp
              Num '1',
            Num '0',
        Return
          Empty
      ]
  ]
```

Code 1.54: PicoC Mon Pass für den Mittelteil

```
File
  Name './example_derived_dts_main_part.reti_blocks',
  [
    Block
      Name 'main.0',
      [
        # // Assign(Name('var'), Array([Num('42')])),
        # Exp(Num('42')),
        SUBI SP 1,
        LOADI ACC 42,
        STOREIN SP ACC 1,
        # Assign(GlobalWrite(Num('0')), Tmp(Num('1'))),
        LOADIN SP ACC 1,
        STOREIN DS ACC 0,
        ADDI SP 1,
        # // Assign(Name('st_first'), Struct([Assign(Name('ar'), Ref(Name('var')))])),
        # Ref(GlobalRead(Num('0'))),
        SUBI SP 1,
        LOADI IN1 0,
        ADD IN1 DS,
        STOREIN SP IN1 1,
        # Assign(GlobalWrite(Num('1')), Tmp(Num('1'))),
        LOADIN SP ACC 1,
        STOREIN DS ACC 1,
        ADDI SP 1,
        # // Exp(Subscr(Subscr(Attr(Name('st_first'), Name('ar')), Num('0')), Num('0'))),
        # Ref(GlobalRead(Num('1'))),
        SUBI SP 1,
        LOADI IN1 1,
```

```
30          ADD IN1 DS,
31          STOREIN SP IN1 1,
32          # Ref(Attr(Tmp(Num('1')), Name('ar'))),
33          LOADIN SP IN1 1,
34          ADDI IN1 0,
35          STOREIN SP IN1 1,
36          # Exp(Num('0')),
37          SUBI SP 1,
38          LOADI ACC 0,
39          STOREIN SP ACC 1,
40          # Ref(Subscr(Tmp(Num('2')), Tmp(Num('1')))),
41          LOADIN SP IN2 2,
42          LOADIN IN2 IN1 0,
43          LOADIN SP IN2 1,
44          MULTI IN2 1,
45          ADD IN1 IN2,
46          ADDI SP 1,
47          STOREIN SP IN1 1,
48          # Exp(Num('0')),
49          SUBI SP 1,
50          LOADI ACC 0,
51          STOREIN SP ACC 1,
52          # Ref(Subscr(Tmp(Num('2')), Tmp(Num('1')))),
53          LOADIN SP IN1 2,
54          LOADIN SP IN2 1,
55          MULTI IN2 1,
56          ADD IN1 IN2,
57          ADDI SP 1,
58          STOREIN SP IN1 1,
59          # Exp(Subscr(Tmp(Num('1')), Num('0'))),
60          LOADIN SP IN1 1,
61          LOADIN IN1 ACC 0,
62          STOREIN SP ACC 1,
63          # Return(Empty()),
64          LOADIN BAF PC -1
65        ]
66    ]
```

Code 1.55: RETI Blocks Pass für den Mittelteil

1.3.6.3 Schlussteil für die verschiedenen Derived Datatypes

```
1  struct st {int attr[2];};
2
3  void main() {
4    int ar1[1][2] = {{42, 314}};
5    struct st ar2[1] = {.attr={42, 314}};
6    int var = 42;
7    int *pntr1 = &var;
8    int **pntr2 = &pntr1;
9
10   ar1[0];
11   ar2[0];
12   *pntr2;
13 }
```

Code 1.56: PicoC Code für den Schlussteil

```
File
  Name './example_derived_dts_final_part.ast',
  [
    StructDecl
      Name 'st',
      [
        Alloc
          Writeable,
          ArrayDecl
            [
              Num '2'
            ],
            IntType 'int',
          Name 'attr'
      ],
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Assign
          Alloc
            Writeable,
            ArrayDecl
              [
                Num '1',
                Num '2'
              ],
              IntType 'int',
            Name 'ar1',
          Array
            [
              Array
                [
                  Num '42',
                  Num '314'
                ]
            ],
        Assign
          Alloc
            Writeable,
            ArrayDecl
              [
                Num '1'
              ],
              StructSpec
                Name 'st',
            Name 'ar2',
          Struct
            [
              Assign
                Name 'attr',
```

```
                Array
                  [
                    Num '42',
                    Num '314'
                  ]
            ],
        Assign
          Alloc
            Writeable,
            IntType 'int',
            Name 'var',
          Num '42',
        Assign
          Alloc
            Writeable,
            PntrDecl
              Num '1',
              IntType 'int',
            Name 'pntr1',
          Ref
            Name 'var',
        Assign
          Alloc
            Writeable,
            PntrDecl
              Num '2',
              IntType 'int',
            Name 'pntr2',
          Ref
            Name 'pntr1',
        Exp
          Subscr
            Name 'ar1',
            Num '0',
        Exp
          Subscr
            Name 'ar2',
            Num '0',
        Exp
          Deref
            Name 'pntr2',
            Num '0'
      ]
  ]
```

Code 1.57: Abstract Syntax Tree für den Schlussteil

```
File
  Name './example_derived_dts_final_part.picoc_mon',
  [
    Block
      Name 'main.0',
      [
        // Assign(Name('ar1'), Array([Array([Num('42'), Num('314')])])),
```

```
 8         Exp
 9           Num '42',
10         Exp
11           Num '314',
12         Assign
13           GlobalWrite
14             Num '0',
15           Tmp
16             Num '2',
17         // Assign(Name('ar2'), Struct([Assign(Name('attr'), Array([Num('42'),
           ↪  Num('314')]))])),
18         Exp
19           Num '42',
20         Exp
21           Num '314',
22         Assign
23           GlobalWrite
24             Num '2',
25           Tmp
26             Num '2',
27         // Assign(Name('var'), Num('42')),
28         Exp
29           Num '42',
30         Assign
31           GlobalWrite
32             Num '4',
33           Tmp
34             Num '1',
35         // Assign(Name('pntr1'), Ref(Name('var'))),
36         Ref
37           GlobalRead
38             Num '4',
39         Assign
40           GlobalWrite
41             Num '5',
42           Tmp
43             Num '1',
44         // Assign(Name('pntr2'), Ref(Name('pntr1'))),
45         Ref
46           GlobalRead
47             Num '5',
48         Assign
49           GlobalWrite
50             Num '6',
51           Tmp
52             Num '1',
53         // Exp(Subscr(Name('ar1'), Num('0'))),
54         Ref
55           GlobalRead
56             Num '0',
57         Exp
58           Num '0',
59         Ref
60           Subscr
61             Tmp
62               Num '2',
63             Tmp
```

```
64              Num '1',
65        Exp
66          Subscr
67            Tmp
68              Num '1',
69            Num '0',
70        // Exp(Subscr(Name('ar2'), Num('0'))),
71        Ref
72          GlobalRead
73            Num '2',
74        Exp
75          Num '0',
76        Ref
77          Subscr
78            Tmp
79              Num '2',
80            Tmp
81              Num '1',
82        Exp
83          Subscr
84            Tmp
85              Num '1',
86            Num '0',
87        // Exp(Subscr(Name('pntr2'), Num('0'))),
88        Ref
89          GlobalRead
90            Num '6',
91        Exp
92          Num '0',
93        Ref
94          Subscr
95            Tmp
96              Num '2',
97            Tmp
98              Num '1',
99        Exp
100         Subscr
101           Tmp
102             Num '1',
103           Num '0',
104       Return
105         Empty
106     ]
107 ]
```

Code 1.58: PicoC Mon Pass für den Schlussteil

```
1 File
2   Name './example_derived_dts_final_part.reti_blocks',
3   [
4     Block
5       Name 'main.0',
6       [
7         # // Assign(Name('ar1'), Array([Array([Num('42'), Num('314')])])),
```

```
 8         # Exp(Num('42')),
 9         SUBI SP 1,
10         LOADI ACC 42,
11         STOREIN SP ACC 1,
12         # Exp(Num('314')),
13         SUBI SP 1,
14         LOADI ACC 314,
15         STOREIN SP ACC 1,
16         # Assign(GlobalWrite(Num('0')), Tmp(Num('2'))),
17         LOADIN SP ACC 1,
18         STOREIN DS ACC 1,
19         LOADIN SP ACC 2,
20         STOREIN DS ACC 0,
21         ADDI SP 2,
22         # // Assign(Name('ar2'), Struct([Assign(Name('attr'), Array([Num('42'),
           ↪  Num('314')]))])),
23         # Exp(Num('42')),
24         SUBI SP 1,
25         LOADI ACC 42,
26         STOREIN SP ACC 1,
27         # Exp(Num('314')),
28         SUBI SP 1,
29         LOADI ACC 314,
30         STOREIN SP ACC 1,
31         # Assign(GlobalWrite(Num('2')), Tmp(Num('2'))),
32         LOADIN SP ACC 1,
33         STOREIN DS ACC 3,
34         LOADIN SP ACC 2,
35         STOREIN DS ACC 2,
36         ADDI SP 2,
37         # // Assign(Name('var'), Num('42')),
38         # Exp(Num('42')),
39         SUBI SP 1,
40         LOADI ACC 42,
41         STOREIN SP ACC 1,
42         # Assign(GlobalWrite(Num('4')), Tmp(Num('1'))),
43         LOADIN SP ACC 1,
44         STOREIN DS ACC 4,
45         ADDI SP 1,
46         # // Assign(Name('pntr1'), Ref(Name('var'))),
47         # Ref(GlobalRead(Num('4'))),
48         SUBI SP 1,
49         LOADI IN1 4,
50         ADD IN1 DS,
51         STOREIN SP IN1 1,
52         # Assign(GlobalWrite(Num('5')), Tmp(Num('1'))),
53         LOADIN SP ACC 1,
54         STOREIN DS ACC 5,
55         ADDI SP 1,
56         # // Assign(Name('pntr2'), Ref(Name('pntr1'))),
57         # Ref(GlobalRead(Num('5'))),
58         SUBI SP 1,
59         LOADI IN1 5,
60         ADD IN1 DS,
61         STOREIN SP IN1 1,
62         # Assign(GlobalWrite(Num('6')), Tmp(Num('1'))),
63         LOADIN SP ACC 1,
```

```
64          STOREIN DS ACC 6,
65          ADDI SP 1,
66          # // Exp(Subscr(Name('ar1'), Num('0'))),
67          # Ref(GlobalRead(Num('0'))),
68          SUBI SP 1,
69          LOADI IN1 0,
70          ADD IN1 DS,
71          STOREIN SP IN1 1,
72          # Exp(Num('0')),
73          SUBI SP 1,
74          LOADI ACC 0,
75          STOREIN SP ACC 1,
76          # Ref(Subscr(Tmp(Num('2')), Tmp(Num('1')))),
77          LOADIN SP IN1 2,
78          LOADIN SP IN2 1,
79          MULTI IN2 2,
80          ADD IN1 IN2,
81          ADDI SP 1,
82          STOREIN SP IN1 1,
83          # Exp(Subscr(Tmp(Num('1')), Num('0'))),
84          # // Exp(Subscr(Name('ar2'), Num('0'))),
85          # Ref(GlobalRead(Num('2'))),
86          SUBI SP 1,
87          LOADI IN1 2,
88          ADD IN1 DS,
89          STOREIN SP IN1 1,
90          # Exp(Num('0')),
91          SUBI SP 1,
92          LOADI ACC 0,
93          STOREIN SP ACC 1,
94          # Ref(Subscr(Tmp(Num('2')), Tmp(Num('1')))),
95          LOADIN SP IN1 2,
96          LOADIN SP IN2 1,
97          MULTI IN2 2,
98          ADD IN1 IN2,
99          ADDI SP 1,
100         STOREIN SP IN1 1,
101         # Exp(Subscr(Tmp(Num('1')), Num('0'))),
102         LOADIN SP IN1 1,
103         LOADIN IN1 ACC 0,
104         STOREIN SP ACC 1,
105         # // Exp(Subscr(Name('pntr2'), Num('0'))),
106         # Ref(GlobalRead(Num('6'))),
107         SUBI SP 1,
108         LOADI IN1 6,
109         ADD IN1 DS,
110         STOREIN SP IN1 1,
111         # Exp(Num('0')),
112         SUBI SP 1,
113         LOADI ACC 0,
114         STOREIN SP ACC 1,
115         # Ref(Subscr(Tmp(Num('2')), Tmp(Num('1')))),
116         LOADIN SP IN2 2,
117         LOADIN IN2 IN1 0,
118         LOADIN SP IN2 1,
119         MULTI IN2 1,
120         ADD IN1 IN2,
```

```
121         ADDI SP 1,
122         STOREIN SP IN1 1,
123         # Exp(Subscr(Tmp(Num('1')), Num('0'))),
124         # Return(Empty()),
125         LOADIN BAF PC -1
126       ]
127   ]
```

Code 1.59: RETI Blocks Pass für den Schlussteil

1.3.7 Umsetzung von Funktionen

1.3.7.1 Funktionen auflösen zu RETI Code

```
1  void main() {
2    return;
3  }
4
5  void fun1() {
6  }
7
8  int fun2() {
9    return 1;
10 }
```

Code 1.60: PicoC Code für 3 Funktionen

```
1  File
2    Name './example_3_funs.ast',
3    [
4      FunDef
5        VoidType 'void',
6        Name 'main',
7        [],
8        [
9          Return
10           Empty
11       ],
12     FunDef
13       VoidType 'void',
14       Name 'fun1',
15       [],
16       [],
17     FunDef
18       IntType 'int',
19       Name 'fun2',
20       [],
21       [
22         Return
23           Num '1'
24       ]
```

```
25  ]
```

Code 1.61: Abstract Syntax Tree für 3 Funktionen

```
1 File
2   Name './example_3_funs.picoc_blocks',
3   [
4     FunDef
5       VoidType 'void',
6       Name 'main',
7       [],
8       [
9         Block
10          Name 'main.2',
11          [
12            Return
13              Empty
14          ]
15      ],
16    FunDef
17      VoidType 'void',
18      Name 'fun1',
19      [],
20      [
21        Block
22          Name 'fun1.1',
23          []
24      ],
25    FunDef
26      IntType 'int',
27      Name 'fun2',
28      [],
29      [
30        Block
31          Name 'fun2.0',
32          [
33            Return
34              Num '1'
35          ]
36      ]
37  ]
```

Code 1.62: PicoC Blocks Pass für 3 Funktionen

```
1 File
2   Name './example_3_funs.picoc_mon',
3   [
4     Block
5       Name 'main.2',
6       [
7         Return
```

```
8           Empty
9       ],
10    Block
11      Name 'fun1.1',
12      [
13        Return
14          Empty
15      ],
16    Block
17      Name 'fun2.0',
18      [
19        // Return(Num('1')),
20        Exp
21          Num '1',
22        Return
23          Tmp
24            Num '1'
25      ]
26  ]
```

Code 1.63: PicoC Mon Pass für 3 Funktionen

```
1 File
2   Name './example_3_funs.reti_blocks',
3   [
4     Block
5       Name 'main.2',
6       [
7         # Return(Empty()),
8         LOADIN BAF PC -1
9       ],
10    Block
11      Name 'fun1.1',
12      [
13        # Return(Empty()),
14        LOADIN BAF PC -1
15      ],
16    Block
17      Name 'fun2.0',
18      [
19        # // Return(Num('1')),
20        # Exp(Num('1')),
21        SUBI SP 1,
22        LOADI ACC 1,
23        STOREIN SP ACC 1,
24        # Return(Tmp(Num('1'))),
25        LOADIN SP ACC 1,
26        ADDI SP 1,
27        LOADIN BAF PC -1
28      ]
29  ]
```

Code 1.64: RETI Blocks Pass für 3 Funktionen

1.3.7.1.1 Sprung zur Main Funktion

```
void fun1() {
}

int fun2() {
  return 1;
}

void main() {
  return;
}
```

Code 1.65: PicoC Code für Funktionen, wobei die main Funktion nicht die erste Funktion ist

```
File
  Name './example_3_funs_main.picoc_mon',
  [
    Block
      Name 'fun1.2',
      [
        Return
          Empty
      ],
    Block
      Name 'fun2.1',
      [
        // Return(Num('1')),
        Exp
          Num '1',
        Return
          Tmp
            Num '1'
      ],
    Block
      Name 'main.0',
      [
        Return
          Empty
      ]
  ]
```

Code 1.66: PicoC Mon Pass für Funktionen, wobei die main Funktion nicht die erste Funktion ist

```
File
  Name './example_3_funs_main.reti_blocks',
  [
    Block
      Name 'fun1.2',
      [
```

```
7         # Return(Empty()),
8         LOADIN BAF PC -1
9       ],
10    Block
11      Name 'fun2.1',
12      [
13        # // Return(Num('1')),
14        # Exp(Num('1')),
15        SUBI SP 1,
16        LOADI ACC 1,
17        STOREIN SP ACC 1,
18        # Return(Tmp(Num('1'))),
19        LOADIN SP ACC 1,
20        ADDI SP 1,
21        LOADIN BAF PC -1
22      ],
23    Block
24      Name 'main.0',
25      [
26        # Return(Empty()),
27        LOADIN BAF PC -1
28      ]
29  ]
```

Code 1.67: PicoC Blocks Pass für Funktionen, wobei die main Funktion nicht die erste Funktion ist

```
1 File
2   Name './example_3_funs_main.reti_patch',
3   [
4     Block
5       Name 'start.3',
6       [
7         Exp
8           GoTo
9             Name 'main.0'
10      ],
11    Block
12      Name 'fun1.2',
13      [
14        # Return(Empty()),
15        LOADIN BAF PC -1
16      ],
17    Block
18      Name 'fun2.1',
19      [
20        # // Return(Num('1')),
21        # Exp(Num('1')),
22        SUBI SP 1,
23        LOADI ACC 1,
24        STOREIN SP ACC 1,
25        # Return(Tmp(Num('1'))),
26        LOADIN SP ACC 1,
27        ADDI SP 1,
28        LOADIN BAF PC -1
```

```
      ],
    Block
      Name 'main.0',
      [
        # Return(Empty()),
        LOADIN BAF PC -1
      ]
  ]
```

Code 1.68: PicoC Patch Pass für Funktionen, wobei die main Funktion nicht die erste Funktion ist

1.3.7.2 Funktionsdeklaration und -definition

```
int fun2(int var);

void fun1() {
}

void main() {
  int var = fun2(42);
  return;
}

int fun2(int var) {
  return var;
}
```

Code 1.69: PicoC Code für Funktionen, wobei eine Funktion vorher deklariert werden muss

```
SymbolTable
  [
    Symbol(
      {
        type qualifier:         Empty()
        datatype:               FunDecl(IntType('int'), Name('fun2'), [Alloc(Writeable(),
        ↪  IntType('int'), Name('var'))])
        name:                   Name('fun2')
        value or address:       Empty()
        position:               Pos(Num('1'), Num('4'))
        size:                   Empty()
      },
    Symbol(
      {
        type qualifier:         Empty()
        datatype:               FunDecl(VoidType('void'), Name('fun1'), [])
        name:                   Name('fun1')
        value or address:       Empty()
        position:               Pos(Num('3'), Num('5'))
        size:                   Empty()
      },
    Symbol(
      {
```

```
23        type qualifier:         Empty()
24        datatype:               FunDecl(VoidType('void'), Name('main'), [])
25        name:                   Name('main')
26        value or address:       Empty()
27        position:               Pos(Num('6'), Num('5'))
28        size:                   Empty()
29      },
30    Symbol(
31      {
32        type qualifier:         Writeable()
33        datatype:               IntType('int')
34        name:                   Name('var@main')
35        value or address:       Num('0')
36        position:               Pos(Num('7'), Num('6'))
37        size:                   Num('1')
38      },
39    Symbol(
40      {
41        type qualifier:         Writeable()
42        datatype:               IntType('int')
43        name:                   Name('var@fun2')
44        value or address:       Num('0')
45        position:               Pos(Num('11'), Num('13'))
46        size:                   Num('1')
47      }
48  ]
```

Code 1.70: Symboltabelle für Funktionen, wobei eine Funktion vorher deklariert werden muss

1.3.7.3 Funktionsaufruf

1.3.7.3.1 Ohne Rückgabewert

```
1  struct st {int attr1; int attr2[2];};
2
3  void stack_fun(struct st param[2][3]);
4
5  void main() {
6    struct st local_var[2][3];
7    stack_fun(local_var);
8    return;
9  }
10
11 void stack_fun(struct st param[2][3]) {
12   int local_var;
13 }
```

Code 1.71: PicoC Code für Funktionsaufruf ohne Rückgabewert

```
1  File
2    Name './example_fun_call_no_return_value.picoc_mon',
```

```
  [
    Block
      Name 'main.1',
      [
        StackMalloc
          Num '2',
        Ref
          GlobalRead
            Num '0',
        NewStackframe
          Name 'stack_fun',
          GoTo
            Name 'addr@next_instr',
        Exp
          GoTo
            Name 'stack_fun.0',
        RemoveStackframe,
        Return
          Empty
      ],
    Block
      Name 'stack_fun.0',
      [
        Return
          Empty
      ]
  ]
```

Code 1.72: PicoC Mon Pass für Funktionsaufruf ohne Rückgabewert

```
File
  Name './example_fun_call_no_return_value.reti_blocks',
  [
    Block
      Name 'main.1',
      [
        # StackMalloc(Num('2')),
        SUBI SP 2,
        # Ref(GlobalRead(Num('0'))),
        SUBI SP 1,
        LOADI IN1 0,
        ADD IN1 DS,
        STOREIN SP IN1 1,
        # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr'))),
        MOVE BAF ACC,
        ADDI SP 3,
        MOVE SP BAF,
        SUBI SP 4,
        STOREIN BAF ACC 0,
        LOADI ACC GoTo
                    Name 'addr@next_instr',
        ADD ACC CS,
        STOREIN BAF ACC -1,
        Exp
```

```
25          GoTo
26            Name 'stack_fun.0',
27        # RemoveStackframe(),
28        MOVE BAF IN1,
29        LOADIN IN1 BAF 0,
30        MOVE IN1 SP,
31        # Return(Empty()),
32        LOADIN BAF PC -1
33      ],
34    Block
35      Name 'stack_fun.0',
36      [
37        # Return(Empty()),
38        LOADIN BAF PC -1
39      ]
40  ]
```

Code 1.73: RETI Blocks Pass für Funktionsaufruf ohne Rückgabewert

```
1  # StackMalloc(Num('2'))
2  SUBI SP 2;
3  # Ref(GlobalRead(Num('0')))
4  SUBI SP 1;
5  LOADI IN1 0;
6  ADD IN1 DS;
7  STOREIN SP IN1 1;
8  # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
9  MOVE BAF ACC;
10 ADDI SP 3;
11 MOVE SP BAF;
12 SUBI SP 4;
13 STOREIN BAF ACC 0;
14 LOADI ACC 14;
15 ADD ACC CS;
16 STOREIN BAF ACC -1;
17 JUMP 5;
18 # RemoveStackframe()
19 MOVE BAF IN1;
20 LOADIN IN1 BAF 0;
21 MOVE IN1 SP;
22 # Return(Empty())
23 LOADIN BAF PC -1;
24 # Return(Empty())
25 LOADIN BAF PC -1;
```

Code 1.74: RETI Pass für Funktionsaufruf ohne Rückgabewert

1.3.7.3.2 Mit Rückgabewert

```
1 void stack_fun() {
2   return 42;
```

```
}

void main() {
  int var = stack_fun();
}
```

Code 1.75: PicoC Code für Funktionsaufruf mit Rückgabewert

```
File
  Name './example_fun_call_with_return_value.picoc_mon',
  [
    Block
      Name 'stack_fun.1',
      [
        // Return(Num('42')),
        Exp
          Num '42',
        Return
          Tmp
            Num '1'
      ],
    Block
      Name 'main.0',
      [
        // Assign(Name('var'), Call(Name('stack_fun'), [])),
        StackMalloc
          Num '2',
        NewStackframe
          Name 'stack_fun',
          GoTo
            Name 'addr@next_instr',
        Exp
          GoTo
            Name 'stack_fun.1',
        RemoveStackframe,
        Assign
          GlobalWrite
            Num '0',
          Tmp
            Num '1',
        Return
          Empty
      ]
  ]
```

Code 1.76: PicoC Mon Pass für Funktionsaufruf mit Rückgabewert

```
File
  Name './example_fun_call_with_return_value.reti_blocks',
  [
    Block
```

```
5       Name 'stack_fun.1',
6       [
7         # // Return(Num('42')),
8         # Exp(Num('42')),
9         SUBI SP 1,
10        LOADI ACC 42,
11        STOREIN SP ACC 1,
12        # Return(Tmp(Num('1'))),
13        LOADIN SP ACC 1,
14        ADDI SP 1,
15        LOADIN BAF PC -1
16      ],
17    Block
18      Name 'main.0',
19      [
20        # // Assign(Name('var'), Call(Name('stack_fun'), [])),
21        # StackMalloc(Num('2')),
22        SUBI SP 2,
23        # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr'))),
24        MOVE BAF ACC,
25        ADDI SP 2,
26        MOVE SP BAF,
27        SUBI SP 2,
28        STOREIN BAF ACC 0,
29        LOADI ACC GoTo
30                    Name 'addr@next_instr',
31        ADD ACC CS,
32        STOREIN BAF ACC -1,
33        Exp
34          GoTo
35            Name 'stack_fun.1',
36        # RemoveStackframe(),
37        MOVE BAF IN1,
38        LOADIN IN1 BAF 0,
39        MOVE IN1 SP,
40        # Assign(GlobalWrite(Num('0')), Tmp(Num('1'))),
41        LOADIN SP ACC 1,
42        STOREIN DS ACC 0,
43        ADDI SP 1,
44        # Return(Empty()),
45        LOADIN BAF PC -1
46      ]
47  ]
```

Code 1.77: RETI Blocks Pass für Funktionsaufruf mit Rückgabewert

```
1 JUMP 7;
2 # // Return(Num('42'))
3 # Exp(Num('42'))
4 SUBI SP 1;
5 LOADI ACC 42;
6 STOREIN SP ACC 1;
7 # Return(Tmp(Num('1')))
8 LOADIN SP ACC 1;
```

```
ADDI SP 1;
LOADIN BAF PC -1;
# // Assign(Name('var'), Call(Name('stack_fun'), []))
# StackMalloc(Num('2'))
SUBI SP 2;
# NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
MOVE BAF ACC;
ADDI SP 2;
MOVE SP BAF;
SUBI SP 2;
STOREIN BAF ACC 0;
LOADI ACC 17;
ADD ACC CS;
STOREIN BAF ACC -1;
JUMP -15;
# RemoveStackframe()
MOVE BAF IN1;
LOADIN IN1 BAF 0;
MOVE IN1 SP;
# Assign(GlobalWrite(Num('0')), Tmp(Num('1')))
LOADIN SP ACC 1;
STOREIN DS ACC 0;
ADDI SP 1;
# Return(Empty())
LOADIN BAF PC -1;
```

Code 1.78: RETI Pass für Funktionsaufruf mit Rückgabewert

1.3.7.3.3 Umsetzung von Call by Sharing für Arrays

```
void stack_fun(int (*param1)[3], int param2[2][3]) {
}

void main() {
  int local_var1[2][3];
  int local_var2[2][3];
  stack_fun(local_var1, local_var2);
}
```

Code 1.79: PicoC Code für Call by Sharing für Arrays

```
File
  Name './example_fun_call_by_sharing_array.picoc_mon',
  [
    Block
      Name 'stack_fun.1',
      [
        Return
          Empty
      ],
    Block
```

```
    Name 'main.0',
    [
      StackMalloc
        Num '2',
      Ref
        GlobalRead
          Num '0',
      Ref
        GlobalRead
          Num '6',
      NewStackframe
        Name 'stack_fun',
        GoTo
          Name 'addr@next_instr',
      Exp
        GoTo
          Name 'stack_fun.1',
      RemoveStackframe,
      Return
        Empty
    ]
]
```

Code 1.80: PicoC Mon Pass für Call by Sharing für Arrays

```
SymbolTable
  [
    Symbol(
      {
        type qualifier:         Empty()
        datatype:               FunDecl(VoidType('void'), Name('stack_fun'),
        ↪  [Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int'))),
        ↪  Name('param1')), Alloc(Writeable(), ArrayDecl([Num('3')], IntType('int')),
        ↪  Name('param2'))])
        name:                   Name('stack_fun')
        value or address:       Empty()
        position:               Pos(Num('1'), Num('5'))
        size:                   Empty()
      },
    Symbol(
      {
        type qualifier:         Writeable()
        datatype:               PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int')))
        name:                   Name('param1@stack_fun')
        value or address:       Num('0')
        position:               Pos(Num('1'), Num('21'))
        size:                   Num('1')
      },
    Symbol(
      {
        type qualifier:         Writeable()
        datatype:               PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int')))
        name:                   Name('param2@stack_fun')
        value or address:       Num('1')
```

```
      position:               Pos(Num('1'), Num('37'))
      size:                   Num('1')
    },
  Symbol(
    {
      type qualifier:         Empty()
      datatype:               FunDecl(VoidType('void'), Name('main'), [])
      name:                   Name('main')
      value or address:       Empty()
      position:               Pos(Num('4'), Num('5'))
      size:                   Empty()
    },
  Symbol(
    {
      type qualifier:         Writeable()
      datatype:               ArrayDecl([Num('2'), Num('3')], IntType('int'))
      name:                   Name('local_var1@main')
      value or address:       Num('0')
      position:               Pos(Num('5'), Num('6'))
      size:                   Num('6')
    },
  Symbol(
    {
      type qualifier:         Writeable()
      datatype:               ArrayDecl([Num('2'), Num('3')], IntType('int'))
      name:                   Name('local_var2@main')
      value or address:       Num('6')
      position:               Pos(Num('6'), Num('6'))
      size:                   Num('6')
    }
]
```

Code 1.81: Symboltabelle für Call by Sharing für Arrays

```
File
  Name './example_fun_call_by_sharing_array.reti_blocks',
  [
    Block
      Name 'stack_fun.1',
      [
        # Return(Empty()),
        LOADIN BAF PC -1
      ],
    Block
      Name 'main.0',
      [
        # StackMalloc(Num('2')),
        SUBI SP 2,
        # Ref(GlobalRead(Num('0'))),
        SUBI SP 1,
        LOADI IN1 0,
        ADD IN1 DS,
        STOREIN SP IN1 1,
        # Ref(GlobalRead(Num('6'))),
```

```
21        SUBI SP 1,
22        LOADI IN1 6,
23        ADD IN1 DS,
24        STOREIN SP IN1 1,
25        # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr'))),
26        MOVE BAF ACC,
27        ADDI SP 4,
28        MOVE SP BAF,
29        SUBI SP 4,
30        STOREIN BAF ACC 0,
31        LOADI ACC GoTo
32                    Name 'addr@next_instr',
33        ADD ACC CS,
34        STOREIN BAF ACC -1,
35        Exp
36          GoTo
37            Name 'stack_fun.1',
38        # RemoveStackframe(),
39        MOVE BAF IN1,
40        LOADIN IN1 BAF 0,
41        MOVE IN1 SP,
42        # Return(Empty()),
43        LOADIN BAF PC -1
44      ]
45  ]
```

Code 1.82: RETI Block Pass für Call by Sharing für Arrays

1.3.7.3.4 Umsetzung von Call by Value für Structs

```
1 struct st {int attr1; int attr2[2];};
2
3 void stack_fun(struct st param) {
4 }
5
6 void main() {
7   struct st local_var;
8   stack_fun(local_var);
9 }
```

Code 1.83: PicoC Code für Call by Value für Structs

```
1 File
2   Name './example_fun_call_by_value_struct.picoc_mon',
3   [
4     Block
5       Name 'stack_fun.1',
6       [
7         Return
8           Empty
9       ],
```

```
    Block
      Name 'main.0',
      [
        StackMalloc
          Num '2',
        Assign
          Tmp
            Num '3',
          GlobalRead
            Num '0',
        NewStackframe
          Name 'stack_fun',
          GoTo
            Name 'addr@next_instr',
        Exp
          GoTo
            Name 'stack_fun.1',
        RemoveStackframe,
        Return
          Empty
      ]
  ]
```

Code 1.84: PicoC Mon Pass für Call by Value für Structs

```
File
  Name './example_fun_call_by_value_struct.reti_blocks',
  [
    Block
      Name 'stack_fun.1',
      [
        # Return(Empty()),
        LOADIN BAF PC -1
      ],
    Block
      Name 'main.0',
      [
        # StackMalloc(Num('2')),
        SUBI SP 2,
        # Assign(Tmp(Num('3')), GlobalRead(Num('0'))),
        SUBI SP 3,
        LOADIN DS ACC 0,
        STOREIN SP ACC 1,
        LOADIN DS ACC 1,
        STOREIN SP ACC 2,
        LOADIN DS ACC 2,
        STOREIN SP ACC 3,
        # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr'))),
        MOVE BAF ACC,
        ADDI SP 5,
        MOVE SP BAF,
        SUBI SP 5,
        STOREIN BAF ACC 0,
        LOADI ACC GoTo
```

```
30                    Name 'addr@next_instr',
31          ADD ACC CS,
32          STOREIN BAF ACC -1,
33          Exp
34            GoTo
35              Name 'stack_fun.1',
36          # RemoveStackframe(),
37          MOVE BAF IN1,
38          LOADIN IN1 BAF 0,
39          MOVE IN1 SP,
40          # Return(Empty()),
41          LOADIN BAF PC -1
42        ]
43    ]
```

Code 1.85: RETI Block Pass für Call by Value für Structs

1.3.8 Umsetzung kleinerer Details

1.4 Fehlermeldungen

1.4.1 Error Handler

1.4.2 Arten von Fehlermeldungen

1.4.2.1 Syntaxfehler

1.4.2.2 Laufzeitfehler

Literatur

Online

- *C Operator Precedence - cppreference.com*. URL: https://en.cppreference.com/w/c/language/operator_precedence (besucht am 27. 04. 2022).