
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

Betreuung:
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	Konkrete Syntax für Lexer erstellen	11
1.1.2	Basic Lexer	12
1.2	Syntaktische Analyse	12
1.2.1	Konkrete 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	Kompositionen von PicoC-Nodes und RETI-Nodes mit besonderer Bedeutung	16
1.2.5.4	Abstrakte Syntax	18
1.2.5.5	Transformer	20
1.2.5.6	Codebeispiel	20
1.3	Code Generierung	21
1.3.1	Übersicht	21
1.3.2	Passes	22
1.3.2.1	PicoC-Shrink Pass	22
1.3.2.1.1	Codebeispiel	22
1.3.2.2	PicoC-Blocks Pass	24
1.3.2.2.1	Abstrakte Syntax	24
1.3.2.2.2	Codebeispiel	24
1.3.2.3	PicoC-Mon Pass	26
1.3.2.3.1	Abstrakte Syntax	26
1.3.2.3.2	Codebeispiel	27
1.3.2.4	RETI-Blocks Pass	29
1.3.2.4.1	Abstrakte Syntax	29
1.3.2.4.2	Codebeispiel	29
1.3.2.5	RETI-Patch Pass	32
1.3.2.5.1	Abstrakte Syntax	32
1.3.2.5.2	Codebeispiel	32
1.3.2.6	RETI Pass	34
1.3.2.6.1	Konkrete und Abstrakte Syntax	34
1.3.2.6.2	Codebeispiel	35
1.3.3	Umsetzung von Pointern	38
1.3.3.1	Referenzierung	38
1.3.3.2	Pointer Dereferenzierung durch Zugriff auf Arrayindex ersetzen	40
1.3.4	Umsetzung von Arrays	41
1.3.4.1	Initialisierung von Arrays	41
1.3.4.2	Zugriff auf Arrayindex	43

1.3.4.3	Zuweisung an Arrayindex	46
1.3.5	Umsetzung von Structs	48
1.3.5.1	Deklaration von Structs	48
1.3.5.2	Initialisierung von Structs	49
1.3.5.3	Zugriff auf Structattribut	53
1.3.5.4	Zuweisung an Structattribut	56
1.3.6	Umsetzung der Derived Datatypes im Zusammenspiel	59
1.3.6.1	Einleitungsteil für Globale Statische Daten und Stackframe	59
1.3.6.2	Mittelteil für die verschiedenen Derived Datatypes	62
1.3.6.3	Schlusssteil für die verschiedenen Derived Datatypes	65
1.3.7	Umsetzung von Funktionen	72
1.3.7.1	Funktionen auflösen zu RETI Code	72
1.3.7.1.1	Sprung zur Main Funktion	75
1.3.7.2	Funktionsdeklaration und -definition	77
1.3.7.3	Funktionsaufruf	78
1.3.7.3.1	Ohne Rückgabewert	78
1.3.7.3.2	Mit Rückgabewert	80
1.3.7.3.3	Umsetzung von Call by Sharing für Arrays	83
1.3.7.3.4	Umsetzung von Call by Value für Structs	86
1.3.8	Umsetzung kleinerer Details	88
1.4	Fehlermeldungen	88
1.4.1	Error Handler	88
1.4.2	Arten von Fehlermeldungen	88
1.4.2.1	Syntaxfehler	88
1.4.2.2	Laufzeitfehler	88

Abbildungsverzeichnis

1.1	Cross-Compiler Kompiliervorgang ausgeschrieben	21
1.2	Cross-Compiler Kompiliervorgang Kurzform	21
1.3	Architektur mit allen Passes ausgeschrieben	22

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 Derivation Tree generiert	20
1.5	PicoC Code für Codebeispiel	22
1.6	Abstract Syntax Tree für Codebeispiel	23
1.7	PicoC Shrink Pass für Codebeispiel	24
1.8	PicoC-Blocks Pass für Codebeispiel	26
1.9	PicoC-Mon Pass für Codebeispiel	29
1.10	RETI-Blocks Pass für Codebeispiel	32
1.11	RETI-Patch Pass für Codebeispiel	34
1.12	RETI Pass für Codebeispiel	37
1.13	PicoC Code für Pointer Referenzierung	38
1.14	Abstract Syntax Tree für Pointer Referenzierung	38
1.15	PicoC Mon Pass für Pointer Referenzierung	39
1.16	RETI Blocks Pass für Pointer Referenzierung	40
1.17	PicoC Code für Pointer Dereferenzierung	40
1.18	Abstract Syntax Tree für Pointer Dereferenzierung	40
1.19	PicoC Shrink Pass für Pointer Dereferenzierung	41
1.20	PicoC Code für Array Initialisierung	41
1.21	Abstract Syntax Tree für Array Initialisierung	42
1.22	Symboltabelle für Array Initialisierung	42
1.23	PicoC Mon Pass für Array Initialisierung	43
1.24	RETI Blocks Pass für Array Initialisierung	43
1.25	PicoC Code für Zugriff auf Arrayindex	44
1.26	Abstract Syntax Tree für Zugriff auf Arrayindex	44
1.27	PicoC Mon Pass für Zugriff auf Arrayindex	45
1.28	RETI Blocks Pass für Zugriff auf Arrayindex	46
1.29	PicoC Code für Zuweisung an Arrayindex	46
1.30	Abstract Syntax Tree für Zuweisung an Arrayindex	47
1.31	PicoC Mon Pass für Zuweisung an Arrayindex	47
1.32	RETI Blocks Pass für Zuweisung an Arrayindex	48
1.33	PicoC Code für Deklaration von Structs	48
1.34	Symboltabelle für Deklaration von Structs	49
1.35	PicoC Code für Initialisierung von Structs	50
1.36	Abstract Syntax Tree für Initialisierung von Structs	51
1.37	Symboltabelle für Initialisierung von Structs	52
1.38	PicoC Mon Pass für Initialisierung von Structs	53
1.39	RETI Blocks Pass für Initialisierung von Structs	53
1.40	PicoC Code für Zugriff auf Structattribut	53
1.41	Abstract Syntax Tree für Zugriff auf Structattribut	54
1.42	PicoC Mon Pass für Zugriff auf Structattribut	55
1.43	RETI Blocks Pass für Zugriff auf Structattribut	56
1.44	PicoC Code für Zuweisung an Structattribut	56
1.45	Abstract Syntax Tree für Zuweisung an Structattribut	57
1.46	PicoC Mon Pass für Zuweisung an Structattribut	58
1.47	RETI Blocks Pass für Zuweisung an Structattribut	58

1.48 PicoC Code für den Einleitungsteil	59
1.49 Abstract Syntax Tree für den Einleitungsteil	60
1.50 PicoC Mon Pass für den Einleitungsteil	61
1.51 RETI Blocks Pass für den Einleitungsteil	61
1.52 PicoC Code für den Mittelteil	62
1.53 Abstract Syntax Tree für den Mittelteil	63
1.54 PicoC Mon Pass für den Mittelteil	64
1.55 RETI Blocks Pass für den Mittelteil	65
1.56 PicoC Code für den Schlussteil	66
1.57 Abstract Syntax Tree für den Schlussteil	67
1.58 PicoC Mon Pass für den Schlussteil	69
1.59 RETI Blocks Pass für den Schlussteil	72
1.60 PicoC Code für 3 Funktionen	72
1.61 Abstract Syntax Tree für 3 Funktionen	73
1.62 PicoC Blocks Pass für 3 Funktionen	73
1.63 PicoC Mon Pass für 3 Funktionen	74
1.64 RETI Blocks Pass für 3 Funktionen	74
1.65 PicoC Code für Funktionen, wobei die main Funktion nicht die erste Funktion ist	75
1.66 PicoC Mon Pass für Funktionen, wobei die main Funktion nicht die erste Funktion ist	75
1.67 PicoC Blocks Pass für Funktionen, wobei die main Funktion nicht die erste Funktion ist	76
1.68 PicoC Patch Pass für Funktionen, wobei die main Funktion nicht die erste Funktion ist	77
1.69 PicoC Code für Funktionen, wobei eine Funktion vorher deklariert werden muss	77
1.70 Symboltabelle für Funktionen, wobei eine Funktion vorher deklariert werden muss	78
1.71 PicoC Code für Funktionsaufruf ohne Rückgabewert	78
1.72 PicoC Mon Pass für Funktionsaufruf ohne Rückgabewert	79
1.73 RETI Blocks Pass für Funktionsaufruf ohne Rückgabewert	80
1.74 RETI Pass für Funktionsaufruf ohne Rückgabewert	80
1.75 PicoC Code für Funktionsaufruf mit Rückgabewert	81
1.76 PicoC Mon Pass für Funktionsaufruf mit Rückgabewert	81
1.77 RETI Blocks Pass für Funktionsaufruf mit Rückgabewert	82
1.78 RETI Pass für Funktionsaufruf mit Rückgabewert	83
1.79 PicoC Code für Call by Sharing für Arrays	83
1.80 PicoC Mon Pass für Call by Sharing für Arrays	84
1.81 Symboltabelle für Call by Sharing für Arrays	85
1.82 RETI Block Pass für Call by Sharing für Arrays	86
1.83 PicoC Code für Call by Value für Structs	86
1.84 PicoC Mon Pass für Call by Value für Structs	87
1.85 RETI Block Pass für Call by Value für Structs	88

Tabellenverzeichnis

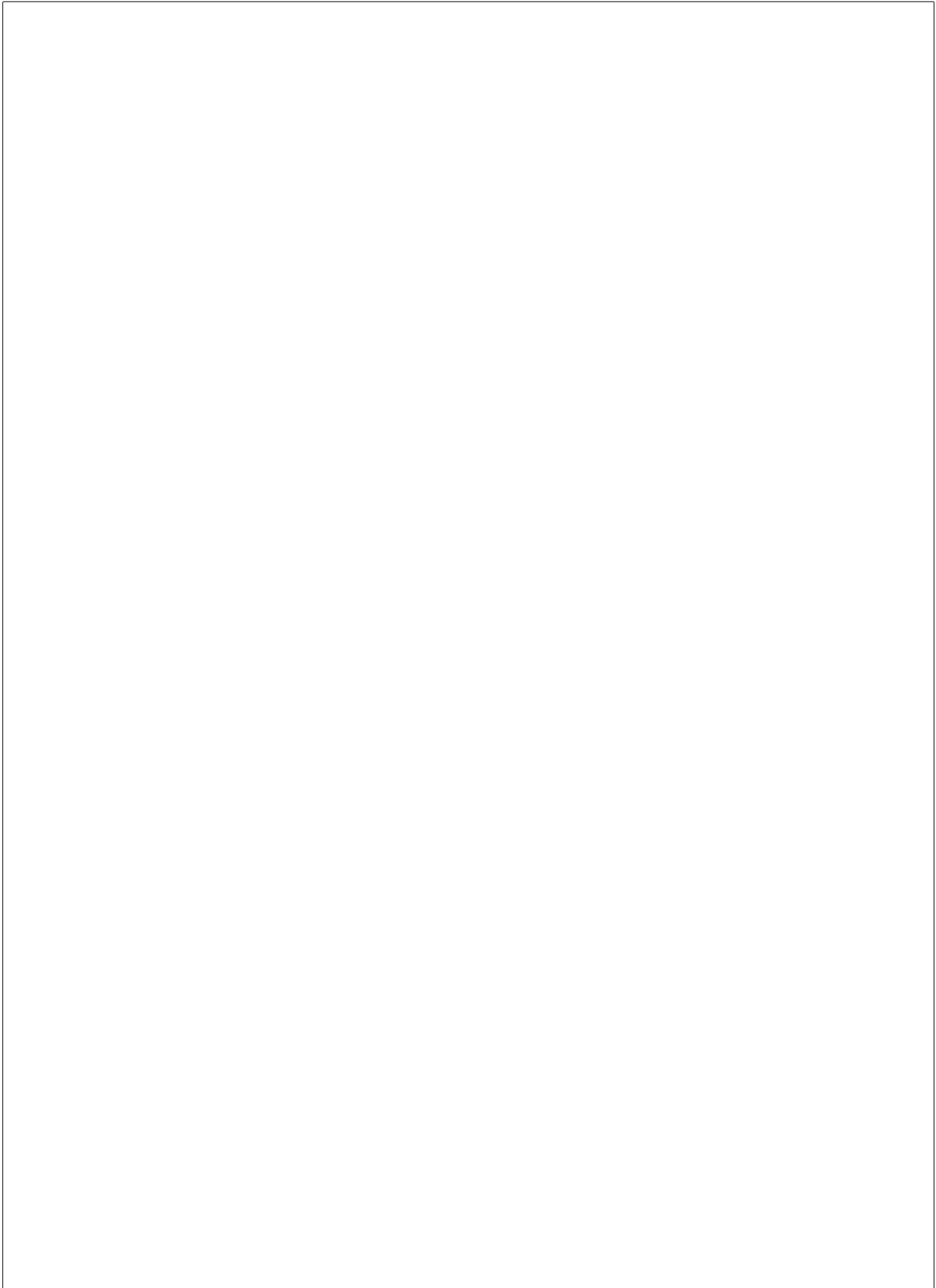
1.1	Präzidenzregeln von PicoC	13
1.2	Kompositionen von PicoC-Nodes und RETI-Nodes mit besonderer Bedeutung	17

Definitionsverzeichnis

1.1	Symboltabelle	27
-----	-------------------------	----

Grammatikverzeichnis

1.1.1 Konkrete Syntax des Lexers in EBNF	11
1.2.1 Konkrete Syntax des Parsers in EBNF, Teil 1	12
1.2.2 Konkrete Syntax des Parsers in EBNF, Teil 2	13
1.2.3 Abstrakte Syntax für L_{PioC}	19
1.3.1 Abstrakte Syntax für L_{PicoC_Blocks}	24
1.3.2 Abstrakte Syntax für L_{PicoC_Mon}	26
1.3.3 Abstrakte Syntax für L_{RETI_Blocks}	29
1.3.4 Abstrakte Syntax für L_{RETI_Patch}	32
1.3.5 Konkrete Syntax für L_{RETI_Lex}	34
1.3.6 Konkrete Syntax für L_{RETI_Parse}	35
1.3.7 Abstrakte Syntax für L_{RETI}	35



1 Implementierung

1.1 Lexikalische Analyse

1.1.1 Konkrete Syntax für Lexer erstellen

<i>COMMENT</i>	::=	"//"/[\backslash n]*/ "/"*//(. \backslash n)*?/"*/	<i>L_Comment</i>
<i>RET_COMMENT.2</i>	::=	"//""?"#[\backslash n]*/	
<i>DIG_NO_0</i>	::=	"1" "2" "3" "4" "5" "6" "7" "8" "9"	<i>L_Arith</i>
<i>DIG_WITH_0</i>	::=	"0" <i>DIG_NO_0</i>	
<i>NUM</i>	::=	"0" <i>DIG_NO_0</i> <i>DIG_WITH_0</i> *	
<i>ASCII_CHAR</i>	::=	"_".~"	
<i>CHAR</i>	::=	"'" <i>ASCII_CHAR</i> "'"	
<i>FILENAME</i>	::=	<i>ASCII_CHAR</i> + ".picoc"	
<i>LETTER</i>	::=	"a"..z" "A"..Z"	
<i>NAME</i>	::=	(<i>LETTER</i> "_") (<i>LETTER</i> — <i>DIG_WITH_0</i> — "_")*	
<i>name</i>	::=	<i>NAME</i> <i>INT_NAME</i> <i>CHAR_NAME</i> <i>VOID_NAME</i>	
<i>NOT</i>	::=	"~"	
<i>REF_AND</i>	::=	"&"	
<i>un_op</i>	::=	<i>SUB_MINUS</i> <i>LOGIC_NOT</i> <i>NOT</i> <i>MUL_DEREF_PNTR</i> <i>REF_AND</i>	
<i>MUL_DEREF_PNTR</i>	::=	"*"	
<i>DIV</i>	::=	"/"	
<i>MOD</i>	::=	"%"	
<i>prec1_op</i>	::=	<i>MUL_DEREF_PNTR</i> <i>DIV</i> <i>MOD</i>	
<i>ADD</i>	::=	"+"	
<i>SUB_MINUS</i>	::=	"-"	
<i>prec2_op</i>	::=	<i>ADD</i> <i>SUB_MINUS</i>	
<i>LT</i>	::=	"<"	<i>L_Logic</i>
<i>LTE</i>	::=	"<="	
<i>GT</i>	::=	">"	
<i>GTE</i>	::=	">="	
<i>rel_op</i>	::=	<i>LT</i> <i>LTE</i> <i>GT</i> <i>GTE</i>	
<i>EQ</i>	::=	"=="	
<i>NEQ</i>	::=	"!="	
<i>eq_op</i>	::=	<i>EQ</i> <i>NEQ</i>	
<i>LOGIC_NOT</i>	::=	"!"	
<i>INT_DT.2</i>	::=	"int"	
<i>INT_NAME.3</i>	::=	"int" (<i>LETTER</i> <i>DIG_WITH_0</i> "_")+	<i>L_Assign_Alloc</i>
<i>CHAR_DT.2</i>	::=	"char"	
<i>CHAR_NAME.3</i>	::=	"char" (<i>LETTER</i> <i>DIG_WITH_0</i> "_")+	
<i>VOID_DT.2</i>	::=	"void"	
<i>VOID_NAME.3</i>	::=	"void" (<i>LETTER</i> <i>DIG_WITH_0</i> "_")+	
<i>prim_dt</i>	::=	<i>INT_DT</i> <i>CHAR_DT</i> <i>VOID_DT</i>	

1.1.2 Basic Lexer

1.2 Syntaktische Analyse

1.2.1 Konkrete Syntax für Parser erstellen

In 1.2.1

<i>prim_exp</i>	::=	<i>name</i> <i>NUM</i> <i>CHAR</i> "(" <i>logic_or</i> ")"	<i>L_Arith</i> +
<i>post_exp</i>	::=	<i>array_subscr</i> <i>struct_attr</i> <i>fun_call</i> <i>input_exp</i> <i>print_exp</i> <i>prim_exp</i>	<i>L_Array</i> + <i>L_Pntr</i> +
<i>un_exp</i>	::=	<i>un_opun_exp</i> <i>post_exp</i>	<i>L_Struct</i> + <i>L_Fun</i>
<i>input_exp</i>	::=	"input" "(" "	<i>L_Arith</i>
<i>print_exp</i>	::=	"print" "(" <i>logic_or</i> ")"	
<i>arith_prec1</i>	::=	<i>arith_prec1</i> <i>prec1_op</i> <i>un_exp</i> <i>un_exp</i>	
<i>arith_prec2</i>	::=	<i>arith_prec2</i> <i>prec2_op</i> <i>arith_prec1</i> <i>arith_prec1</i>	
<i>arith_and</i>	::=	<i>arith_and</i> "&" <i>arith_prec2</i> <i>arith_prec2</i>	
<i>arith_oplus</i>	::=	<i>arith_oplus</i> "^" <i>arith_and</i> <i>arith_and</i>	
<i>arith_or</i>	::=	<i>arith_or</i> " " <i>arith_oplus</i> <i>arith_oplus</i>	
<i>rel_exp</i>	::=	<i>rel_exp</i> <i>rel_op</i> <i>arith_or</i> <i>arith_or</i>	<i>L_Logic</i>
<i>eq_exp</i>	::=	<i>eq_exp</i> <i>eq_oprel_exp</i> <i>rel_exp</i>	
<i>logic_and</i>	::=	<i>logic_and</i> "&&" <i>eq_exp</i> <i>eq_exp</i>	
<i>logic_or</i>	::=	<i>logic_or</i> " " <i>logic_and</i> <i>logic_and</i>	
<i>type_spec</i>	::=	<i>prim_dt</i> <i>struct_spec</i>	<i>L_Assign_Alloc</i>
<i>alloc</i>	::=	<i>type_spec</i> <i>pntr_decl</i>	
<i>assign_stmt</i>	::=	<i>un_exp</i> "=" <i>logic_or</i> ";"	
<i>initializer</i>	::=	<i>logic_or</i> <i>array_init</i> <i>struct_init</i>	
<i>init_stmt</i>	::=	<i>alloc</i> "=" <i>initializer</i> ";"	
<i>const_init_stmt</i>	::=	"const" <i>type_spec</i> <i>name</i> "=" <i>NUM</i> ";"	
<i>pntr_deg</i>	::=	"*" *	<i>L_Pntr</i>
<i>pntr_decl</i>	::=	<i>pntr_deg</i> <i>array_decl</i> <i>array_decl</i>	
<i>array_dims</i>	::=	("[" <i>NUM</i> "]") *	<i>L_Array</i>
<i>array_decl</i>	::=	<i>name</i> <i>array_dims</i> "(" <i>pntr_decl</i> ")" <i>array_dims</i>	
<i>array_init</i>	::=	"{" <i>initializer</i> ("," <i>initializer</i>) * "}"	
<i>array_subscr</i>	::=	<i>post_exp</i> "[" <i>logic_or</i> "]"	
<i>struct_spec</i>	::=	"struct" <i>name</i>	<i>L_Struct</i>
<i>struct_params</i>	::=	(<i>alloc</i> ";") +	
<i>struct_decl</i>	::=	"struct" <i>name</i> "{" <i>struct_params</i> "}"	
<i>struct_init</i>	::=	"{" " " <i>name</i> "=" <i>initializer</i> ("," " " <i>name</i> "=" <i>initializer</i>) * "}"	
<i>struct_attr</i>	::=	<i>post_exp</i> "." <i>name</i>	
<i>if_stmt</i>	::=	"if" "(" <i>logic_or</i> ")" <i>exec_part</i>	<i>L_If_Else</i>
<i>if_else_stmt</i>	::=	"if" "(" <i>logic_or</i> ")" <i>exec_part</i> "else" <i>exec_part</i>	
<i>while_stmt</i>	::=	"while" "(" <i>logic_or</i> ")" <i>exec_part</i>	<i>L_Loop</i>
<i>do_while_stmt</i>	::=	"do" <i>exec_part</i> "while" "(" <i>logic_or</i> ")" ";"	

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

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

Grammar 1.2.2: Konkrete 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 1.1 aufgelistet.

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

Tabelle 1.1: Präzidenzregeln von PicoC

¹C 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     ptr_decl
10    ptr_deg *
11    array_decl
12    array_dims
13    5
14    6
15    ptr_decl
16    ptr_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    ptr_deg
27    name main
28    fun_params
29    decl_exec_stmts
30    decl_part
31    decl_exp_stmt
32    alloc
33    ptr_decl
34    ptr_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 Kompositionen von PicoC-Nodes und RETI-Nodes mit besonderer Bedeutung

Hier sind jegliche **Kompositionen** von **PicoC-Nodes** und **RETI-Nodes** aufgelistet, die eine **besondere Bedeutung** haben und nicht bereits in der **Abstrakten Syntax 1.2.1** enthalten sind.

Komposition	Beschreibung
Ref(GlobalRead(Num('addr')))	Speichert Adresse der Speicherzelle, die Num('addr') Speicherzellen relativ zum Datensegment Register DS steht auf den Stack .
Ref(StackRead(Num('addr')))	Speichert Adresse der Speicherzelle, die Num('addr') Speicherzellen relativ zum Begin-Aktive-Funktion Register BAF steht auf den Stack .
Ref(Subscr(Tmp(Num('addr1')), Tmp(Num('addr2'))))	Berechnet die nächste Adresse aus der Adresse , die an Speicherzelle Tmp(Num('addr1')) steht und dem Subscript Index , der an Speicherzelle Tmp(Num('addr2')) steht und speichert diese auf den Stack . Die Berechnung ist abhängig davon ob der Datentyp ArrayDecl(datatype) oder PntrDecl(datatype) ist. Der Datentyp ist ein verstecktes Attribut von Ref(exp).
Ref(Attr(Tmp(Num('addr1')), Name('attr')))	Berechnet die nächste Adresse aus der Adresse , die an Speicherzelle Tmp(Num('addr1')) steht und dem Attributnamen Name('attr') und speichert diese auf den Stack . Zur Berechnung ist der Name des Struct in StructSpec(Name('st')) notwendig, dessen Attribut Name('attr') ist. StructSpec(Name('st')) ist ein verstecktes Attribut von Ref(exp).
Assign(Tmp(Num('size')), GlobalRead(Num('addr')))	Schreibt Num('size') viele Speicherzellen, die ab GlobalRead(Num('addr')) relativ zum Datensegment Register DS stehen, versetzt genauso auf den Stack .
Assign(Tmp(Num('size')), StackRead(Num('addr')))	Schreibt Num('size') viele Speicherzellen, die ab GlobalRead(Num('addr')) relativ zum Begin-Aktive-Funktion Register BAF stehen, versetzt genauso auf den Stack .
Exp(GlobalRead(Num('addr')))	Speichert Inhalt der Speicherzelle, die Num('addr') Speicherzellen relativ zum Datensegment Register DS steht auf den Stack .
Exp(StackRead(Num('addr')))	Speichert Inhalt der Speicherzelle, die Num('addr') Speicherzellen relativ zum Begin-Aktive-Funktion Register BAF steht auf den Stack .
Exp(Tmp(Num('addr')))	Speichert Inhalt der Speicherzelle, die Num('addr') Speicherzellen relativ zum Stackpointer Register SP steht auf den Stack .
Assign(Tmp(Num('addr1')), Tmp(Num('addr2')))	Speichert Inhalt der Speicherzelle Tmp(Num('addr2')), die Num('addr2') Speicherzellen relativ zum Stackpointer Register SP steht an der Adresse in der Speicherzelle, die Num('addr1') Speicherzellen relativ zum Stackpointer Register SP steht.
Assign(GlobalWrite(Num('addr')), Tmp(Num('size')))	Schreibt Num('size') viele Speicherzellen, die auf dem Stack stehen, versetzt genauso auf die Speicherzellen ab Num('addr') relativ zum Datensegment Register DS.
Assign(StackWrite(Num('addr')), Tmp(Num('size')))	Schreibt Num('size') viele Speicherzellen, die auf dem Stack stehen, versetzt genauso auf die Speicherzellen ab Num('addr') relativ zum Begin-Aktive-Funktion Register BAF.
Exp(Reg(reg))	Schreibt den aktuellen Wert des Registers reg auf den Stack
Instr(Loadi(), [Reg(Acc()), Goto()])	

Tabelle 1.2: Kompositionen von PicoC-Nodes und RETI-Nodes mit besonderer Bedeutung

Um die obige Tabelle 1.2 nicht mit unnötig viel repetitiven Inhalt zu füllen, wurden die zahlreichen Kompositionen **ausgelassen**, bei denen einfach nur **exp** durch $\text{Tmp}(\text{Num}('x')), x \in \mathbb{N}$ ersetzt wurde.

Zudem sind auch jegliche Kombinationen ausgelassen, bei denen einfach nur eine **Expression** an ein $\text{Exp}(\text{exp})$ bzw. $\text{Ref}(\text{exp})$ drangehängt wurde.

Ersetzt ein **exp**, dessen Wert bereits **vorher** berechnet wurde und x Speicherzellen **relativ** zum **Stackpointer** SP steht.

1.2.5.4 Abstrakte Syntax

<i>un_op</i>	::=	<i>Minus()</i> <i>Not()</i>	<i>L_Arith</i>
<i>bin_op</i>	::=	<i>Add()</i> <i>Sub()</i> <i>Mul()</i> <i>Div()</i> <i>Mod()</i> <i>Oplus()</i> <i>And()</i> <i>Or()</i>	
<i>exp</i>	::=	<i>Name(str)</i> <i>Num(str)</i> <i>Char(str)</i> <i>BinOp</i> (<i><exp></i> , <i><bin_op></i> , <i><exp></i>) <i>UnOp</i> (<i><un_op></i> , <i><exp></i>) <i>Call</i> (<i>Name('input')</i> , <i>None</i>)	
<i>exp_stmts</i>	::=	<i>Alloc</i> (<i><type_qual></i> , <i><datatype></i> , <i>Name(str)</i>) <i>Call</i> (<i>Name('print')</i> , <i><exp></i>)	
<i>un_op</i>	::=	<i>LogicNot()</i>	<i>L_Logic</i>
<i>rel</i>	::=	<i>Eq()</i> <i>NEq()</i> <i>Lt()</i> <i>LtE()</i> <i>Gt()</i> <i>GtE()</i>	
<i>bin_op</i>	::=	<i>LogicAnd()</i> <i>LogicOr()</i>	
<i>exp</i>	::=	<i>Atom</i> (<i><exp></i> , <i><rel></i> , <i><exp></i>) <i>ToBool</i> (<i><exp></i>)	
<i>type_qual</i>	::=	<i>Const()</i> <i>Writeable()</i>	<i>L_Assign_Alloc</i>
<i>datatype</i>	::=	<i>IntType()</i> <i>CharType()</i> <i>VoidType()</i>	
<i>assign_lhs</i>	::=	<i>Alloc</i> (<i><type_qual></i> , <i><datatype></i> , <i>Name(str)</i>) <i><rel_loc></i>	
<i>exp_stmts</i>	::=	<i>Alloc</i> (<i><type_qual></i> , <i><datatype></i> , <i>Name(str)</i>)	
<i>stmt</i>	::=	<i>Assign</i> (<i><assign_lhs></i> , <i><exp></i>) <i>Exp</i> (<i><exp_stmts></i>)	
<i>datatype</i>	::=	<i>PntrDecl</i> (<i>Num(str)</i> , <i><datatype></i>)	<i>L_Pntr</i>
<i>deref_loc</i>	::=	<i>Ref</i> (<i><ref_loc></i>) <i><ref_loc></i>	
<i>ref_loc</i>	::=	<i>Name(str)</i> <i>Deref</i> (<i><deref_loc></i> , <i><exp></i>) <i>Subscr</i> (<i><deref_loc></i> , <i><exp></i>) <i>Attr</i> (<i><ref_loc></i> , <i>Name(str)</i>)	
<i>exp</i>	::=	<i>Deref</i> (<i><deref_loc></i> , <i><exp></i>) <i>Ref</i> (<i><ref_loc></i>)	
<i>datatype</i>	::=	<i>ArrayDecl</i> (<i>Num(str)</i> +, <i><datatype></i>)	<i>L_Array</i>
<i>exp</i>	::=	<i>Subscr</i> (<i><deref_loc></i> , <i><exp></i>) <i>Array</i> (<i><exp></i> +)	
<i>datatype</i>	::=	<i>StructSpec</i> (<i>Name(str)</i>)	<i>L_Struct</i>
<i>exp</i>	::=	<i>Attr</i> (<i><ref_loc></i> , <i>Name(str)</i>) <i>Struct</i> (<i>Assign</i> (<i>Name(str)</i> , <i><exp></i>) +)	
<i>decl_def</i>	::=	<i>StructDecl</i> (<i>Name(str)</i> , <i>Alloc</i> (<i>Writeable()</i> , <i><datatype></i> , <i>Name(str)</i>) +)	
<i>stmt</i>	::=	<i>If</i> (<i><exp></i> , <i><stmt></i> *) <i>IfElse</i> (<i><exp></i> , <i><stmt></i> *, <i><stmt></i> *)	<i>L_If_Else</i>
<i>stmt</i>	::=	<i>While</i> (<i><exp></i> , <i><stmt></i> *) <i>DoWhile</i> (<i><exp></i> , <i><stmt></i> *)	<i>L_Loop</i>
<i>exp</i>	::=	<i>Call</i> (<i>Name(str)</i> , <i><exp></i> *)	<i>L_Fun</i>
<i>exp_stmts</i>	::=	<i>Call</i> (<i>Name(str)</i> , <i><exp></i> *)	
<i>stmt</i>	::=	<i>Return</i> (<i><exp></i>)	
<i>decl_def</i>	::=	<i>FunDecl</i> (<i><datatype></i> , <i>Name(str)</i> , <i>Alloc</i> (<i>Writeable()</i> , <i><datatype></i> , <i>Name(str)</i>)*) <i>FunDef</i> (<i><datatype></i> , <i>Name(str)</i> , <i>Alloc</i> (<i>Writeable()</i> , <i><datatype></i> , <i>Name(str)</i>)*, <i><stmt></i> *)	
<i>file</i>	::=	<i>File</i> (<i>Name(str)</i> , <i><decl_def></i> *)	<i>L_File</i>

Grammar 1.2.3: Abstrakte Syntax für L_{PiocC}

1.2.5.5 Transformer

1.2.5.6 Codebeispiel

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

```
1 File
2   Name './example_dt_simple_ast_gen_array_decl_and_alloc.ast',
3   [
4     StructDecl
5       Name 'st',
6       [
7         Alloc
8           Writeable,
9           PtrDecl
10            Num '1',
11            ArrayDecl
12              [
13                Num '5',
14                Num '6'
15              ],
16            PtrDecl
17              Num '1',
18              IntType 'int',
19            Name 'attr'
20          ],
21      FunDef
22        VoidType 'void',
23        Name 'main',
24        [],
25        [
26          Exp
27            Alloc
28              Writeable,
29              PtrDecl
30                Num '1',
31                ArrayDecl
32                  [
33                    Num '3',
34                    Num '2'
35                  ],
36              PtrDecl
37                Num '1',
38                StructSpec
39                  Name 'st',
40                Name 'var'
41            ]
42        ]
43    ]
44  ]
```

Code 1.4: Abstract Syntax Tree aus vereinfachtem Derivation Tree generiert

1.3 Code Generierung

1.3.1 Übersicht

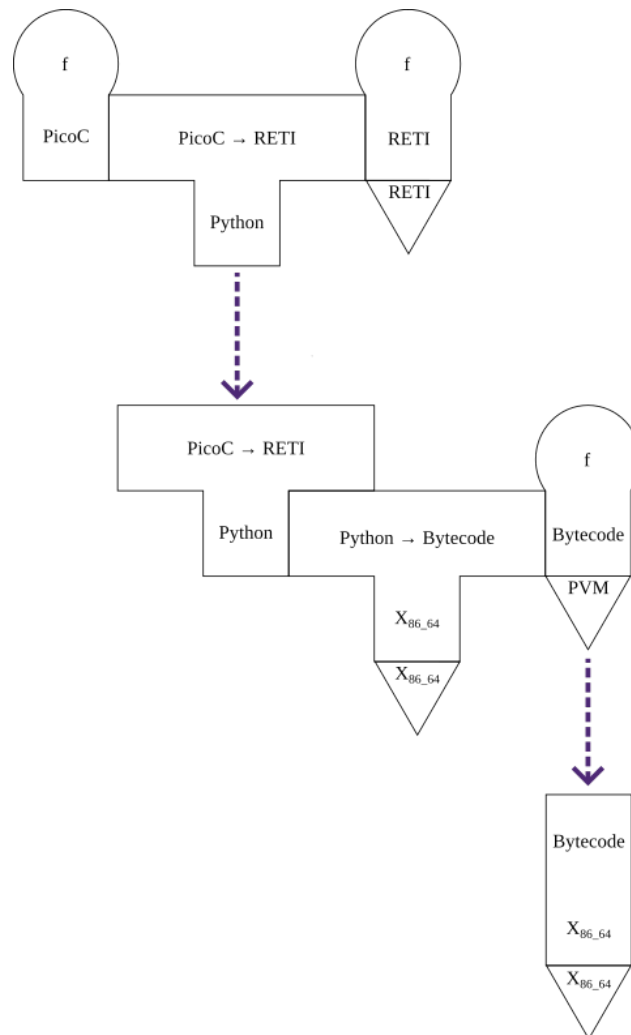


Abbildung 1.1: Cross-Compiler Kompiliervorgang ausgeschrieben

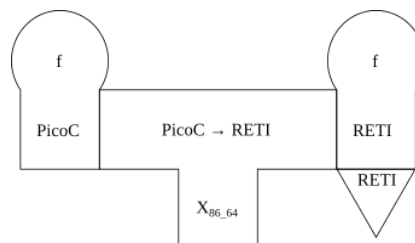


Abbildung 1.2: Cross-Compiler Kompiliervorgang Kurzform

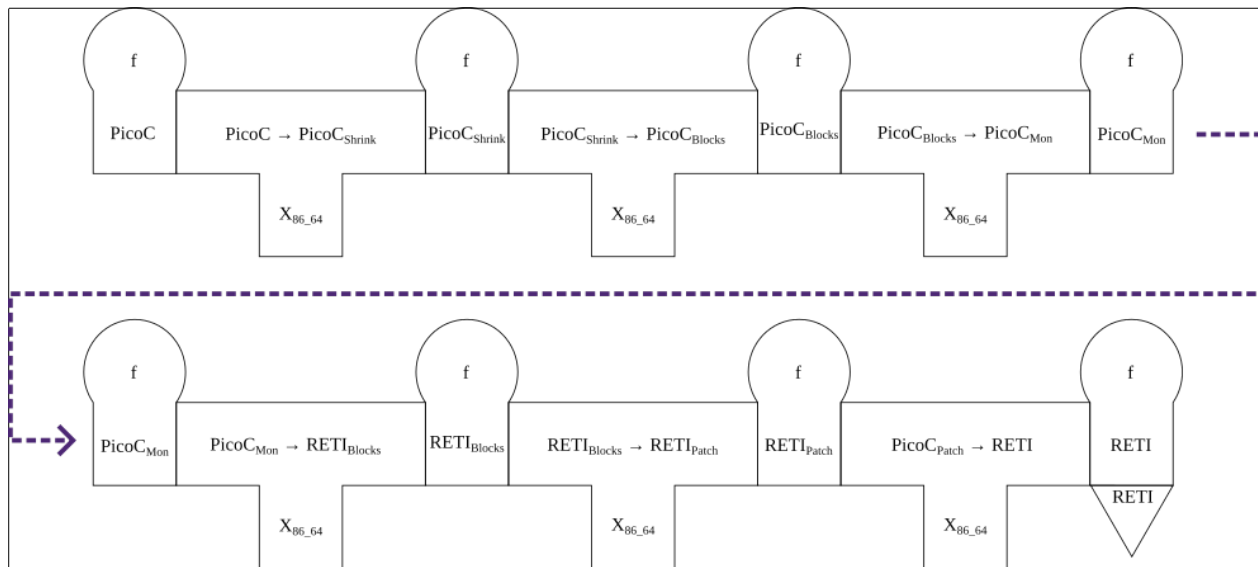


Abbildung 1.3: Architektur mit allen Passes ausgeschrieben

1.3.2 Passes

1.3.2.1 PicoC-Shrink Pass

1.3.2.1.1 Codebeispiel

```

1 // Author: Christoph Scholl, from the Operating Systems Lecture
2
3 void main() {
4     int n = 4;
5     int res = 1;
6     while (1) {
7         if (n == 1) {
8             return;
9         }
10        res = n * res;
11        n = n - 1;
12    }
13 }

```

Code 1.5: PicoC Code für Codebeispiel

```

1 File
2   Name './example_faculty_it.ast',
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 1.6: Abstract Syntax Tree für Codebeispiel

```

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 Codebeispiel

1.3.2.2 PicoC-Blocks Pass

1.3.2.2.1 Abstrakte Syntax

<i>decl_def</i>	$::=$	<i>FunDef</i> ($\langle datatype \rangle$, <i>Name</i> (<i>str</i>), <i>Alloc</i> (<i>Writeable</i> () , $\langle datatype \rangle$, <i>Name</i> (<i>str</i>))* , $\langle block \rangle$ *)	<i>L_Fun</i>
<i>block</i>	$::=$	<i>Block</i> (<i>Name</i> (<i>str</i>), $\langle stmt \rangle$ *)	<i>L_Blocks</i>
<i>stmt</i>	$::=$	<i>Goto</i> (<i>Name</i> (<i>str</i>)) <i>NewStackframe</i> (<i>Name</i> () , <i>Goto</i> (<i>str</i>)) <i>RemoveStackframe</i> () <i>SetScope</i> (<i>Name</i> (<i>str</i>)) <i>SingleLineComment</i> (<i>str</i> , <i>str</i>)	

Grammar 1.3.1: Abstrakte Syntax für L_{PicoC_Blocks}

1.3.2.2.2 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 Codebeispiel

1.3.2.3 PicoC-Mon Pass

1.3.2.3.1 Abstrakte Syntax

<i>ref_loc</i>	$::=$ <i>Tmp</i> (<i>Num</i> (<i>str</i>)) <i>StackRead</i> (<i>Num</i> (<i>str</i>)) <i>StackWrite</i> (<i>Num</i> (<i>str</i>)) <i>GlobalRead</i> (<i>Num</i> (<i>str</i>)) <i>GlobalWrite</i> (<i>Num</i> (<i>str</i>))	<i>L_Assign_Alloc</i>
<i>error_data</i>	$::=$ $\langle exp \rangle$ <i>Pos</i> (<i>Num</i> (<i>str</i>), <i>Num</i> (<i>str</i>))	
<i>exp</i>	$::=$ <i>Stack</i> (<i>Num</i> (<i>str</i>)) <i>Ref</i> ($\langle ref_loc \rangle$, $\langle datatype \rangle$, $\langle error_data \rangle$)	
<i>stmt</i>	$::=$ <i>Exp</i> ($\langle exp \rangle$) <i>Assign</i> (<i>Alloc</i> (<i>Writeable</i> ()), <i>StructSpec</i> (<i>Name</i> (<i>str</i>)), <i>Name</i> (<i>str</i>)), <i>Struct</i> (<i>Assign</i> (<i>Name</i> (<i>str</i>), $\langle exp \rangle$)+, $\langle datatype \rangle$)) <i>Assign</i> (<i>Alloc</i> (<i>Writeable</i> ()), <i>ArrayDecl</i> (<i>Num</i> (<i>str</i>)+, $\langle datatype \rangle$), <i>Name</i> (<i>str</i>), <i>Array</i> ($\langle exp \rangle$ +, $\langle datatype \rangle$))	
<i>symbol_table</i>	$::=$ <i>SymbolTable</i> ($\langle symbol \rangle$)	<i>L_Symbol_Table</i>
<i>symbol</i>	$::=$ <i>Symbol</i> ($\langle type_qual \rangle$, $\langle datatype \rangle$, $\langle name \rangle$, $\langle val \rangle$, $\langle pos \rangle$, $\langle size \rangle$)	
<i>type_qual</i>	$::=$ <i>Empty</i> ()	
<i>datatype</i>	$::=$ <i>BuiltIn</i> () <i>SelfDefined</i> ()	
<i>name</i>	$::=$ <i>Name</i> (<i>str</i>)	
<i>val</i>	$::=$ <i>Num</i> (<i>str</i>) <i>Empty</i> ()	
<i>pos</i>	$::=$ <i>Pos</i> (<i>Num</i> (<i>str</i>), <i>Num</i> (<i>str</i>)) <i>Empty</i> ()	
<i>size</i>	$::=$ <i>Num</i> (<i>str</i>) <i>Empty</i> ()	

Grammar 1.3.2: Abstrakte Syntax für L_{PicoC_Mon}

Definition 1.1: Symoltabelle

1.3.2.3.2 Codebeispiel

```

1 File
2   Name './example_faculty_it.picoc_mon',
3   [
4     Block
5       Name 'main.5',
6       [
7         // Assign(Name('n'), Num('4')),
8         Exp
9           Num '4',
10        Assign
11          GlobalWrite
12            Num '0',
13          Tmp
14            Num '1',
15          // Assign(Name('res'), Num('1')),
16          Exp
17            Num '1',
18          Assign
19            GlobalWrite
20              Num '1',
21            Tmp
22              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')),
47          ↪ GoTo(Name('if_else_after.1'))),
48          Exp
49            GlobalRead
50              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 Codebeispiel

1.3.2.4 RETI-Blocks Pass

1.3.2.4.1 Abstrakte Syntax

<i>program</i>	$::=$	$Program(Name(str), \langle block \rangle^*)$	$L_Program$
<i>exp_stmts</i>	$::=$	$Goto(str)$	L_Blocks
<i>instrs_before</i>	$::=$	$Num(str)$	
<i>num_instrs</i>	$::=$	$Num(str)$	
<i>block</i>	$::=$	$Block(Name(str), \langle instr \rangle^*, \langle instrs_before \rangle, \langle num_instrs \rangle)$	
<i>instr</i>	$::=$	$Goto(Name(str))$	

Grammar 1.3.3: Abstrakte Syntax für L_{RETI_Blocks}

1.3.2.4.2 Codebeispiel

```

1 File
2     Name './example_faculty_it.reti_blocks',
3     [
4         Block
5             Name 'main.5',
6             [
7                 # // Assign(Name('n'), Num('4')),
8                 # Exp(Num('4')),
9                 SUBI SP 1,
10                LOADI ACC 4,
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('res'), Num('1')),
17                # Exp(Num('1')),

```

```

18     SUBI SP 1,
19     LOADI ACC 1,
20     STOREIN SP ACC 1,
21     # Assign(GlobalWrite(Num('1')), Tmp(Num('1'))),
22     LOADIN SP ACC 1,
23     STOREIN DS ACC 1,
24     ADDI SP 1,
25     # // While(Num('1'), []),
26     Exp
27     GoTo
28     Name 'condition_check.4'
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     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')),
52     ↪ GoTo(Name('if_else_after.1'))),
53     # Exp(GlobalRead(Num('0'))),
54     SUBI SP 1,
55     LOADIN DS ACC 0,
56     STOREIN SP ACC 1,
57     # Exp(Num('1')),
58     SUBI SP 1,
59     LOADI ACC 1,
60     STOREIN SP ACC 1,
61     LOADIN SP ACC 2,
62     LOADIN SP IN2 1,
63     SUB ACC IN2,
64     JUMP== 3;,
65     LOADI ACC 0,
66     JUMP 2;,
67     LOADI ACC 1,
68     STOREIN SP ACC 2,
69     ADDI SP 1,
70     # IfElse(Tmp(Num('1')), GoTo(Name('if.2')), GoTo(Name('if_else_after.1'))),
71     LOADIN SP ACC 1,
72     ADDI SP 1,
73     JUMP== GoTo
74     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 Codebeispiel

1.3.2.5 RETI-Patch Pass

1.3.2.5.1 Abstrakte Syntax

$$\text{stmt} ::= \text{Exit}(\text{Num}(\text{str}))$$
Grammar 1.3.4: Abstrakte Syntax für L_{RETI_Patch}

1.3.2.5.2 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')),
49         ↪ GoTo(Name('if_else_after.1'))),
50         # Exp(GlobalRead(Num('0'))),
51         SUBI SP 1,
52         LOADIN DS ACC 0,
53         STOREIN SP ACC 1,
54         # Exp(Num('1')),
55         SUBI SP 1,
56         LOADI ACC 1,
57         STOREIN SP ACC 1,
58         LOADIN SP ACC 2,
59         LOADIN SP IN2 1,
60         SUB ACC IN2,
61         JUMP== 3;,
62         LOADI ACC 0,
63         JUMP 2;,
64         LOADI ACC 1,
65         STOREIN SP ACC 2,
66         ADDI SP 1,
67         # IfElse(Tmp(Num('1')), GoTo(Name('if.2')), GoTo(Name('if_else_after.1'))),
68         LOADIN SP ACC 1,
69         ADDI SP 1,
70         JUMP== GoTo
71             Name 'if_else_after.1';
72 ],
73 Block
74     Name 'if.2',
75     [
76         # Return(Empty()),
77         LOADIN BAF PC -1
78     ],
79 Block
80     Name 'if_else_after.1',
81     [
82         # // Assign(Name('res'), BinOp(Name('n'), Mul('*'), Name('res'))),
83         # Exp(GlobalRead(Num('0'))),
84         SUBI SP 1,
85         LOADIN DS ACC 0,
86         STOREIN SP ACC 1,
87         # Exp(GlobalRead(Num('1'))),
88         SUBI SP 1,
89         LOADIN DS ACC 1,
90         STOREIN SP ACC 1,
91         LOADIN SP ACC 2,
92         LOADIN SP IN2 1,
93         MULT ACC IN2,
94         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 Codebeispiel

1.3.2.6 RETI Pass

1.3.2.6.1 Konkrete und Abstrakte Syntax

<i>dig_no_0</i>	::=	"1" "2" "3" "4" "5" "6"	<i>L_Program</i>
		"7" "8" "9"	
<i>dig_with_0</i>	::=	"0" <i>dig_no_0</i>	
<i>num</i>	::=	"0" <i>dig_no_0</i> <i>dig_with_0</i> * "-" <i>dig_with_0</i> *	
<i>letter</i>	::=	"a"..."Z"	
<i>name</i>	::=	<i>letter</i> (<i>letter</i> <i>dig_with_0</i> _)*	
<i>reg</i>	::=	"ACC" "IN1" "IN2" "PC" "SP"	
		"BAF" "CS" "DS"	
<i>arg</i>	::=	<i>reg</i> <i>num</i>	
<i>rel</i>	::=	"==" "!=" "<" "<=" ">"	
		">=" "_NOP"	

Grammar 1.3.5: Konkrete Syntax für *L_{RETI.Lex}*


```

16 LOADIN SP ACC 1;
17 STOREIN DS ACC 1;
18 ADDI SP 1;
19 # // While(Num('1'), [])
20 # // IfElse(Num('1'), GoTo(Name('while_branch.3')), GoTo(Name('while_after.0')))
21 # Exp(Num('1'))
22 SUBI SP 1;
23 LOADI ACC 1;
24 STOREIN SP ACC 1;
25 # IfElse(Tmp(Num('1')), GoTo(Name('while_branch.3')), GoTo(Name('while_after.0')))
26 LOADIN SP ACC 1;
27 ADDI SP 1;
28 JUMP== 49;
29 # // If(Atom(Name('n'), Eq('=='), Num('1')), [])
30 # // IfElse(Atom(Name('n'), Eq('=='), Num('1')), GoTo(Name('if.2')),
    ↪ GoTo(Name('if_else_after.1')))
31 # Exp(GlobalRead(Num('0')))
32 SUBI SP 1;
33 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 Codebeispiel

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.

```

1 File
2   Name './example_pntr_ref.picoc_mon',
3   [
4     Block
5       Name 'main.0',
6       [
7         // Assign(Name('var'), Num('42')),
8         Exp
9           Num '42',
10        Assign
11          GlobalWrite
12            Num '0',
13          Tmp
14            Num '1',
15        // Assign(Name('pntr'), Ref(Name('var'))),
16        Ref
17          GlobalRead
18            Num '0',
19        Assign
20          GlobalWrite
21            Num '1',
22          Tmp
23            Num '1',
24        Return
25          Empty
26      ]
27  ]

```

Code 1.15: PicoC Mon Pass für Pointer Referenzierung

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

```

1 File
2   Name './example_pntr_ref.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('pntr'), 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 1.16: 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

```

1 File
2   Name './example_pntr_deref.picoc_shrink',
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              Subscr
26                Name 'pntr',
27                Num '0'
28          ]
29   ]

```

Code 1.19: PicoC Shrink Pass für Pointer Dereferenzierung

1.3.4 Umsetzung von Arrays

1.3.4.1 Initialisierung von Arrays

```

1 void main() {
2   int ar[2][1] = {{4}, {2}};
3 }

```

Code 1.20: PicoC Code für Array Initialisierung

```

1 File
2   Name './example_array_init.ast',
3   [
4     FunDef
5       VoidType 'void',
6       Name 'main',
7       [],

```

```

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

```

1 File
2   Name './example_array_init.picoc_mon',
3   [
4     Block
5       Name 'main.0',
6       [
7         // Assign(Name('ar'), Array([Array([Num('4')]), Array([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        Return
18          Empty
19      ]
20  ]

```

Code 1.23: PicoC Mon Pass für Array Initialisierung

```

1 File
2   Name './example_array_init.reti_blocks',
3   [
4     Block
5       Name 'main.0',
6       [
7         # // Assign(Name('ar'), Array([Array([Num('4')]), Array([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        # Return(Empty()),
23        LOADIN BAF PC -1
24      ]
25  ]

```

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:

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

Code 1.25: PicoC Code für Zugriff auf Arrayindex

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

Code 1.26: Abstract Syntax Tree für Zugriff auf Arrayindex

```
1 File
2   Name './example_array_access.picoc_mon',
3   [
4     Block
5       Name 'main.0',
6       [
7         // Assign(Name('ar'), Array([Num('1'), Num('2')]))),
8         Exp
9           Num '1',
10        Exp
11          Num '2',
```

```

12     Assign
13     GlobalWrite
14     Num '0',
15     Tmp
16     Num '2',
17     // Exp(Subscr(Name('ar'), Num('2'))),
18     Ref
19     GlobalRead
20     Num '0',
21     Exp
22     Num '2',
23     Ref
24     Subscr
25     Tmp
26     Num '2',
27     Tmp
28     Num '1',
29     Exp
30     Subscr
31     Tmp
32     Num '1',
33     Num '0',
34     Return
35     Empty
36 ]
37 ]

```

Code 1.27: PicoC Mon Pass für Zugriff auf Arrayindex

```

1 File
2 Name './example_array_access.reti_blocks',
3 [
4     Block
5     Name 'main.0',
6     [
7         # // Assign(Name('ar'), Array([Num('1'), Num('2')])),
8         # Exp(Num('1')),
9         SUBI SP 1,
10        LOADI ACC 1,
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        # // Exp(Subscr(Name('ar'), Num('2'))),
23        # Ref(GlobalRead(Num('0'))),
24        SUBI SP 1,
25        LOADI IN1 0,

```

```

26      ADD IN1 DS,
27      STOREIN SP IN1 1,
28      # Exp(Num('2')),
29      SUBI SP 1,
30      LOADI ACC 2,
31      STOREIN SP ACC 1,
32      # Ref(Subscr(Tmp(Num('2')), Tmp(Num('1')))),
33      LOADIN SP IN1 2,
34      LOADIN SP IN2 1,
35      MULTI IN2 1,
36      ADD IN1 IN2,
37      ADDI SP 1,
38      STOREIN SP IN1 1,
39      # Exp(Subscr(Tmp(Num('1')), Num('0'))),
40      LOADIN SP IN1 1,
41      LOADIN IN1 ACC 0,
42      STOREIN SP ACC 1,
43      # Return(Empty()),
44      LOADIN BAF PC -1
45  ]
46 ]

```

Code 1.28: RETI Blocks Pass für Zugriff auf Arrayindex

1.3.4.3 Zuweisung an Arrayindex

```

1 void main() {
2     int ar[2];
3     ar[2] = 42;
4 }

```

Code 1.29: PicoC Code für Zuweisung an Arrayindex

```

1 File
2   Name './example_array_assignment.ast',
3   [
4     FunDef
5       VoidType 'void',
6       Name 'main',
7       [],
8       [
9         Exp
10          Alloc
11          Writeable,
12          ArrayDecl
13            [
14              Num '2'
15            ],
16          IntType 'int',
17          Name 'ar',
18        Assign
19        Subscr

```

```

20     Name 'ar',
21     Num '2',
22     Num '42'
23   ]
24 ]

```

Code 1.30: Abstract Syntax Tree für Zuweisung an Arrayindex

```

1 File
2   Name './example_array_assignment.picoc_mon',
3   [
4     Block
5       Name 'main.0',
6       [
7         // Assign(Subscr(Name('ar'), Num('2')), Num('42')),
8         Exp
9           Num '42',
10        Ref
11          GlobalRead
12            Num '0',
13        Exp
14          Num '2',
15        Ref
16          Subscr
17            Tmp
18              Num '2',
19            Tmp
20              Num '1',
21        Assign
22          Subscr
23            Tmp
24              Num '1',
25            Num '0',
26          Tmp
27            Num '2',
28        Return
29          Empty
30      ]
31  ]

```

Code 1.31: PicoC Mon Pass für Zuweisung an Arrayindex

```

1 File
2   Name './example_array_assignment.reti_blocks',
3   [
4     Block
5       Name 'main.0',
6       [
7         # // Assign(Subscr(Name('ar'), Num('2')), Num('42')),
8         # Exp(Num('42')),
9         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')], PtrDecl(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(),
16       ↪ ArrayDecl([Num('3')], PtrDecl(Num('1'), IntType('int'))), Name('ar'))])
17     name:            Name('st1')
18     value or address: [Name('ar@st1')]
19     position:        Pos(Num('1'), Num('7'))
20     size:            Num('3')
21   },
22   Symbol(
23   {
24     type qualifier:  Empty()
25     datatype:        StructSpec(Name('st1'))
26     name:            Name('st@st2')
27     value or address: Empty()
28     position:        Pos(Num('3'), Num('23'))
29     size:            Num('3')
30   },
31   Symbol(
32   {
33     type qualifier:  Empty()
34     datatype:        StructDecl(Name('st2'), [Alloc(Writeable(),
35       ↪ StructSpec(Name('st1')), Name('st'))])
36     name:            Name('st2')
37     value or address: [Name('st@st2')]
38     position:        Pos(Num('3'), Num('7'))
39     size:            Num('3')
40   },
41   Symbol(
42   {
43     type qualifier:  Empty()
44     datatype:        FunDecl(VoidType('void'), Name('main'), [])
45     name:            Name('main')
46     value or address: Empty()
47     position:        Pos(Num('5'), Num('5'))
48     size:            Empty()
49   }
50 ]

```

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

```
1 File
2   Name './example_struct_init.ast',
3   [
4     StructDecl
5       Name 'st1',
6       [
7         Alloc
8           Writeable,
9           ArrayDecl
10            [
11              Num '1'
12            ],
13          PtrDecl
14            Num '1',
15            IntType 'int',
16          Name 'pntr'
17        ],
18      StructDecl
19        Name 'st2',
20        [
21          Alloc
22            Writeable,
23            StructSpec
24              Name 'st1',
25              Name 'st'
26        ],
27      FunDef
28        VoidType 'void',
29        Name 'main',
30        [],
31        [
32          Assign
33            Alloc
34              Writeable,
35              IntType 'int',
36              Name 'var',
37              Num '42',
38          Assign
39            Alloc
40              Writeable,
41              StructSpec
42                Name 'st1',
43                Name 'st',
44            Struct
45              [
46                Assign
47                  Name 'st',
48                  Struct
49                    [
50                      Assign
51                        Name 'pntr',
52                        Array
```

```

53         [
54             Array
55             [
56                 Ref
57                 Name 'var'
58             ]
59         ]
60     ]
61 ]
62 ]
63 ]

```

Code 1.36: Abstract Syntax Tree für Initialisierung von Structs

```

1 SymbolTable
2 [
3     Symbol(
4     {
5         type qualifier:      Empty()
6         datatype:            ArrayDecl([Num('1')], PtrDecl(Num('1'), IntType('int')))
7         name:                Name('ptr@st1')
8         value or address:    Empty()
9         position:            Pos(Num('1'), Num('17'))
10        size:                Num('1')
11    },
12    Symbol(
13    {
14        type qualifier:      Empty()
15        datatype:            StructDecl(Name('st1'), [Alloc(Writeable(),
16        ↪ ArrayDecl([Num('1')], PtrDecl(Num('1'), IntType('int'))), Name('ptr'))])
17        name:                Name('st1')
18        value or address:    [Name('ptr@st1')]
19        position:            Pos(Num('1'), Num('7'))
20        size:                Num('1')
21    },
22    Symbol(
23    {
24        type qualifier:      Empty()
25        datatype:            StructSpec(Name('st1'))
26        name:                Name('st@st2')
27        value or address:    Empty()
28        position:            Pos(Num('3'), Num('23'))
29        size:                Num('1')
30    },
31    Symbol(
32    {
33        type qualifier:      Empty()
34        datatype:            StructDecl(Name('st2'), [Alloc(Writeable(),
35        ↪ StructSpec(Name('st1')), Name('st'))])
36        name:                Name('st2')
37        value or address:    [Name('st@st2')]
38        position:            Pos(Num('3'), Num('7'))
39        size:                Num('1')
40    },

```

```

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 Symbol(
49     {
50         type qualifier:      Writeable()
51         datatype:            IntType('int')
52         name:                 Name('var@main')
53         value or address:     Num('0')
54         position:             Pos(Num('6'), Num('6'))
55         size:                 Num('1')
56     },
57 Symbol(
58     {
59         type qualifier:      Writeable()
60         datatype:            StructSpec(Name('st1'))
61         name:                 Name('st@main')
62         value or address:     Num('1')
63         position:             Pos(Num('7'), Num('13'))
64         size:                 Num('1')
65     }
66 ]

```

Code 1.37: Symboltabelle für Initialisierung von Structs

```

1 File
2   Name './example_struct_init.picoc_mon',
3   [
4     Block
5       Name 'main.0',
6       [
7         // Assign(Name('var'), Num('42')),
8         Exp
9           Num '42',
10        Assign
11          GlobalWrite
12            Num '0',
13          Tmp
14            Num '1',
15        // Assign(Name('st'), Struct([Assign(Name('st'), Struct([Assign(Name('pntr'),
16        ↪   Array([Array([Ref(Name('var'))]))]))]),
17        Ref
18          GlobalRead
19            Num '0',
20        Assign
21          GlobalWrite
22            Num '1',
23        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.O',
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')),
17      ↪   Array([Array([Ref(Name('var'))])])])]),
18      # Ref(GlobalRead(Num('0'))),
19      SUBI SP 1,
20      LOADI IN1 0,
21      ADD IN1 DS,
22      STOREIN SP IN1 1,
23      # Assign(GlobalWrite(Num('1')), Tmp(Num('1'))),
24      LOADIN SP ACC 1,
25      STOREIN DS ACC 1,
26      ADDI SP 1,
27      # Return(Empty()),
28      LOADIN BAF PC -1
29    ]
30  ]

```

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

```

1 File
2   Name './example_struct_attr_access.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            Exp
37              Attr
38                Name 'st',
39                Name 'y'
40          ]
41      ]

```

Code 1.41: Abstract Syntax Tree für Zugriff auf Structattribut

```

1 File
2   Name './example_struct_attr_access.picoc_mon',
3   [
4     Block
5       Name 'main.0',
6       [
7         // Assign(Name('st'), Struct([Assign(Name('x'), Num('4')), Assign(Name('y'),
8         ↪ Num('2'))])),
9       Exp

```

```

9      Num '4',
10     Exp
11     Num '2',
12     Assign
13     GlobalWrite
14     Num '0',
15     Tmp
16     Num '2',
17     Ref
18     GlobalRead
19     Num '0',
20     Ref
21     Attr
22     Tmp
23     Num '1',
24     Name 'y',
25     Exp
26     Subscr
27     Tmp
28     Num '1',
29     Num '0',
30     Return
31     Empty
32 ]
33 ]

```

Code 1.42: PicoC Mon Pass für Zugriff auf Structattribut

```

1 File
2 Name './example_struct_attr_access.reti_blocks',
3 [
4     Block
5     Name 'main.0',
6     [
7         # // Assign(Name('st'), Struct([Assign(Name('x'), Num('4')), Assign(Name('y'),
8         ↪ Num('2'))]))),
9         # Exp(Num('4')),
10        SUBI SP 1,
11        LOADI ACC 4,
12        STOREIN SP ACC 1,
13        # Exp(Num('2')),
14        SUBI SP 1,
15        LOADI ACC 2,
16        STOREIN SP ACC 1,
17        # Assign(GlobalWrite(Num('0')), Tmp(Num('2'))),
18        LOADIN SP ACC 1,
19        STOREIN DS ACC 1,
20        LOADIN SP ACC 2,
21        STOREIN DS ACC 0,
22        ADDI SP 2,
23        # Ref(GlobalRead(Num('0'))),
24        SUBI SP 1,
25        LOADI IN1 0,
26        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'),
8       ↪ Num('2'))])),
9     Exp
10    Num '4',
11    Exp
12    Num '2',
13    Assign
14    GlobalWrite
15    Num '0',
16    Tmp
17    Num '2',
18    // Assign(Attr(Name('st'), Name('y')), Num('42')),
19    Exp
20    Num '42',
21    Ref
22    GlobalRead
23    Num '0',
24    Ref
25    Attr
26    Tmp
27    Num '1',
28    Name 'y',
29    Assign
30    Subscr
31    Tmp
32    Num '1',
33    Num '0',
34    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'),
8         ↪ Num('2'))])),
9         # Exp(Num('4')),
10        SUBI SP 1,
11        LOADI ACC 4,
12        STOREIN SP ACC 1,
13        # Exp(Num('2')),
14        SUBI SP 1,
15        LOADI ACC 2,
16        STOREIN SP ACC 1,
17        # Assign(GlobalWrite(Num('0')), Tmp(Num('2'))),
18        LOADIN SP ACC 1,
19        STOREIN DS ACC 1,
20        LOADIN SP ACC 2,
21        STOREIN DS ACC 0,
22        ADDI SP 2,
23        # // Assign(Attr(Name('st'), Name('y')), Num('42')),
24        # Exp(Num('42')),
25        SUBI SP 1,
26        LOADI ACC 42,
27        STOREIN SP ACC 1,
28        # Ref(GlobalRead(Num('0'))),
29        SUBI SP 1,
30        LOADI IN1 0,
31        ADD IN1 DS,
32        STOREIN SP IN1 1,
33        # Ref(Attr(Tmp(Num('1')), Name('y'))),
34        LOADIN SP IN1 1,
35        ADDI IN1 1,
36        STOREIN SP IN1 1,
37        # Assign(Subscr(Tmp(Num('1')), Num('0')), Tmp(Num('2'))),
38        LOADIN SP IN1 1,
39        LOADIN SP ACC 2,
40        ADDI SP 2,
41        STOREIN IN1 ACC 0,
42        # Return(Empty()),
43        LOADIN BAF PC -1
44      ]
45    ]

```

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         PtrDecl
39         Num '1',
40         ArrayDecl
41         [
42             Num '3'
43         ],
44         PtrDecl
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     PtrDecl
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',

```

```
3  [
4    Block
5      Name 'main.1',
6      [
7        Exp
8          GlobalRead
9            Num '9',
10         Return
11           Empty
12       ],
13     Block
14       Name 'fun.0',
15       [
16         Exp
17           StackRead
18             Num '9',
19         Return
20           Empty
21       ]
22   ]
```

Code 1.50: PicoC Mon Pass für den Einleitungsteil

```
1 File
2   Name './example_derived_dts_introduction_part.reti_blocks',
3   [
4     Block
5       Name 'main.1',
6       [
7         # Exp(GlobalRead(Num('9'))),
8         SUBI SP 1,
9         LOADIN DS ACC 9,
10        STOREIN SP ACC 1,
11        # Return(Empty()),
12        LOADIN BAF PC -1
13      ],
14     Block
15       Name 'fun.0',
16       [
17         # Exp(StackRead(Num('9'))),
18         SUBI SP 1,
19         LOADIN BAF ACC -11,
20         STOREIN SP ACC 1,
21         # Return(Empty()),
22         LOADIN BAF PC -1
23      ]
24   ]
```

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           PtrDecl
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

```



```

36         Subscr
37         Tmp
38         Num '2',
39         Tmp
40         Num '1',
41     Exp
42     Num '0',
43     Ref
44     Subscr
45     Tmp
46     Num '2',
47     Tmp
48     Num '1',
49     Exp
50     Subscr
51     Tmp
52     Num '1',
53     Num '0',
54     Return
55     Empty
56 ]
57 ]

```

Code 1.54: PicoC Mon Pass für den Mittelteil

```

1 File
2   Name './example_derived_dts_main_part.reti_blocks',
3   [
4     Block
5     Name 'main.0',
6     [
7       # // Assign(Name('var'), Array([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_first'), Struct([Assign(Name('ar'), 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      # // Exp(Subscr(Subscr(Attr(Name('st_first'), Name('ar')), Num('0')), Num('0'))),
27      # Ref(GlobalRead(Num('1'))),
28      SUBI SP 1,
29      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

```
1 File
2   Name './example_derived_dts_final_part.ast',
3   [
4     StructDecl
5       Name 'st',
6       [
7         Alloc
8           Writeable,
9           ArrayDecl
10            [
11              Num '2'
12            ],
13            IntType 'int',
14            Name 'attr'
15        ],
16      FunDef
17        VoidType 'void',
18        Name 'main',
19        [],
20        [
21          Assign
22            Alloc
23              Writeable,
24              ArrayDecl
25                [
26                  Num '1',
27                  Num '2'
28                ],
29                IntType 'int',
30                Name 'ar1',
31              Array
32                [
33                  Array
34                    [
35                      Num '42',
36                      Num '314'
37                    ]
38                ],
39            Assign
40              Alloc
41                Writeable,
42                ArrayDecl
43                  [
44                    Num '1'
45                  ],
46                StructSpec
47                  Name 'st',
48                  Name 'ar2',
49                Struct
50                  [
51                    Assign
52                      Name 'attr',
```

```

53         Array
54         [
55             Num '42',
56             Num '314'
57         ]
58     ],
59     Assign
60     Alloc
61     Writeable,
62     IntType 'int',
63     Name 'var',
64     Num '42',
65     Assign
66     Alloc
67     Writeable,
68     PtrDecl
69     Num '1',
70     IntType 'int',
71     Name 'ptr1',
72     Ref
73     Name 'var',
74     Assign
75     Alloc
76     Writeable,
77     PtrDecl
78     Num '2',
79     IntType 'int',
80     Name 'ptr2',
81     Ref
82     Name 'ptr1',
83     Exp
84     Subscr
85     Name 'ar1',
86     Num '0',
87     Exp
88     Subscr
89     Name 'ar2',
90     Num '0',
91     Exp
92     Deref
93     Name 'ptr2',
94     Num '0'
95 ]
96 ]

```

Code 1.57: Abstract Syntax Tree für den Schlussteil

```

1 File
2   Name './example_derived_dts_final_part.picoc_mon',
3   [
4     Block
5     Name 'main.0',
6     [
7       // 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'),
18     ↪ Num('314')]))])),
19     Exp
20     Num '42',
21     Exp
22     Num '314',
23     Assign
24     GlobalWrite
25     Num '2',
26     Tmp
27     Num '2',
28     // Assign(Name('var'), Num('42')),
29     Exp
30     Num '42',
31     Assign
32     GlobalWrite
33     Num '4',
34     Tmp
35     Num '1',
36     // Assign(Name('pntr1'), Ref(Name('var'))),
37     Ref
38     GlobalRead
39     Num '4',
40     Assign
41     GlobalWrite
42     Num '5',
43     Tmp
44     Num '1',
45     // Assign(Name('pntr2'), Ref(Name('pntr1'))),
46     Ref
47     GlobalRead
48     Num '5',
49     Assign
50     GlobalWrite
51     Num '6',
52     Tmp
53     Num '1',
54     // Exp(Subscr(Name('ar1'), Num('0'))),
55     Ref
56     GlobalRead
57     Num '0',
58     Exp
59     Num '0',
60     Ref
61     Subscr
62     Tmp
63     Num '2',
64     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'),
23     ↪ Num('314')]))])),
24     # Exp(Num('42')),
25     SUBI SP 1,
26     LOADI ACC 42,
27     STOREIN SP ACC 1,
28     # Exp(Num('314')),
29     SUBI SP 1,
30     LOADI ACC 314,
31     STOREIN SP ACC 1,
32     # Assign(GlobalWrite(Num('2')), Tmp(Num('2'))),
33     LOADIN SP ACC 1,
34     STOREIN DS ACC 3,
35     LOADIN SP ACC 2,
36     STOREIN DS ACC 2,
37     ADDI SP 2,
38     # // Assign(Name('var'), Num('42')),
39     # Exp(Num('42')),
40     SUBI SP 1,
41     LOADI ACC 42,
42     STOREIN SP ACC 1,
43     # Assign(GlobalWrite(Num('4')), Tmp(Num('1'))),
44     LOADIN SP ACC 1,
45     STOREIN DS ACC 4,
46     ADDI SP 1,
47     # // Assign(Name('pntr1'), Ref(Name('var'))),
48     # Ref(GlobalRead(Num('4'))),
49     SUBI SP 1,
50     LOADI IN1 4,
51     ADD IN1 DS,
52     STOREIN SP IN1 1,
53     # Assign(GlobalWrite(Num('5')), Tmp(Num('1'))),
54     LOADIN SP ACC 1,
55     STOREIN DS ACC 5,
56     ADDI SP 1,
57     # // Assign(Name('pntr2'), Ref(Name('pntr1'))),
58     # Ref(GlobalRead(Num('5'))),
59     SUBI SP 1,
60     LOADI IN1 5,
61     ADD IN1 DS,
62     STOREIN SP IN1 1,
63     # Assign(GlobalWrite(Num('6')), Tmp(Num('1'))),
64     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   ]

```

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

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

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

```
1 File  
2   Name './example_3_funs_main.picoc_mon',  
3   [  
4     Block  
5       Name 'fun1.2',  
6       [  
7         Return  
8         Empty  
9       ],  
10    Block  
11      Name 'fun2.1',  
12      [  
13        // Return(Num('1')),  
14        Exp  
15          Num '1',  
16        Return  
17          Tmp  
18            Num '1'  
19      ],  
20    Block  
21      Name 'main.0',  
22      [  
23        Return  
24        Empty  
25      ]  
26  ]
```

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

```
1 File  
2   Name './example_3_funs_main.reti_blocks',  
3   [  
4     Block  
5       Name 'fun1.2',  
6       [  
7
```

```

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

```

```

29     ],
30     Block
31     Name 'main.0',
32     [
33         # Return(Empty()),
34         LOADIN BAF PC -1
35     ]
36 ]

```

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

1.3.7.2 Funktionsdeklaration und -definition

```

1 int fun2(int var);
2
3 void fun1() {
4 }
5
6 void main() {
7     int var = fun2(42);
8     return;
9 }
10
11 int fun2(int var) {
12     return var;
13 }

```

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

```

1 SymbolTable
2 [
3     Symbol(
4         {
5             type qualifier:      Empty()
6             datatype:            FunDecl(IntType('int'), Name('fun2'), [Alloc(Writeable(),
7                                     ↪ IntType('int'), Name('var'))])
8             name:                Name('fun2')
9             value or address:    Empty()
10            position:            Pos(Num('1'), Num('4'))
11            size:                Empty()
12        },
13        Symbol(
14            {
15                type qualifier:    Empty()
16                datatype:          FunDecl(VoidType('void'), Name('fun1'), [])
17                name:              Name('fun1')
18                value or address:  Empty()
19                position:          Pos(Num('3'), Num('5'))
20                size:              Empty()
21            },
22            Symbol(
23                {

```

```

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',

```

```

3  [
4    Block
5      Name 'main.1',
6      [
7        StackMalloc
8          Num '2',
9        Ref
10         GlobalRead
11         Num '0',
12        NewStackframe
13         Name 'stack_fun',
14        GoTo
15         Name 'addr@next_instr',
16      Exp
17        GoTo
18         Name 'stack_fun.0',
19      RemoveStackframe,
20      Return
21        Empty
22    ],
23  Block
24    Name 'stack_fun.0',
25    [
26      Return
27        Empty
28    ]
29 ]

```

Code 1.72: PicoC Mon Pass für Funktionsaufruf ohne Rückgabewert

```

1 File
2   Name './example_fun_call_no_return_value.reti.blocks',
3   [
4     Block
5       Name 'main.1',
6       [
7         # StackMalloc(Num('2')),
8         SUBI SP 2,
9         # Ref(GlobalRead(Num('0'))),
10        SUBI SP 1,
11        LOADI IN1 0,
12        ADD IN1 DS,
13        STOREIN SP IN1 1,
14        # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr'))),
15        MOVE BAF ACC,
16        ADDI SP 3,
17        MOVE SP BAF,
18        SUBI SP 4,
19        STOREIN BAF ACC 0,
20        LOADI ACC GoTo
21          Name 'addr@next_instr',
22        ADD ACC CS,
23        STOREIN BAF ACC -1,
24      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;

```

```

3 }
4
5 void main() {
6     int var = stack_fun();
7 }

```

Code 1.75: PicoC Code für Funktionsaufruf mit Rückgabewert

```

1 File
2   Name './example_fun_call_with_return_value.picoc_mon',
3   [
4     Block
5       Name 'stack_fun.1',
6       [
7         // Return(Num('42')),
8         Exp
9           Num '42',
10        Return
11          Tmp
12            Num '1'
13      ],
14    Block
15      Name 'main.0',
16      [
17        // Assign(Name('var'), Call(Name('stack_fun'), [])),
18        StackMalloc
19          Num '2',
20        NewStackframe
21          Name 'stack_fun',
22          GoTo
23            Name 'addr@next_instr',
24        Exp
25          GoTo
26            Name 'stack_fun.1',
27        RemoveStackframe,
28        Assign
29          GlobalWrite
30            Num '0',
31          Tmp
32            Num '1',
33        Return
34          Empty
35      ]
36  ]

```

Code 1.76: PicoC Mon Pass für Funktionsaufruf mit Rückgabewert

```

1 File
2   Name './example_fun_call_with_return_value.reti_blocks',
3   [
4     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;

```

```

9 ADDI SP 1;
10 LOADIN BAF PC -1;
11 # // Assign(Name('var'), Call(Name('stack_fun'), []))
12 # StackMalloc(Num('2'))
13 SUBI SP 2;
14 # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr'))))
15 MOVE BAF ACC;
16 ADDI SP 2;
17 MOVE SP BAF;
18 SUBI SP 2;
19 STOREIN BAF ACC 0;
20 LOADI ACC 17;
21 ADD ACC CS;
22 STOREIN BAF ACC -1;
23 JUMP -15;
24 # RemoveStackframe()
25 MOVE BAF IN1;
26 LOADIN IN1 BAF 0;
27 MOVE IN1 SP;
28 # Assign(GlobalWrite(Num('0')), Tmp(Num('1'))))
29 LOADIN SP ACC 1;
30 STOREIN DS ACC 0;
31 ADDI SP 1;
32 # Return(Empty())
33 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

```

1 void stack_fun(int (*param1)[3], int param2[2][3]) {
2 }
3
4 void main() {
5     int local_var1[2][3];
6     int local_var2[2][3];
7     stack_fun(local_var1, local_var2);
8 }

```

Code 1.79: PicoC Code für Call by Sharing für Arrays

```

1 File
2   Name './example_fun_call_by_sharing_array.piloc_mon',
3   [
4     Block
5       Name 'stack_fun.1',
6       [
7         Return
8         Empty
9       ],
10    Block

```

```

11     Name 'main.0',
12     [
13         StackMalloc
14         Num '2',
15         Ref
16         GlobalRead
17         Num '0',
18         Ref
19         GlobalRead
20         Num '6',
21     NewStackframe
22         Name 'stack_fun',
23         GoTo
24         Name 'addr@next_instr',
25     Exp
26         GoTo
27         Name 'stack_fun.1',
28     RemoveStackframe,
29     Return
30     Empty
31 ]
32 ]

```

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

```

1 SymbolTable
2 [
3     Symbol(
4     {
5         type qualifier:      Empty()
6         datatype:            FunDecl(VoidType('void'), Name('stack_fun'),
7                               ↪ [Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int'))),
8                               ↪ Name('param1')), Alloc(Writeable(), ArrayDecl([Num('3')], IntType('int')),
9                               ↪ Name('param2'))])
10        name:                Name('stack_fun')
11        value or address:     Empty()
12        position:             Pos(Num('1'), Num('5'))
13        size:                 Empty()
14    },
15    Symbol(
16    {
17        type qualifier:      Writeable()
18        datatype:            PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int')))
19        name:                Name('param1@stack_fun')
20        value or address:     Num('0')
21        position:             Pos(Num('1'), Num('21'))
22        size:                 Num('1')
23    },
24    Symbol(
25    {
26        type qualifier:      Writeable()
27        datatype:            PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int')))
28        name:                Name('param2@stack_fun')
29        value or address:     Num('1')

```

```

27     position:      Pos(Num('1'), Num('37'))
28     size:          Num('1')
29   },
30   Symbol(
31   {
32     type qualifier: Empty()
33     datatype:       FunDecl(VoidType('void'), Name('main'), [])
34     name:           Name('main')
35     value or address: Empty()
36     position:       Pos(Num('4'), Num('5'))
37     size:           Empty()
38   },
39   Symbol(
40   {
41     type qualifier: Writeable()
42     datatype:       ArrayDecl([Num('2'), Num('3')], IntType('int'))
43     name:           Name('local_var1@main')
44     value or address: Num('0')
45     position:       Pos(Num('5'), Num('6'))
46     size:           Num('6')
47   },
48   Symbol(
49   {
50     type qualifier: Writeable()
51     datatype:       ArrayDecl([Num('2'), Num('3')], IntType('int'))
52     name:           Name('local_var2@main')
53     value or address: Num('6')
54     position:       Pos(Num('6'), Num('6'))
55     size:           Num('6')
56   }
57 ]

```

Code 1.81: Symboltabelle für Call by Sharing für Arrays

```

1 File
2   Name './example_fun_call_by_sharing_array.reti_blocks',
3   [
4     Block
5       Name 'stack_fun.1',
6       [
7         # Return(Empty()),
8         LOADIN BAF PC -1
9       ],
10    Block
11      Name 'main.0',
12      [
13        # StackMalloc(Num('2')),
14        SUBI SP 2,
15        # Ref(GlobalRead(Num('0'))),
16        SUBI SP 1,
17        LOADI IN1 0,
18        ADD IN1 DS,
19        STOREIN SP IN1 1,
20        # 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             ],

```

```

10  Block
11      Name 'main.0',
12      [
13          StackMalloc
14              Num '2',
15          Assign
16              Tmp
17              Num '3',
18              GlobalRead
19              Num '0',
20          NewStackframe
21              Name 'stack_fun',
22              GoTo
23                  Name 'addr@next_instr',
24          Exp
25              GoTo
26                  Name 'stack_fun.1',
27          RemoveStackframe,
28          Return
29              Empty
30      ]
31 ]

```

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

```

1 File
2 Name './example_fun_call_by_value_struct.reti_blocks',
3 [
4     Block
5         Name 'stack_fun.1',
6         [
7             # Return(Empty()),
8             LOADIN BAF PC -1
9         ],
10    Block
11        Name 'main.0',
12        [
13            # StackMalloc(Num('2')),
14            SUBI SP 2,
15            # Assign(Tmp(Num('3')), GlobalRead(Num('0'))),
16            SUBI SP 3,
17            LOADIN DS ACC 0,
18            STOREIN SP ACC 1,
19            LOADIN DS ACC 1,
20            STOREIN SP ACC 2,
21            LOADIN DS ACC 2,
22            STOREIN SP ACC 3,
23            # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr'))),
24            MOVE BAF ACC,
25            ADDI SP 5,
26            MOVE SP BAF,
27            SUBI SP 5,
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     # 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).