
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	Motivation	6
1.1	PicoC und RETI	6
1.2	Problemstellung	6
2	Einführung	7
2.1	Compiler und Interpreter	7
2.1.1	T-Diagramme	7
2.2	Sprachen	7
2.2.1	Mehrdeutige Grammatiken	8
2.2.2	Präzidenz und Assoziativität	8
2.3	Lexikalische Analyse	8
2.4	Syntaktische Analyse	9
2.5	Code Generation	10
2.6	Fehlermeldungen	10
3	Implementierung	11
3.1	PicoC und RETI	11
3.2	Grammatiken	11
3.2.1	Umsetzung von Präzidenz	11
3.3	Lexikalische Analyse	12
3.3.1	Lark	12
3.4	Syntaktische Analyse	12
3.4.1	Lark	12
3.4.2	Early Algorithmus	12
3.5	Code Generation	12
3.5.1	Passes	12
3.5.2	Umsetzung von Pointern und Arrays	12
3.5.3	Umsetzung von Structs	12
3.5.4	Umsetzung von Funktionen	12
3.5.5	Umsetzung kleinerer Details	12
3.5.6	Fehlermeldungen	12
3.5.7	Error Handler	12
4	Ergebnisse und Ausblick	13
4.1	Funktionsumfang	13
4.2	Qualitätskontrolle	13
4.3	Kommentierter Kompilervorgang	13
4.4	Erweiterungsideen	13
A	Appendix	14
A.1	Konkrete und Abstrakte Syntax	14
A.2	Bedienungsanleitungen	14
A.2.1	PicoC-Compiler	14
A.2.2	Showmode	14
A.2.3	Entwicklertools	14

Abbildungsverzeichnis

Tabellenverzeichnis

3.1 Präzidenzregeln von PicoC	11
---	----

Definitionen

2.1	Compiler	7
2.2	Interpreter	7
2.3	T-Diagram	7
2.4	Chromsky Hierarchie	7
2.5	Grammatik	7
2.6	Reguläre Sprachen	7
2.7	Kontextfreie Sprachen	7
2.8	Ableitungsbaum	8
2.9	Mehrdeutige Grammatik	8
2.10	Assoziativität	8
2.11	Präzidenz	8
2.12	Pattern	8
2.13	Lexeme	8
2.14	Lexer (bzw. Scanner)	9
2.15	Parser	9
2.16	Konkrete Syntax	9
2.17	Derivation Tree	9
2.18	Abstrakte Syntax	9
2.19	Abstrakte Syntax Tree	9
2.20	Transformer	10
2.21	Visitor	10
2.22	Pass	10
3.1	Symboltabelle	12

1 Motivation

1.1 PicoC und RETI

1.2 Problemstellung

2 Einführung

2.1 Compiler und Interpreter

Definition 2.1: Compiler

Definition 2.2: Interpreter

2.1.1 T-Diagramme

Definition 2.3: T-Diagram

2.2 Sprachen

Definition 2.4: Chomsky Hierarchie

Definition 2.5: Grammatik

Definition 2.6: Reguläre Sprachen

Definition 2.7: Kontextfreie Sprachen

2.2.1 Mehrdeutige Grammatiken

Definition 2.8: Ableitungsbaum

Definition 2.9: Mehrdeutige Grammatik

2.2.2 Präzidenz und Assoziativität

Definition 2.10: Assoziativität

Definition 2.11: Präzidenz

2.3 Lexikalische Analyse

Die **Lexikalische Analyse** bildet üblicherweise die erste Ebene innerhalb der **Pipe Architektur** bei der Implementierung von Compilern. Die Aufgabe der lexikalischen Analyse ist vereinfacht gesagt, in einem Inputstring, z.B. dem Inhalt einer Datei, welche in **UTF-8** codiert ist, Folgen endlicher Symbole (auch **Wörter** genannt) zu finden, die bestimmte **Pattern** (Definition 2.12) matchen, die durch eine **reguläre Grammatik** spezifiziert sind.

Definition 2.12: Pattern

Beschreibung aller möglichen **Lexeme** einer Menge \mathbb{P}_T , die einem bestimmten **Token** T zugeordnet werden. Die Menge \mathbb{P}_T ist eine möglicherweise unendliche Menge von **Wörtern**, die sich mit den Regeln einer **regulären Grammatik** G_{Lex} einer **regulären Sprache** L_{Lex} beschreiben lassen^a, die für die Beschreibung eines **Tokens** T zuständig sind.^b

^aAls Beschreibungswerkzeug können aber auch z.B. reguläre Ausdrücke hergenommen werden.

^bWhat is the difference between a token and a lexeme?

Diese Folgen endlicher Symbole werden auch **Lexeme** (Definition 2.13) genannt.

Definition 2.13: Lexeme

Ein **Lexeme** ist ein **Wort** aus dem Inputstring, welches das **Pattern** für eines der **Token** T einer **Sprache** L_{Lex} matched.^a

^aWhat is the difference between a token and a lexeme?

Diese **Lexeme** werden vom **Lexer** im **Inputstring** identifiziert und **Tokens** T zugeordnet (Definition 2.14).

Definition 2.14: Lexer (bzw. Scanner)

Ein **Lexer** ist eine **rechtseindeutige** Funktion $lex : \Sigma^* \rightarrow (N \times V)^*$, welche ein **Wort** aus Σ^* auf ein **Token** T von einem **Token Name** N und einem **Token Value** V abbildet, falls diese Folge von Symbolen sich unter der **regulären Grammatik** G_{Lex} der **regulären Sprache** L_{Lex} ableiten lässt.^a

^alecture-notes-2021.

Die

Die vom **Lexer** identifizierten **Token** der **Sprache** werden

Die **reguläre Grammatik** G_{Lex} , die zur Beschreibung der Token T einer regulären Sprache L_{Lex} verwendet wird, ist üblicherweise **regulär**, da ein typischer **Lexer** immer nur **ein oder wenige Symbole** vorausschaut^a, unabhängig davon, was für Symbole davor aufgetaucht sind. Die übliche Implementierung eines **Lexers** merkt sich nicht, was für Symbole davor aufgetaucht sind, der **Kontext** in dem ein Symbol auftaucht ist also **nicht wichtig**.

^aMan nennt das auch einem **Lookahead** von 1 oder k

2.4 Syntaktische Analyse

Ein

Der **Parser** nutzt **Token** T als Wegweiser, um herauszufinden,

Definition 2.15: Parser

^a

^aWhat is the difference between a token and a lexeme?

Definition 2.16: Konkrete Syntax**Definition 2.17: Derivation Tree****Definition 2.18: Abstrakte Syntax****Definition 2.19: Abstrakte Syntax Tree**

Definition 2.20: Transformer**Definition 2.21: Visitor**

2.5 Code Generation

Definition 2.22: Pass

2.6 Fehlermeldungen

3 Implementierung

3.1 PicoC und RETI

ASTNode

3.2 Grammatiken

3.2.1 Umstzung von Präzidenz

Die PicoC Sprache hat dieselben Präzidenzregeln implementiert, wie die Sprache C¹. Die Präzidenzregeln von PicoC sind in Tabelle 3.2.1 aufgelistet.

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

Tabelle 3.1: Präzidenzregeln von PicoC

¹C *Operator Precedence* - cppreference.com.

3.3 Lexikalische Analyse

3.3.1 Lark

3.4 Syntaktische Analyse

3.4.1 Lark

3.4.2 Early Algorithmus

3.5 Code Generation

3.5.1 Passes

PicoC-Shrink Pass

PicoC-Blocks Pass

PicoC-Mon Pass

Definition 3.1: Symboltabelle

RETI-Blocks Pass

RETI-Patch Pass

RETI Pass

3.5.2 Umsetzung von Pointern und Arrays

3.5.3 Umsetzung von Structs

3.5.4 Umsetzung von Funktionen

3.5.5 Umsetzung kleinerer Details

3.5.6 Fehlermeldungen

3.5.7 Error Handler

4 Ergebnisse und Ausblick

4.1 Funktionsumfang

4.2 Qualitätskontrolle

4.3 Kommentierter Kompiliervorgang

4.4 Erweiterungsideen

A Appendix

A.1 Konkrete und Abstrakte Syntax

A.2 Bedienungsanleitungen

A.2.1 PicoC-Compiler

A.2.2 Showmode

A.2.3 Entwicklertools

Literatur

Online

- *C Operator Precedence* - *cppreference.com*. URL: https://en.cppreference.com/w/c/language/operator_precedence (besucht am 27.04.2022).
- *lecture-notes-2021*. 20. Jan. 2022. URL: <https://github.com/Compiler-Construction-Uni-Freiburg/lecture-notes-2021/blob/56300e6649e32f0594bbbd046a2e19351c57dd0c/material/lexical-analysis.pdf> (besucht am 28.04.2022).
- *What is the difference between a token and a lexeme?* NewbeDEV. URL: <http://newbedev.com/what-is-the-difference-between-a-token-and-a-lexeme> (besucht am 17.06.2022).