Albert Ludwigs Universität Freiburg

Technische Fakultät

PicoC-Compiler

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

Bachelorarbeit

Abgabedatum: 28th April 2022

Author:
Jürgen Mattheis

Gutachter:
Prof. Dr. Scholl

Betreung:
M.Sc. Seufert

Eine Bachelorarbeit am Lehrstuhl für
Betriebssysteme

ERKLÄRUNG

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

Inhaltsverzeichnis

1 Motivation **7**
- 1.1 PicoC und RETI . . . 7
- 1.2 Problemstellung . . . 7

2 Einführung **8**
- 2.1 Compiler und Interpreter . . . 8
  - 2.1.1 T-Diagramme . . . 8
- 2.2 Grammatiken . . . 8
  - 2.2.1 Chromsky Hierarchie . . . 8
  - 2.2.2 Präzidenz und Assoziativität . . . 8
  - 2.2.3 Mehrdeutige Grammatiken . . . 8
  - 2.2.4 Ableitungsbaum . . . 8
  - 2.2.5 Linksrekursiv und Rechtrekursiv . . . 8
- 2.3 Lexikalische Analyse . . . 8
- 2.4 Syntaktische Analyse . . . 9
- 2.5 Code Generation . . . 10
  - 2.5.1 Passes . . . 10
- 2.6 Fehlemeldungen . . . 10
  - 2.6.1 Kategorien von Fehlermeldungen . . . 10

3 Implementierung **11**
- 3.1 Grammatiken . . . 11
  - 3.1.1 PicoC . . . 11
  - 3.1.2 RETI . . . 12
  - 3.1.3 Mehrdeutigkeit . . . 12
  - 3.1.4 Präzidenz und Assoziativität . . . 12
  - 3.1.5 Linksrekursivität . . . 12
- 3.2 Lexikalische Analyse . . . 12
  - 3.2.1 Lark . . . 12
  - 3.2.2 LL(1) Recursive-Descent Lexer . . . 12
- 3.3 Syntax Analyse . . . 12
  - 3.3.1 Lark . . . 12
  - 3.3.2 Normalized Heterogeneous ASTNode . . . 12
  - 3.3.3 Early Algorithmus . . . 12
  - 3.3.4 Visitor und Transformer . . . 12
- 3.4 Code Generation . . . 12
  - 3.4.1 Symbol Table for Nested Scopes . . . 12
  - 3.4.2 PicoC-Shrink Pass . . . 12
  - 3.4.3 PicoC-Blocks Pass . . . 12
  - 3.4.4 PicoC-Mon Pass . . . 12
  - 3.4.5 RETI-Blocks Pass . . . 12
  - 3.4.6 RETI-Patch Pass . . . 12
  - 3.4.7 RETI Pass . . . 12
- 3.5 Fehlermeldungen . . . 12
  - 3.5.1 Error Handler . . . 12

4 Ergebnisse und Ausblick **13**
4.1 Vergleich eigener Parser und Lark 13
4.2 Beispielhafte Ausführung 13
4.3 Qualitätskontrolle 13
4.4 Erweiterungsideen 13

A Appendix **15**

Abbildungsverzeichnis

Tabellenverzeichnis

3.1 Präzidenzregeln von PicoC 11

Definitionen

2.1 Pattern 8
2.2 Lexeme 8
2.3 Lexer (bzw. Scanner) 9
2.4 Parser 9
2.5 Konkrette Syntax 9
2.6 Derivation Tree 9
2.7 Abstrakte Syntax 10
2.8 Abstrakte Syntax Tree 10
2.9 Transformer 10
2.10 Visitor 10

1 Motivation

1.1 PicoC und RETI

1.2 Problemstellung

2 Einführung

2.1 Compiler und Interpreter

2.1.1 T-Diagramme

2.2 Grammatiken

2.2.1 Chromsky Hierarchie

2.2.2 Präzidenz und Assoziativität

2.2.3 Mehrdeutige Grammatiken

2.2.4 Ableitungsbaum

2.2.5 Linksrekursiv und Rechtrekursiv

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 2.1) matchen, die durch eine **reguläre Grammatik** spezifiziert sind.

Definition 2.1: 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]

[a] Als Beschreibungswerkzeug können aber auch z.B. reguläre Ausdrücke hergenommen werden.
[b] *What is the difference between a token and a lexeme?*

Diese Folgen endlicher Symoble werden auch **Lexeme** (Definition 2.2) genannt.

Definition 2.2: Lexeme

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

[a] *What is the difference between a token and a lexeme?*

Diese **Lexeme** werden vom **Lexer** im **Inputstring** identifziert und **Tokens** T zugeordnet (Definition 2.3).

Definition 2.3: Lexer (bzw. Scanner)

*Ein **Lexer** ist eine **rechtseindeutige** Funktion $lex : \sum^* \rightharpoonup (N \times V)^*$, welche ein **Wort** aus $\sum^*$ 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]

[a] *lecture-notes-2021*.

Die **Lekikalische Analyse** filtert jegliche für die Weiterverarbeitung unwichtige Symbole, wie Leerzeichen ␣, Newline `\n`[1] und Tabs `\t` aus dem Inputstring heraus und

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**.

[a] Man nennt das auch einem **Lookahead** von 1 oder k

2.4 Syntaktische Analyse

Dadurch, dass der **Lexer** in der **Lekikalischen Analyse**, wie in 2.3 beschrieben alle **Token** `T` der Sprache $L_{\texttt{Lex}}$ im Inutstring identifziert hat, ist die Weiterverarbeitung durch die **Syntaktische Analyse**, mittels eines **Parsers** deutlich einfacher.

Ein

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

Definition 2.4: Parser

a

[a] *What is the difference between a token and a lexeme?*

Definition 2.5: Konkrette Syntax

Definition 2.6: Derivation Tree

[1] In Unix Systemen wird für Newline das ASCII Symbol **line feed**, in Windows hingegen die ASCII Symbole **carriage return** und **line feed** nacheinander verwendet. Das wird aber meist durch die verwendete Porgrammiersprache, die man zur Inplementierung des Lexers nutzt wegabstrahiert.

Definition 2.7: Abstrakte Syntax

Definition 2.8: Abstrakte Syntax Tree

Definition 2.9: Transformer

Definition 2.10: Visitor

2.5 Code Generation

2.5.1 Passes

2.6 Fehlemeldungen

2.6.1 Kategorien von Fehlermeldungen

3 Implementierung

3.1 Grammatiken

3.1.1 PicoC

Die PicoC Sprache hat dieselben Präzidenzregeln implementiert, wie die Sprache C[1]. Die Präzidenzregeln von PicoC sind in Tabelle 3.1.1 aufgelistet.

Präzidenz	Operator	Beschreibung	Assoziativität

1	`a()`
	`a[]`
	`a.b`
		Funktionsaufruf
		Indexzugriff
		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`	Logiches 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

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

3.1.2 RETI

3.1.3 Mehrdeutigkeit

3.1.4 Präzidenz und Assoziativität

3.1.5 Linksrekursivität

3.2 Lexikalische Analyse

3.2.1 Lark

3.2.2 LL(1) Recursive-Descent Lexer

3.3 Syntax Analyse

3.3.1 Lark

3.3.2 Normalized Heterogeneous ASTNode

3.3.3 Early Algorithmus

Das LL(k) Recursive-Descent Parser[2] Pattern ist

3.3.4 Visitor und Transformer

3.4 Code Generation

3.4.1 Symbol Table for Nested Scopes

3.4.2 PicoC-Shrink Pass

3.4.3 PicoC-Blocks Pass

3.4.4 PicoC-Mon Pass

3.4.5 RETI-Blocks Pass

3.4.6 RETI-Patch Pass

3.4.7 RETI Pass

3.5 Fehlermeldungen

3.5.1 Error Handler

[2] Parr, *Language Implementation Patterns*.

4 Ergebnisse und Ausblick

4.1 Vergleich eigener Parser und Lark

4.2 Beispielhafte Ausführung

4.3 Qualitätskontrolle

4.4 Erweiterungsideen

Test, ob der Spellchecker funktioniert Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor.

Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

A Appendix

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).