EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Spezielle Kapitel der Praktischen Informatik: Compilerbau, INF3199A

Dokumentation

für die Prüfung zum

Master / Bachelor of Science

des Studiengangs Informatik

an der Universität Tübingen

von

Arwed Mett, Steffen Lindner, Robin Heinz, Pavel Karasik, Florian Engel

Matrikelnummer	4170745, 3912506, 3971071, 3991213, 3860700
Semester	WS17/18
Betreuer	Prof. Dr. Martin Plümicke
Gutachter	Prof. Dr. Martin Plümicke

Inhaltsverzeichnis

1 Einleitung

Im folgenden werden die Ergebnisse des Compilerbau Projektes der Vorlesung „Spezielle Kapitel der Praktischen Informatik: Compilerbau, INF3199A“ dokumentiert. Ziel war es einen Java Compiler zu entwickeln der eine Teilmenge der Programmiersprache Java [3] zu Java Bytecode[2] compiliert. Der Bytecode kann anschließend mit einer Java Virtual Machine (JVM)[1] ausgeführt werden. Des weiteren wurde ein Testframework entwickelt um die Teilfunktionen des Compilers zu validieren.

Dieses Dokument dient als Dokumentation des Compilers, um einen groben Überblick der Struktur des Projektes zu bekommen, den Compiler zu bauen, das Testframework auszuführen und zur Dokumentation der Aufgabenverteilung.

Das Projekt wurde innerhalb des Wintersemesters 17/18 an der Universität Tübingen in einer Gruppe von 5 Leuten implementiert. Hierfür wurde das Projekt in Parser/Lexer, Typchecker, Codegenerierer und Tester unterteilt.

2 Installation

Im folgenden werden die Schritte zur Installation des Compilers beschrieben.

2.1 Voraussetzungen

Es werden folgende Programme benötigt.

- GHC >= v8.0.1
- Cabal >= v1.24.0
- JRE >= 1.8
- Git >= 2.14.3

2.2 Erstellungsprozess

Um den Java Compiler zu bauen muss zuerst das Repository heruntergeladen werden. Hierzu kann git verwendet werden. Falls Git nicht auf dem Rechner installiert ist, kann das Repository alternativ auch über das WebUI von Github heruntergeladen werden.

```
1 git clone \
2   -b 'v1.0.0' --single-branch --depth 1 \
3   git@github.com:Pfeifenjoy/compilerbau-WS17-18.git
```

Listing 2.1: Download des Projektes

Dadurch wird das Repository in den Ordner „compilerbau-WS17-18“ geladen.

Anschließend muss in den Ordner des Quellcodes gewechselt werden. Dieser kann mit Hilfe von Cabal erstellt und installiert werden. Durch `cabal install` wird das Kommando `jc` installiert. Falls sich `jc` nicht im Pfad befindet kann es alternativ mit `cabal exec jc` ausgeführt werden.

```
cd compilerbau-WS17-18/project
cabal configure
cabal build
cabal install
```

Listing 2.2: Bauen des jc Kommandos

2.3 Ausführen des Testframeworks

Das Testframework ist eine Haskell Testsuite, welche in Cabal erst aktiviert werden muss. Nachdem, oder während, das Project installiert wurde wie in Listing 2.2 beschrieben, müssen folgende Befehle ausgeführt werden.

```
cabal configure --enable-tests
```

Listing 2.3: Aktivierung des Testframeworks

Anschließend kann mit dem Befehl `cabal test` das Testframework ausgeführt werden oder mit `cabal repl test-core` in den interaktiven Modus gesprungen werden.

```
cabal repl test-core
> main -- Anzeiger der Testinformationen
```

Listing 2.4: Interaktiver Modus: Testframework

Mithilfe der main Funktion können dann die Testfälle ausgeführt werden und deren Ergebnisse angezeigt werden.

2.4 Bedienung des Compilers

Das Programm `jc` bekommt als Argument eine Liste von Java Quelldateien, und generiert daraus Class-Files. Mithilfe des flag `-l LOG`, e.g. `cabal exec jc -- -l log *.java`, wird eine log-Datei erstellt, welche die abstrakte Syntax etc. beinhaltet. Eine genauere Beschreibung der Abstrakten Syntax ist unter `compilerbau-WS17-18/project/src/ABSTree.hs` zu finden.

Der Hilfetext aus Listing 2.5 kann mittels `cabal exec jc -- -h` generiert werden.

```
usage : jc [source-files...] [-l  LOG] [-v] [-h] [--version]

mandatory arguments:
 source-files                  Path to the sources which are
                               compiled.
```

```
6
7 optional arguments:
8  -l, --log  LOG                Specify log file.
9  -v, --verbose                 Show extra information
10 -h, --help                    show this help message and exit
11 --version                     print the program version and exit
```

Listing 2.5: Hilfe jc

3 Architektur

Das folgende Kapitel ist eine Übersicht der Architektur des Projektes. Es hat **nicht** den Anspruch eine Technische Dokumentation der Architektur zu sein, sondern einen Überblick der Komponenten des Projektes zu verschaffen.

Bemerkung: Im folgenden werden alle Pfade relativ zu `compilerbau-WS17-18/project` angegeben.

Insgesamt besteht das Projekt aus drei Unterprogrammen. Um maximale Widerverwendung zu gewährleisten wurden die Kernfunktionen in einer Bibliothek (Library) zusammengefasst. Diese Bibliothek ist unter `src` zu finden. Sie besteht wiederum aus folgenden Komponenten:

1. Lexer
2. Parser
3. Typchecker
4. Codegenerierer

Des weiteren wurde ein Terminal User Interface (TUI) entwickelt, womit die Funktionen des Compilers ausgeführt werden können. Da die Funktionen in der Bibliothek implementiert sind, linked das TUI gegen die Bibliothek. Es basiert auf der argparser Bibliothek [1] und ist unter `cli/main.hs` zu finden.

Das Letzte Unterprogramm ist das Testframework, welches unter `test` zu finden ist und gleichermaßen wie das TUI gegen die Bibliothek linked.

3.1 Implementierung der Hauptbibliothek

Ziel der Entwicklung der Hauptbibliothek war es einzelne Funktionen für das Lexen, Parsen, etc. zur Verfügung zu stellen. Als gemeinsame Datenstruktur wurde eine abstrakte Syntax verwendet. Die abstrakte Syntax ist unter `src/ABSTree.hs` zu finden. Des weiteren gibt es eine `Lexer.lex` methode, welche eine Liste an Tokens erzeugt. Anschließend erstellt

[1] https://hackage.haskell.org/package/argparser-0.3.4/docs/System-Console-ArgParser.html

der Parser mit `Parser.parse` aus dieser Liste eine abstrakte Syntax. Allgemein ist eine abstrakte Syntax eine Liste von Klassen. Da Java eine typisierte Sprache ist transformiert der Typchecker die abstrakte Syntax zu einer getypten abstrakten Syntax mittels `Typechecker.checkTypes`. Der Codegenerator erstellt dann mit `Codegen.GenerateClassFile.genClass` anhand der getypten abstrakten Syntax den Bytecode.

Die Implementierung des jeweiligen Schrittes ist unter den Ordnern aus Tabelle 3.1 zu finden.

Komponente	**Pfad**
Lexer	`src/Lexer/Lexer.x`
Parser	`src/Parser/Parser.y`
Typchecker	`src/TypeChecker.hs`
Codegenerator	`src/Codegen/*`

Tabelle 3.1: Quellcode pro Komponente

3.2 Implementierung des Testframework

4 Organisation

4.1 Teammitglieder

4.2 Aufgabenverteilung

Abkürzungsverzeichnis

JVM	Java Virtual Machine
TUI	Terminal User Interface

Literatur

[1] *Kostenloser Java-Download*. URL: https://java.com/de/download/ (besucht am 16. 02. 2018).

[2] Tim Lindholm u. a. *The Java Virtual Machine Specification, Java SE 8 Edition*. 1st. Addison-Wesley Professional, 2014. ISBN: 013390590X, 9780133905908.

[3] Prof. Dr. Martin Plümicke. „Compilerbau Prüfungsleistung“. In: (2017). URL: http://www2.ba-horb.de/~pl/Compilerbau_WS2017_18/Compiler_Aufgabe.pdf.