HUMBOLDT-UNIVERSITÄT ZU BERLIN

Entwicklung eines Compilers in Rust zu Lehrzwecken

Daniel Bucher · 8. November 2023

Übersicht

- Zielsetzung & Motivation
- Entwicklung des Compilers
- Präsentation der Aufgaben

Zielsetzung & Motivation

Zielsetzung der Arbeit

- Erstellen eines Compilers/Interpreters in Rust
  - Bietet die Grundlage für Übungsaufgaben
  - Verschiedene Konzepte des Compilerbaus sollen abgedeckt werden
  - Der Aufbau soll für Lernende verständlich sein
- Ableitung von 5 Übungsaufgaben aus dem Compiler
  - Abdeckung relevanter Felder des Compilerbaus
  - Inhaltlich und strukturell aufbauende Aufgaben
  - Auch für Rust-Anfänger geeignet

Zielsetzung & Motivation

Motivation

- Compilerbau wenden theoretische Informatik praktisch an
- Bisherige Ansätze bieten Verbesserungspotenzial
  - Ansatz in C ist seit mehr als 15 Jahren im Einsatz
  - Konvertierter in Rust erstmals im SOSE 2022 genutzt
- Ausnutzen des Potenzials von Rust

Zielsetzung & Motivation

Rust als Programmiersprache

- Hohe Beliebtheit bei Programmierern
  - Seit 2016 beliebteste Sprache im *Stack Overflow Developer Survey*
- Compiler-Überwachte Speicherverwaltung
  - Referenzen als sichere Pointer
  - *Ownership* und *Borrowing* für sichere Nebenläufigkeit
- Direkte Integration von Design-Pattern
- Einfache Projektverwaltung mit `Cargo`

Entwicklung des Compilers

Entwicklung des Compilers

Compiler - Struktur

- Aufbau
  - Lexikalische Analyse
  - Syntaktische Analyse
  - Semantische Analyse
  - Codeoptimierung
  - Codegenerierung / Interpreter

Entwicklung des Compilers

Interpreter - Umsetzung

- Zielsprache: C1
- Aufbau
  - Lexikalische Analyse (Bibliothek: logos)
  - Syntaktische Analyse (Bibliothek: LALRPOP)
  - Semantische Analyse (Visitor Design Pattern)
  - ~~Codeoptimierung~~
  - ~~Codegenerierung/~~ Interpreter (Visitor Design Pattern)

Entwicklung des Compilers

Herausforderung: Mehrdeutige Grammatik

- Problemstellung: LALRPOP erlaubt keine mehrdeutig ableitbaren Grammatiken

Entwicklung des Compilers

Herausforderung: Mehrdeutige Grammatik

- Problemstellung: LALRPOP erlaubt keine mehrdeutig ableitbaren Grammatiken
  - Bekannt als: Dangling Else

```
1  void main() {
2      if (true)
3          if (true)
4              printf("If");
5          else
6              printf("Else");
7  }
```

Entwicklung des Compilers

Lösungsansatz: Dangling Else

- Ziel: Das dangling Else entfernen, aber die Grammatik nicht einschränken
- Ansatz: Formulierung einer Bindungsvorschrift:

Der Block eines `if`-Statements **mit** Else-Block darf rekursiv nicht zu einem `if`-Statement **ohne** Else-Block abgeleitet werden.

Entwicklung des Compilers

Auflösung der Mehrdeutigkeit 1

Die Regel zur Ableitung des IF-Statements:

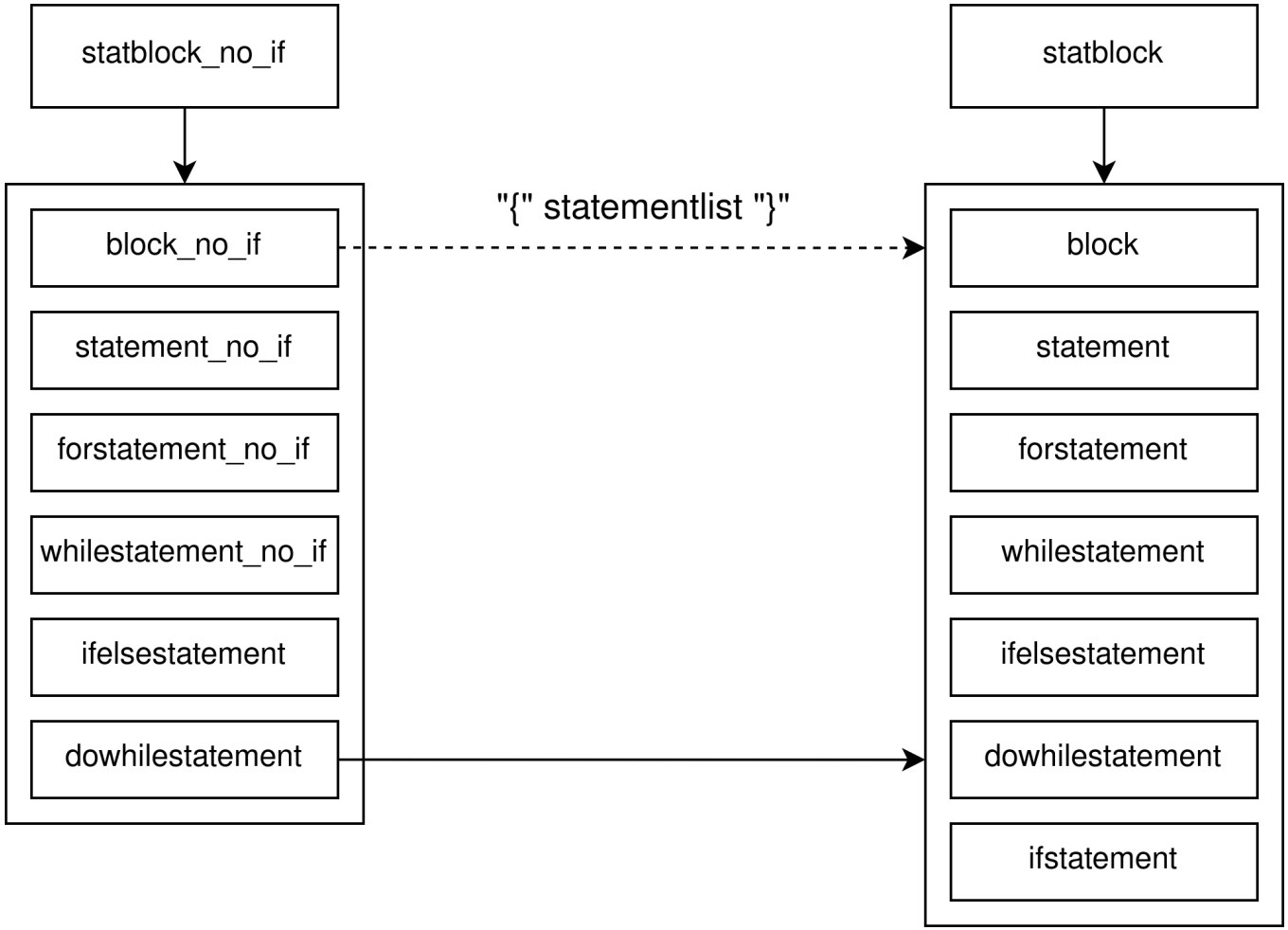
$$ifstatement ::= < KW\_IF > \text{"("} assignment \text{")"} statblock (< KW\_ELSE > statblock)?$$

Umgeformte Regel mit aufgelöster Mehrdeutigkeit:

$$\begin{aligned} ifstatement ::= & < KW\_IF > \text{"("} assignment \text{")"} statblock \\ & | < KW\_IF > \text{"("} assignment \text{")"} stateblock\_no\_if < KW\_ELSE > statblock \end{aligned}$$

Entwicklung des Compilers

Auflösung der Mehrdeutigkeit 2



Entwicklung des Compilers

Herausforderung: Strukturierung

- Aktueller Schritt: Anwendung der semantischen Analyse auf dem Syntaxbaum vom Parser
- Vorgehen: Syntaxbaum soll alle Knoten durchlaufen und dabei eine neue Struktur erzeugen

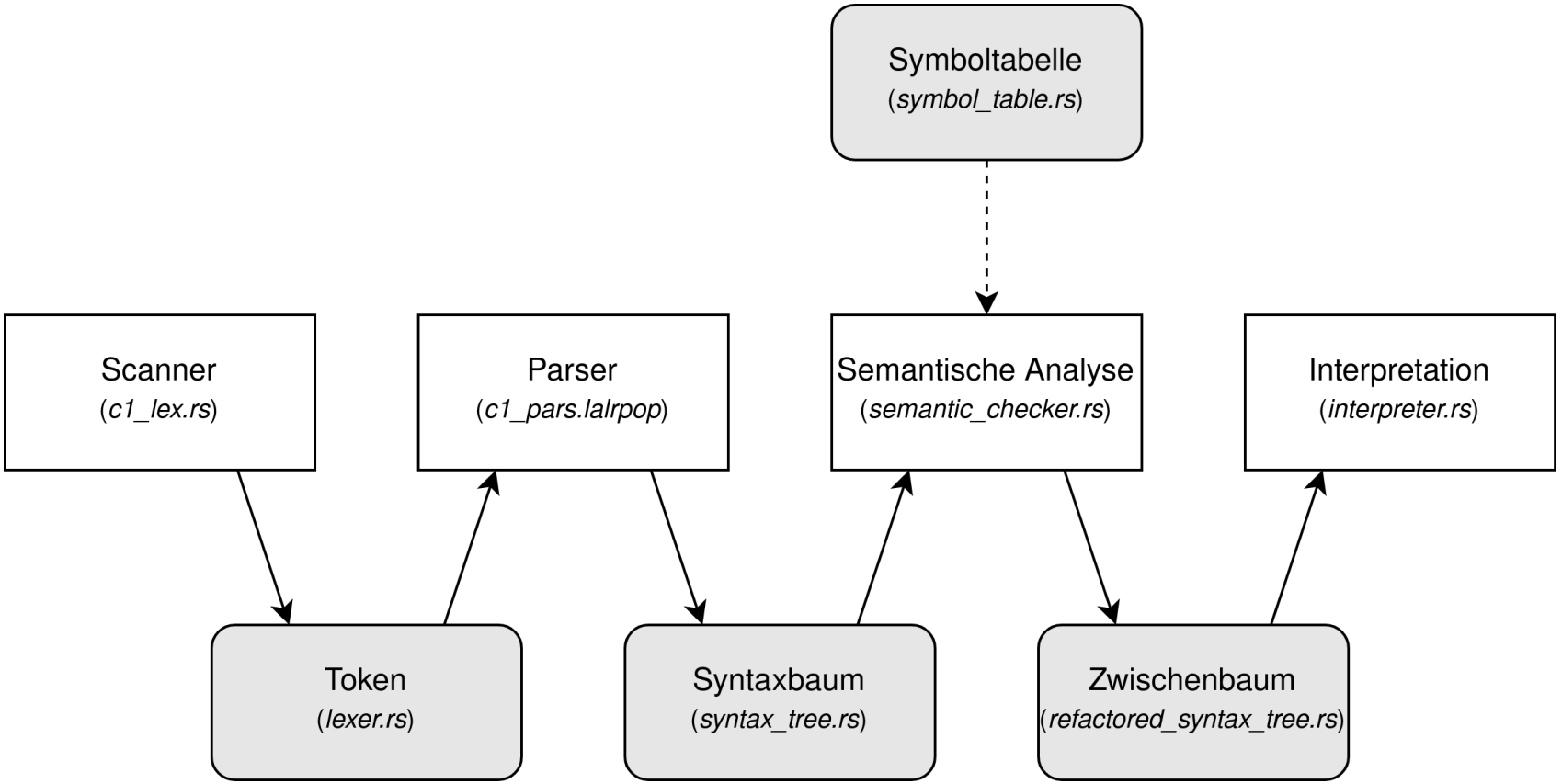
Entwicklung des Compilers

Herausforderung: Strukturierung

- Aktueller Schritt: Anwendung der semantischen Analyse auf dem Syntaxbaum vom Parser
- Vorgehen: Syntaxbaum soll alle Knoten durchlaufen und dabei eine neue Struktur erzeugen
- Idee: Nutzung des Visitor Design Pattern
  - Aufgrund der Funktionen von Rust ist die Verwendung des Visitor Pattern nativ möglich.

Entwicklung des Compilers

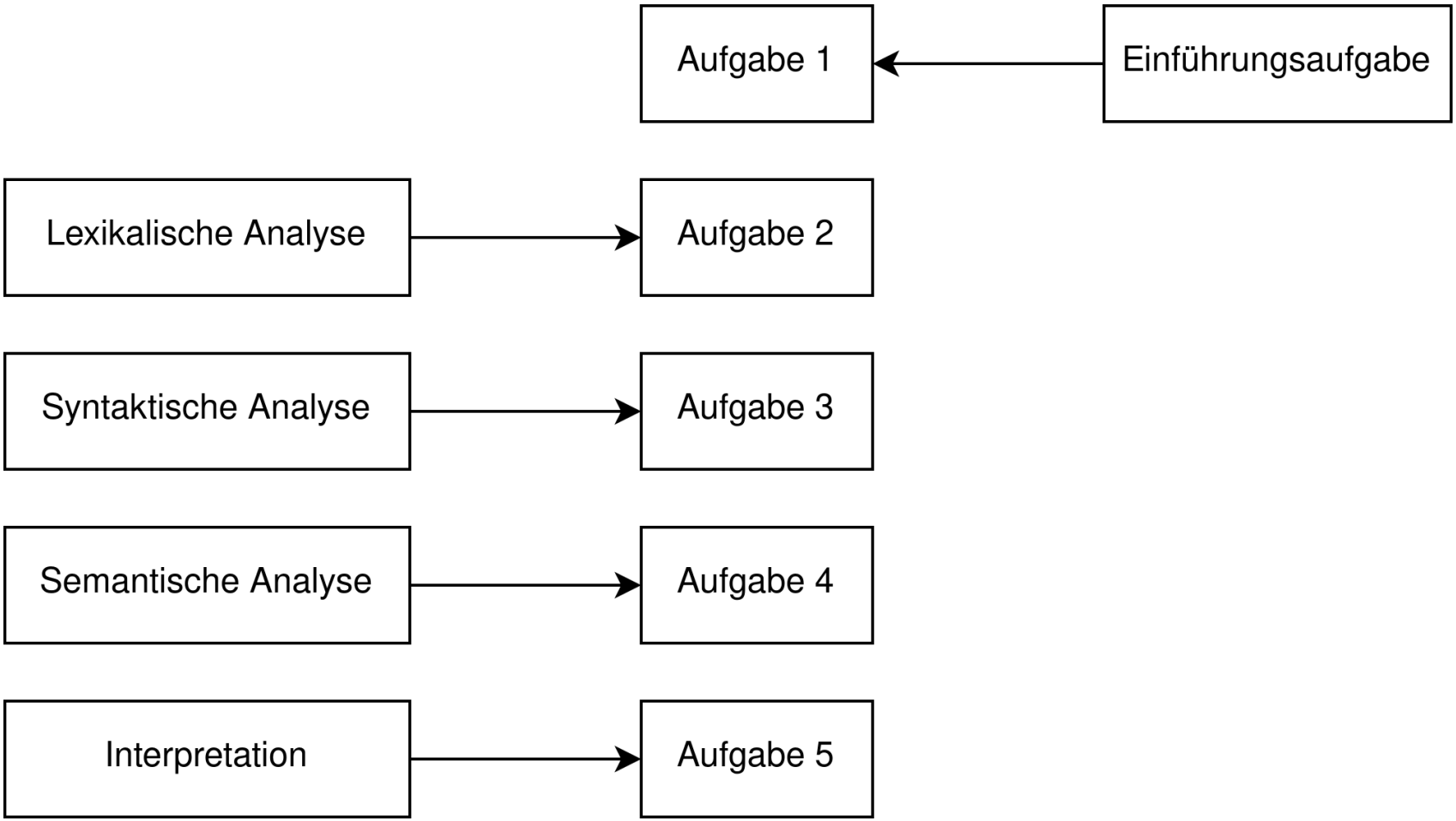
Zustand des Compilers



Präsentation der Aufgaben

Präsentation der Aufgaben

Aufteilung der Aufgaben



Präsentation der Aufgaben

Generelle Struktur in allen Aufgaben

- Einheitlich strukturierte Aufgabenstellungen
- Vorgabe eines Rust-Projekts (`cargo`) mit den vorgegebenen Strukturen und Dateien
- Mitgeliefert werden jeweils mehrere Testfälle zur Selbstkontrolle und zur Bewertung

Präsentation der Aufgaben

Aufgabe 1 - Einführung

- Lernziel: Vertiefung des Umgangs mit Rust & Anwenden des Visitor Pattern in Rust
- Aufgabe:
  - Vervollständigung eines Rechners für Grundrechenaufgaben mit Variablen
  - Repräsentation der Aufgaben als Baumstruktur
  - Implementation zweier Visitor als Rechner und Pretty-Printer
  - Ähnlichkeit zu einem Interpreter

Präsentation der Aufgaben

Aufgabe 2 - Lexer

- Lernziel: Umgang mit dem Lexer `logos` & Konstruktion von Tokens mit regulären Ausdrücken
- Aufgabe:
  - Vervollständigung der Eingabe des Lexers
  - Erstellung von Tokens aus Keywords und regulären Ausdrücken
  - Konstruktion der regulären Ausdrücke aus textueller Beschreibung

Präsentation der Aufgaben

Aufgabe 3 - Parser

- Lernziel: Umgang mit dem Parsergenerator `LALRPOP` & Auseinandersetzung mit mehrdeutigen Grammatiken
- Aufgabe:
  - Eintragung der Grammatikregeln in den Parsergenerator
  - Auflösung des *dangling else* mit Hilfestellungen
  - Konstruktion des Syntaxbaums beim Parsen der Tokens

Präsentation der Aufgaben

Aufgabe 4 - Semantische Analyse

- Lernziel: Anwendung der semantischen Analyse mit Nutzung einer Symboltabelle & vertiefender Umgang mit dem Visitor-Pattern
- Aufgabe:
  - Ausfüllen der Visitor-Methoden mit semantischer Prüfung
  - Bereitgestellt wird die Symboltabelle als Hilfsstruktur
  - Konstruktion einer Zwischenstruktur aus dem Syntaxbaum

Präsentation der Aufgaben

Aufgabe 5 - Interpretation

- Lernziel: Verständnis des Zusammenspiels der einzelnen Komponenten eines vollständigen Interpreters
- Aufgabe:
  - Interpretation auf Basis der Zwischenstruktur aus der semantischen Analyse
  - Mitgeliefert wird eine stackbasierte Speicherstruktur (ähnlich zu C)
  - Wenige Einschränkungen für die Bearbeitung

Präsentation der Aufgaben

Erreichte Verbesserungen

- Besserer Fokus aus Compilerbau als im C-Ansatz
- Verbesserte Ausnutzung von Rust-Features als im Rust-Ansatz
- Aufbauende Struktur zum Verständnis der Zusammenhänge im Compilerbau

Vielen Dank für die Aufmerksamkeit

Gibt es Fragen?