



The Conversion of Source Code to Machine Code

Silas Groh, Mik Müller

17. März 2023

Carl-Fuhlrott-Gymnasium

Einstieg & Motivation

- Computerprogrammer werden oft in speziell entwickelten Programmiersprachen verfasst
- Vorteile eines hohen Abstraktionsgrades [Dan05, S. 9]:
 - Entwicklung ist einfacher und schneller
 - Programme sind portabel
 - Instandhaltung ist einfacher

```
1 fn main() {  
2     foo(2);  
3 }  
4  
5 fn foo(n: int) {  
6     let mut m = 3;  
7     exit(n + m);  
8 }
```

Übersetzung



```
1 ; RISC-V binary  
2 00000000 457f 464c 0102  
3 00000010 0002 00f3 0001  
4 00000020 0040 0000 0000  
5 00000030 0005 0000 0040  
6 00000040 0001 0000 0005  
7 00000050 0000 0001 0000  
8 00000060 0124 0000 0000  
9 00000070 1000 0000 0000  
10 00000080 00b0 0000 0000
```

- Programme sollten einfach zu schreiben sein

⇒ Ein Computer muss diese jedoch auch *einfach* verarbeiten

- Man unterscheidet zwischen *Compilern* und *Interpretern*
- Ein Compiler (auch *Übersetzer*) übersetzt die Sprache in ein zielspezifisches Format, sodass ein Computer dieses verstehen kann
- Ein Interpreter führt das Programm direkt aus, ohne es vorher zu bearbeiten



Abbildung 1 – Etappen der Übersetzung.

[Wir05, S. 6–7]

Etappen der Übersetzung (angepasst)



Abbildung 2 – Etappen der Übersetzung (angepasst).

[Wir05, S. 6–7]

```
1  fn main() {  
2      exit(fib(10));  
3  }  
4  
5  fn fib(n: int) -> int {  
6      if n < 2 {  
7          n  
8      } else {  
9          fib(n - 2) + fib(n - 1)  
10     }  
11 }
```

1.1 – Berechnung von Fibonaccizahlen in rush

Tabelle 1 – Die wichtigsten Fähigkeiten von rush.

Bezeichnung	Beispiel
Schleife	<code>loop { }</code>
while-Schleife	<code>while x < 5 { }</code>
for-Schleife	<code>for i = 0; i < 5; i += 1 { }</code>
if-Verzweigung	<code>if true { } else { }</code>
Funktionsdefinition	<code>fn foo(n: int) { }</code>
infix-expression	<code>1 + n; 5 ** 2</code>
prefix-expression	<code>!false; -n</code>
let-statement	<code>let mut answer = 42</code>
cast-expression	<code>42 as float</code>

Tabelle 2 – Datentypen in rush.

Bezeichnung	Instanziierung einer Variable
'int'	<code>let a: int = 0;</code>
'float'	<code>let b: float = 3.14;</code>
'bool'	<code>let c: bool = true;</code>
'char'	<code>let d: char = 'a';</code>
'()'	<code>let e: () = main();</code>
'!'	<code>let f = exit(42);</code>

- Im Commit 4de569a umfasste das Projekt 17456 Zeilen Programmtext¹
- Das Projekt enthält einen Lexer, einen Parser, fünf Compiler und einen Interpreter

¹Leerzeilen und Kommentare werden nicht gezählt.

Tabelle 3 – Zeilen Programmtext pro Komponente.

Komponente	Zeilen Programmtext
Lexer / Parser	2737
Tree-walking interpreter	578
VM compiler / runtime	1281
WASM compiler	1584
LLVM compiler	1450
RISC-V compiler	2275
x86 compiler	2751

Lexikalische & Syntaktische Analyse

- Gruppieren des Programmtextes in Tokens
- Analyse der Syntax des Programms
- Generation eines abstrakten Syntaxbaums
- Festlegen der formalen Regeln in Form einer Grammatik

```
1 Expression = Term , { ( '+' | '-' ) , Term } ;
2 Term       = Factor , { ( '*' | '/' ) , Factor } ;
3 Factor      = ( integer
4               | '(' , Expression , ')' ) , [ '**' , Factor ] ;
5 integer     = { '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8'
               ↪ | '9' }- ;
```

1.2 – Ein Beispiel für eine kontextfreie Grammatik (EBNF).

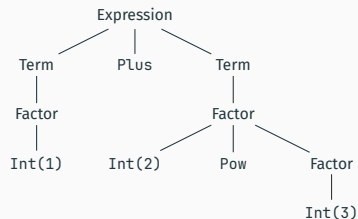


Abbildung 3 – Abstrakter Syntaxbaum für '1+2**3'.

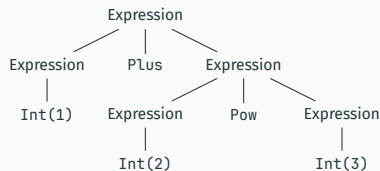


Abbildung 4 – Abstrakter Syntaxbaum für '1+2**3', erstellt durch Pratt-Parsing.

Interpreter

TODO: Write this

Kompilierung zu high-level Architekturen

TODO: Write this

Kompilierung zu low-level Architekturen

TODO: Write this

Finale Anmerkungen & Fazit

TODO: Write this

Programmblockverzeichnis

Literatur



Sivarama P Dandamudi. *Guide to RISC processors*. Ottawa, Canada: Springer International Publishing, Feb. 2005. ISBN: 0-387-21017-2.



Niklaus Wirth. *Compiler Construction*. Zürich, 2005. ISBN: 0-201-40353-6.