

Vorlesung (SoSe2018)

Teil 2: Einführung




Compilerbau Einführung

Vorlesung des BA-Studiums
Prof. Johann Christoph Freytag, Ph.D.
Institut für Informatik, Humboldt-Universität zu Berlin
SoSe2018

Handys bitte ausschalten

© Prof. J.C. Freytag, Ph.D. 2.1



Ziel des Abschnittes

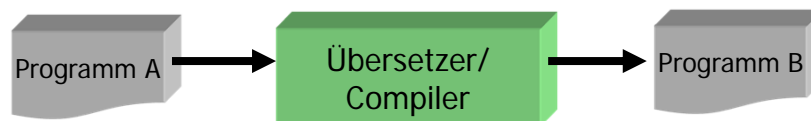
- Einführung in Übersetzer (Compiler)
 - Wichtige Grundlagen
 - Wesentlicher Aufbau (Architektur)
 - Komponenten & Strukturen
 - Aufgaben einzelner Komponenten
 - Beispiele

© Prof. J.C. Freytag, Ph.D. 2.2

Übersetzer (Compiler)

Was ist ein Compiler?

- ... ist ein Programm, das ein (ausführbares) Programm in einer Sprache A in ein ausführbares Programm einer anderen Sprache B übersetzt



- Erwartung: das erzeugte B-Programm ist in einem gewissen Sinne "besser" als das originale A-Programm

Übersetzer - Interpreter

- Was ist ein Interpreter?
 - Ein Programm, das ein ausführbares Programm liest und ein Ergebnis durch seine Ausführung liefert
- Was ist ein Übersetzer (Compiler)?
 - Ein Programm, das ein Programm in ein anderes transformiert
- Unterscheidung nicht immer offensichtlich
 - Rechner/HW: "Interpreter"
 - Taschenrechner: Interpreter
 - Viele ähnliche Probleme zu lösen
- Diese Vorlesung: **Fokus auf Übersetzer (Compiler)**

Vorlesung (SoSe2018)

Teil 2: Einführung



Motivation

- Warum werden Übersetzer gebaut?
 - Notwendig zur (effizienten) Ausführung von Programmen, die nicht in maschinen ausführbarer Form vorliegen
- Warum wird Übersetzerbau gelehrt?
 - Wohldefiniertes Gebiet mit Theorie, Datenstrukturen & Algorithmen;
 - Viele in der (praktischen) Informatik wichtigen Aspekte finden sich im Compilerbau wieder und lassen sich gut darstellen/üben;
 - Viele Aufgaben der „realen“ Welt lassen sich auf Compilerprobleme zurückführen; (effiziente) Lösungen sind (in vielen Fällen) bekannt.
- Warum diese Vorlesung besuchen??
 - Grundlegende Kenntnisse/Fähigkeiten werden vermittelt (siehe vorheriger Punkt);
 - Grundlage für andere Bereiche/Vorlesungen

© Prof. J.C. Freytag, Ph.D.

2.5



Perspektive

- Compilerbau nutzt viele Konzepte und Erkenntnisse anderer Informatikbereiche

| | |
|-------------------------------|------------------------------------|
| Künstliche Intelligenz | Lern- & Suchalgorithmen |
| Datenstrukturen & Algorithmen | Graphenalgorithmen |
| Theorie | Endlicher Automat / Kellerautomat |
| Systeme | Allocation/Synchronisation |
| Architektur | Pipeline Mgmt./ Befehlssatznutzung |

© Prof. J.C. Freytag, Ph.D.

2.6

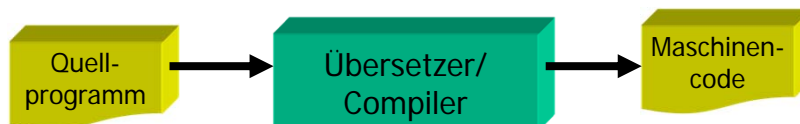


Qualitäten eines Compilers

- Erzeugung korrekten Codes
- Erzeugtes Programm soll effizient sein
- Compiler ist effizient
- Übersetzungszeit ~ Programmgröße
- Unterstützung separater Übersetzungen
- Gute Diagnostik bei Fehlern
- Gute Integration mit dem Debugger
- Aufruf von Programmen in anderer Sprache möglich
- Konsistente, vorhersehbare Optimierung



Abstrakte Sicht



- Konsequenz
 - Erkennen korrekter Programme
 - Generieren von korrektem Code
 - Speicherallokation für alle Variablen und Maschinencode
 - Festlegung des Formates für Objektcode (Assemblercode)

Vorlesung (SoSe2018)

Teil 2: Einführung

Herausforderung

- Können wir $n \times m$ Compiler mit $n+m$ Komponenten implementieren?
 - Alle Eigenschaften einer Programmiersprache muss im *Frontend* gekapselt sein
 - Alle Eigenschaften der Maschinsprache müssen im *Backend* gekapselt sein
 - Zwischencode muss alle "erdenklichen" Konzepte von Programmiersprachen und Maschinsprachen darstellen können

© Prof. J.C. Freytag, Ph.D. 2.9

Traditioneller 2-Phasen-Compiler

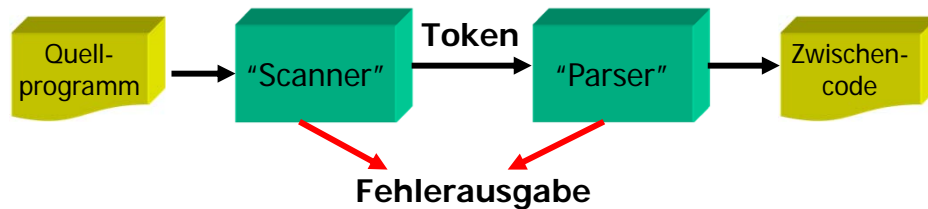
- Konsequenz
 - Zwischenrepräsentation ("Zwischencode")
 - **Frontend** erzeugt vom Quellprogramm korrekten *Zwischencode*
 - **Backend** erzeugt vom Zwischencode korrekten Maschinencode
 - Unterstützung mehrerer Quellsprachen/Maschinsprachen
 - Mehrere Phasen erzeugen meist besseren Code

© Prof. J.C. Freytag, Ph.D. 2.10

Vorlesung (SoSe2018)

Teil 2: Einführung

Frontend

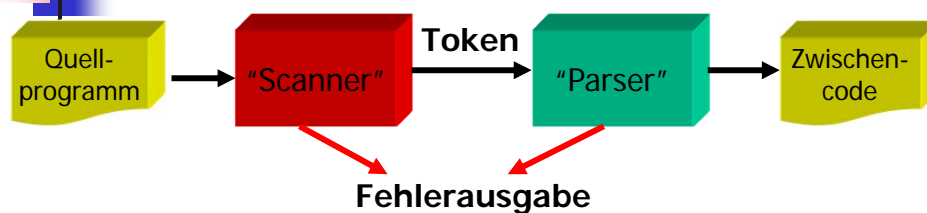


- Frontend ist verantwortlich für
 - Erkennen von korrekten Programmen
 - Fehlererkennung/-erholung
 - Erzeugen von Zwischen-code
 - Eine für das Backend "angenehme"/erleichterte Codeerzeugung

© Prof. J.C. Freytag, Ph.D.

2.11

Frontend (cont.)



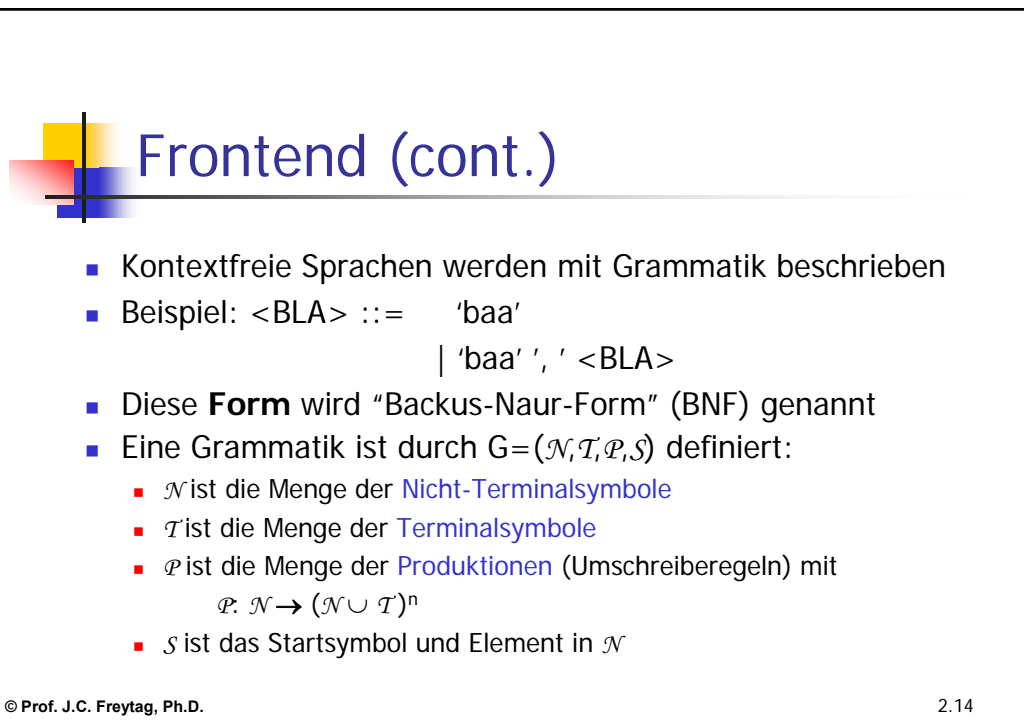
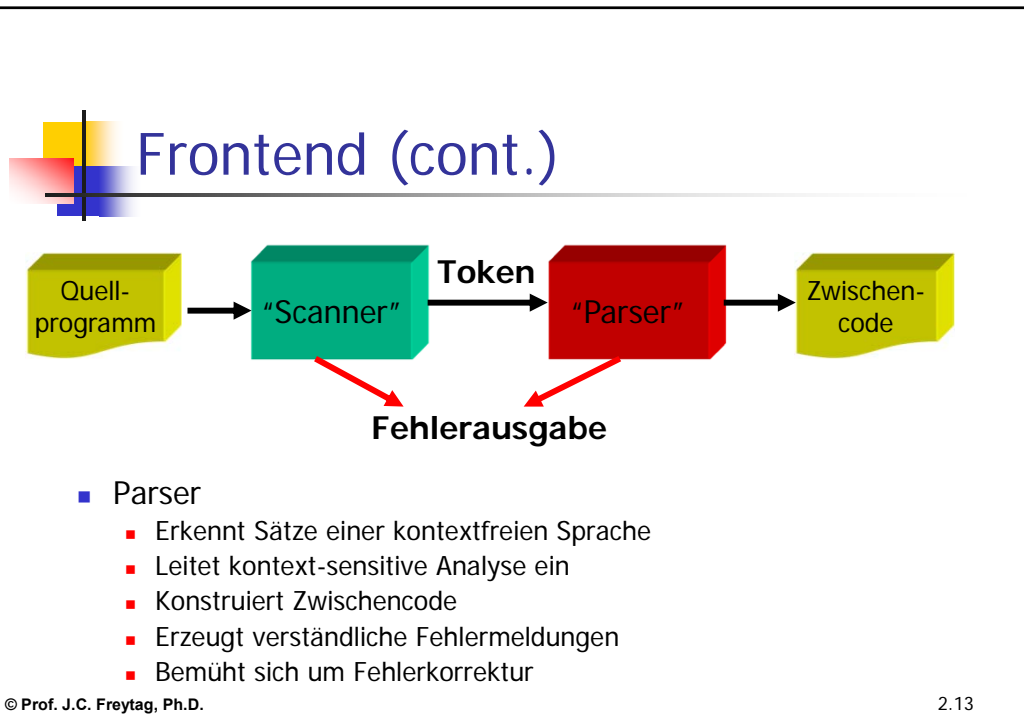
- Scanner
 - Überführt Zeichenketten in *Token*
 - *Token*: atomare Einheiten für die Syntaxanalyse
 - Beispiel: $x = x + y$ wird $\langle id, 'x' \rangle = \langle id, 'x' \rangle + \langle id, 'y' \rangle$
 - Zeichenkettenwert für ein Token wird "Lexem" genannt
 - Typische Token: number, id, +, -, do, end
 - Entfernen von Zwischenzeichen: Blank, Tab, Kommentaren, CRLF
 - Wichtig: Scanner muss effizient (schnell) sein

© Prof. J.C. Freytag, Ph.D.

2.12

Vorlesung (SoSe2018)

Teil 2: Einführung



Vorlesung (SoSe2018)

Teil 2: Einführung

Beispiel

- Kontextfreie Grammatik für einfache Ausdrücke
 1. $\langle \text{goal} \rangle ::= \langle \text{expr} \rangle$
 2. $\langle \text{expr} \rangle ::= \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$
 3. $\quad \quad \quad | \langle \text{term} \rangle$
 4. $\langle \text{term} \rangle ::= \text{number}$
 5. $\quad \quad \quad | \text{id}$
 6. $\langle \text{op} \rangle ::= +$
 7. $\quad \quad \quad | -$
- Zuordnung
 - $S = \langle \text{goal} \rangle$
 - $T = \{\text{number}, \text{id}, +, -\}$
 - $N = \{\langle \text{goal} \rangle, \langle \text{expr} \rangle, \langle \text{term} \rangle, \langle \text{op} \rangle\}$
 - $P = \{1, 2, 3, 4, 5, 6, 7\}$

© Prof. J.C. Freytag, Ph.D.

2.15

Beispiel (cont.)

- Zulässige Sätze der Sprache können durch "Substitution" abgeleitet werden

| Prod | Ergebnis |
|------|---|
| | $\langle \text{goal} \rangle$ |
| 1 | $\langle \text{expr} \rangle$ |
| 2 | $\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$ |
| 5 | $\langle \text{expr} \rangle \langle \text{op} \rangle y$ |
| 7 | $\langle \text{expr} \rangle - y$ |
| 2 | $\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{term} \rangle - y$ |
| 4 | $\langle \text{expr} \rangle \langle \text{op} \rangle 2 - y$ |
| 6 | $\langle \text{expr} \rangle + 2 - y$ |
| 3 | $\langle \text{term} \rangle + 2 - y$ |
| 5 | $x + 2 - y$ |

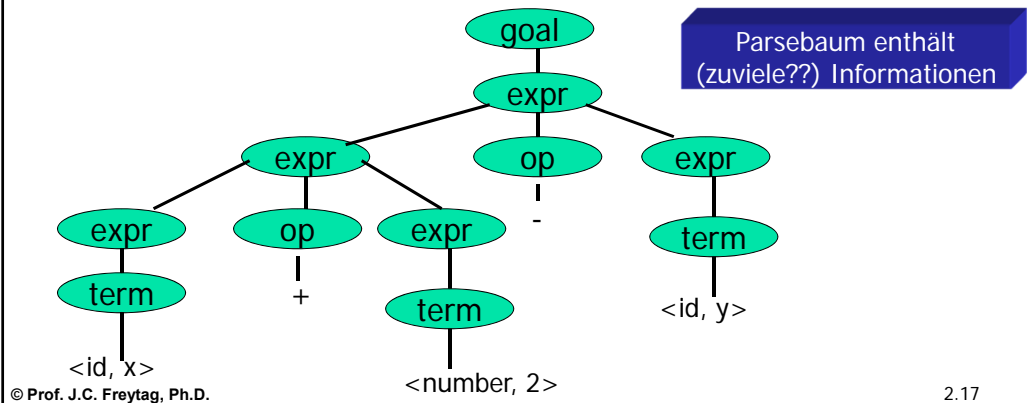
Um einen zulässigen Satz zu erkennen, muss dieser Prozess umgekehrt werden ("parsing")

© Prof. J.C. Freytag, Ph.D.

2.16

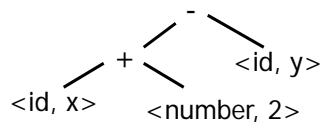
Parsebaum (Parse Tree)

- Erzeugungs-/Erkennungsprozess kann durch einen "Parsebaum" beschrieben werden



Abstrakter Syntaxbaum (AST)

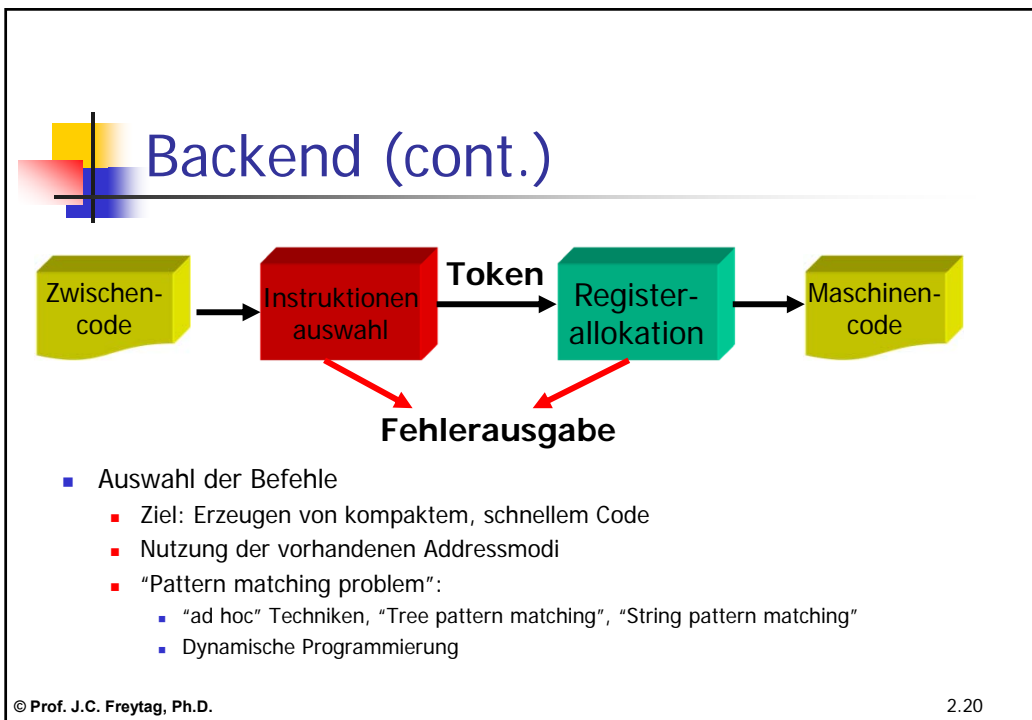
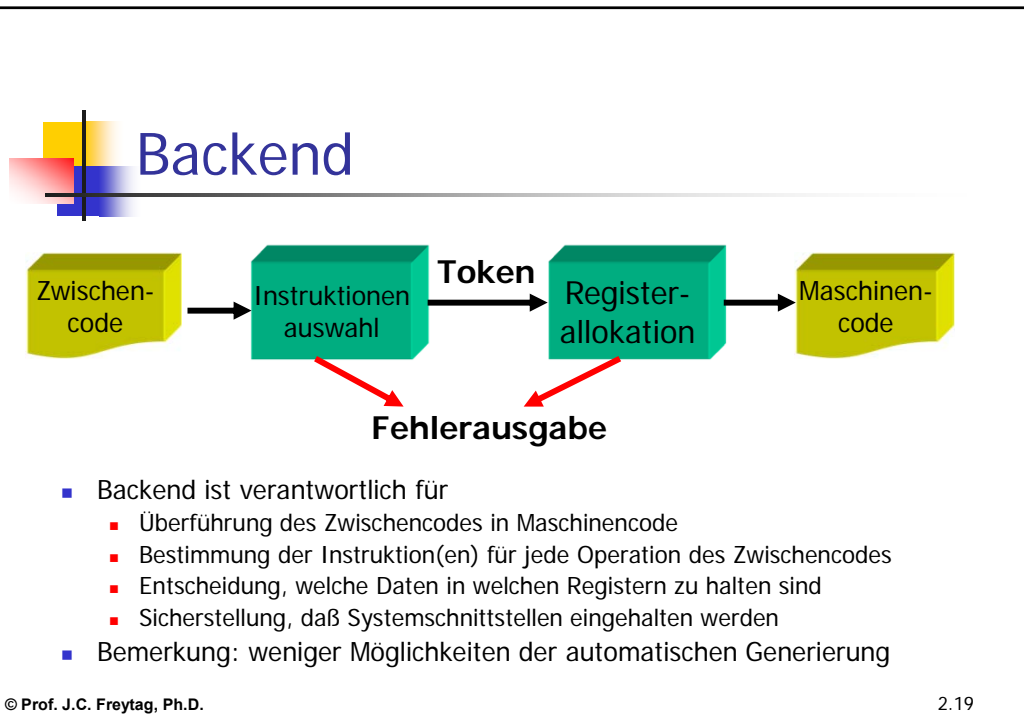
- Compiler benutzen oft abstrakte Syntaxbäume (ASTs)



- Vorteil:
 - Kompaktere Darstellung
 - ASTs werden oft als Zwischendarstellung zwischen Frontend und Backend benutzt

Vorlesung (SoSe2018)

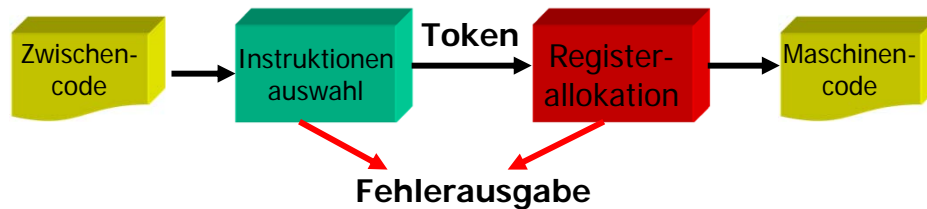
Teil 2: Einführung



Vorlesung (SoSe2018)

Teil 2: Einführung

Backend (cont.)

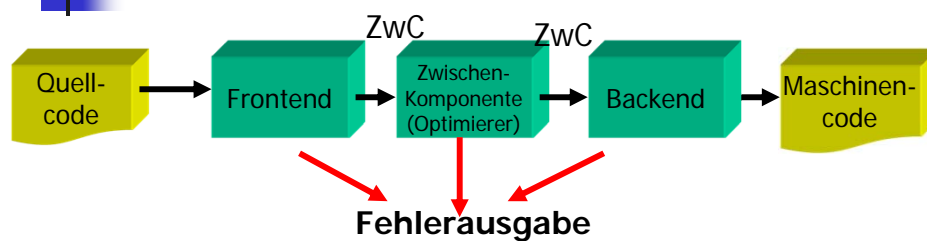


- Registerallokation
 - Ziel: Daten sollen in den Registern zu finden sein, wenn diese benötigt werden
 - Nur endliche Anzahl an Registern vorhanden
 - Abhängig von Registernutzung (oder nicht) unterschiedliche Codeerzeugung
 - Optimale Allokation ist sehr schwierig
 - NP-vollständiges Problem für ein oder k Register
 - Moderne Registerallokation benutzt Lösungen zur "Graphenfärbung"

© Prof. J.C. Freytag, Ph.D.

2.21

Traditioneller 3-Phasen-Compiler



- Zwischenkomponente:
 - Ziel: Codeverbesserung/-optimierung zur Laufzeitverkürzung / Verringerung des Ressourcenverbrauches
 - Optimierung darf Korrektheit des Programmes nicht zerstören

© Prof. J.C. Freytag, Ph.D.

2.22

Vorlesung (SoSe2018)

Teil 2: Einführung



Optimierer

- Moderne Compiler: Meist Mehr-Phasen-Optimierer
- Typische Phasen
 - Konstanten propagieren und zusammenfassen
 - Code reorganisieren
 - Teilausdrücke erkennen und eliminieren
 - Redundanzen erkennen und eliminieren
 - "Toten Code" erkennen und eliminieren

© Prof. J.C. Freytag, Ph.D.

2.23



Beispiel-Grammatik

"einfache" Statements

```
Stmt ::= Stmt ; Stmt
Stmt ::= Id := Exp
Stmt ::= print ( ExpList )
Exp ::= id
Exp ::= num
Exp ::= Exp Binop Exp
Exp ::= ( Stmt , Exp )
ExpList ::= Exp , ExpList
ExpList ::= Exp
Binop ::= +
Binop ::= -
Binop ::= X
Binop ::= /
```

Compound Statement

Assignment Statement

Print Statement

Id Expression

Number Expression

OpExpression

ESeqEpression

PairExpList

LastExpList

Plus

Minus

Times

Div

© Prof. J.C. Freytag, Ph.D.

2.24

Vorlesung (SoSe2018)

Teil 2: Einführung

Beispiel-Programm

Programm

```
a := 5 + 3; b := (print(a, a-1), 10 x a);  
print(b)
```

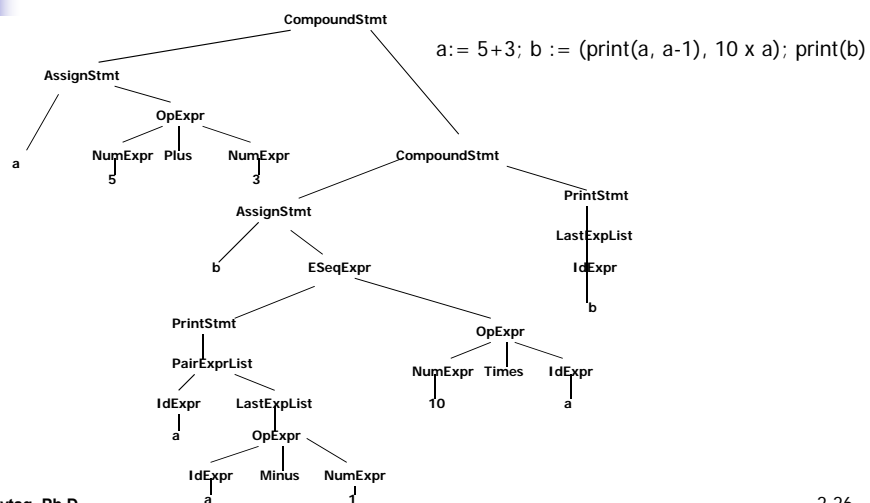
Erzeugt

```
8 7  
80
```

© Prof. J.C. Freytag, Ph.D.

2.25

Baumrepräsentation



© Prof. J.C. Freytag, Ph.D.

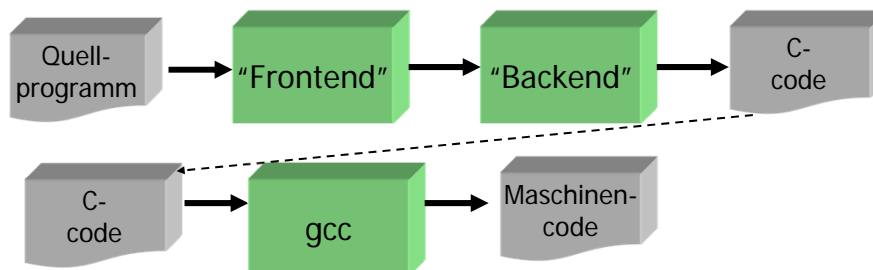
2.26

Vorlesung (SoSe2018)

Teil 2: Einführung

Perspektive (Forts.)

- häufige Aufgabe eines Informatikers:
- Quelltextanalysen, Quelltexttransformationen
- seltener (aber sehr anspruchsvoll): Codegenerierung
- Vereinfachung (aber schwierig genug): Code in höherer Programmiersprache



© Prof. J.C. Freytag, Ph.D.

2.27

Literatur

- Kernighan, Ritchie
The C Programming Language, Prentice Hall (2.Auflage, 1988, 1990)
- Aho, Sethi, Ullman:
Compilers: Principles, Techniques und Tools, Addison-Wesley Publishing Company, 1987
(bekannt als „Drachenbuch“)
- Wilhelm, Maurer:
Übersetzerbau: Theorie, Konstruktion, Generierung, Springer Verlag, 2. Auflage
- Fischer/LeBlanc:
Crafting a Compiler with C, The Benjamin Cummings Publishing Company, 1991

© Prof. J.C. Freytag, Ph.D.

2.28



Zusammenfassung

- Überblick
 - Architektur
 - Frontend
 - Backend
 - Einzelaufgaben (grob) & Konzepte
 - Scanning, Parsing,



Fragen ??

