Übersetzergenerierung mit lex und yacc

Jan Bredereke

WiSe 2007/08, Universität Bremen

0. Überblick und Organisatorisches

Motivation

- Übersetzer: Grundlegende Werkzeuge
  - „welche Fehler kann er finden?“
  - „Konstrukt wie effizient übersetzen?“
- „Eingabe → Ausgabe“ = Übersetzer
  - auch: Benutzerinteraktion
  - strukturierte Ein-/Ausgabe: domain specific languages (DSLs)
  - ständig neue „kleine“ Sprachen
- Übersetzer großenteils automatisch generierbar
  - Generator-Eingabe: DSL-Beschreibung (in Meta-Sprache)

Inhalte der Vorlesung

1. Einführung
2. Lexikalische Analyse
3. Der Textstrom-Editor sed
4. Der Scanner-Generator lex
5. Syntaxanalyse und der Parser-Generator yacc
6. Syntaxgesteuerte Übersetzung
7. Übersetzungssteuerung mit make

Organisatorisches

- Vortragender
  - PD Dr. Jan Bredereke, brederek@tzi.de
    - Astrium GmbH, Bereich Space Transportation, Abteilung GNC & On-Board Software Engineering
- Zeiten
  - montags 17:00–18:30 oder 17:15–18:45?
- Voraussetzung
  - Vordiplom, oder mindestens Theoretische Informatik I und Praktische Informatik 2
- Web-Seiten
  - www.tzi.de/agbs/lehre/ws0708/uegen/

Literatur

- Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman. **Compilerbau, Teil 1.** Oldenbourg, 2. Aufl. (Dez. 1999). ISBN 3-486-25294-1.
- N. N. **sed(1) Manual-Seite.** In: LunetIX Linuxhandbuch (Juli 1993).
- Mike Haertel, James A. Woods und David Olson. **grep(1) Manual-Seite.** In: LunetIX Linuxhandbuch (Juli 1993).

Literatur (2)

- John R. Levine, Tony Mason und Doug Brown. **lex & yacc.** O'Reilly, zweite korrigierte Auflage (1995). ISBN 1-56592-000-7.
- Vern Paxson. **Flex, version 2.5 - A fast scanner generator.** University of California (1990).
- Charles Donelly and Richard Stallman. **Bison - The YACC-compatible parser generator.** Version 1.75. Free Software Foundation (2002). ISBN 1-882114-44-2. GNU Free Documentation License.

Literatur (3)

- R. Stallman, R. McGrath und P. Smith. **GNU Make - A Program for Directing Recompilation.** Version 3.80. Free Software Foundation (Juli 2002). ISBN 1-882114-81-7. GNU Free Documentation License.

Software

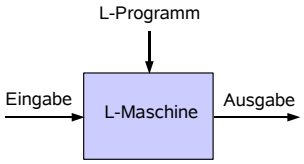
- lex & yacc
  - flex & bison
    - auf allen Uni-Rechnern
    - in allen großen Linux-Distributionen
    - Windows: im freien Cygwin-Paket
- sed, grep, make
  - ebenso verfügbar
- Download-Links
  - www.tzi.de/agbs/lehre/ws0708/uegen/

Scheinkriterien-Vorschlag

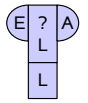
- keine Übungsaufgaben, da reine Vorlesung
  - deswegen 2 ECTS
- mündliche Prüfung am Ende
  - 20-30 min. pro Kandidat
  - auf Wunsch mit Beisitzer

1. Einführung

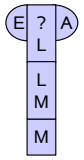
Universelle programmierbare Rechenmaschine



- L-Maschine
  - kann beliebige Programme einer Sprache L ausführen
- Ausführung eines Programms
  - symbolisiert durch:



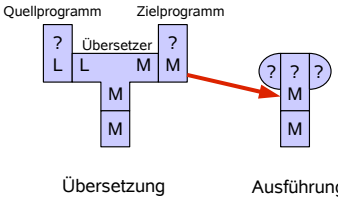
Interpreter



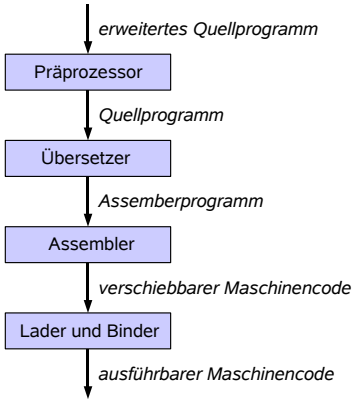
- simuliert L-Maschine durch M-Programm
  - L-Maschine: problemorientiert, virtuell
  - M-Maschine: existiert real

Übersetzer

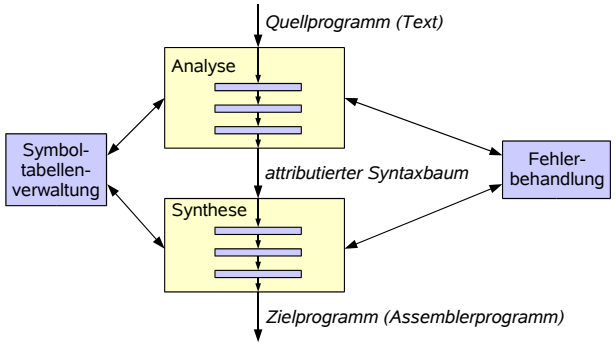
- übersetzt L-Programme in M-Programme
- M-Programm ist direkt ausführbar



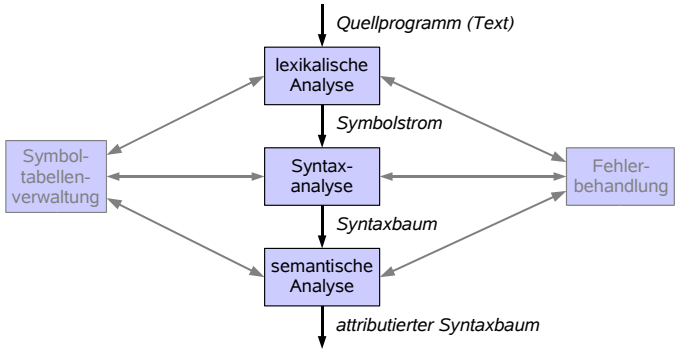
Programme rund um einen Übersetzer



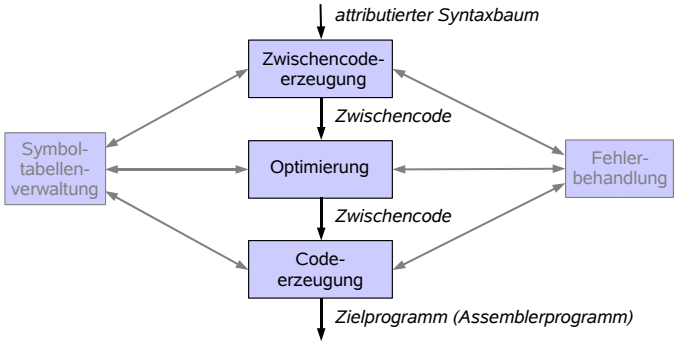
Die Phasen der Übersetzung



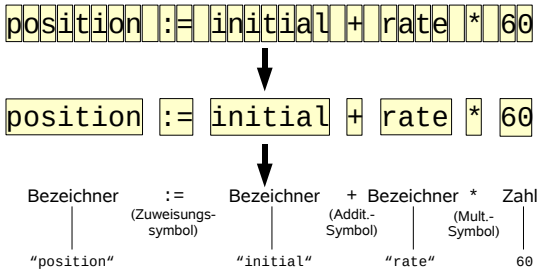
Die Analysephasen



Die Synthesephasen



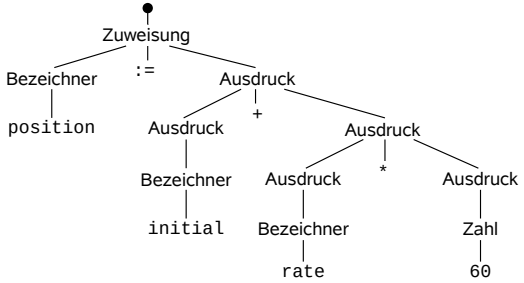
Lexikalische Analyse



- Gruppierung der Eingabezeichen in Lexeme
  - Leerzeichen werden entfernt
- Zuordnung Lexem → Symbol
  - Symbole (Token): Bezeichner, :=, +, *, Zahl, ...
  - Symbol hat z.T. Wert als Attribut

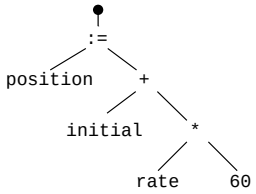
Syntaxanalyse (Parsing)

Parse-Baum:



- hierarchische Gruppierung der Symbole
  - mit Hilfe einer kontextfreien Grammatik

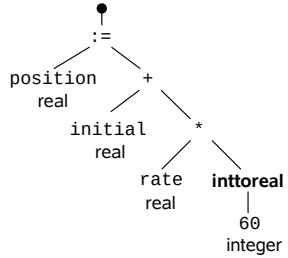
Syntaxbaum



- ist komprimierte Darstellung des Parse-Baums

Semantische Analyse

attributierter Syntaxbaum:



- sammelt Typinformationen für Codeerzeugung
  - nutzt hierarchische Struktur des Syntaxbaums
- Typüberprüfungen
  - ggf. automatische Typkonversionen

Symboltabellenverwaltung

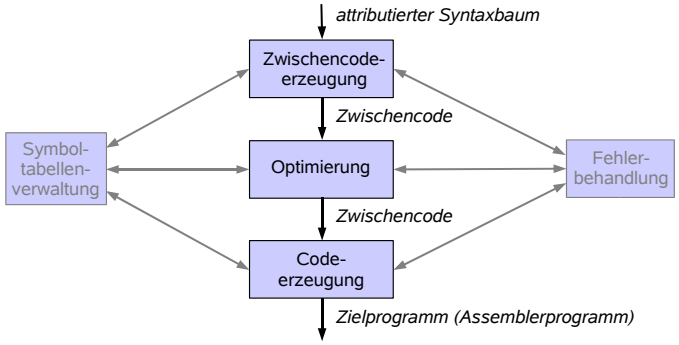
Nr.	Name	Typ	...
1	position	real	...
2	initial	real	...
3	rate	real	...
4	...		...

- sammelt und speichert die Attribute der Bezeichner
  - Typ
  - Gültigkeitsbereich
  - bei Prozedurnamen: Anzahl & Typen der Argumente, ...
  - Details zum Speicherbereich (bei Codeerzeugung)
  - ...

Fehlerbehandlung

- in jeder Phase
- jeder Fehler muss behandelt werden
  - Ziel: weiterarbeiten, um möglichst viele Fehler zu finden

Erinnerung: Die Synthesephasen



Zwischencodeerzeugung

```
temp1 := inttoreal(60)
temp2 := bezeichner3 * temp1
temp3 := bezeichner2 + temp2
bezeichner1 := temp3
```

- Zwischencode: Programm für eine abstrakte Maschine
  - leicht zu erzeugen
  - leicht in Zielsprache zu übersetzen
  - hier: Drei-Adress-Code
    - jede Instruktion hat höchstens drei Operanden

Optimierung

```
temp1 := bezeichner3 * 60.0
bezeichner1 := bezeichner2 + temp1
```

- versucht, Zwischencode zu verbessern
  - schnellere Ausführung
  - weniger Speicherplatz

Codeerzeugung

```
MOVF bezeichner3, R2
MULF #60.0, R2
MOVF bezeichner2, R1
ADDF R2, R1
MOVF R1, bezeichner1
```

- Variable → Speicherplatz
- Zwischencode-Instruktion → Folge von Maschinenbefehlen
- Register → Variable

Front-End

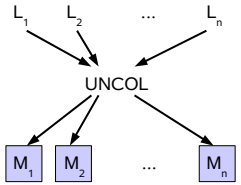
- Phasen(-anteile), die von Quellsprache abhängen
  - lexikalische Analyse
  - syntaktische Analyse
  - Symboltabellenerstellung
  - semantische Analyse
  - Zwischencodeerzeugung
  - Optimierung, maschinenunabhängig
  - Fehlerbehandlung für obige Phasen

Back-End

- Phasen(-anteile), die von Zielsprache abhängen
  - Optimierung, maschinenabhängig
  - Codeerzeugung
  - weitere Symboltabellenoperationen
  - Fehlerbehandlung für obige Phasen

UNCOL-Ansatz

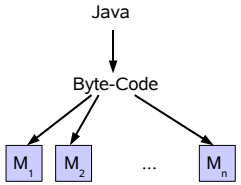
- Universal Communication Oriented Language
  - universelle Zwischensprache (Idee ca. 1960):



  - Vorteil: viel weniger Übersetzer nötig
- leider gibt es keine *effiziente* Sprache UNCOL
  - Sprachen zu verschieden
  - Rechner zu verschieden

Zwischensprache für nur eine Quellsprache

- Beispiel Java:



- braucht nur *einen* Übersetzer Java → Byte-Code
- geht
  - aber gute Definition eines Zwischencodes immer noch schwierig

Läufe vs. Phasen

- Läufe (Pässe):
  nacheinander ablaufende Teilprogramme des Übersetzers
- Phasen:
  funktional unabhängige Teilaufgaben
  - kein Feedback
- oft mehrere Phasen in einem Lauf
  - effizienter
  - Beispiel: lexikalische + syntaktische Analyse
    - Puffer für max. ein Lexem Vorausschau
    - Scanner liefert Lexem auf Anforderung des Parsers

Übersetzerbauwerkzeuge

- allgemeine Software-Werkzeuge
- Scanner-Generatoren — lex
- Parser-Generatoren
- syntaxgesteuerte Übersetzungsmaschinen — yacc
  - Parse-Baum → Zwischencode
- automatische Codegeneratoren
- Datenflußmaschinen
  - Datenflußanalyse für Codeoptimierung

Inhalte der Vorlesung