Seminar „Compilergeneratoren“

The GENTLE Compiler Construction System

Mathias Peters & Eik Villwock

17. Juli 2003

Ablauf

1.1. Installation

- Download `http://www.first.gmd.de/gentle/`
- entpacken der Datei **gentle-97.tar.gz**
- ausführen von **build** in den Verzeichnissen

  gentle
  lib
  reflex

Voraussetzung: C-Compiler

- ggf. lex, yacc Installation
- lauffähig unter allen verbreiteten UNIX-Systemen und DOS

- kostenlose Nutzung für Ausbildung
- kommerzielle Anwendungen nur mit Lizenzvertrag

1.2. Überblick

- wurde von F.W. Schröer 1989 entwickelt (GMD)
- fasst die wichtigsten dieser Methoden und Werkzeuge zusammen
- Schnittstelle zur Programmiersprache C
- Ergänzung durch lex und yacc (LALR(1)) Spezifikationen möglich
- ist zur Erstellung "ernsthafte Produktionscompiler" gedacht
- deklarative Sprache wie Prolog oder SQL
- generiert schnell und effiziente Compiler

Gentle System:

1. Stark problemorientierte Sprache für den Compilerbau
2. Compiler, der Gentle-Programme in C-Programme übersetzt
3. einige Standardmodule

1.3. Allgemeines

- basierend auf rekursiven Definitionen und strukturierten Ersetzungen
- Programm besteht im Wesentlichen aus **Regeln** die **Prädikate** beschreiben

• Token-Prädikate **'TOKEN'**
Erkennung von Lexemen im Quellprogramm und Übersetzung in Token zur Generierung des Scanners

• Zwischensymbol-Prädikate **'NONTERM'**
Beschreibung der kontextfreien Grammatik zur Generierung des Parsers

- Aktions-Prädikate **'ACTION'**
  Übersetzungen und Aktionen(z.B. Ausgaben) - Spezifikation in Gentle oder C

- Bedingungs-Prädikate **'CONDITION'**
  Überprüfung von Bedingungen

- Feger-Prädikate **'SWEEP'**
  Durchsuchen komplexer Strukturen (z.B. Bäume)

- Wahl-Prädikate **'CHOICE'**
  Lösung von Problemen bei Codeerzeugung und -optimierung

- Schlüsselwörter von Apostroph umschlossen
  'TOKEN'
- Bezeichner: Buchstabe gefolgt von Buchstaben, Zahlen, Unterstrich
  Var_1

- Fehlerbehandlung
  Abbruch nach dem ersten Fehler bei Benutzung von lex und yacc

- Kommentare

--	bis Zeilenende
/* comment */	mehrzeilig

• Parameter

'nonterm' expr1 (INT ->) - Eingabeparameter vom Typ INT
'nonterm' expr2 (-> INT) - Ausgabeparamter vom Typ INT
'nonterm' expr3 (INT -> INT) - Ein- und Ausgabeparamter vom Typ INT

• Variablen

implizite Deklaration je nach Verwendung

```
'nonterm' ahne2(-> INT)
   'rule' ahne2(-> N+1) : “gross“ ahne1(-> N)
```

• Wurzelprädikat

```
      'root' expression(-> X) print (X)
```

'TYPE' - interne, externe Typdeklarationen

```
'TYPE' GESCHLECHT
m
f
```

'MODULE' - Kapselung einzelner Programmteile

'VAR' - globale Variablen

'TABLE' - Tabellendeklaration

1.4. Vordefinierte Prädikate

Können ohne Deklaration benutzt werden

- Vergleichsprädikate **`eq, ne, gt, ge, lt, le`**
- Ausgabeprädikat **`print`**
- Variablenüberprüfung und -definition **`where`**
- **`@-Prädikat`**
  kann nach einem **`'NONTERM'-`** oder **`'TOKEN'-`**Prädikat benutzt werden

übergibt die Koordinate aus dem Quelltext

Erzeugung aussagefähiger Fehlermeldungen

```
'rule' Stmt: ”IF” @(->POS) Expr ”THEN” Stmt
```

Ausgabe: **`file 1, line 3, col 6`**

Optimale Regelauswahl - $

- alternative Regeln werden unter Benutzung des **'choice'** Prädikates angegeben
- Zuweisung einer positiven ganzen Zahl zu den einzelnen alternativen Regeln
- abhängig von der Eingabe wird die günstigere Variante gewählt

```
'choice' Encode(Stmnt)
      'rule' Encode(assign(V,X)):
      StackCode(X)
      POP(V)
      $ 10
      'rule' Encode(assign(V,X)):
      AccuCode(X)
      STOREACCU(V)
      $ 10
```

1.5. Vergleich Gentle - Prolog

- Gentle besitzt im Gegensatz zu Prolog **verschiedene Prädikate**
- Parameter von Prädikaten müssen als **Eingabeparameter** oder **Ausgabeparameter** festgelegt werden
- Wiederaufsetzungsverfahren:
  benutzt wird **flaches Wiederaufsetzen**(shallow backtracking)
  anstelle von **tiefem Wiederaufsetzen**(deep backtracking)
- Aufruf kann bei Gentle als Funktion betrachtet werden,
  bei Prolog als Relation

→ hohe Ausführungsgeschwindigkeit
→ Mächtigkeit geringer, aber ausreichend für Compilererstellung

```
'type' PERSON a b c d e
'condition' Mag(PERSON -> PERSON)
      'rule' Mag(d -> c): .
      'rule' Mag(d -> e): .
```

flaches Wiederaufsetzen(shallow backtracking):

- es wird nur die **erste** gelungene Ausführung eines Aufrufes betrachtet, anstatt alle **möglichen gelungenen** Ausführungen

→ bei Erfolg der ersten Regel wird abgebrochen

tiefem Wiederaufsetzen(deep backtracking):

- es werden nach der ersten gelungenen Ausführung alle weiteren Regeln untersucht

→ es wird auch die zweite Regel überprüft

Ablauf

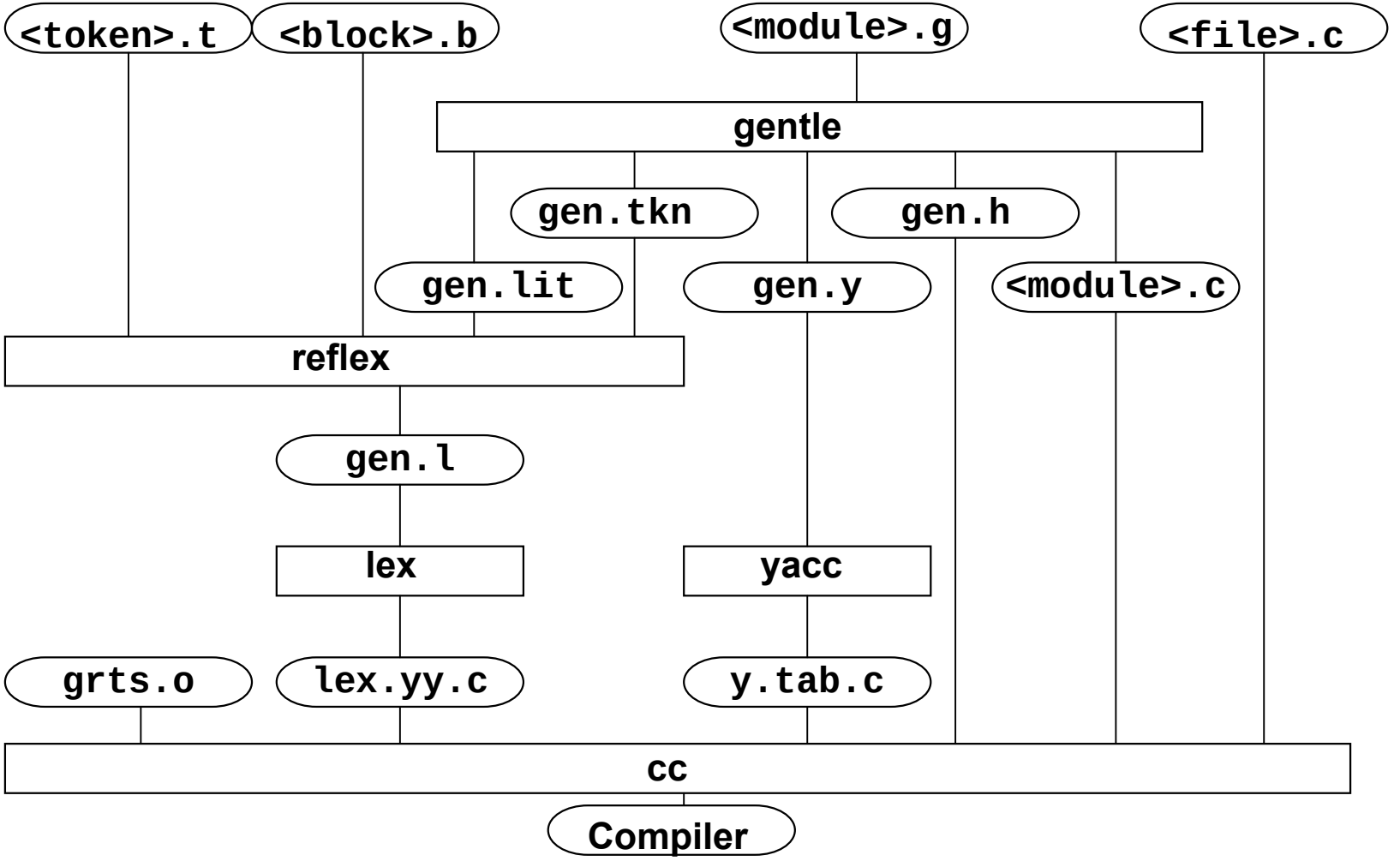
- ✓ **1. Vorstellung Gentle**
  - ✓ 1.1. Installation
  - ✓ 1.2. Überblick
  - ✓ 1.3. Allgemeines
  - ✓ 1.4. Vordefinierte Prädikate
  - ✓ 1.5. Vergleich Gentle - Prolog
- **2. Kleines Beispiel - Rechner**
  - 2.1. Beispiel Rechner - konkrete Syntax
  - 2.2. Compilergenerierung
  - 2.3. Listen und Bäume
  - 2.4. Ausdrücke und Muster
  - 2.5. Beispiel Rechner - abstrakte Syntax
- **3. Komplexes Beispiel - Minako**
- **4. Fazit**
- **5. Quellen**

2.1. Beispiel Rechner - konkrete Syntax

Grammatik EBNF

```
expression      ::= expression '+' expr2
                  | expression '-' expr2
                  | expr2
expr2            ::= expr2 '*' expr3
                  | expr2 '/' expr3
                  | expr3
expr3           ::= Number
                  | '-' expr3
                  | '+' expr3
                  | ' (' expression ') '
```

2.2. Compilergenerierung



Compilergenerierung

zu schreibende Dateien:

- **`*.g`** - Gentle Datei
- **`*.t`** - Flex Spezifikation der Token
- **`*.b`** - Spezialdefinition für Flex
- **`*.c`** - C-Code für spezielle Teile des Compilers

erzeugte Dateien:

- **`gen.lit`** - Lex Spezifikation der Terminalsymbole
- **`gen.tkn`** - Auflistung der Tokenprädikate
- **`gen.y`** - Yacc Parser-Spezifikation
- **`gen.h`** - Tokendefinitionen
- **`module.c`** - in C übersetzte Gentle Datei

- **`gen.l`** - Lex Scanner-Spezifikation
- **`lex.yy.c`** - Scanner C-Datei
- **`y.tab.c`** - Parser C-Datei

- **`grts.c`** - Gentle Laufzeitsystem (gentle run time system)

2.3. Listen und Bäume

-bietet durch Definition eigener Datentypen einfache Möglichkeit mit Listen und Bäumen zu arbeiten

-Listendefinition:

```
'type' LISTE
leer
list(Element: INT, Rest: LISTE)
```

-verschiedene Ausprägungen von Listen:

```
leer
list(15, leer)
list(15, list(3, leer))
list(15, list(3, list(17, leer)))
```

Bäume:

-Handhabung entspricht der von Listen

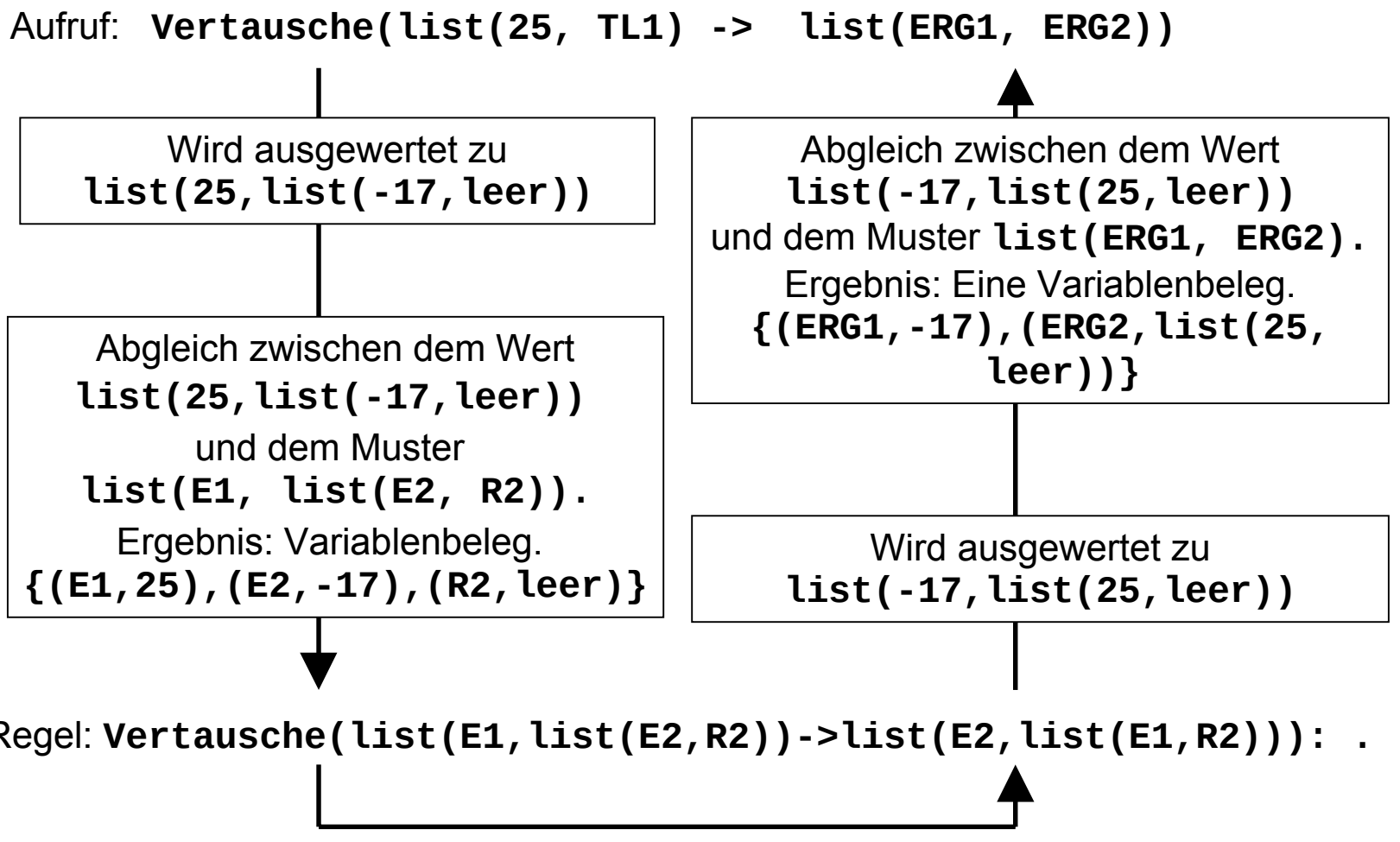
-Definition eines Baumtyps:

```
'TYPE' Baum
leer
baum(Name : STRING, LKnoten : Baum, RKnoten:Baum)
```

-Ausprägungen von Bäumen:

```
leer
baum(BBaum, leer, leer)
baum(BBaum, baum(Lkind, leer, leer),
            baum(Rkind, leer, leer))
```

2.4. Ausdrücke und Muster



2.5. Beispiel Rechner - abstrakte Syntax

-konkrete Syntax nützlich, um die Grammatik zu parsen

-jedoch meist zu unhandlich, um damit zu arbeiten(etwa bei Termumformungen)

-durch Definieren eigener Typen, möglich von der konkreten Syntax zu abstrahieren

-wesentliche Informationen der Daten(die Struktur) bleiben jedoch erhalten

-geeignet als Zwischendarstellung(siehe Beispiel praefix.g)

Ablauf

- ✓ **1. Vorstellung Gentle**
  - ✓ 1.1. Installation
  - ✓ 1.2. Überblick
  - ✓ 1.3. Allgemeines
  - ✓ 1.4. Vordefinierte Prädikate
  - ✓ 1.5. Vergleich Gentle - Prolog
- ✓ **2. Kleines Beispiel - Rechner**
  - ✓ 2.1. Beispiel Rechner - konkrete Syntax
  - ✓ 2.2. Compilergenerierung
  - ✓ 2.3. Listen und Bäume
  - ✓ 2.4. Ausdrücke und Muster
  - ✓ 2.5. Beispiel Rechner - abstrakte Syntax
- **3. Komplexes Beispiel - Minako**
- **4. Fazit**
- **5. Quellen**

3. Minako

```
                                .
                                .
                                .
'nonterm' program
    'rule' program : declassignment ";“
    'rule' program : functiondefinition
    'rule' program : declassignment ";" program
    'rule' program : functiondefinition program
'nonterm' functiondefinition
    'rule' functiondefinition : type id
             "(" paraml ")" "{" statementlist "}“
'nonterm' paraml
    'rule' paraml
    'rule' paraml : parameterlist
                                .
                                .
                                .
```

Ablauf

- ✓ **1. Vorstellung Gentle**
  - ✓ 1.1. Installation
  - ✓ 1.2. Überblick
  - ✓ 1.3. Allgemeines
  - ✓ 1.4. Vordefinierte Prädikate
  - ✓ 1.5. Vergleich Gentle - Prolog
- ✓ **2. Kleines Beispiel - Rechner**
  - ✓ 2.1. Beispiel Rechner - konkrete Syntax
  - ✓ 2.2. Compilergenerierung
  - ✓ 2.3. Listen und Bäume
  - ✓ 2.4. Ausdrücke und Muster
  - ✓ 2.5. Beispiel Rechner - abstrakte Syntax
- ✓ **3. Komplexes Beispiel - Minako**
- **4. Fazit**
- **5. Quellen**

4. Fazit

- Standardfehlerbehandlung nicht optimal, da Abbruch nach Erkennung des ersten Fehlers
  - @-Prädikat kann für eigene Fehlerausgabe benutzt werden
- ungewohnter Programmierstil, aber Möglichkeit einen Compiler zum große Teil auf einer sehr hohen und übersichtlichen Abstraktionsebene zu schreiben
- generiert schnell und effiziente Compiler
- Ergänzung durch C-Programmteile

5. Quellen

The GENTLE Compiler Construction System
Friedrich Wilhelm Schröer; R. Oldenbourg, Munich and Vienna, 1997
ISBN 3-486-247034-4

`http://www.first.gmd.de/gentle/`

The Compiler Construction System GENTLE - Manual and Tutorial
Jürgen Vollmer, GMD; Revision 3.9

`http://www.informatik-vollmer.de/publications/gentle-manual.pdf`

Einführung in Gentle
Ulrich Grude, Technische Fachhochschule Berlin

`http://www.tfh-berlin.de/~grude/SkriptGentle.pdf`

Compilerbau mit Gentle
Ulrich Grude, Technische Fachhochschule Berlin

`http://www.tfh-berlin.de/~grude/CBmitGentle.pdf`