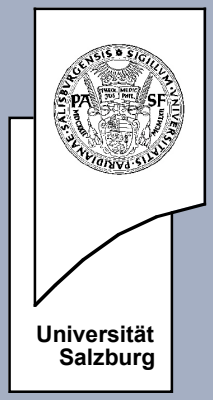

Attributierte Grammatiken

O.Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Pree
Universität Salzburg
www.SoftwareResearch.net

SOFTWARE RESEARCH LAB

Inhalt

- Merkmale attributierter Grammatiken
- Beispiel

Merkmale attributierter Grammatiken (ATG)

Allgemeine Merkmale von ATG

- Datenorientierte Entwurfsmethode
- Grundidee: Eingangsinformation steuert Ver-arbeitungsprozeß maßgeblich
- Beschreibung der Daten durch eine Grammatik
- Erweitern der Grammatik durch "Übersetzungsaktionen"
- Verwendung bewährter Techniken des Übersetzerbaus(lexikalische Analyse, Syntaxanalyse)

Wann sind ATG anwendbar?

- wenn im wesentlichen *ein* Eingabestrom vorliegt
- wenn Eingabeinformation genügend strukturiert ist

ATG sind geeignet für

- Entwurf von Programmen
- Spezifikation von Programmen
- Dokumentation von Programmen

Algorithmische Interpretation von ATG (I)

- Grammatikregeln werden als Erkennungsprozesse aufgefaßt.
- Das aus der Grammatik abgeleitete Programm heißt *Syntaxanalysator (Parser)*
- Es setzt einen *lexikalischen Analysator* voraus, der Terminalsymbole erkennt und liefert.

Arbeitsweise des Syntaxanalysators

- Jede Ableitungsregel wird von links nach rechts durchlaufen
- Zur Alternativenauswahl wird das nächste Eingabesymbol verwendet
- Bei jedem Terminalsymbol fordert der Syntaxanalysator vom lexikalischen Analysator das nächste Eingabesymbol an
- Bei jedem Nonterminalsymbol
  - unterbricht der Syntaxanalysator die Abarbeitung der laufenden Regel
  - arbeitet das angetroffene Nonterminalsymbol ab
  - setzt dann an der Abbruchstelle fort.

Semantische Aktionen in Ableitungsregeln

Schreibweise: Pseudocode zwischen "sem" und "endsem"

Beispiel 1: Zählung der Buchstaben eines Wortes

```
Wort = Buchstabe     sem b:=1 endsem
       { Buchstabe   sem b:=b+1 endsem
       }         sem Write(b," Buchstaben") endsem.
```

Beispiel 2: Binärzahl in umgekehrte Reihenfolge transformieren

```
BinZahl = "0" [BinZahl]    sem Write("0") endsem

        | "1" [BinZahl]    sem Write("1") endsem.
```

Semantische Werte der Grammatiksymbole

- **Ausgangsattribute**
  - entstehen bei der Erkennung eines Symbols
  - entsprechen Ausgangsparametern
- **Eingangsattribute**
  - werden einem Symbol zu seiner Erkennung mitgegeben
  - entsprechen Eingangsparametern

- **Beispiele**

Ziffer ↑ wert
Telegramm ↑ words ↑ longWords
BewegungsSatz ↓ stammNr

Entwurfsvorgang mit ATG

- Syntaktische Struktur der Eingabe als kontextfreie Grammatik darstellen
- Attribute für den Verarbeitungsprozeß finden
  - Semantische Werte von Symbolen überlegen
  - Angabe mit Name, Typ und Bedeutung
- Attributierte Grammatiksymbole spezifizieren
- Semantische Prozeduren spezifizieren
  - ausserhalb der Grammatik definierte Prozeduren
  - Attributierte Grammatik aufstellen

ATG-Beispiel: Bewegung eines Roboterarms

ATG-Beispiel

Verbale Formulierung

- Die Eingabe beginnt mit den Koordinaten des Startpunkts, auf die beliebig viele lineare und kreisförmige Teilbewegungen folgen können. Die Eingabe wird mit dem Code “E” abgeschlossen.

- Die Beschreibung einer linearen Teilbewegung beginnt mit der Anzahl der Einzelschritte, die benutzt werden sollen, um den Zielpunkt anzusteuern. Darauf folgen der Code “L” und die Koordinaten des Zielpunkts.

- Die Beschreibung einer Kreisbewegung beginnt ebenfalls mit der Anzahl der (linearen) Einzelschritte, auf die der Code “C” und die Koordinaten eines Übergangspunktes und des Zielpunkts folgen. Der Zielpunkt soll in einer Kreisbahn angesteuert werden, auf der der Übergangspunkt liegt.

ATG-Beispiel

- Punkt-Koordinaten werden durch drei ganze Zahlen zwischen 0 und 2000 angegeben, die in runde Klammern eingeschlossen und durch Kommas voneinander getrennt werden. Jede Zahl bedeutet eine absolute Millimeterangabe. Wenn ihr ein “+” oder “–” vorangestellt ist, bedeutet die Zahl eine (zum jeweiligen Ausgangspunkt der Teilbewegung) relative Distanz.

- Beispiel:

(200,100,100)
5 L (–100,+200,0)
20 C (+700,+700,+1200) (+0,+1400,+0)
E

ATG-Beispiel

- Bedeutung:

Start beim Punkt

x=200, y=100, z=100.

Lineare Bewegung in 5 Schritten zum Punkt

x=100, y=300, z=0.

Kreisbewegung in 20 Schritten zum Punkt

x=100, y=1700, z=0,

wobei der Punkt

x=800, y=1000, z=1200

auf der Kreisbahn liegen soll.

- Nachteile von verbalen Beschreibungen:
  - oft unklar; nur durch Beispiele verständlich
  - ungenau (wo sind Leerzeichen vorgeschrieben, bzw. wo dürfen welche stehen?)
  - Formulierungsregeln mit Nebenbedingungen und Bedeutungsangaben vermischt

ATG-Beispiel

Robo = Position { Move } “E” .

Position = “(” Coordinate “,” Coordinate “,” Coordinate “)” .

Move = Number (“L” Position | “C” Position Position) .

Coordinate = [“+” | “–”] Number .

Number = digit { digit } .

Terminalsymbole: “L”, “C”, “E”, “+”, “–”, “(”, “,”, “)”, “ ”
Terminalklasse: digit
Nonterminalsymbole: Robo, Position, Move, Coordinate, Number, Space

SOFTWARE RESEARCH LAB
Universität Salzburg

ATG-Beispiel

Robo erkennt eine vollständige Bewegungsbeschreibung

Position <x0, y0, z0, x, y, z>

erkennt eine Position mit den Koordinaten x, y und z.

x0, y0 und z0 bedeuten die Ursprungsposition, auf die sich relative Koordinatenangaben beziehen.

Move <x0, y0, z0, x, y, z>

erkennt eine Bewegung beginnend bei (x0,y0,z0) und liefert den Endpunkt (x,y,z) der Bewegung.

Coordinate <v0, v>

erkennt eine Koordinate mit dem Wert v. v0 ist ein Ursprungswert, auf den sich relative Angaben beziehen.

Number <v>

erkennt eine ganze Dezimalzahl mit dem Wert v.

Space

erkennt eine beliebig lange Folge von Leerzeichen, die auch leer sein kann.

digit <d> erkennt eine Ziffer mit dem Wert d.

ATG-Beispiel

```
Robo =
    Space
    Position <0, 0, 0, x0, y0, z0>
                    sem
                            InitMoves(x0, y0, z0)
                    endsem
    {     Move <x0, y0, z0, x, y, z>
                    sem
                            x0:=x; y0:=y; z0:=z
                    endsem
    }     “E”  Space  .
```

ATG-Beispiel

```
Position <x0, y0, z0, x, y, z> =
    "("    Space  Coordinate <x0, x>
    ","    Space  Coordinate <y0, y>
    ","    Space  Coordinate <z0, z>
    ")"    Space  .

Coordinate <v0, v> =
                    sem sign:=' ' endsem
    [     "+"       sem sign:='+' endsem
    |     "–"       sem sign:='–' endsem
    ]
    Number <v>  sem
                          if sign='+' then v:=v0+v;
                          else if sign='–' then v:=v0–v
                    endsem
    .
```


SOFTWARE RESEARCH LAB

Universität Salzburg

ATG-Beispiel

```
Move <x0, y0, z0, x, y, z> =
    Number <steps>
    (     "L"  Space
          Position <x0, y0, z0, x, y, z>
                sem
                      AddLine(x, y, z, steps)
                endsem
    |     "C"  Space
          Position <x0, y0, z0, x1, y1, z1>
          Position <x0, y0, z0, x, y, z>
                sem
                      AddArc(x, y, z,
                         x1, y1, z1, steps)
                endsem
    ) .
```


SOFTWARE RESEARCH LAB

Universität Salzburg

ATG-Beispiel

```
Number <v> =
    digit <d>      sem v:=d endsem
    {     digit <d>          sem v:=10*v+d endsem
    }
    Space  .

Space =
    { " " } .
```


SOFTWARE RESEARCH LAB

Universität Salzburg

ATG-Beispiel

Semantische Routinen:

Datenkapsel Moves zur Verwaltung einer Liste von Teilbewegungen mit folgenden Zugriffsroutinen:

InitMoves(x0, y0, z0)
Initialisiert die Teilbewegungsliste mit dem Startpunkt (x0,y0,z0).

AddLine(x, y, z, steps)
Erweitert die Teilbewegungsliste um eine lineare Bewegung zum Punkt (x,y,z) in steps Schritten.

AddArc(x, y, z, x1, y1, z1, steps)
Erweitert die Teilbewegungsliste um eine kreisförmige Bewegung über den Punkt (x1,y1,z1) zum Punkt (x,y,z) in steps Schritten.