4. Syntaktische Analyse

Eingabe: Symbolfolge

Aufgaben:

Ableitung bilden mit konkreter Grammatik (parsing)

Strukturbaum aufbauen mit abstrakter Grammatik

Fehlerbehandlung

Ausgabe: Strukturbaum

Übersetzermodul Parser (dt. Zerteiler):

Kellerautomat; hier nur deterministische

kann bei Anwenden einer Produktion Aktion ausführen

(Baumknoten erzeugen)

spezifiziert durch konkrete KFG mit Aktionen

Zielbezogene Parser:

Linksableitung; Baum top-down oder bottom-up

Quellbezogene Parser:

Rechtsableitung rückwärts; Baum bottom-up

Strukturbaum (vergröberter Ableitungsbaum):

als Datenstruktur für Translationsphase, oder

implizit, Aufrufe von Operationen der Translationsphase

Vorlesung Übersetzer WS 97/98 / Folie 20

Ziele:

- Zusammenhang zwischen den Aufgaben "Ableitung bilden" und "Baum aufbauen" erkennen.

in der Vorlesung:

- Beispiele für Grammatik mit Aktionen, Ableitung, Strukturbaum

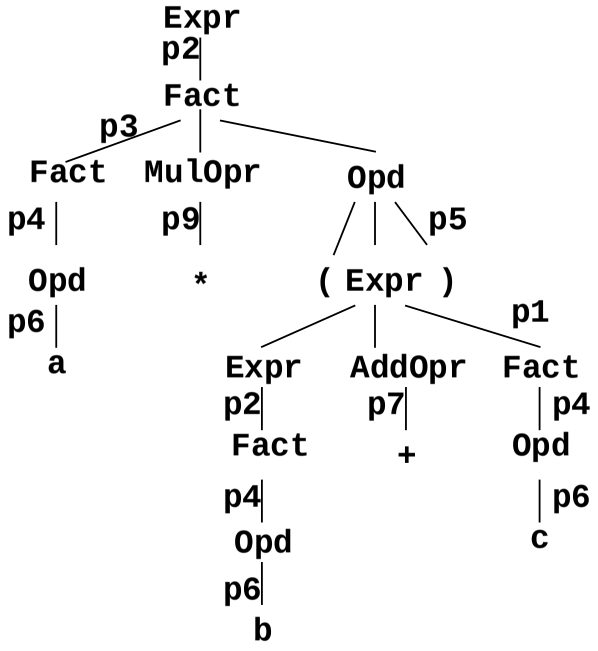
nachlesen:

Kastens / Übersetzerbau, Abschnitt 4.1; kontext-freie Grammatiken aus Theoretischer Informatik wiederholen; Präfix, Postfix aus Informatik A wiederholen

Beispiel: Ausdrucksgrammatik

Name	Produktion		Aktion
p1:	Expr	::= Expr AddOpr Fact	BinEx
p2:	Expr	::= Fact	
p3:	Fact	::= Fact MulOpr Opd	BinEx
p4:	Fact	::= Opd	
p5:	Opd	::= '(' Expr ')'	
p6:	Opd	::= Ident	IdEx
p7:	AddOpr	::= '+'	PlusOpr
p8:	AddOpr	::= '-'	MinusOpr
p9:	MulOpr	::= '*'	TimesOpr
p10:	MulOpr	::= '/'	DivOpr

Ableitungsbaum für `a * (b + c)`



Konkrete und abstrakte Syntax

konkrete Syntax	**abstrakte Syntax**
KFG	KFG
definiert Struktur der Quellprogramme	definiert Strukturbäume
eindeutig	i.a. mehrdeutig
spezifiziert Parser, Ableitung	Grundlage für Translationsphase
Aktionen spezifizieren	Baumaufbau
Kettenproduktionen `Expr ::= Fact`	unnötige weglassen
haben keine Aktion	
beteiligte Symbole	zu Symbolklasse vereinigt `Exp = {Expr, Fact}`
gleiche Aktion zu strukturgleichen Produktionen: `Expr ::= Expr AddOpr Fact`	`&BinEx`
`Fact ::= Fact MulOpr Opd`	`&BinEx`
Terminale	unnötige weglassen, keine Baumknoten

Aus konkreter Syntax und Symbolklassen sind die abstrakte Syntax und die Aktionen automatisch generierbar.

Vorlesung Übersetzer WS 97/98 / Folie 21

Ziele:

- Rolle der abstrakten Syntax
- Mit kontext-freien Grammatiken kann man Bäume spezifizieren!

in der Vorlesung:

- Beispiel: Ausdrucksgrammatik

nachlesen:

Kastens / Übersetzerbau, Abschnitt 4.1

Übungsaufgaben:

- Abstrakte Syntax aus konkreter und Symbolklassen herstellen.
- Dafür Eli anwenden. Aufgabe 9

Verständnisfragen:

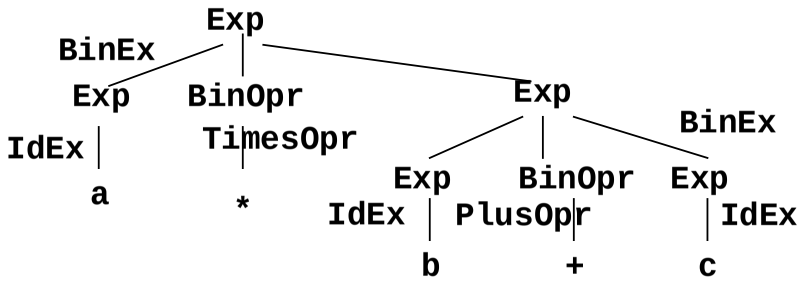
- Warum stört es nicht, daß; eine abstrakte Syntax i.a. mehrdeutig ist?
- Warum geht bei der Repräsentation von Ausdrücken durch einen abstrakten Strukturbaum keine Information verloren?
- Geben Sie andere Beispiele, in denen sich konkrete und abstrakte Syntax unterscheiden.
- Welche Rolle hat die abstrakte Syntax im Kalkül "denotationale Semantik"?

Beispiel: abstrakte Ausdrucksgrammatik

Name **Produktion**

```
BinEx:     Exp    ::= Exp BinOpr Exp
IdEx:      Exp    ::= Ident
PlusOpr:   BinOpr ::= '+'
MinusOpr:  BinOpr ::= '-'
TimesOpr:  BinOpr ::= '*'
DivOpr:    BinOpr ::= '/'
```

Strukturbaum für `a * (b + c)`



Symbolklassen:

```
Exp = { Expr, Fact, Opd }
BinOpr = { AddOpr, MulOpr }
```

Aktionen der konkreten Syntax
Produktionen der abstrakten Syntax

keine Aktion an konkreter Produktion
kein Knoten im Strukturbaum

Recursive Descent Parser

dt.: rekursiver Abstieg; zielorientierte, prädiktive Methode

KFG in Menge von Funktionen transformieren:

Nichtterminal	Funktion
alternative Produktionen	Zweige im Rumpf
Nichtterminal auf rechter Seite	Aufruf
Terminal auf rechter Seite	akzeptiere Symbol und lies
Entscheidungsmenge zu Produktion	Verzweigungsentscheidung

Beispiel:

```
p1:  Stmt ::= Variable ':=' Expr
p2:       |   'while' Expr 'do' Stmt

void Stmt ()
{  switch (CurrSymbol)
   { case Entscheidungsmenge für p1:
          Variable ();
          accept (assignSym);
          Expr ();
          break;
     case Entscheidungsmenge für p2:
          accept (whileSym);
          Expr ();
          accept (doSym);
          Stmt ();
          break;
     default: Fehlerbehandlung;
} }
```

Vorlesung Übersetzer WS 97/98 / Folie 22

Ziele:

- Konstruktionsschema: Recursive Descent

in der Vorlesung:

- Erläuterung des Prinzips

nachlesen:

Kastens / Übersetzerbau, Abschnitt 4.2

Verständnisfragen:

- Wo ist der Keller des Kellerautomaten, und was sind seine Zustände?
- Wo fügt man Aktionen in die Funktionen ein, um Postfix oder Präfix zu erzeugen?

Grammatikbedingung für Recursive Descent

Eine Grammatik ist **stark LL(1)**, wenn für alle Produktionenpaare mit gleicher linker Seite die Entscheidungmengen disjunkt sind:

	Entscheidungmengen
A ::= u	Anf (u Folge (A))
A ::= v	Anf (v Folge (A))

Anfangs- und Folgemengen:

Anf (u) := {t ∈ T | es gibt u $\not\Rightarrow^*$ tv } ∪ {ε} falls u ⇒* ε existiert

Folge (A) := {t ∈ T | es gibt S ⇒* uAv und t ∈ Anf (v)}

Konsequenz:

Starke LL(1)-Grammatiken haben keine links-rekursiven Produktionen, und keine Produktionen mit gleichen Anfängen.

EBNF-Konstrukte, z.B.:

A ::= u (v)* w

in Funktion für A:

... u ... while (CurrSymbol in Anf (v)) { ... v ...} ...w ...

Es muß gelten: Anf (v) und Anf (w Folge (A)) sind disjunkt.

Vorlesung Übersetzer WS 97/98 / Folie 23

Ziele:

- einfach zu prüfende Grammatikbedingung

in der Vorlesung:

- Voraussetzung und Grenzen von Recursive Descent zeigen
- Erweiterung für EBNF

nachlesen:

Kastens / Übersetzerbau, Abschnitt 4.2, LL(k)-Bedingungen, Berechnung von Anfangs- und Folgemengen

Übungsaufgaben:

- Geben Sie analoge Regeln für andere EBNF-Konstrukte an.

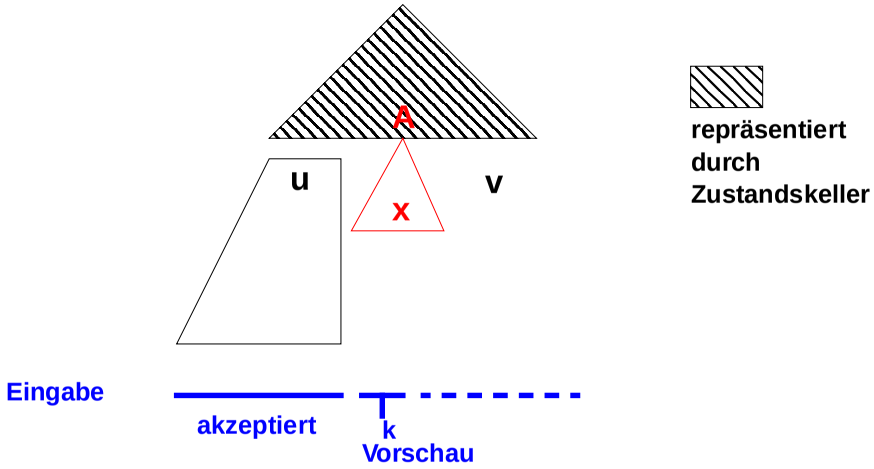
Verständnisfragen:

- Warum ist die EBNF-Erweiterung für die Anwendbarkeit besonders wichtig?

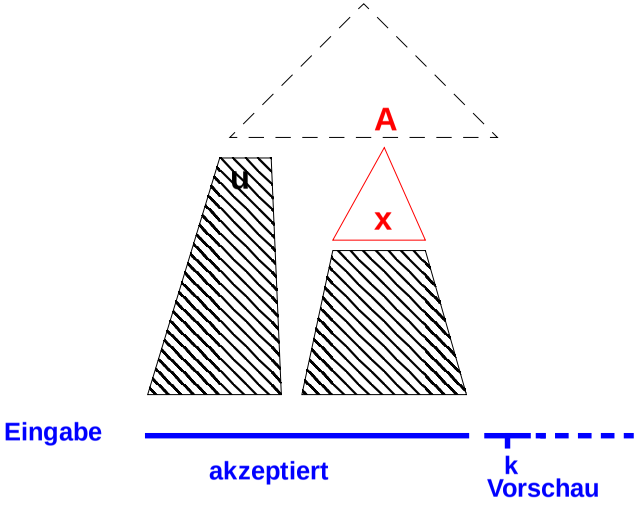
Vergleich: quellbezogen - zielbezogen

Information des Kellerautomaten bei der Entscheidung, die Produktion `A ::= x` anzuwenden.

zielbezogen, prädiktiv, Linksableitung:



quellbezogen, Rechtsableitung rückwärts:



Vorlesung Übersetzer WS 97/98 / Folie 24

Ziele:

- Entscheidungssituation der Kellerautomaten
- Quellbezogene sind mächtiger, ihre Grammatiken weniger restriktiv

in der Vorlesung:

- Hinweis auf die Rolle des Kellers

nachlesen:

Kastens / Übersetzerbau, Abschnitt Text zu Abb. 4.2-1, 4.3-1

LR(k)-Bedingung

Eine kontext-freie Grammatik ist eine LR(k)-Grammatik mit

k ≥ 0, wenn für alle Paare von Rechtsableitungen

Z =>* u A w => u x w **u, u', x, x' ∈ V***

Z =>* u' B w' => u' x' w' **w, w', v ∈ T***

gilt:

falls es ein v gibt, so daß

u x v = u' x' w'

und die ersten k Zeichen von w und v gleich sind,

dann sind auch A = B, x = x', u = u'

(d. h. der letzte Ableitungsschritt ist gleich)

Diese Bedingung ist nicht konstruktiv und so i.a. nicht prüfbar!

Automat konstruieren, deterministisch g.d.w. LR(k).

Mächtigkeit der Grammatikklasse:

LL(k) => LR(k) => eindeutig

Praktische Bedeutung haben Unterklassen wie LALR(1).

Vorlesung Übersetzer WS 97/98 / Folie 25

Ziele:

- LR(k)-Bedingung verstehen

in der Vorlesung:

- Struktur der Bedingung erläutern, Beispiel
- Entscheidungsschema verfeinern

nachlesen:

Kastens / Übersetzerbau, Abschnitt 4.3

Verständnisfragen:

- Warum ist die LR(k)-Bedingung so nicht allgemein prüfbar?
- Warum muß man die Ableitungen aus der LR(k)-Bedingung rückwärts lesen, um den LR(k)-Algorithmus zu verstehen?

LR(1)-Automat

Zustände werden gekellert.

Operationen: **shift** lesen und Zustand kellern

reduce mit Produktion reduzieren, Zustände entkellern

Ein **Zustand** repräsentiert eine **Menge von Situationen**.

Eine **Situation** repräsentiert den Analysestand bzgl. einer Produktion:

[A ::= u . v R]

Analyseposition **.** **R** erwarteter **Rechtskontext**
Menge von Terminalen, die folgen können, wenn mit der Produktion reduziert ist.

Reduktionssituation:

[A ::= u v . R]

Automat reduziert,
falls nächstes Zeichen der Eingabe in R

Vorlesung Übersetzer WS 97/98 / Folie 26

Ziele:

- Struktur des LR-Automaten

in der Vorlesung:

- Menge von Situationen, Rechtskontext erläutern

nachlesen:

Kastens / Übersetzerbau, Abschnitt 4.3

Verständnisfragen:

- Was enthält die Rechtskontextmenge bei einem LR(3)-Automaten?
- Wie kodiert man einen LR-Zustand bei der Automatenimplementierung?