



Compiler

Kapitel 4

Syntaktische Analyse

Folie: 1

Kapitel 4

Syntaktische Analyse: LL Parsing



Autor:
Aho et al.

© Pearson Studium 2008



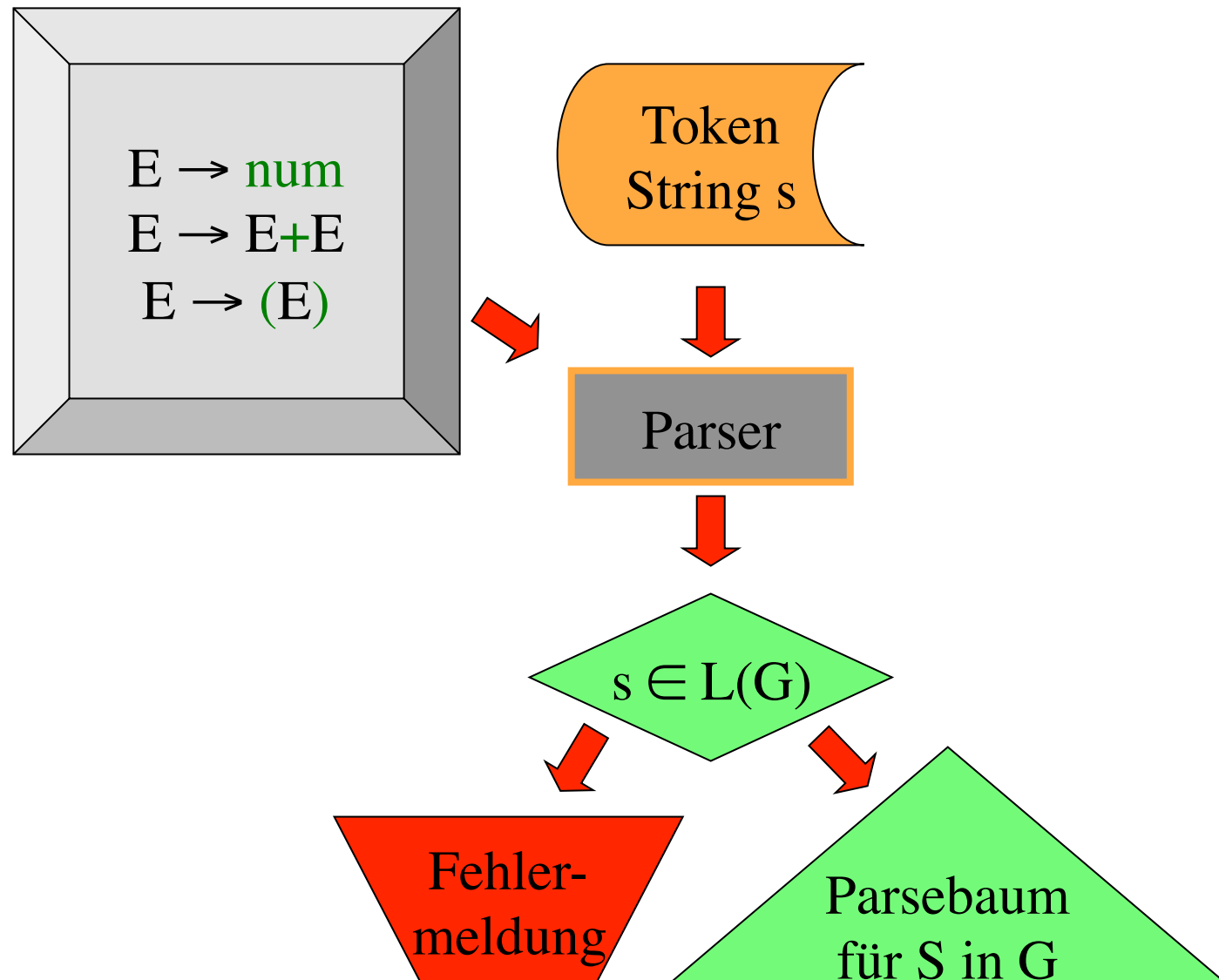
Compiler

Kapitel 4

Syntaktische Analyse

2

Was ist Parsing?





Compiler

Kapitel 4

Syntaktische Analyse

Beispiel

- $S \rightarrow (S)S \mid \varepsilon$
- Was ist die generierte Sprache?
- Beweis?



Autor:
Aho et al.

© Pearson Studium 2008

Siehe Aho, Beispiel 4.5, Seite 247



Compiler

Kapitel 4

Syntaktische Analyse

Top-Down/Bottom-up Parsing

$$Ex \rightarrow Nat \mid (Ex) \mid Ex + Ex \mid Ex * Ex$$

- **Top-down:**

- Man fängt mit dem “Startsymbol” an
- Man versucht den String abzuleiten

- **Bottom-up:**

- Man fängt mit dem String an
- Man versucht das Startsymbol zu erzeugen
- Produktionen werden von rechts nach links angewandt

Nat + Nat
Ex + Nat
Ex + Ex
Ex



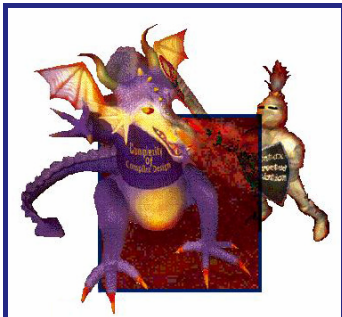
Compiler

Kapitel 4

Syntaktische Analyse

Left-to-right/Right-to-left

- Left-to-right \Rightarrow
 - Untersuche den String von links nach rechts
- Right-to-left \Leftarrow
 - Man fängt mit dem letzten Symbol an und arbeitet rückwärts
 - Nicht für Compiler verwendet (Effizienz)



Compiler

Kapitel 4

Syntaktische Analyse

Einfaches Beispiel

- $S \rightarrow a S b \mid a S c \mid d$
- Strings
 - aadbc
 - aaadbccb
- Welche Regeln müssen angewandt werden?
- Komplexität?



Compiler

Kapitel 4

Syntaktische Analyse

Folie: 7

Top-Down Parsing

$$E \rightarrow T E'$$

$$E' \rightarrow + T E' \mid \epsilon$$

$$T \rightarrow F T'$$

$$T' \rightarrow * F T' \mid \epsilon$$

$$F \rightarrow (E) \mid id$$

id+id*id



Compiler

Kapitel 4

Syntaktische Analyse

Folie: 8



Autor:
Aho et al.

© Pearson Studium 2008

id+id*id

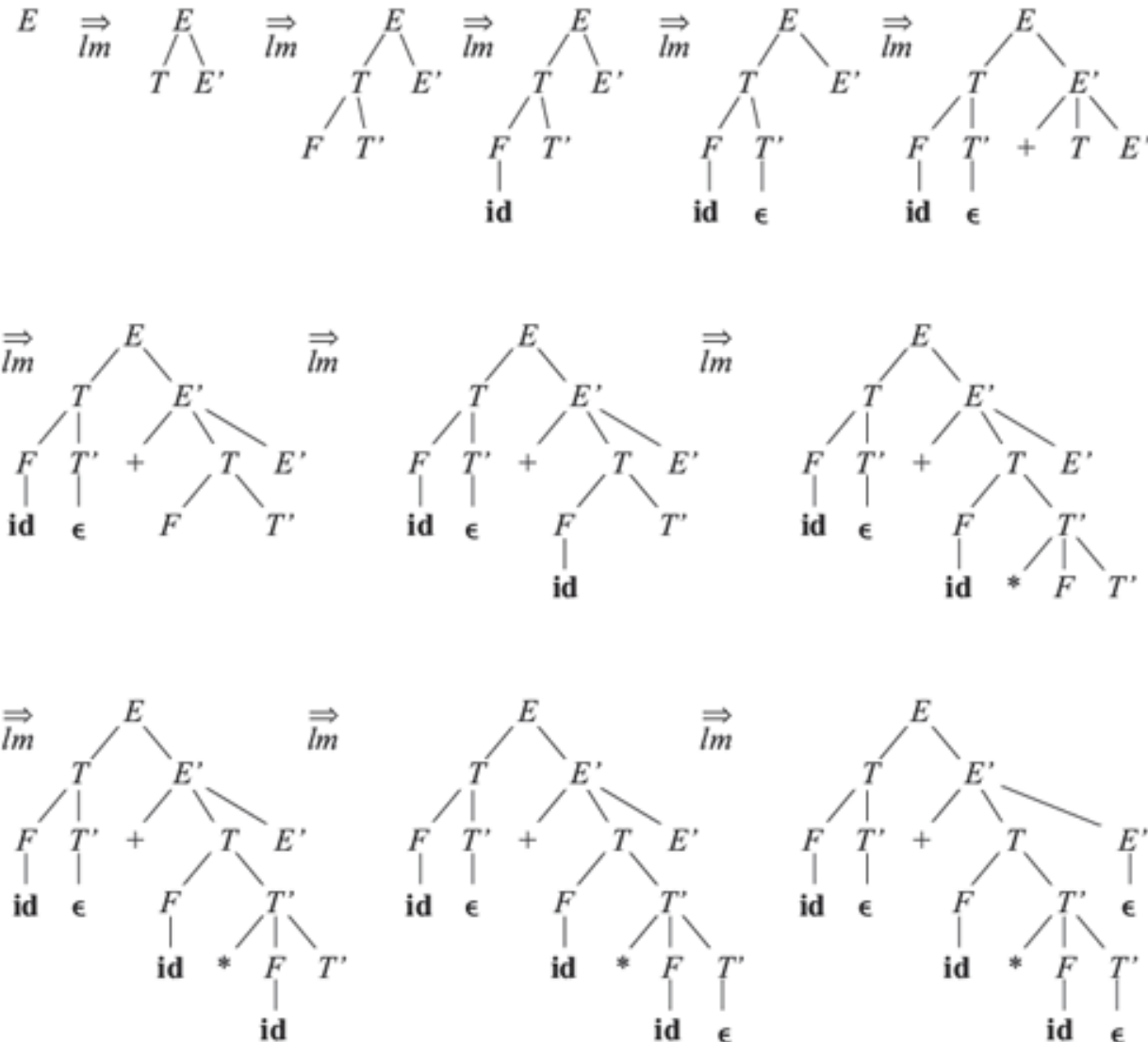
Top-Down Parsing

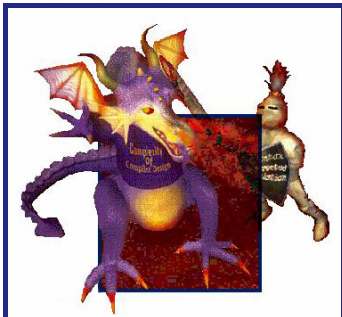
$$E \rightarrow T E'$$

$$E' \rightarrow + T E' \mid \epsilon$$

$$T \rightarrow F T'$$

$$T' \rightarrow * F T' \mid \epsilon$$

$$F \rightarrow (E) \mid \text{id}$$




Compiler

Kapitel 4

Syntaktische Analyse

Folie: 9

PEARSON
Studium **it**
informatik

Autor:
Aho et al.

© Pearson Studium 2008

Rekursiv absteigendes Parsing

- 1 Prozedur pro Nichtterminal

```
void A() {
```

```
1)   Wähle eine A-Produktion,  $A \rightarrow X_1 X_2 \dots X_k$ ;
```

```
2)   for (  $i = 1$  bis  $k$  ) {
```

```
3)       if (  $X_i$  ist ein Nichtterminal )
```

```
4)           rufe die Prozedur  $X_i()$  auf;
```

```
5)       else if (  $X_i$  ist gleich dem aktuellen Eingabesymbol  $a$  )
```

```
6)           setze die Eingabe auf das folgende Symbol;
```

```
7)       else /
```

```
        }
```

```
    }
```

mit Backtracking:

wähle eine andere A-Produktion aus;
Fehler falls alle Produktionen probiert



Compiler

Kapitel 4

Syntaktische Analyse

Folie: 10

PEARSON
Studium **it**
informatik

Autor:
Aho et al.

© Pearson Studium 2008

Beispiel

- String: "cad"

$$S \rightarrow c A d$$

$$A \rightarrow a b \mid a$$

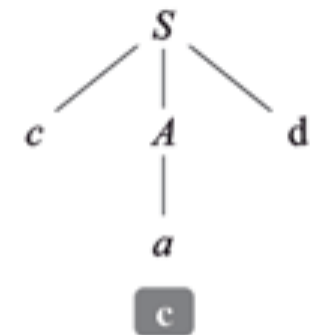
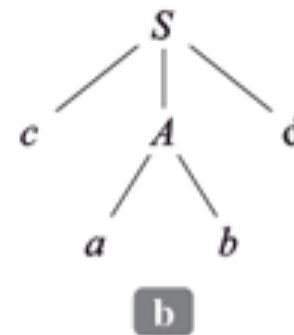
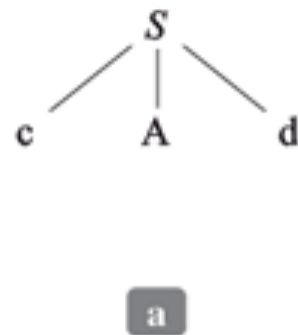
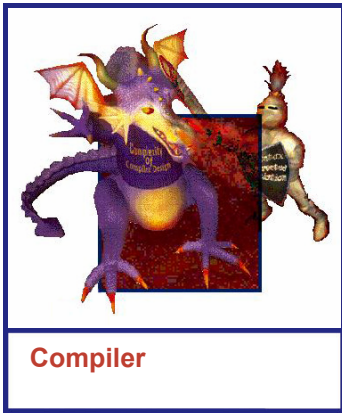


Abbildung 4.14: Schritte beim Top-Down-Parsing



Compiler

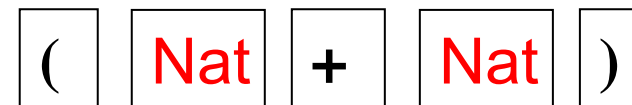
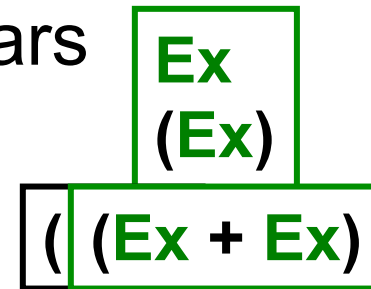
Kapitel 4

Syntaktische Analyse

Prädiktives rekursiv absteigendes Parsing

$Ex \rightarrow Nat \mid (Ex) \mid Ex + Ex$

- Top-down, leftmost derivation, left-to-right
- **Predict** which production to apply based on current token (or next k tokens)
- Only works for certain grammars
 - LL(1), LL(k)
- Easy to implement by hand
- Efficient (if it works):
 - no backtracking





Compiler

Kapitel 4

Syntaktische Analyse

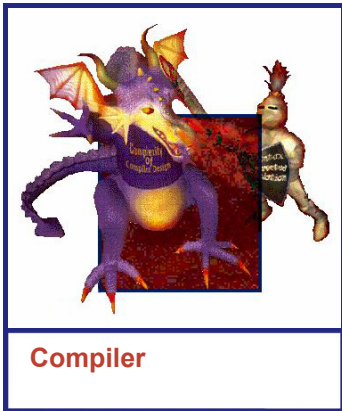
Ein kleines Beispiel

- Hier: für jedes Nichtterminal
 - Erstes Symbol der rechten Seite jeweils:
eindeutiges Terminalsymbol !

```
S → if E then S else S
S → begin S L
S → print E
L → end
L → ; S L
E → num = num
```

```
S
begin S L
begin print E L
begin print num = num L
begin print num = num end
```

```
begin print num = num end
```



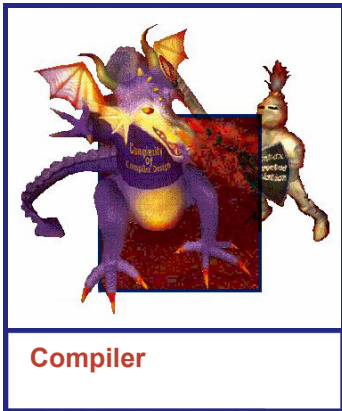
Übersetzung nach Java/C

- Für jedes Nichtterminal
 - Eine Prozedur mit
 - 1 Switch,
 - 1 Case pro Produktion

S → **if E then S else S**
S → **begin S L**
S → **print E**
L → **end**
L → **; S L**
E → **num = num**

```
void S() {switch(tok) {  
    case IF:  eat(IF); E();  
              eat(THEN); S(); eat(ELSE); S(); break;  
    case BEGIN: eat(BEGIN); S(); L(); break;  
    case PRINT: eat(PRINT); E(); break;  
    default: error();  
}}
```

```
void eat(ex) { if(tok== ex) tok = System.in.read(); else  
              { System.out.print("expected:"); System.out.print(ex);} }
```



Erweitertes Beispiel

```
S → if S then S else S  
S → begin S L  
S → Syscall  
Syscall → print E  
Syscall → read V  
Syscall → open FD  
...
```

- Erstes Symbol nicht immer ein **Terminal**
- Prädiktives Parsing trotzdem anwendbar
 - Bestimme $\text{FIRST}(\gamma)$
 - $\{a \in T \mid \gamma \Rightarrow^* a\alpha\}$
 - $\text{FIRST}(\text{Syscall}) = \{\text{print}, \text{read}, \text{open}, \dots\}$
 - $\text{FIRST}(\text{if S then S else S}) = \{\text{if}\}$
 - $\text{First}(\cdot)$ of all rhs for the same non-terminal symbol should be disjoint



Erweitertes Beispiel II

Comp

Kapitel

Syntak

- $S \rightarrow \text{if } E \text{ then } S \text{ else } S$
- $S \rightarrow \text{begin } S L$
- $S \rightarrow \text{Syscall}$
- $\text{Syscall} \rightarrow \text{print } E$
- $\text{Syscall} \rightarrow \text{read } V$
- $\text{Syscall} \rightarrow \text{open } FD$
- ...

$\text{First}(\text{if } E \text{ then } S \text{ else } S) = \{\text{if}\}$

$\text{First}(\text{begin } S L) = \{\text{begin}\}$

$\text{First}(\text{Syscall}) = \{\text{print, read, open}\}$

\Rightarrow C/Java Code für S ?

```
void S() {switch(tok) {
    case IF:    eat(IF); E();
               eat(THEN); S(); eat(ELSE); S(); break;
    case BEGIN: eat(BEGIN); S(); L(); break;
    case PRINT: case READ: case OPEN:
               Syscall(); break;
    default:   error();
}}}
```



Compiler

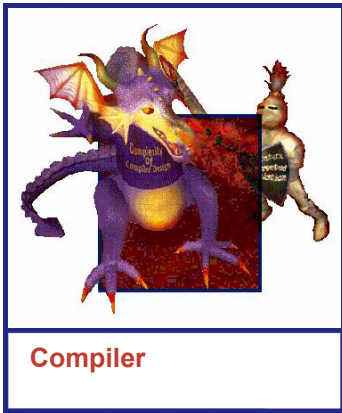
Kapitel 4

Syntaktische Analyse

ϵ -Produktionen: Ein einfaches Beispiel

- Was passiert hier:
- $T \rightarrow \text{program } S \text{ end}$
- $S \rightarrow \text{if } S \text{ then } S E$
- $S \rightarrow \text{begin } S \text{ end}$
- $S \rightarrow \epsilon$
- $E \rightarrow \text{else } S \text{ fi}$
- $E \rightarrow \text{fi}$

- Wann wendet man Regel 3 für S an ??



Compiler

Kapitel 4

Syntaktische Analyse

Nullable Symbols

- FIRST reicht nicht aus (bei ϵ):

- Ableitung

- FDef
- Fun (Arg) : Type ;
- **function ID** (Arg) : Type ;

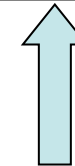
- Welche Regel für Arg ?

- FIRST(**ID** : Type ; Arg) = {**ID**}, FIRST(ϵ) = {}
- **Report error in input ??**

- **Nein: ϵ ist nullable und “)” can auf Arg folgen**

```
FDef → Fun ( Arg ) : Type ;  
Arg → ID : Type ; Arg  
Arg →  $\epsilon$   
Type → integer  
Type → real  
Type → boolean  
Fun → function ID
```

```
function ID ( ) : real ;
```



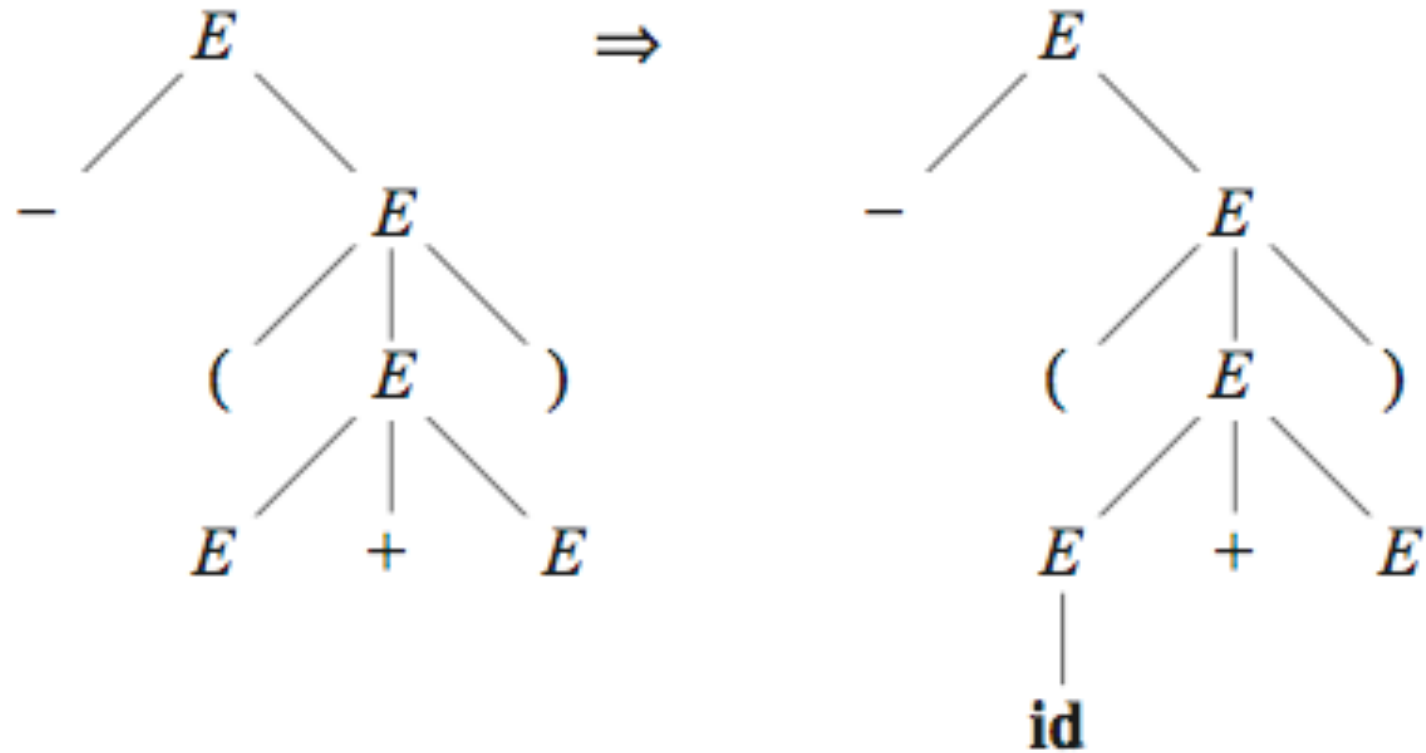


Compiler

Kapitel 4

Syntaktische Analyse

Folie: 18





Compiler

Kapitel 4

Syntaktische Analyse

PEARSON
Studium **it**
informatik

Autor:
Aho et al.

© Pearson Studium 2008

Bausteine für prädiktives rekursiv absteigendes Parsing:

- nullable(**X**):
 - wahr falls für Nichtterminal **X** gilt: $X \Rightarrow^* \epsilon$
- FIRST(γ):
 - set of terminal symbols that can begin any string produced by γ
 - $\{a \in T \mid \gamma \Rightarrow^* a\alpha\}$
- FOLLOW(**X**):
 - set of terminal symbols **t** that can immediately follow **X**; i.e.,
 - $= \{t \in T \mid S \Rightarrow^* \alpha X t \beta\} \cup \{\$ \mid S \Rightarrow^* \alpha X\}$



Compiler

Kapitel 4

Syntaktische Analyse

Folie: 20

First und Follow

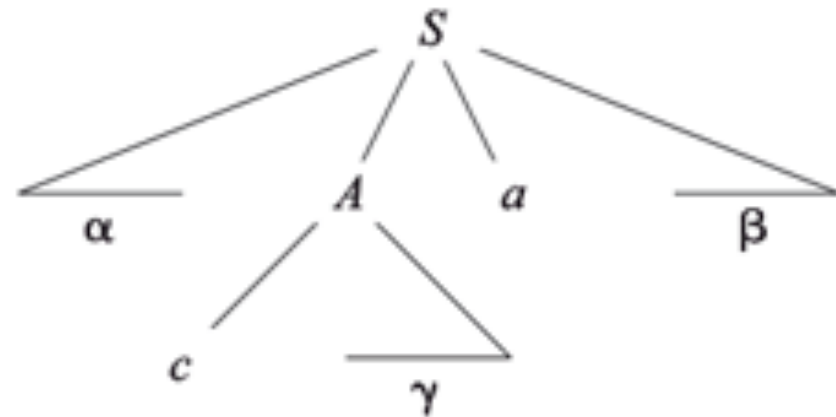


Abbildung 4.15: Terminal c ist in $\text{FIRST}(A)$ und a in $\text{FOLLOW}(A)$.



Compiler

Kapitel 4

Syntaktische Analyse

Folie: 21

LL(1)

- Grammatik ist LL(1) wenn für alle Produktionen $A \rightarrow \alpha$, $A \rightarrow \beta$ gilt:
 - $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$
 - $\neg \alpha \Rightarrow^* \epsilon \vee \neg \beta \Rightarrow^* \epsilon$
 - wenn $\alpha \Rightarrow^* \epsilon$ dann gilt:
 $\text{FOLLOW}(A) \cap \text{FIRST}(\beta) = \emptyset$
 - wenn $\beta \Rightarrow^* \epsilon$ dann gilt
 $\text{FOLLOW}(A) \cap \text{FIRST}(\alpha) = \emptyset$



Compiler

Kapitel 4

Syntaktische Analyse

Berechnung von nullable(**X**)

```
for all symbols X do
```

```
    nullable(X) := false;
```

```
repeat
```

```
    change := false;
```

```
    for every production  $X \rightarrow s_1 \dots s_k$ 
```

```
    do
```

```
        if nullable(X)=false and
```

```
         $s_1 \dots s_k$  are all nullable then
```

```
            {nullable(X) := true;
```

```
              change := true;}  
    until change = false
```

true if k=0 !

```
until change = false
```

$Z \rightarrow d$

$Z \rightarrow XYZ$

$Y \rightarrow \epsilon$

$Y \rightarrow c$

$X \rightarrow Y$

$X \rightarrow a$



Compiler

Kapitel 4

Syntaktische Analyse

Berechnung von FIRST

```
for all non-terminals X do FIRST  
  (X) := {};
```

```
for all terminals T do FIRST(T) :=  
  {T};
```

repeat

```
for every production X → s1 ... sk  
do
```

```
{ FIRST(X) := FIRST(X) ∪ FIRST(s1);
```

```
for i := 2 to k do
```

```
if nullable(si-1) then
```

```
  FIRST(X) := FIRST(X) ∪ FIRST
```

```
(si);
```

```
else exit; }
```

until no more changes

Y: {c}

X: {a,c}

Z: {a,c,d}

Übung:

Z → d

Y → ε

X → **Y**

Z → **XYZ**

Y → c

X → a



Compiler

Kapitel 4

Syntaktische Analyse

FIRST(γ) für Strings

- $\text{FIRST}(\mathbf{X}\gamma) = \text{FIRST}(\mathbf{X})$
if not nullable(\mathbf{X})
- $\text{FIRST}(\mathbf{X}\gamma) = \text{FIRST}(\mathbf{X}) \cup \text{FIRST}(\gamma)$
if nullable(\mathbf{X})
- Beispiel: $\text{FIRST}(\mathbf{X}\mathbf{Y}\mathbf{Z}) =$
 - $\{a,c\} \cup \text{FIRST}(\mathbf{Y}\mathbf{Z}) =$
 - $\{a,c\} \cup \{c\} \cup \text{FIRST}(\mathbf{Z}) =$
 - $\{a,c\} \cup \{c\} \cup \{a,c,d\} = \{a,c,d\}$

$\mathbf{Y}: \{c\}$
 $\mathbf{X}: \{a,c\}$
 $\mathbf{Z}: \{a,c,d\}$

$\mathbf{Z} \rightarrow d$ $\mathbf{Y} \rightarrow \epsilon$ $\mathbf{X} \rightarrow \mathbf{Y}$
 $\mathbf{Z} \rightarrow \mathbf{X}\mathbf{Y}\mathbf{Z}$ $\mathbf{Y} \rightarrow c$ $\mathbf{X} \rightarrow a$



Berechnung von FOLLOW

FIRST:
 Y: {c}
 X: {a,c}
 Z: {a,c,d}

Y: {a,c,d}
 X: {a,c,d}
 Z: {}

```

for all symbols X do FOLLOW(X) := {};
repeat
  for every production X → s1 ... sk do
    for i := 1 to k do
      { FOLLOW(si) := FOLLOW(si) ∪ FIRST(si+1);
        for j := i+2 to k do
          if nullable(si+1) and ... nullable(sj-1)
          then
            FOLLOW(si) := FOLLOW(si) ∪ FIRST(sj);
          if i=k or
            (nullable(si+1) and...nullable(sk)) then
            FOLLOW(si) := FOLLOW(si) ∪ FOLLOW
              (X);
        }
  until no more changes
  
```

$Z \rightarrow d$ $Y \rightarrow \epsilon$ $X \rightarrow Y$
 $Z \rightarrow XYZ$ $Y \rightarrow c$ $X \rightarrow a$

Übung:



Compiler

Kapitel 4

Syntaktische Analyse

Benutzung von nullable, First, Follow: Erstellen eines Parsers

- Erstellen einer **Parsertabelle**:
 - Reihen: Nichtterminale **X**
 - Kolonnen:
Terminalsymbole **c** on input
 - Produktion **X** \rightarrow γ anwendbar
 - **c** \in FIRST(γ) oder
 - Nullable(γ) und **c** \in FOLLOW(**X**)
- Übersetzung nach C/Java/...:
 - 1 rekursive Prozedure pro Nichtterminal (siehe vorher)



Compiler

Unser Beispiel:

FIRST

Y: {c}

X: {a,c}

Z: {a,c,d}

Prädiktive Parsertabelle:

	a	c	d
X	$X \rightarrow a$ $X \rightarrow Y$	$X \rightarrow Y$	$X \rightarrow Y$
Y	$Y \rightarrow \epsilon$	$Y \rightarrow \epsilon$ $Y \rightarrow c$	$Y \rightarrow \epsilon$
Z	$Z \rightarrow XYZ$	$Z \rightarrow XYZ$	$Z \rightarrow XYZ$ $Z \rightarrow d$

FOLLOW

Y: {a,c,d}

X: {a,c,d}

Z: {}

Konflikte !!

$Z \rightarrow d$	$Y \rightarrow \epsilon$	$X \rightarrow Y$
$Z \rightarrow XYZ$	$Y \rightarrow c$	$X \rightarrow a$

Kapitel 4

Syntaktische Analyse

PEARSON

Studium

informatik

Autor:
Aho et al.

© Pearson Studium 2008



Compiler

Kapitel 4

Syntaktische Analyse

Folie: 28

Anderes Beispiel:

$$E \rightarrow T E'$$

$$E' \rightarrow + T E' \mid \epsilon$$

$$T \rightarrow F T'$$

$$T' \rightarrow * F T' \mid \epsilon$$

$$F \rightarrow (E) \mid id$$

Nichtterminal	Eingabesymbol					
	id	+	*	()	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

Abbildung 4.17: Parsertabelle M zu Beispiel 4.18



Compiler

Kapitel 4

Syntaktische Analyse

Folie: 29

Noch ein anderes Beispiel:

$$S \rightarrow i E t S S' \mid \epsilon a$$

$$S' \rightarrow e S \mid \epsilon$$

$$E \rightarrow b$$

Nichtterminal	Eingabesymbol					
	<i>a</i>	<i>b</i>	<i>e</i>	<i>i</i>	<i>t</i>	<i>\$</i>
<i>S</i>	$S \rightarrow a$			$S \rightarrow iEtSS'$		
<i>E'</i>			$S' \rightarrow \epsilon$ $S' \rightarrow eS$			$S' \rightarrow \epsilon$
<i>E</i>		$E' \rightarrow b$				



Compiler

Kapitel 4

Syntaktische Analyse

Ein komplettes Beispiel: Parsing von Ausdrücken

- $E \rightarrow (E) EC$
- $E \rightarrow \text{Num } EC$
- $EC \rightarrow + E \mid - E \mid * E \mid / E$ Anmerkung:
Keine Prioritäten!
- $EC \rightarrow \varepsilon$
- $\text{Num} \rightarrow 0 \mid 1 \text{ DigStar} \mid 2 \text{ DigStar} \mid \dots$
- $\text{DigStar} \rightarrow \varepsilon$
- $\text{DigStar} \rightarrow 0 \text{ DigStar} \mid 1 \text{ DigStar} \mid \dots$



Compiler

Kapitel 4

Syntaktische Analyse

PEARSON
Studium **it**
informatik

Autor:
Aho et al.

© Pearson Studium 2008

Beispiel: nullable, first, follow

- nullable: EC, DigStar
- $\text{First}(\text{Num}) = \text{First}(\text{DigStar}) = \{0, 1, 2, 3, \dots, 9\}$
- $\text{First}(\text{EC}) = \{+, -, *, /\}$ $\text{First}(E) = \{ (, 0, 1, 2, 3, \dots, 9 \}$
- $\text{Follow}(E) = \{) \}$ $\text{Follow}(\text{EC}) = \{) \}$
- $\text{Follow}(\text{Num}) = \{+, -, *, /,)\}$
- $\text{Follow}(\text{DigStar}) = \{+, -, *, /,)\}$

$E \rightarrow (E) EC$

$E \rightarrow \text{Num} EC$

$EC \rightarrow + E \mid - E \mid * E \mid / E$

$EC \rightarrow \varepsilon$

$\text{Num} \rightarrow 0 \mid 1 \text{ DigStar} \mid 2 \text{ DigStar} \mid \dots$

$\text{DigStar} \rightarrow \varepsilon$

$\text{DigStar} \rightarrow 0 \text{ DigStar} \mid 1 \text{ DigStar} \mid \dots$



Compiler

Kapitel 4

Syntaktische Analyse

Übersetzung nach Java: Infrastruktur

```
public static void advance() throws java.io.IOException {  
    if(tokasint != -1) { tokasint = System.in.read();  
        tok = (char) tokasint; }  
}
```

```
public static void eat(char c) throws java.io.IOException {  
    if(tok==c) advance();  
    else { System.out.print("*** parser error, expected <");  
        System.out.print(c);  
        System.out.print("> instead of <");  
        if(tokasint== -1)  
            System.out.print("EOF");  
        else  
            System.out.print(tok);  
        System.out.println(">");  
    }  
}
```



Compiler

Kapitel 4

Syntaktische Analyse

PEARSON
Studium **it**
informatik

Autor:
Aho et al.

© Pearson Studium 2008

Der resultierende Parser I

```
public static void Expr() throws java.io.IOException
{
    switch(tok) {
        case '(': /* Expr --> ( Expr ) ExprCont */
            eat('('); Expr(); eat(')'); ExprCont();
            break;
        case '0': case '1': case '2': case '3': case '4':
        case '5': case '6': case '7': case '8': case '9':
            /* Expr --> Num ExprCont */
            Num(); ExprCont();
            break;
        default: eat('('); /* generate error message */
    }
}
```



Compiler

Kapitel 4

Syntaktische Analyse

Der resultierende Parser II

```
public static void ExprCont() throws java.io.IOException
{
    switch(tok) {
    case ')': case (char) (-1): /* eof */
        /* ExprCont --> epsilon */ break;
    case '+': case '*': case '/': case '-':
        /* ExprCont --> (+|*|/|-) Expr */
        advance(); Expr();
        break;
    default: eat(')'); /* generate error message */
    }
}
```



Autor:
Aho et al.

© Pearson Studium 2008

Remember: Follow(EC) = { **)** }, nullable(EC) !!



Compiler

Kapitel 4

Syntaktische Analyse

Der resultierende Parser III

```
public static void Num() throws java.io.IOException
{
    switch(tok) {
    case '0': advance(); break; /* Num --> 0 */
    case '1': case '2': case '3': case '4':
    case '5': case '6': case '7': case '8': case '9':
        /* Num --> (1|2|3|4|5|6|7|8|9) DigStar */
        advance(); DigStar(); break;
    default: eat('0'); /* generate error message */

    }
}
```



Compiler

Kapitel 4

Syntaktische Analyse

PEARSON
Studium **it**
informatik

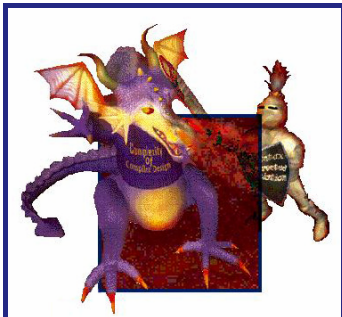
Autor:
Aho et al.

© Pearson Studium 2008

Der resultierende Parser IV

```
public static void DigStar() throws java.io.IOException
{
    switch(tok) {
        case ')': case '+': case '*': case '/': case '-':
        case (char) (-1): /* eof */
            break; /* DigStar --> epsilon */
        case '0': case '1': case '2': case '3': case '4':
        case '5': case '6': case '7': case '8': case '9':
            /* DigStar --> (1|2|3|4|5|6|7|8|9) DigStar */
            advance(); DigStar(); break;
        default: eat('0'); /* generate error message */
    }
}
```

Remember: Follow(EC) = { **),+,-,*,/** }, nullable(DigStar) !!



Compiler

Kapitel 4

Syntaktische Analyse

PEARSON
Studium **it**
informatik

Autor:
Aho et al.

© Pearson Studium 2008

Der resultierende Parser IV

```
public static void main(String[] args)
    throws java.io.IOException {
    System.out.println("LL(1) Parser");
    System.out.println("-----");
    advance();
    System.out.println("Starting parse process...");
    Expr();
    System.out.println("Finished.");
}
```

und nun eine Vorführung !

(Quellcode auf der Webseite erhältlich)



Compiler

Kapitel 4

Syntaktische Analyse

Probleme für das rekursiv absteigende Parsing

- Mehrdeutige Grammatiken
 - Gibt immer Konflikte in Parsertabelle!
- Linksrekursion
 - $E \rightarrow E + ID$ $\text{FIRST}(ID) \subseteq \text{FIRST}(E)$
 - $E \rightarrow ID$ \Rightarrow **Konflikt !**
- Gemeinsamer Präfix
 - $S \rightarrow \text{if } E \text{ then } S \text{ else } S$
 - $S \rightarrow \text{if } E \text{ then } S$ \Rightarrow **Konflikt !**



Compiler

Kapitel 4

Syntaktische Analyse

Linksfaktorisierung

- Wiederhole bis keine Änderung mehr:
 - Für jedes Nicht-Terminal X :
 - Finde den längsten gemeinsamen Präfix α von 2 oder mehr Alternativen
 - Falls $\alpha \neq \varepsilon$, dann ersetze

$$X \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_k \mid \gamma_1 \mid \dots \mid \gamma_n$$

$$X \rightarrow \alpha X' \mid \gamma_1 \mid \dots \mid \gamma_n$$

$$X' \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_k$$



Compiler

Kapitel 4

Syntaktische Analyse

Linksfaktorisierung 2

- $S \rightarrow \text{if } E \text{ then } S \text{ else } S$
- $S \rightarrow \text{if } E \text{ then } S$

Nach Linksfaktorisierung:

$S \rightarrow \text{if } E \text{ then } S R$

$R \rightarrow \text{else } S$

$R \rightarrow \varepsilon$

*Immer noch mehrdeutig
Lösung??*



Compiler

Kapitel 4

Syntaktische Analyse

Dangling Else Problem

- $S \rightarrow \text{Matched} \mid \text{Unmatched}$
- $\text{Matched} \rightarrow \text{if Expr then Matched else Matched} \mid \text{OtherS}$
- $\text{Unmatched} \rightarrow \text{if Expr then S} \mid \text{if Expr then Matched else Unmatched}$



Compiler

Kapitel 4

Syntaktische Analyse

Eliminieren der Linksrekursion

– Umschreiben nach Rechtsrekursion:

- $E \rightarrow E + T$
- $E \rightarrow T$

$$E \rightarrow T E'$$

$$E' \rightarrow + T E'$$

$$E' \rightarrow \varepsilon$$

- $X \rightarrow X \gamma_1$
- $X \rightarrow X \gamma_2$
- $X \rightarrow \alpha_1$
- $X \rightarrow \alpha_2$

$$X \rightarrow \alpha_1 X'$$

$$X \rightarrow \alpha_2 X'$$

$$X' \rightarrow \gamma_1 X'$$

$$X' \rightarrow \gamma_2 X'$$

$$X' \rightarrow \varepsilon$$



Compiler

Kapitel 4

Syntaktische Analyse

Mehrdeutige Grammatiken

$$\text{Ex} \rightarrow \text{Nat} \mid \text{ID} \mid (\text{Ex}) \mid \text{Ex} + \text{Ex} \mid \text{Ex} * \text{Ex}$$

- Lösung 1: Parser “hacken”
 - Prioritäten/Konflikte im Parser Explizit behandeln; nicht empfohlen
- Lösung 2: Grammatik umschreiben
 - Verlangt Nachdenken (siehe “dangling else” Problem von vorher)
 - Nicht immer möglich

$$\begin{aligned} \text{Ex} &\rightarrow \text{Ex} + \text{Term} \mid \text{Term} \\ \text{Term} &\rightarrow \text{Term} * \text{Factor} \mid \text{Factor} \\ \text{Factor} &\rightarrow \text{Nat} \mid \text{ID} \mid (\text{Ex}) \end{aligned}$$



Compiler

Kapitel 4

Syntaktische Analyse

Fehlerbehandlung

- Exception generieren
- Insertion (Einfügung)
 - e.g., print error, pretend everything ok
- Deletion (Löschung)
 - e.g., skip until a token in FOLLOW is reached

```
void S() {switch(tok) {  
  case IF:  eat(IF); E();  
            eat(THEN); S(); eat(ELSE); S(); break;  
  case BEGIN: eat(BEGIN); S(); L(); break;  
  case PRINT: eat(PRINT); E(); break;  
  default:  error_recovery ;           }}  
}
```

PEARSON
Studium

Autor:
Aho et al.

© Pearson Studium 2006



Compiler

Kapitel 4

Syntaktische Analyse

Grammatiken in JavaCC

```
options {
    IGNORE_CASE = true;
}
PARSER_BEGIN(MyParser)
class MyParser {
    public static void main(String args[])
        throws ParseException {
        MyParser parser = new MyParser(System.in);
        parser.Start();
    }
}
PARSER_END(MyParser)
```

```
TOKEN: {
    < IF: "if" >
    | < #DIGIT: ["0"-"9"]>
    | < ID: ["a"-"z"] (["a"-"z"]|<DIGIT>)* >
    | < NUM: (<DIGIT>)+ >
    | < REAL: ( (<DIGIT>)+ "." (<DIGIT>)* ) | ((<DIGIT>)* "." (<DIGIT>)+ ) >
}
... Parser Grammar Rules
```

Start Symbol der
Grammatik



Autor:
Aho et al.

© Pearson Studium 2008



Compiler

Kapitel 4

Syntaktische Analyse

JavaCC Grammars 2/2

```
void Start() :
{
    /* LOCAL VARIABLE DECLARATIONS ... */
}
{ NonTerminal() <TOKEN> ... }

void NonTerminal() :
{
    /* LOCAL VARIABLE DECLARATIONS ... */
}
{ "+" NonTerminal() | ... }  <-- regular expression constructs can be
    used
```



Autor:
Aho et al.

© Pearson Studium 2008



Compiler

Kapitel 4

Syntaktische Analyse

Sehr einfaches Beispiel

- `PARSER_BEGIN(Simple3)`
- `public class Simple3 {`
- `public static void main(String args[]) throws ParseException {`
- `Simple3 parser = new Simple3(System.in);`
- `parser.Input();`
- `}}`
- `PARSER_END(Simple3)`
- `SKIP :`
- `{ " " | "\t" | "\n" | "\r" }`
- `TOKEN :{ <LBRACE: "{">| <RBRACE: ">">}`
- `void Input() :`
- `{ int count; }`
- `{ count=MatchedBraces() <EOF>`
- `{ System.out.println("The levels of nesting is " + count); }`
- `}`
- `int MatchedBraces() :`
- `{ int nested_count=0; }`
- `{ [<LBRACE> nested_count=MatchedBraces() {nested_count++;} <RBRACE>]`
- `{ return nested_count; }`
- `}`

Zusammenfassung



Compiler

Kapitel 4

Syntaktische Analyse

- Recursive Descent Parsing:
 - Principles, limitations
 - Computing nullable, First, Follow
 - Writing a parser “by hand”
 - Problems: how to solve them **(in book!)**



Autor:
Aho et al.

© Pearson Studium 2008