Compilerbau

LL(K)-Parser 2

Prof. Dr. Franz-Karl Schmatzer
schmatzf@dhbw-loerrach.de

Literatur

- C.Wagenknecht, M.Hielscher; Formale Sprachen, abstrakte Automaten und Compiler; 3.Aufl. Springer Vieweg 2022;
- U.Meyer; Grundkurs Compilerbau; Rheinwerkverlag, 1. Aufl. 2021
- A.V.Aho, M.S.Lam,R.Savi,J.D.Ullman, *Compiler – Prinzipien,Techniken und Werkzeuge*. 2. Aufl., Pearson Studium, 2008.
- Güting, Erwin; *Übersetzerbau –Techniken, Werkzeuge, Anwendungen*, Springer Verlag 1999

Top-Down-Analyse

Agenda

- LL(k)-Parser
  - First und Follow-Mengen
  - Prädikativer Parser

LL(K) Grammatiken

Charakterisierung

- Problem wie findet man bei einer Produktion

  $A \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \ldots \mid \alpha_n$

  die richtige rechte Seite, wenn man k Terminalzeichen vorausschauen kann.
- Dazu definiert man zuerst die Menge
  - $FIRST_k(\alpha)$ und
  - $FOLLOW_k(A)$
- Darauf aufbauend definiert man Steuermengen $D(A \rightarrow \alpha_i)$
- Eine starke LL(K) Grammatik hat disjunkte Steuermengen, so dass eindeutig eine Produktion ausgewählt werden kann.

LL(K) Grammatiken

Definition $FIRST_k(\alpha)$ und $FOLLOW_k(A)$

Sei $G = (N,\Sigma,P,S)$ eine kontextfreie Grammatik, $\alpha \in (N \cup \Sigma)^*$, $k > 0$. Dann ist:

$FIRST_k(\alpha) := start_k(\{w \mid \alpha \Rightarrow^* w\})$

Die Menge $FIRST_k(\alpha)$ beschreibt gerade die Anfangsstücke bis zur Länge k von aus α ableitbaren Terminalworte

Sei $G = (N,\Sigma,P,S)$ eine kontextfreie Grammatik, $A \in N$, $k > 0$. Dann ist:

$FOLLOW_k(A) := \{w \mid S \Rightarrow^* uAv \text{ und } w = FIRST_k(v)\}$

$FOLLOW_k(A)$ beschreibt also Terminalzeichenfolgen bis zur Länge k, die innerhalb einer Ableitung in G auf das Nichtterminal A folgen können.

LL(k) Grammatik

Berechnung von FIRST-Mengen

Algorithmus:

- Sei a ein Terminal dann ist: First(a) = {a}
- Sei A ein Nichtterminal mit $A \to \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \ldots \mid \alpha_n$ dann ist:
  $First(A) = First(\alpha_1) \cup First(\alpha_2) \cup First(\alpha_3) \ldots \cup First(\alpha_n)$
- Sei A ein Nichtterminal mit $A \to \alpha_1\alpha_2\alpha_3\alpha_n$ dann ist:
  $First(A) = First(\alpha_1)First(\alpha_2)First(\alpha_3) \ldots First(\alpha_n)$ (Konkatenation)
  $a \in FIRST(A)$, falls $a \in FIRST(\alpha_k)$ und $\varepsilon \in FIRST(\alpha_i)$ für alle $1 \leq i \leq k$
  $\varepsilon \in FIRST(A)$, falls $\varepsilon \in FIRST(\alpha_i)$ für alle $1 \leq i \leq n$
- Rekursiv anwenden bis nur noch Terminale auftreten.

LL(k) Grammatik

Beispiel: Berechnung von FIRST-Mengen

- Beispiel G = ({S,A,B},{a,b,c,d},P,S) mit P:

  $S \to ABcd$

  $A \to a \mid B$

  $B \to b \mid \varepsilon$

- Die FIRST-Mengen für k=1
  - $First_1(B) = \{b,\varepsilon\}$
  - $First_1(A) = \{a\} \cup FIRST_1(B) = \{a, b, \varepsilon\}$
  - $First_1(S) = FIRST_1(ABcd) = \{a, b, c\}$
- **Bauen Sie die Grammatik in FLACI nach und Bestimmen Sie die First-Mengen**

LL(k) Grammatik

Berechnung von FOLLOW-Mengen

Algorithmus:

- Initialisiere FOLLOW(S) mit {$}

  ($ ist ein Eingabeende-Zeichen und nicht Teil des Eingabealphabet)

- Für jede Produktion $A \to \alpha B \beta$ und $\beta \neq \varepsilon$:

  Füge alle Symbole von FIRST(β) ohne {ε} in FOLLOW(B),

- Für jede Produktion $A \to \alpha B$ oder $A \to \alpha B \beta$ mit $\varepsilon \in$ FIRST(B):

  Füge alle Symbole aus FOLLOW(A) in FOLLOW(B) ein.

- Rekursiv anwenden bis nur noch Terminale auftreten.

LL(k) Grammatik

Beispiel: Berechnung von FOLLOW-Mengen

- Beispiel G = ({S,A,B},{a,b,c,d},P,S) mit P:

  $S \rightarrow ABcd$

  $A \rightarrow a \mid B$

  $B \rightarrow b \mid \varepsilon$

- Berechnen der FOLLOW-Mengen für k=1
  - $FOLLOW_1(S) = \{\$\}$
  - $FOLLOW_1(A) = FIRST_1(Bcd) = \{b, c\}$
  - $FOLLOW_1(B) = FIRST_1(cd) \cup FOLLOW_1(A) = \{b,c\}$
- Die FIRST-Mengen
  - $First_1(B) = \{b,\varepsilon\}$
  - $First_1(A) = \{a, b, \varepsilon\}$
  - $First_1(S) = \{a, b, c\}$

LL(K) Grammatiken

Definition Steuermengen $D(A \to \alpha_i)$

Sei $G = (N,\Sigma,P,S)$ eine kontextfreie Grammatik, $A \in N$, $k > 0$ und sei

$A \to \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \dots \mid \alpha_n$ die Mengen der A Produktionen. Dann ist für

$1 \leq i \leq n$ die Steuermenge $D_k(A \to \alpha_i)$ definiert als:

$$\mathbf{D_k(A \to \alpha_i) := start_k(FIRST_k(\alpha_i)FOLLOW_k(A))}$$

- Bem: Die Steuermenge besteht aus den First-Mengen und falls die Worte kleiner k sind werden die Follow-Mengen konkateniert.
- Man kann zeigen, dass genau dann wenn die Steuermengen für eine Produktion A disjunkt sind, es sich um eine starke LL(K) Grammatik handelt.
- Im folgend betrachten wir LL(1) Grammatiken näher.
  - Bestimmen FIRST- und FOLLOW-Mengen so wie die Steuermengen D

LL(K) Grammatiken

Berechnen der Steuermengen $D(A \rightarrow \alpha_i)$

- Beispiel G = ({S,A,B},{a,b,c,d},P,S) mit P:

  $S \rightarrow ABcd$

  $A \rightarrow a \mid B$

  $B \rightarrow b \mid \varepsilon$

Die Steuermengen für G

Firstmengen	Produktion		Steuermengen	Followmengen
{b,ε}	B →	b |	{b}	{b,c}
		ε	{b,c}	
{a, b, ε}	A →	a |	{a}	{b,c}
		B	{b, c}	
{a,b,c}	S →	ABcd	{a, b, c}	{$}

Aufgabe First-Follow-Mengen

Gegeben sei die folgende Grammatik $G=(N, \Sigma, P, E)$

- Berechnen Sie die FIRST-Mengen.
- Bestimmen Sie die Follow-Mengen.
- Bestimmen Sie die initialen Steuermengen.
- $N = \{E, E', T, T', F\}$, $\Sigma = \{+, *, (,), id\}$ und $P = \{ E \rightarrow TE', E' \rightarrow +TE' \mid \varepsilon, T \rightarrow FT', T' \rightarrow *FT' \mid \varepsilon, F \rightarrow (E) \mid id\}$
- $N = \{S, A, B\}$, $\Sigma = \{a, b, c, d\}$ und $P = \{S \rightarrow Ac \mid dS \mid \varepsilon, A \rightarrow aB \mid cA \mid d, B \rightarrow b \mid \varepsilon \}$
- **Führen Sie die Beispiele auch mit FLACI aus.**

LL(1) Grammatik

Steuermengen

- Eine kontextfreie Grammatik ist genau dann eine LL(1)-Grammatik, wenn für jedes Nichtterminal A mit $A \to \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \dots \mid \alpha_n$ gilt:
  - Die Mengen $FIRST_1(\alpha_1), \dots, FIRST_1(\alpha_n)$ sind paarweise disjunkt,
  - Genaue eine Menge $FIRST_1(\alpha_i)$ darf das leere Wort enthalten. Dafür muss dann die zugehörige $FOLLOW_1(A)$ disjunkt zu allen anderen $FIRST_1(\alpha_j)$ sein.
- Die Steuermengen vereinfachen sich zu:
  - $D(A \to \alpha_i) := FIRST_1(\alpha_i)$ falls $\varepsilon \notin FIRST_1(\alpha_i)$ oder

    $(FIRST_1(\alpha_i) - \{\varepsilon\}) \cup FOLLOW_1(A)$ sonst

Aufgaben LL(1)

Gehören folgende Grammatiken zu LL(1)?

- $G(N, \Sigma, P, S)$ mit $N = \{A,B, S\}$, $\Sigma=\{x,y,z\}$ und $P = \{S \rightarrow A \mid B,\ A \rightarrow x\,A \mid y,\ B \rightarrow x\,B \mid z\ \}$
- $G(N, \Sigma, P, K)$ mit $N = \{K,E, S\}$, $\Sigma=\{a,b,d,c\}$ und $P = \{K \rightarrow S \mid \varepsilon,\ S \rightarrow a\,S\,b \mid E,\ E\rightarrow d \mid c\,E\ \}$

Begründen Sie ihre Aussagen. Überprüfen Sie dies mit FLACI.

LL(1) Grammatik

Berechnung von FIRST-Mengen

- Systematisches Verfahren:
  - Bestimmen einer Menge N_ε aller Nichtterminale aus denen das leere Wort abgeleitet werden kann $N_\varepsilon := \{ X \in N \mid X \Rightarrow^* \varepsilon \}$
  - Man zeichne einen Graphen, dessen Knoten die Nichtterminale sind. Für jede Produktion $A \to X_1 \dots X_m$ mit dem Nichtterminal X_1 füge eine gerichtete $A \to X_1$ Kante ein.

    Falls $X_1 \in N_\varepsilon$ und X_2 ein Nichtterminal füge eine Kante $A \to X_2$ hinzu. Falls $X_1 \in N_\varepsilon$ und $X_2 \in N_\varepsilon$ und X_3 ein Nichtterminal füge eine Kante $A \to X_3$ hinzu, usw.
- Eine Kante $A \to B$ drückt aus: FIRST(B) sollte vor FIRST(A) berechnet werden.

Beispiel Grammatik G_2

- Betrachte folgende Grammatik G_2

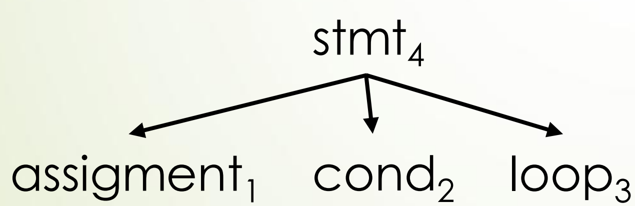
stmt	→	assignment | cond | loop
assignment	→	id := expr
cond	→	if boolexpr then stmt cond-rest
cond-rest	→	fi | else stmt fi
loop	→	while boolexpr do stmt od
expr	→	boolexpr | numexpr
boolexpr	→	numexpr cop numexpr
numexpr	→	term nexpr | term
nexpr	→	+ term nexpr | ε
term	→	factor nterm | factor
nterm	→	* factor nterm | ε
factor	→	id | const | (numexpr)

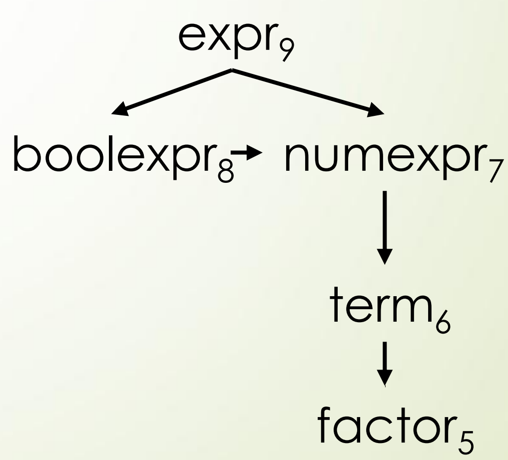
- Die Menge N_ε := {nterm, nexpr}

Beispiel Grammatik G_2

Graphen zu den FIRST-Mengen

- Graphen zu der Berechnung der FIRST-Mengen
  - Berechnen der FIRST-Mengen von assigment, cond und loop zuerst und dann die FIRST-Menge von stmt
  - Berechnen der FIRST-Menge von factor, dann die von term, dann die von numexpr, dann die von boolexpr. Am Schluss wird die FIRST-Menge von expr berechnet.
  - Die FIRST-Mengen zu cond-rest, nexpr, nterm sind unabhängig





Beispiel Grammatik G_2

Die FIRST- und Steuermengen

FIRST-Mengen	Produktion		Steuermengen D
{id}	assignment →	id := expr	{id}
{if}	cond → cond-rest	if boolexpr then stmt	{if}
{fi, else}	cond-rest →	fi | else stmt fi	{fi} {else}
{while}	loop →	while boolexpr do stmt od	{while}
{id, if,while}	stmt →	assigment | cond | loop	{id} {if} {while}
{id,const,(}	factor →	id | const | (expr)	{id} {const} {(}
{id,const,(}	term →	factor nterm	{id,const,(}
{id,const,(}	numexpr →	term nexpr	{id,const,(}
{id,const,(}	boolexpr →	numexpr cop numexpr	{id,const,(}
{id,const,(}	expr →	boolexpr | numexpr	{id,const,(} {id,const,(}

Beispiel Grammatik G_{2n}

G_{2n} gehört zu LL(1)

- Beobachtung:
  - Die Steuermengen für die Produktion expr sind nicht disjunkt.

  $\Rightarrow$ die Grammatik G_2 gehört nicht zu LL(1)
- Das Problem liegt an der Produktion von expr

expr → boolexpr | numexpr

Ein Token-Vorausschau reicht nicht um zwischen den beiden Produktionen zu entscheiden.

Das lässt sich beheben durch folgende Produktionen:

expr → numexpr bool-rest

boolrest → cop numexpr | ε

Beispielgrammatik G_{2n}

- Betrachte folgende Grammatik G_{2n}

  stmt → assignment | cond | loop

  assignment → id := expr

  cond → if boolexpr then stmt cond-rest

  cond-rest → fi | else stmt fi

  loop→ while boolexpr do stmt od

  expr → numexpr bool-rest

  boolrest → cop numexpr | ε

  boolexpr → numexpr cop numexpr

  numexpr → term nexpr

  nexpr → + term nexpr | ε

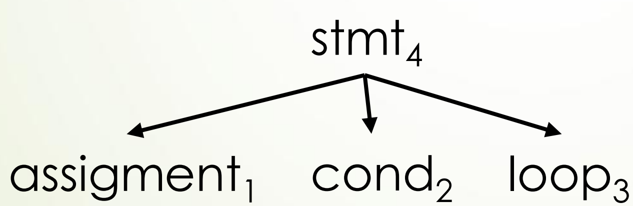
  term → factor nterm

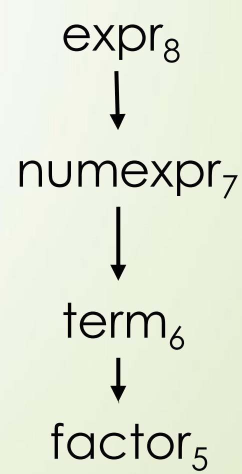
  nterm → * factor nterm | ε

  factor → id | const | (expr)

Beispielgrammatik G_{2n}

- Die Menge N_ε := {nterm, nexpr, boolrest}
  - Für diese Produktionen müssen die FOLLOW-Mengen bestimmt werden.
- Die Graphen zur Berechnung der FIRST-Mengen





Beispiel Grammatik G_{2n}

Die FIRST- und Steuermengen

FIRST-Mengen	Produktion		Steuermengen D
{id, if,while}	stmt →	assigment | cond | loop	{id} {if} {while}
{id}	assignment →	id := expr	{id}
{if}	cond → rest	if boolexpr then stmt cond-	{if}
{fi, else}	cond-rest →	fi | else stmt fi	{fi} {else}
{while}	loop →	while boolexpr do stmt od	{while}
{id,const,(}	expr →	boolexpr bool-rest	{id,const,(}
{cop}	bool-rest→	cop numexpr | ε	{cop} {$, od, fi, else,)}
{id,const,(}	boolexpr →	numexpr cop numexpr	{id,const,(}

Beispiel Grammatik G_{2n}

Die FIRST- und Steuermengen

FIRST-Mengen	Produktion		Steuermengen D
{id,const,(}	numexpr →	term nexpr	{id,const,(}
{+}	nexpr→	+ term nexpr | ε	{+} {$, od, fi, else,), then, do,cop}
{id,const,(}	term →	factor nterm	{id,const,(}
{*}	nterm →	* factor nterm | ε	{*} {$, od, fi, else,), then, do,cop,+}
{id,const,(}	factor →	id | const | (expr)	{id} {const} {(}

Aufgabe

- Erstellen Sie die beiden vorherigen Grammatiken mit FLACI und überprüfen Sie die First-Mengen.

Berechnung der FOLLOW-Menge

- Systematisches Verfahren

1. Trage alle Nichtterminale als Knoten in einen Graphen. Der Graph hat noch keine Kanten. Markiere den Knoten mit dem Startsymbol mit dem Symbol \$ (Ende der Eingabe)
2. Betrachte der Reihe nach, alle Produktionen in P. Für jede Produktion jedes Nichtterminal B auf der rechten Seite.
    1. $A \to \alpha B \beta$ mit $\beta \neq \varepsilon$: Markiere den Knoten B mit allen Symbolen, die in FIRST(β) liegen. Falls $\varepsilon \in$ FIRST(β), dann füge eine Kante $A \to B$ hinzu.
    2. $A \to \alpha B$: Füge die Kante $A \to B$ hinzu.
3. Falls es Zyklen gibt, werden alle Knoten des Zyklus gleich behandelt
4. Die FOLLOW-Menge eines Nichtterminals ist die Vereinigung seiner eigenen Markierungen mit den Markierungen aller seiner Vorgänger im Graphen.

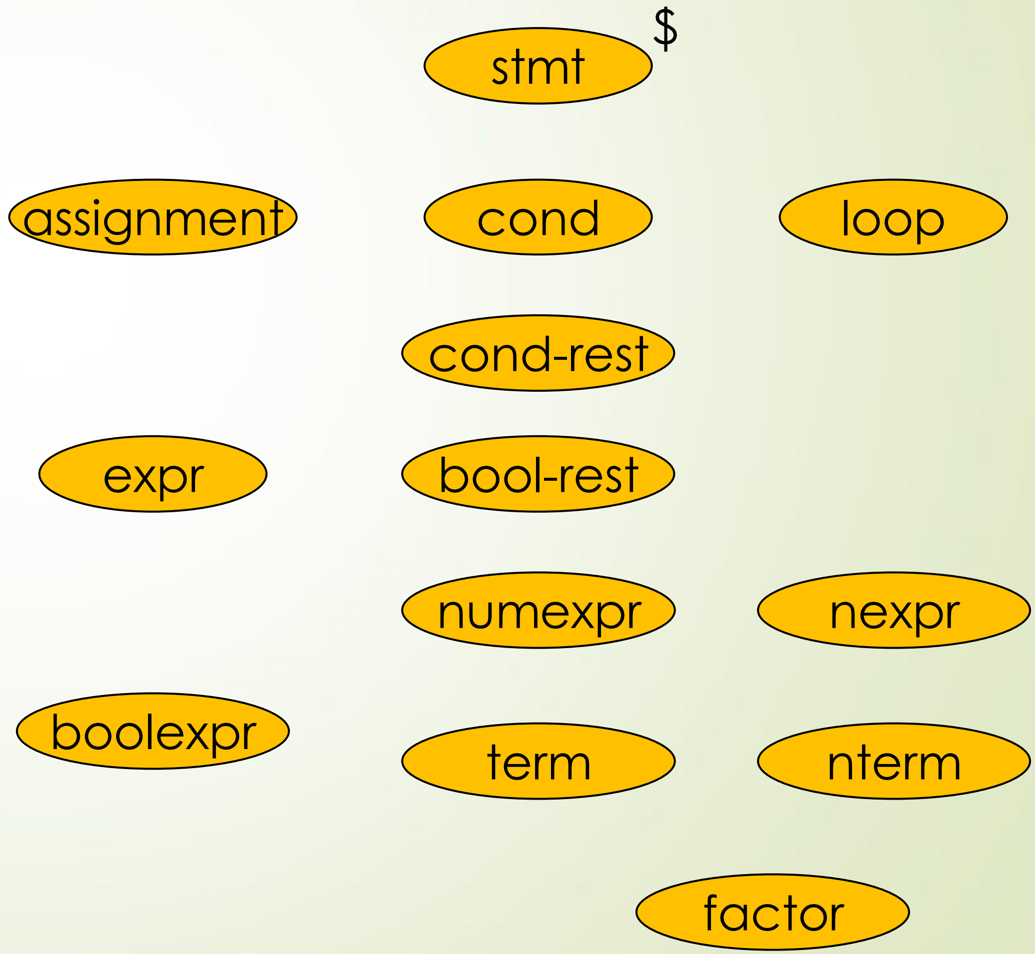
Beispiel Grammatik G_{2n}

Die Follow-Mengen als Graphen

Aufbau des Graphen laut Vorschrift der vorherigen Folie

Nach dem 1. Schritt (alle Nichterminale + $ an Startknoten)

stmt → assignment | cond | loop
assignment → id := expr
cond → if boolexpr then stmt cond-rest
cond-rest → fi | else stmt fi
loop → while boolexpr do stmt od
expr → numexpr bool-rest
boolrest → cop numexpr | ε
boolexpr → numexpr cop numexpr
numexpr → term nexpr
nexpr → + term nexpr | ε
term → factor nterm
nterm → * factor nterm | ε
factor → id | const | (expr)



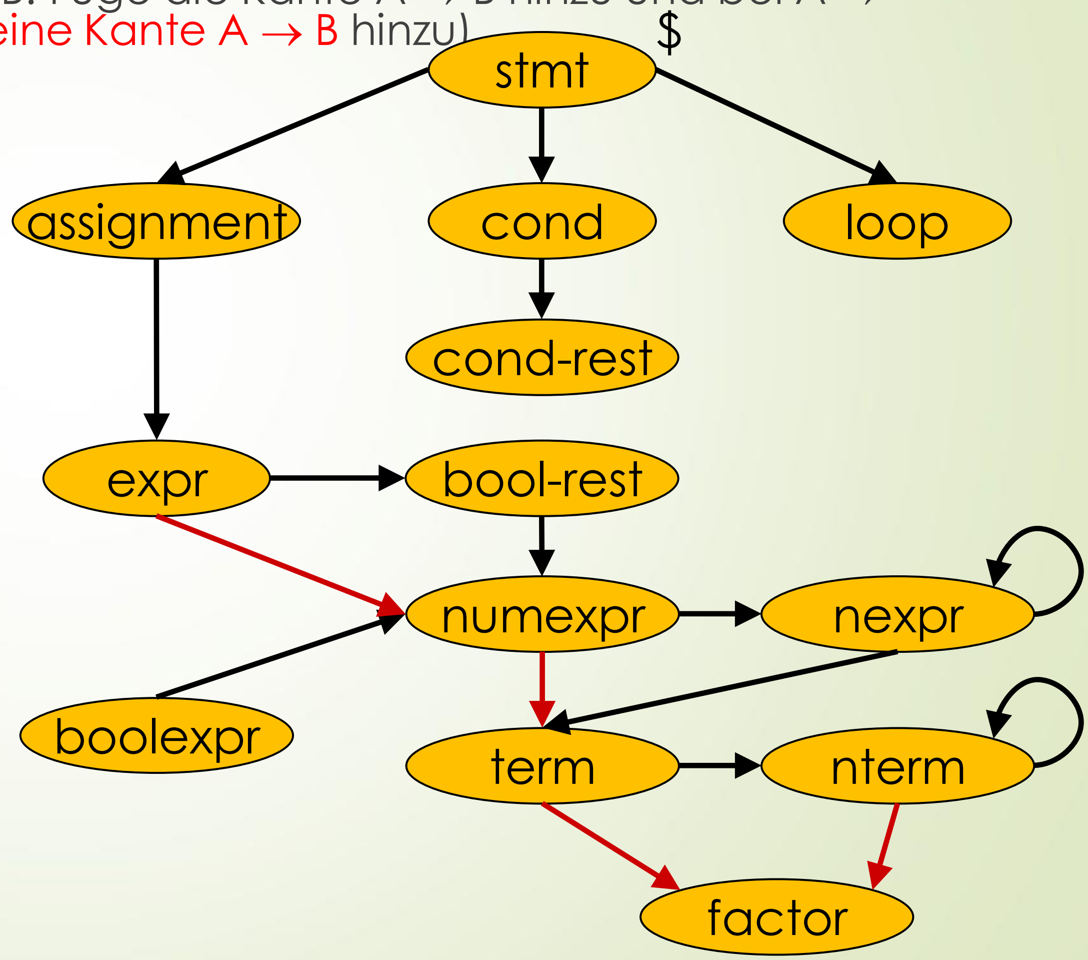
Beispiel Grammatik G_{2n}

Die Follow-Mengen als Graphen

Aufbau des Graphen laut Vorschrift der vorherigen Folie

- Während des 2. Schrittes ($A \to \alpha B$: Füge die Kante $A \to B$ hinzu und bei $A \to \alpha B$, falls $\varepsilon \in$ FIRST(β), dann füge eine Kante $A \to B$ hinzu)

stmt	→	assignment | cond | loop
assignment	→	id := expr
cond	→	if boolexpr then stmt cond-rest
cond-rest	→	fi | else stmt fi
loop	→	while boolexpr do stmt od
expr	→	numexpr bool-rest
boolrest	→	cop numexpr | ε
boolexpr	→	numexpr cop numexpr
numexpr	→	term nexpr
nexpr	→	+ term nexpr | ε
term	→	factor nterm
nterm	→	* factor nterm | ε
factor	→	id | const | (expr)



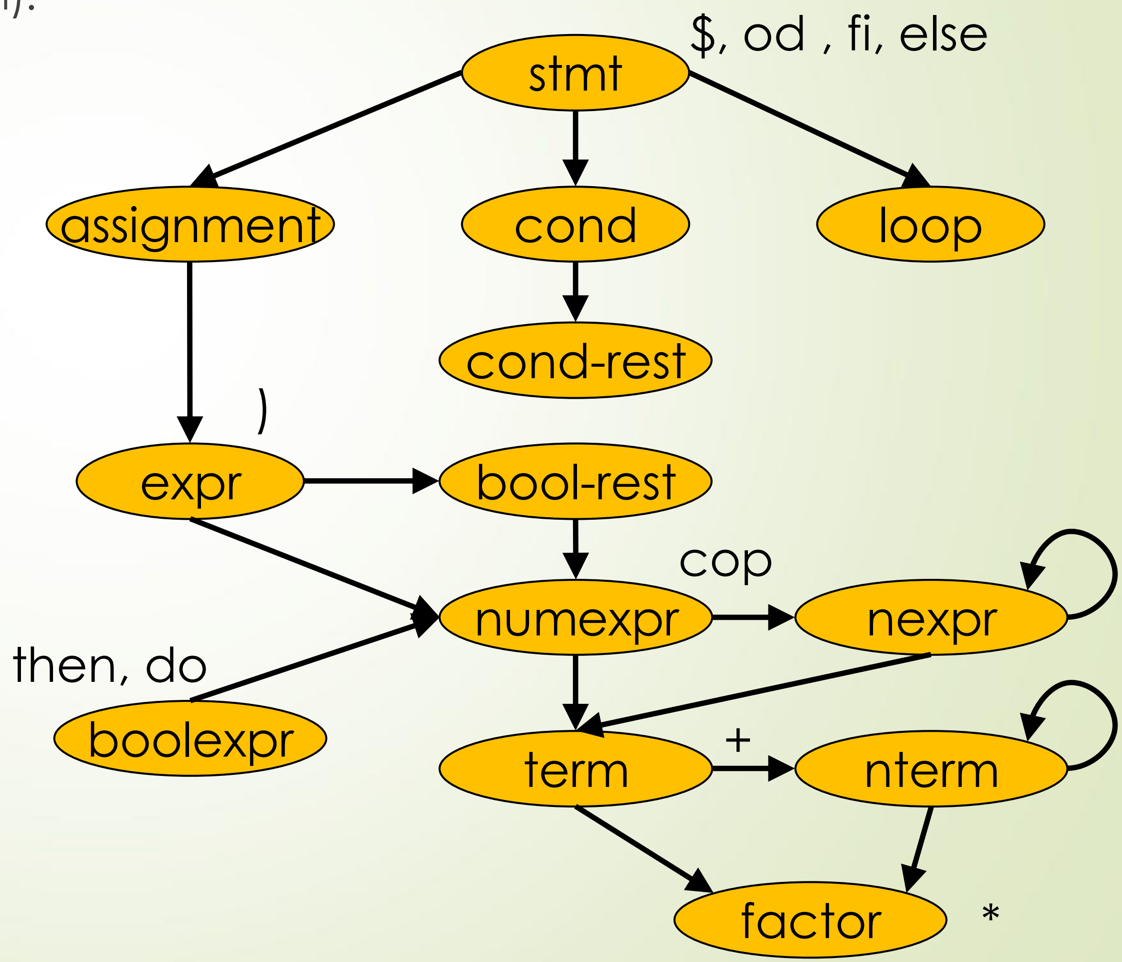
Beispiel Grammatik G_{2n}

Die Follow-Mengen als Graphen

Aufbau des Graphen laut Vorschrift der vorherigen Folie

- Nach dem 2. Schritt ($A \to \alpha B\beta$ mit $\beta \neq \varepsilon$: Markiere den Knoten B mit allen Symbolen, die in FIRST(β) liegen).

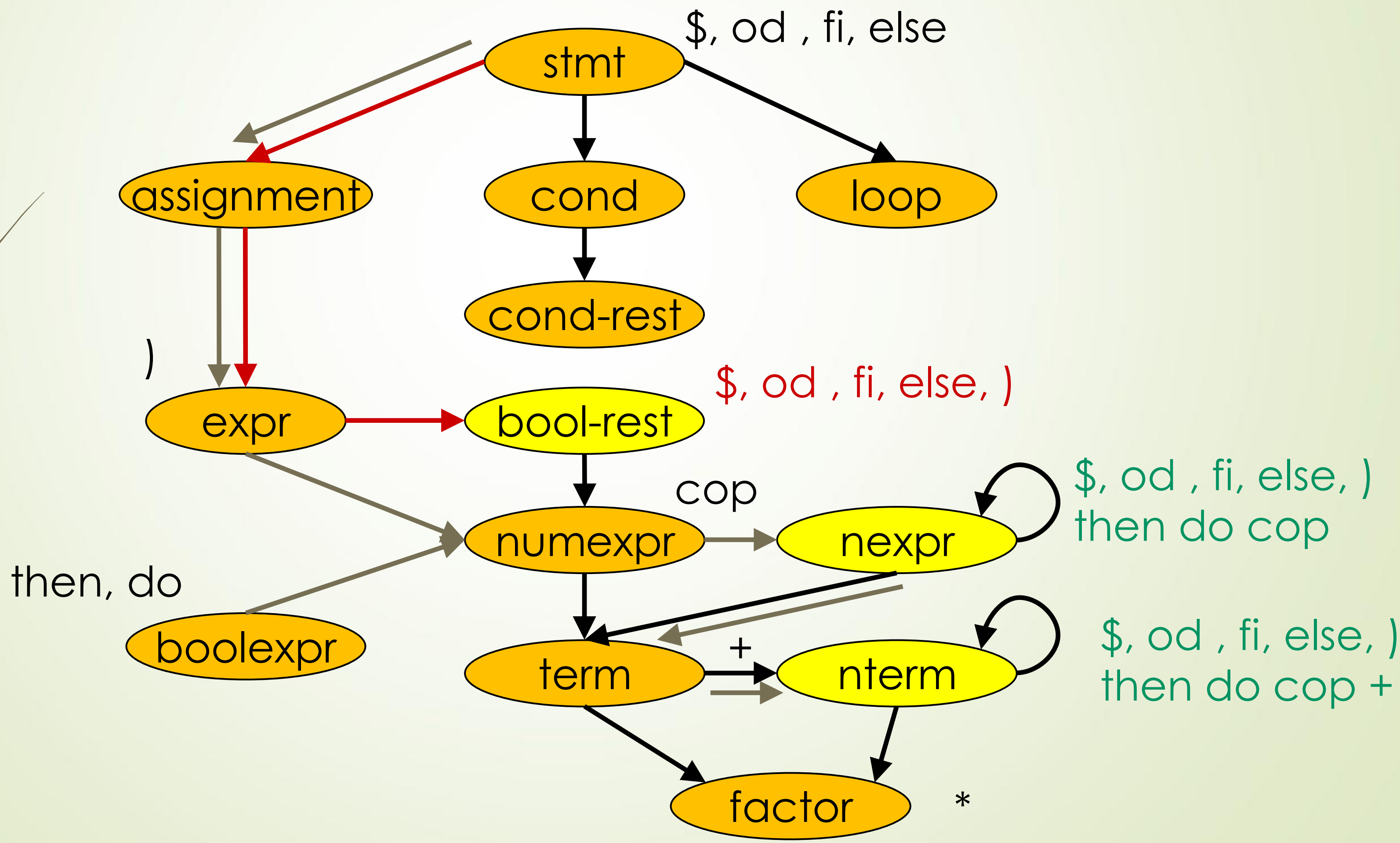
stmt	→	assignment | cond | loop
assignment	→	id := expr
cond	→	if boolexpr then stmt cond-rest
cond-rest	→	fi | else stmt fi
loop	→	while boolexpr do stmt od
expr	→	numexpr bool-rest
boolrest	→	cop numexpr | ε
boolexpr	→	numexpr cop numexpr
numexpr	→	term nexpr
nexpr	→	+ term nexpr | ε
term	→	factor nterm
nterm	→	* factor nterm | ε
factor	→	id | const | (expr)



Beispiel Grammatik G_{2n}

Die Follow-Mengen als Graphen

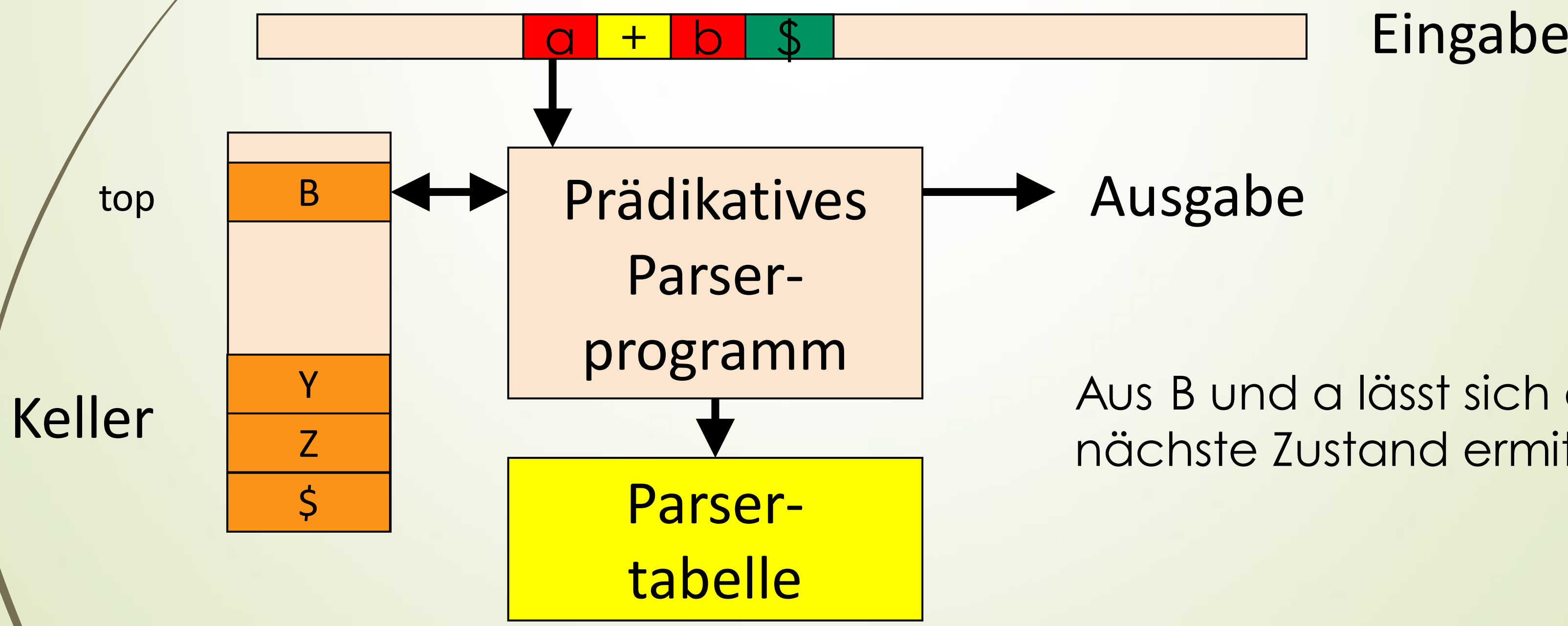
- Aufbau des Graphen laut Vorschrift der vorherigen Folie
  - Nach dem 4. Schritt



Aufbau eines prädikativen Analysators

Analysetabelle - Prinzip

- Für eine LL(1) Grammatik lassen sich prädikative Parser recht einfach erstellen.
  - Die richtigen Produktionen lassen sich anhand des gelesene Eingabezeichen eindeutig bestimmen.
  - Man kann damit ein PDA erstellen



Aus B und a lässt sich der nächste Zustand ermitteln

Aufbau eines prädikativen Analysators

Analysetabelle- Beispiel

- Betrachte folgende reduzierte Grammatik G_a (Teilgrammatik G_{2n})

Grammatik-Regeln			**Nummer**	**Steuermenge**
numexpr	→	term nexpr	(1)	{id,const,(}
nexpr	→	+ term nexpr |	(2)	{+}
		ε	(3)	{$,)}
term	→	factor nterm |	(4)	{id,const,(}
nterm	→	* factor nterm |	(5)	{*}
		ε	(6)	{$,),+}
factor	→	id |	(7)	{id}
		const |	(8)	{const}
		(expr)	(9)	{(}

- Damit lassen sich Ausdrücke formulieren wie
  - id*(id+id*(id+id))
  - id+id*const

Aufbau eines prädikativen Analysators

Analysetabelle- Tabelle

- Analysetabelle: für jedes Nichtterminal und jedes Eingabezeichen
  - Spalte zeigt die Nummer der Produktion an, die gewählt wird
  - Wo keine Einträge sind, liegt ein Fehlerzustand vor.

Nichtterminal	Eingabesymbol						
	id	const	+	*	(	)	$
numexpr(E)	1	1			1		
nexpr(N)			2			3	3
Term (T)	4	4			4		
nterm(R)			6	5		6	6
factor(F)	7	8			9		

Aufbau eines prädikativen Analysators

Analysetabelle- Funktionsweise des Kellerautomaten

- Betrachte den Eingabestring: id+id*id

Stack	Eingabe	Regel
E$	id+id*id$	
TN$	id+id*id$	1
FRN$	id+id*id$	4
idRN$	id+id*id$	7
RN$	+id*id$	Match(id)
N$	+id*id$	6
+TN$	+id*id$	4
+TN$	+id*id$	Match(+)
FRN$	id*id$	4
idRN$	id*id$	7
RN$	*id$	Match(id)

Stack	Eingabe	Regel
RN$	*id$	Match(id)
*FRN$	*id$	5
FRN$	id$	Match(*)
idRN$	id$	7
RN$	$	Match(id)
N$	$	6
$	$	6
		Match($)

Aufbau eines prädikativen Analysators

rekursiver Abstieg - Prinzip

- Rekursives Parsing besteht aus einer Gruppe von Prozeduren.
  - Eine Prozedur für jedes Nichtterminal
  - Eine Startprozedur, welche die anderen Prozeduren aufruft und anhält wenn der Eingabestring erfolgreich gelesen wurde.
- Pseudocode:

```
void procedure A() { // Prozedur zum Nichtterminal A
    // Wähle eine A-Produktion A→X1..Xn;
    for(i = 1 bis n) {
        if(X ein Nichtterminal) call X(); // rufe Prozedur zu X;
        else if (X  = dem aktuelle Eingabezeichen) call nexttoken();
        else  // Fehleroutine aufrufen !
    }
}
```

Aufbau eines prädikativen Analysators

rekursiver Abstieg - Beispiel

- Betrachte den Ausschnitt

stmt	→	assignment |	{id}
		cond |	{if}
		loop	{while}
assignment	→	id := expr	{id}

- Aufbau der Prozeduren

```
procedure stmt() {
        if (token == ‘id‘) assigment();
        elseif(token == ‘if‘) cond();
        elseif(token== ‘while‘) loop;
        else error();
        }
```

```
procedure assigment() {
    if(token == ‘id‘) {
     match(‘id‘);
     match(‘:=‘);
     expr();
     }
    else error();
    }
```

```
procedure match(token t){
    if(token == t) nexttoken();
    else error;
    }
```