Grammar Engineering

Forderungen:

- deterministische Grammatik: zu einer Eingabe existiert höchstens ein Syntaxbaum
- Operatorprioritäten: Grammatik erzeugt Syntaxbaum gemäß Prioritäten

Beispiel auf Folien Syntaxanalyse Seite 12.

Gegenbeispiel:

$$A \rightarrow A + A \mid A * A \mid \textbf{bez} \mid (A)$$

x+y+x hat 2 Syntaxbäume, **x*y+z** auch - sogar einen, der „Punkt vor Strich" ignoriert.

Grammar Engineering

Weiteres Gegenbeispiel: „Dangling Else"

$$
\begin{array}{rcl}
S & \rightarrow & \textbf{if } E \textbf{ then } S \\
 & | & \textbf{if } E \textbf{ then } S \textbf{ else } S
\end{array}
$$

if A then if B then C else D
hat 2 verschiedene Syntaxbäume. In der Praxis gehört ein **else** aber immer zum letzten **if**.

- Parsergeneratoren erkennen Mehrdeutigkeiten
- aber in Grammatiken mit hunderten Produktionen sind Mehrdeutigkeiten schwer zu beheben

Faustregeln

- ein Nichtterminal pro Prioritätsebene
- nicht zweimal dasselbe Nichtterminal auf der rechten Seite
- Links- oder Rechtsassoziaitivität von Operatoren wird durch links- bzw. rechtsrekursive Regeln ausgedrückt

Abstrakte Syntax als abstrakte Algebra

Heute fasst man eine abstrakte Syntax als Signatur einer ordnungssortierten Termalgebra auf, und einen AST als Term gemäß dieser Signatur.

- Klassen der abstrakten Syntax entsprechen Sorten der Algebra
- innere Baumknoten entsprechen Operatoren (Funktionssymbolen) der Algebra inkl. Signatur.

Beispiel: abstrakte Syntax für Expressions und Statements

$$
\begin{array}{rcl}
\mathit{Stmt} & = & \mathit{IfStmt} \mid \mathit{IfElseStmt} \mid \mathit{WhileStmt} \mid \mathit{Assignment} \mid \mathit{Block} \ldots \\
\mathit{IfStmt} & :: & \mathit{Expr}\ \mathit{Stmt} \\
\mathit{IfElseStmt} & :: & \mathit{Expr}\ \mathit{Stmt}\ \mathit{Stmt} \\
\mathit{WhileStmt} & :: & \mathit{Expr}\ \mathit{Stmt} \\
\mathit{Block} & :: & \mathit{Decls}\ \mathit{StmtList} \\
\mathit{StmtList} & :: & \mathit{Stmt}\ + \\
\mathit{Assignment} & :: & \mathit{Var}\ \mathit{Expr} \\
\mathit{Var} & = & \mathbf{Bez} \mid \ldots
\end{array}
\qquad
\begin{array}{rcl}
\mathit{Expr} & = & \mathit{Addop} \mid \mathit{MultOp} \mid \mathit{Var} \mid \\
\mathit{Addop} & :: & \mathit{Expr}\ \mathit{Expr} \\
\mathit{Multop} & :: & \mathit{Expr}\ \mathit{Expr}
\end{array}
$$

Abstrakte Syntax als abstrakte Algebra

Entsprechende Bäume können auch als Terme dargestellt werden.
Schreibweise zur Konstruktion von Termen z.B.

Beispiele

$Addop(Bez(\mathbf{hinz}), Bez(\mathbf{kunz}))$
$IfStmt(Bez(\mathbf{test}), Assignment(Bez(\mathbf{x}), Addop(Bez(\mathbf{x}), Bez(\mathbf{y}))))$
$Block(Decls(\ldots), StmtList(s_1, s_2, \ldots, s_{42}))$

Assoziativitäten/Präzedenzen werden durch Termstruktur dargestellt
Achtung: Die abstrakte Syntax enthält keine semantischen Bedingungen z.B. „Typ einer **If**-Expression muss boolsch sein"

Parsertabelle

Grammatik:

$$
\begin{aligned}
E &\rightarrow T\ E' \\
E' &\rightarrow +\ T\ E' \mid \epsilon \\
T &\rightarrow F\ T' \\
T' &\rightarrow *\ F\ T' \mid \epsilon \\
F &\rightarrow (\ E\ ) \mid \mathbf{id}
\end{aligned}
$$

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: Parsertabelle

Parsertabelle

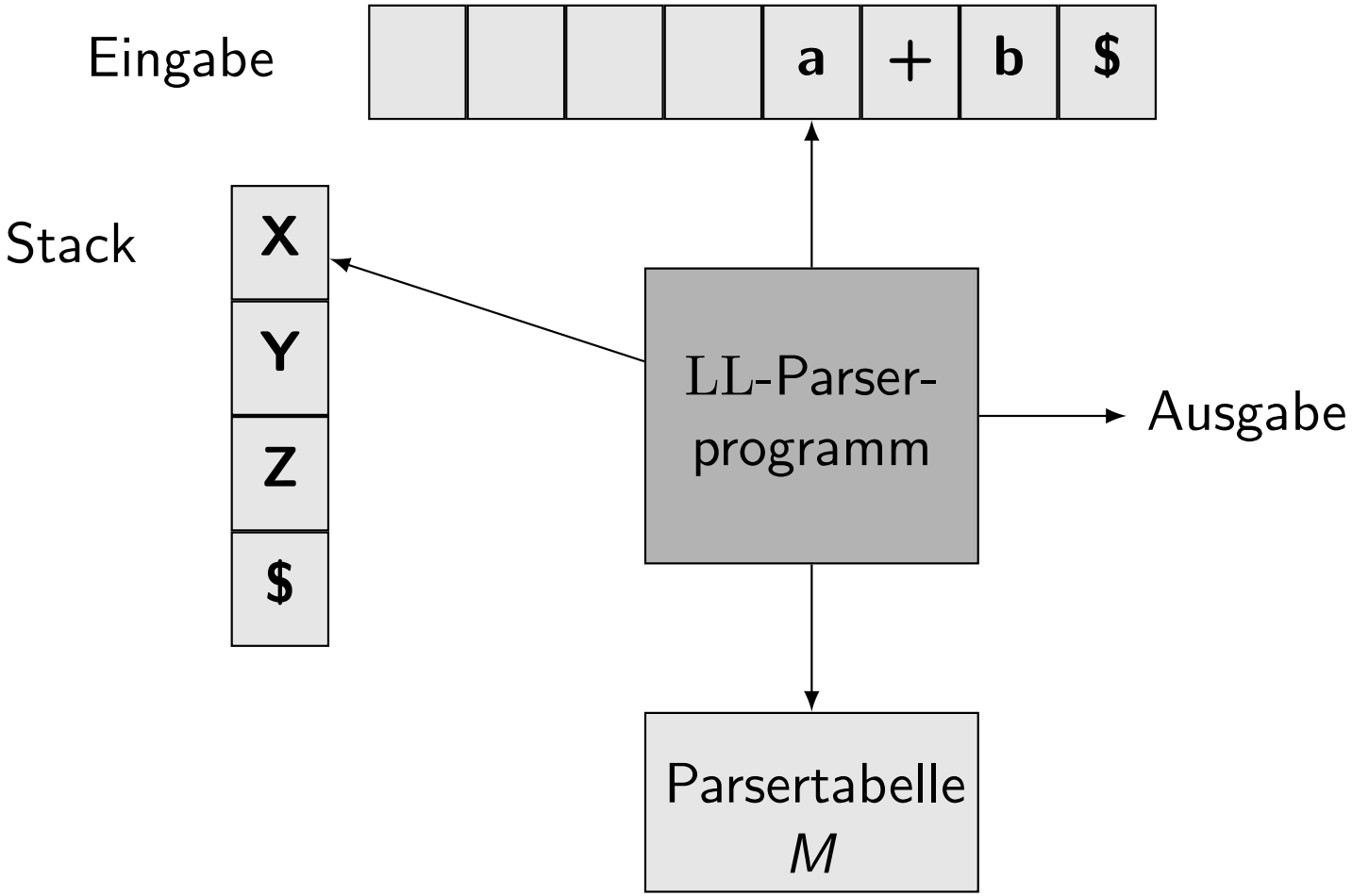
Grammatik:

$$
\begin{aligned}
S &\rightarrow \mathbf{i}\ E\ \mathbf{t}\ S\ S' \mid \mathbf{a} \\
S' &\rightarrow \mathbf{e}\ S \mid \epsilon \\
E &\rightarrow \mathbf{b}
\end{aligned}
$$

Nichtterminal	**Eingabesymbol**					
	a	**b**	**e**	**i**	**t**	$
S	$S \rightarrow \mathbf{a}$			$S \rightarrow \mathbf{i}E\mathbf{t}SS'$		
S'			$S' \rightarrow \mathbf{e}S$			$S' \rightarrow \epsilon$
E		$E' \rightarrow \mathbf{b}$				

Abbildung: Parsertabelle

Modell eines tabellengesteuerten LL-Parsers



Verhalten eines LL-Parsers

Übereinstimmung	Stack	Eingabe	Aktion
	$E\$$	**id+id*id$**	
	$TE'\$$	**id+id*id$**	Ausgabe von $E \rightarrow TE'$
	$FT'E'\$$	**id+id*id$**	Ausgabe von $T \rightarrow FT'$
	id $T'E'\$$	**id+id*id$**	Ausgabe von $F \rightarrow$ **id**
id	$T'E'\$$	**+id*id$**	Übereinstimmung mit **id**
id	$E'\$$	**+id*id$**	Ausgabe von $T' \rightarrow \epsilon$
id	$+TE'\$$	**+id*id$**	Ausgabe von $E' \rightarrow +TE'$
id+	$TE'\$$	**id*id$**	Übereinstimmung mit **+**
id+	$FT'E'\$$	**id*id$**	Ausgabe von $T \rightarrow FT'$
id+	**id** $T'E'\$$	**id*id$**	Ausgabe von $F \rightarrow$ **id**
id+id	$T'E'\$$	***id$**	Übereinstimmung mit **id**
id+id	$*FT'E'\$$	***id$**	Ausgabe von $T' \rightarrow *FT'$
id+id*	$FT'E'\$$	**id$**	Übereinstimmung mit *****
id+id*	**id** $T'E'\$$	**id$**	Ausgabe von $F \rightarrow$ **id**
id+id*id	$T'E'\$$	$\$$	Übereinstimmung mit **id**
id+id*id	$E'\$$	$\$$	Ausgabe von $T' \rightarrow \epsilon$
id+id*id	$\$$	$\$$	Ausgabe von $E' \rightarrow \epsilon$

Konstruktion der $\mathrm{LL}(1)$-Tabelle

$$\begin{aligned}\mathrm{LL}[X, \mathbf{a}] = \quad & \{X \to X_1 \ldots X_n \in P \mid \\ & \mathbf{a} \in \mathsf{FIRST}(X_1 \ldots X_n \cdot \mathsf{FOLLOW}(X_1 \ldots X_n))\}\end{aligned}$$

Es muss gelten $|\mathrm{LL}[X, \mathbf{a}]| = 1$ für alle X, $\mathbf{a}$, sonst ist die Grammatik nicht $\mathrm{LL}(1)$.

LL(1) Parseralgorithmus

```
push('$'); push(Z); t = next_token();
while (t != '$') {
    if (stackEmpty()) { error("end of input expected"); }
    else if (top() ∈ T) {
        if (top() == t) {
            pop(); t = next_token();
        } else { error(top() + " expected"); pop(); }
    } else if (LL[top(), t] == ⊥) {
        error("illegal Symbol " + t);
    } else {
        (X → X_1 ... X_n) = LL[top(), t];
        pop();
        for(i = n; i >= 1; --i)
            push(X_i);
    }
}
if (top() != '$')
    error("unexpected end of input");
```

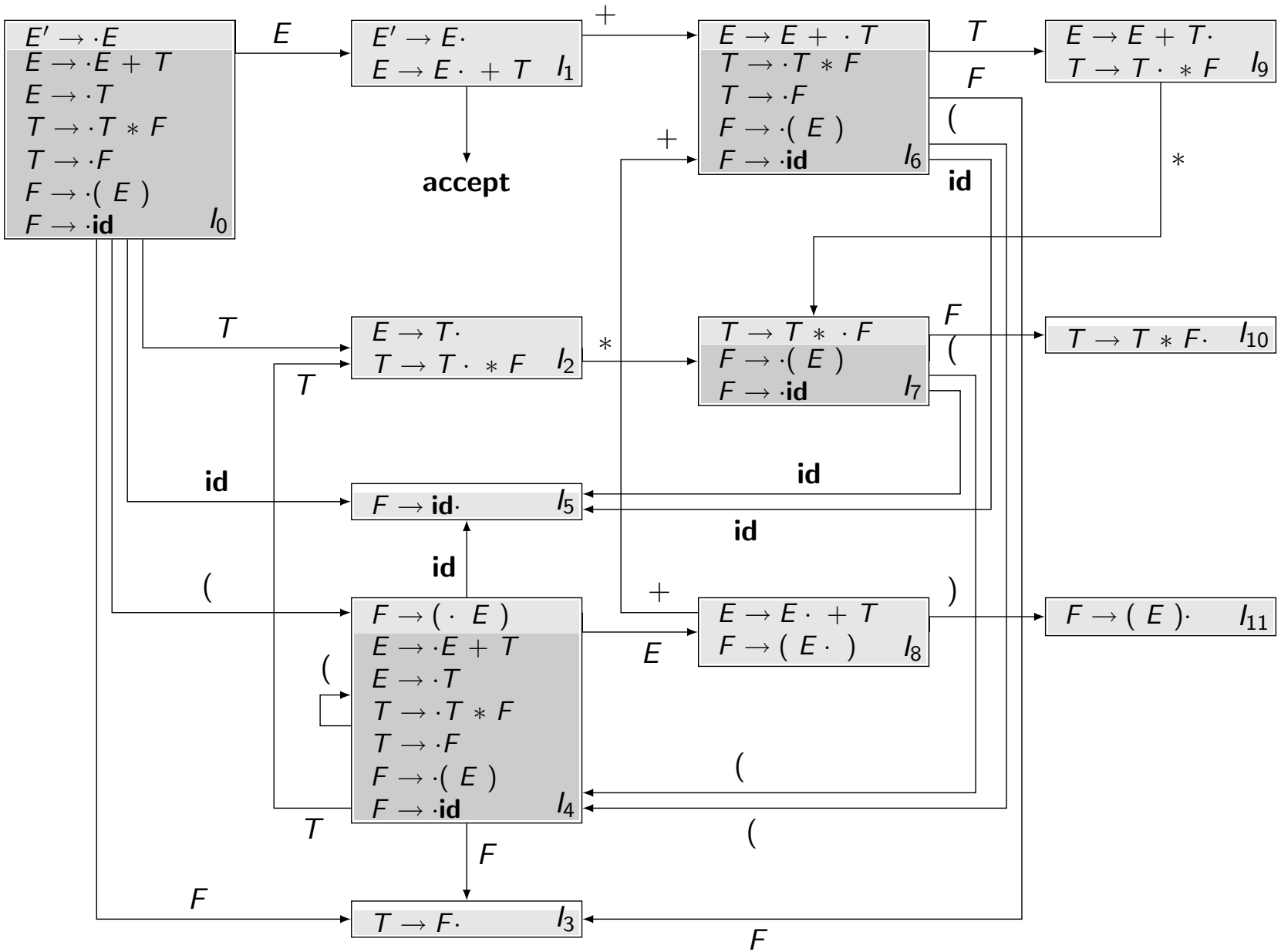

Verhalten eines Shift-Reduce-Parsers

Stack	**Eingabe**	**Aktion**
\$	$\mathbf{id}_1 * \mathbf{id}_2$ \$	Verschieben (shift)
\$ $\mathbf{id}_1$	$* \mathbf{id}_2$ \$	Reduzieren durch $F \longrightarrow \mathbf{id}$
\$ F	$* \mathbf{id}_2$ \$	Reduzieren durch $T \longrightarrow F$
\$ T	$* \mathbf{id}_2$ \$	Verschieben
\$ $T *$	$\mathbf{id}_2$ \$	Verschieben
\$ $T * \mathbf{id}_2$	\$	Reduzieren durch $F \longrightarrow \mathbf{id}$
\$ $T * F$	\$	Reduzieren durch $T \longrightarrow T * F$
\$ T	\$	Reduzieren durch $E \longrightarrow T$
\$ E	\$	Akzeptieren

Beispiel: Grammatik G

$$
\begin{aligned}
E' &\rightarrow E \\
E &\rightarrow E + T \mid T \\
T &\rightarrow T * F \mid F \\
F &\rightarrow (E) \mid \mathbf{id}
\end{aligned}
$$

LR(0)-Automat für Grammatik G



Berechnung von CLOSURE

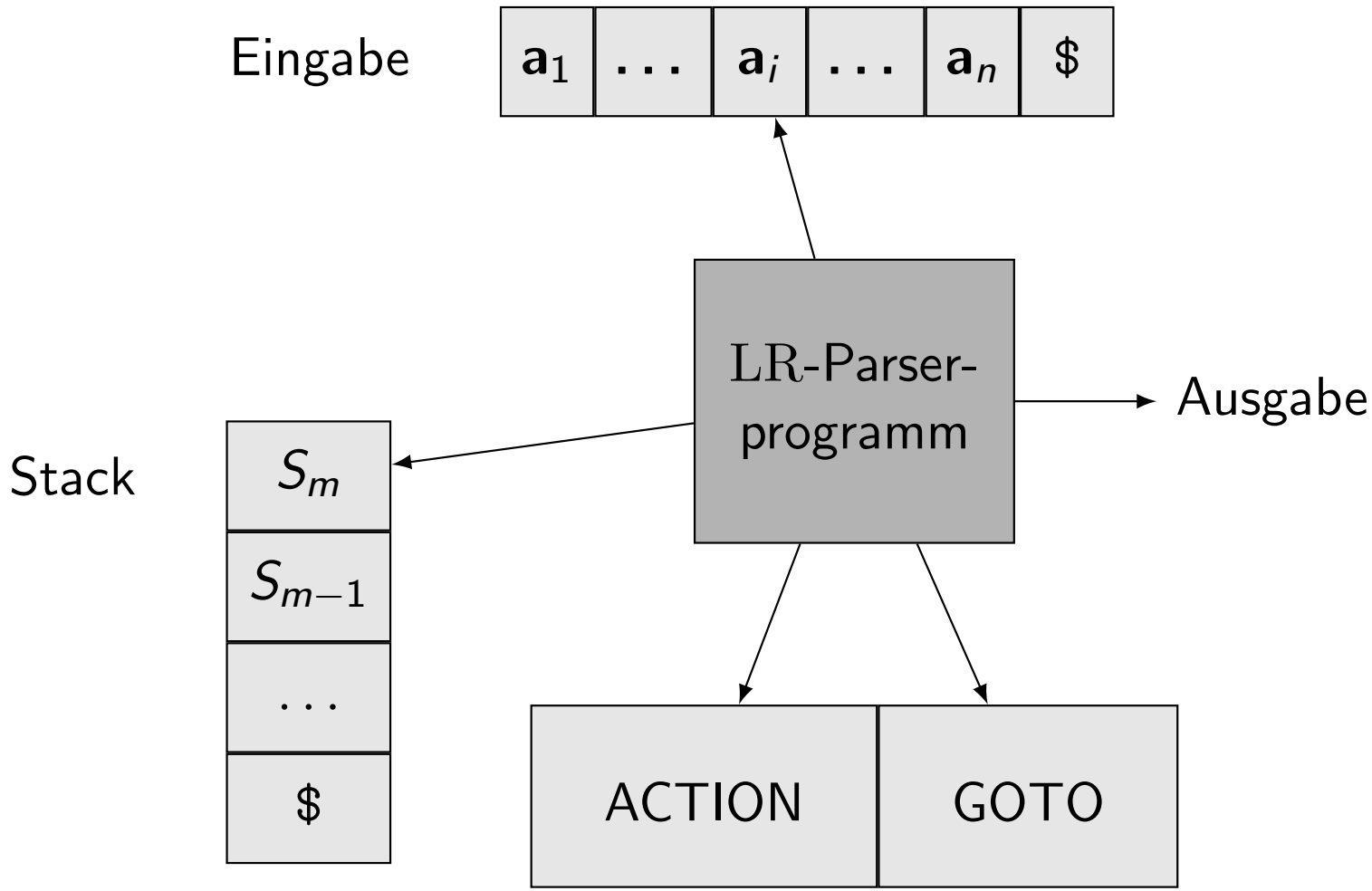
```
set<production> closure(production I)
{
    J = {I};
    do {
        changed = false;
        foreach((A → α · Bβ) ∈ J) {
            foreach((B → γ) ∈ G) {
                if((B → ·γ) ∉ J) {
                    J = J ∪ {B → ·γ};
                    changed = true;
                }
            }
        }
    } while (changed);

    return J;
}
```


Syntaxanalyse von **id * id**

Stack	**Symbole**	**Eingabe**	**Aktion**
0	\$	**id** $*$ **id** \$	Verschieben (shift)
0 5	\$ **id**	$*$ **id** \$	Reduzieren durch $F \rightarrow$ **id**
0 3	\$ F	$*$ **id** \$	Reduzieren durch $T \rightarrow F$
0 2	\$ T	$*$ **id** \$	Verschieben
0 2 7	\$ T $*$	**id** \$	Verschieben
0 2 7 5	\$ T $*$ **id**	\$	Reduzieren durch $F \rightarrow$ **id**
0 2 7 10	\$ T $*$ F	\$	Reduzieren durch $T \rightarrow T*F$
0 2	\$ T	\$	Reduzieren durch $E \rightarrow T$
0 1	\$ E	\$	Akzeptieren

Modell eines LR-Parsers



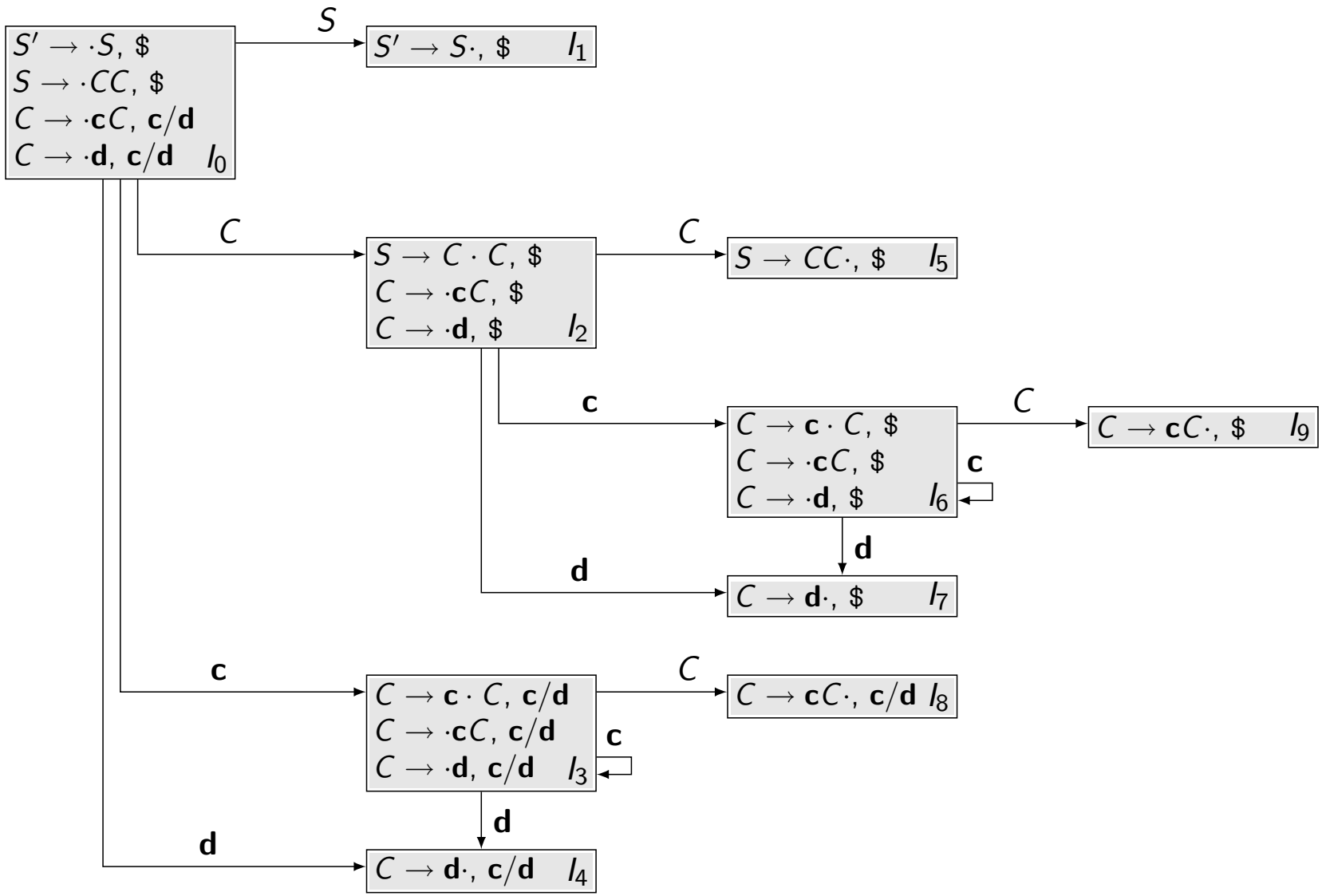
LR(1) Parseralgorithmus

```
push(0);
a = next_token();
while (true) {
    if (ACTION[top(), a] == shift t) {
        push(t);
        a = next_token();
    } else if(ACTION[top(), a] == reduce A → β) {
        for (i = 0; i < |β|; ++i)
            pop();
        push(GOTO[top(), A]);
    } else if(ACTION[top(), a] == accept) {
        break;
    } else {
        report_error();
    }
}
```


Beispiel: Grammatik G'

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow C\ C \\
C &\rightarrow \mathbf{c}\ C \mid \mathbf{d}
\end{aligned}
$$

GOTO-Graph für die Grammatik G'



Beispiel Situationen

Ist I die Menge der beiden Situationen $\{[E' \to E\cdot], [E \to E \cdot + T]\}$, dann enthält $\mathrm{GOTO}(I, +)$ folgende Situationen:

$$
\begin{array}{lcl}
E & \to & E + \cdot T \\
T & \to & \cdot T * F \\
T & \to & \cdot F \\
F & \to & \cdot(E) \\
F & \to & \cdot \mathbf{id}
\end{array}
$$

Berechnung der kanonischen $\mathrm{LR}(0)$-Situationsmengen

```
void items(grammar G) {
    C = { closure( {[S0 → ·S]} ) };
    do {
        changed = false;
        foreach(set<item> I ∈ C) {
            foreach(non_terminal X) {
                if (GOTO(I, X) ≠ ∅ && GOTO(I, X) ∉ C) {
                    C = C ∪ { closure(GOTO(I, X)) };
                    changed = true;
                }
            }
        }
    } while(changed);
}
```


Parsertabelle für Ausdrucksgrammatik

Zustand	ACTION						GOTO		
	id	+	*	(	)	$	*E*	*T*	*F*
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Verhalten eines LR-Parsers

Stack	Symbole	Eingabe	Aktion
0		**id** $*$ **id** $+$ **id** \$	Verschieben
0 5	**id**	$*$ **id** $+$ **id** \$	Reduzieren durch $F \rightarrow$ **id**
0 3	F	$*$ **id** $+$ **id** \$	Reduzieren durch $T \rightarrow F$
0 2	T	$*$ **id** $+$ **id** \$	Verschieben
0 2 7	T $*$	**id** $+$ **id** \$	Verschieben
0 2 7 5	T $*$ **id**	$+$ **id** \$	Reduzieren durch $F \rightarrow$ **id**
0 2 7 10	T $*$ F	$+$ **id** \$	Reduzieren durch $T \rightarrow T * F$
0 2	T	$+$ **id** \$	Reduzieren durch $E \rightarrow T$
0 1	E	$+$ **id** \$	Verschieben
0 1 6	E $+$	**id** \$	Verschieben
0 1 6 5	E $+$ **id**	\$	Reduzieren durch $F \rightarrow$ **id**
0 1 6 3	E $+$ F	\$	Reduzieren durch $T \rightarrow F$
0 1 6 9	E $+$ T	\$	Reduzieren durch $E \rightarrow E + T$
0 1	E	\$	Akzeptieren

Fehlerbehandlung für LL-Parser

Panikmodus: Überlese Tokens bis nächstes Token in synchronisierender Menge (Ankermenge). Ankermengen werden beim rekursiven Abstieg mitübergeben bzw. stehen zu einem Nichtterminal auf dem Stack.
Berechnung der Ankermenge für Nichtterminal A: Ank(A)

1. erste Näherung: FOLLOW(A). Aber das reicht nicht: z.B. fehlendes Semikolon würde zu Folgefehlern führen.
2. Erweiterung von Ank(A) um solche Tokens, die „übergeordnete" Strukturen beginnen. Beispiel: *statement* ist übergeordnet zu *expression*: FIRST(*statement*) wird zu Ank(*expression*) hinzugefügt.
3. wenn $\epsilon \in L(A)$, wird an Fehlerstelle $A \longrightarrow \epsilon$ expandiert und normal fortgefahren.
4. wenn ein Topstack-Terminal nicht kommt, wird es gepoppt nebst Meldung „Terminal xx expected".

Fehlerbehandlung für LR-Parser

Beispiel für Grammatik:

$$E \rightarrow E{+}E \mid E{*}E \mid (E) \mid \mathbf{id}$$

Erzeuge 4 Aktionen zur Fehlerbehebung:

e1 **id** erwartet aber nicht gefunden.
Aktion: Shift 3; Nachricht „`Operand fehlt`".

e2 **)** gefunden ohne vorherige **(**.
Aktion: Überspringe Token; Nachricht „`) ohne Gegenstück`".

e3 Operator erwartet aber **id** oder **)** gefunden.
Aktion: Shift 4; Nachricht „`Operator fehlt`".

e4 $ gefunden aber noch Klammern geöffnet.
Aktion: Nachricht „`) fehlt`".

LR-Parsertabelle mit Fehlerroutinen

Zustand	ACTION						GOTO
	id	+	*	(	)	$	*E*
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc.	
2	s3	e1	e1	s2	e2	e1	6
3	r4	r4	r4	r4	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	r1	r1	s5	r1	r1	r1	
8	r2	r2	r2	r2	r2	r2	
9	r3	r3	r3	r3	r3	r3	

Earley Parser

- Algorithmus zum parsen kontextfreier Grammatiken mit dynamischer Programmierung.
- Verarbeitet mehrdeutige Grammatiken und erzeugt alle möglichen Parsebäume eines Satzes.
- Laufzeit liegt im allgemeinen in $O(n^3)$; in $O(n^2)$ für eindeutige Grammatiken; in $O(n)$ für fast alle $\mathrm{LR}(k)$ Grammatiken.
- Häufiger Einsatz in der Computerlinguistik.

Algorithmus

- Erweitere die Grammatik um eine Regel $S' \rightarrow S$.
- Für jede Eingabeposition wird Menge S_i von Situationen $[X \rightarrow \alpha \cdot \beta, j]$ berechnet. j gibt die Eingabeposition an. Initiale Menge: $\{[S' \rightarrow \cdot S, 0]\}$.
- Für jede Eingabeposition i wende 3 Schritte iterativ bis zum Fixpunkt an:
  - **Prediction**: Für jede Situation $[X \rightarrow \alpha \cdot A\ \beta, j]$ in S_i füge eine neue Situation $[A \rightarrow \cdot\gamma, i]$ für jede Produktion der Grammatik mit A auf der linken Seite ein (transitive Hülle bilden).
  - **Completion**: Für jede Situation $[A \rightarrow \gamma\cdot, j]$ in S_i füge für jede Situation $[X \rightarrow \alpha \cdot A\ \beta, k]$ in S_j eine neue Situation $[X \rightarrow \alpha\ A \cdot \beta, k]$ in S_i ein.
  - **Scanning**: Sei **a** das nächste Token. Füge für jede Situation $[X \rightarrow \alpha \cdot \mathbf{a}\ \beta, j]$ in S_i eine Situation $[X \rightarrow \alpha\ \mathbf{a} \cdot \beta, j]$ in S_{i+1} ein.

Beispiel

Grammatik:

$$
\begin{aligned}
S &\rightarrow S \textbf{ und } S \mid S \textbf{ oder } S \\
S &\rightarrow \textbf{blau} \mid \textbf{gestreift} \mid \textbf{glatt} \mid \textbf{teuer}
\end{aligned}
$$

Eingabe: **blau und gestreift oder glatt und teuer**

Beispiel

Eingabe: **blau und gestreift oder glatt und teuer** \$ (Pos. 0)

Situationen S_0:
$[S' \rightarrow \cdot S, 0]$

Iteration:
$[S \rightarrow \cdot S \textbf{ und } S, 0]$
$[S \rightarrow \cdot S \textbf{ oder } S, 0]$
$[S \rightarrow \cdot \textbf{blau}, 0]$
$[S \rightarrow \cdot \textbf{gestreift}, 0]$
$[S \rightarrow \cdot \textbf{glatt}, 0]$
$[S \rightarrow \cdot \textbf{teuer}, 0]$

Beispiel

Eingabe: **blau und gestreift oder glatt und teuer** \$ (Pos. 1)

Situationen S_1:
$[S \rightarrow \textbf{blau}\cdot, 0]$

Iteration:
$[S' \rightarrow S\cdot, 0]$
$[S \rightarrow S \ \cdot \ \textbf{und}\ S, 0]$
$[S \rightarrow S \ \cdot \ \textbf{oder}\ S, 0]$

Beispiel

Eingabe: **blau und gestreift oder glatt und teuer** \$ (Pos. 2)

Situationen S_2:
$[S \rightarrow S$ **und** $\cdot S, 0]$

Iteration:
$[S \rightarrow \cdot S$ **und** $S, 2]$
$[S \rightarrow \cdot S$ **oder** $S, 2]$
$[S \rightarrow \cdot$**blau**$, 2]$
$[S \rightarrow \cdot$**gestreift**$, 2]$
$[S \rightarrow \cdot$**glatt**$, 2]$
$[S \rightarrow \cdot$**teuer**$, 2]$

Beispiel

Eingabe: **blau und gestreift oder glatt und teuer** \$ (Pos. 3)

Situationen S_3:
$[S \rightarrow \textbf{gestreift}, 2]$

Iteration:
$[S \rightarrow S\ \textbf{und}\ S\cdot, 0]$
$[S' \rightarrow S\cdot, 0]$ $[S \rightarrow S\ \cdot\ \textbf{und}\ S, 0]$
$[S \rightarrow S\ \cdot\ \textbf{oder}\ S, 0]$
$[S \rightarrow S\ \cdot\ \textbf{und}\ S, 2]$
$[S \rightarrow S\ \cdot\ \textbf{oder}\ S, 2]$

Beispiel

Eingabe: **blau und gestreift oder glatt und teuer** \$ (Pos. 4)

Situationen S_4:
$[S \rightarrow S$ **oder** $\cdot\, S, 0]$
$[S \rightarrow S$ **oder** $\cdot\, S, 2]$

Iteration:
$[S \rightarrow \cdot S$ **und** $S, 4]$
$[S \rightarrow \cdot S$ **oder** $S, 4]$
$[S \rightarrow \cdot$**blau**$, 4]$
$[S \rightarrow \cdot$**gestreift**$, 4]$
$[S \rightarrow \cdot$**glatt**$, 4]$
$[S \rightarrow \cdot$**teuer**$, 4]$

Beispiel

Eingabe: **blau und gestreift oder glatt und teuer** \$ (Pos. 5)

Situationen S_5:
$[S \rightarrow \textbf{glatt}\cdot, 4]$

Iteration:

$[S \rightarrow S \textbf{ oder } S\cdot, 0]$
$[S \rightarrow S \textbf{ oder } S\cdot, 2]$
$[S \rightarrow S \ \cdot \textbf{und } S, 4]$
$[S \rightarrow S \ \cdot \textbf{oder } S, 4]$

$[S' \rightarrow S\cdot, 0]$
$[S \rightarrow S \ \cdot \textbf{und } S, 0]$
$[S \rightarrow S \ \cdot \textbf{oder } S, 0]$
$[S \rightarrow S \textbf{ und } S\cdot, 0]$
$[S \rightarrow S \ \cdot \textbf{und } S, 2]$
$[S \rightarrow S \ \cdot \textbf{oder } S, 2]$

Beispiel

Eingabe: **blau und gestreift oder glatt und teuer** $ (Pos. 6)

Situationen S_6:

$[S \rightarrow S\ \textbf{und}\ \cdot S, 4]$
$[S \rightarrow S\ \textbf{und}\ \cdot S, 0]$
$[S \rightarrow S\ \textbf{und}\ \cdot S, 2]$

Iteration:

$[S \rightarrow \cdot S\ \textbf{und}\ S, 6]$
$[S \rightarrow \cdot S\ \textbf{oder}\ S, 6]$
$[S \rightarrow \cdot \textbf{blau}, 6]$
$[S \rightarrow \cdot \textbf{gestreift}, 6]$
$[S \rightarrow \cdot \textbf{glatt}, 6]$
$[S \rightarrow \cdot \textbf{teuer}, 6]$

Beispiel

Eingabe: **blau und gestreift oder glatt und teuer** $ (Pos. 7)

Situationen S_7:
$[S \rightarrow \textbf{teuer}\cdot, 6]$

Iteration:

$[S \rightarrow S \ \cdot \textbf{und}\ S, 6]$
$[S \rightarrow S \ \cdot \textbf{oder}\ S, 6]$
$[S \rightarrow S\ \textbf{und}\ S\cdot, 4]$
$[S \rightarrow S\ \textbf{und}\ S\cdot, 0]$
$[S \rightarrow S\ \textbf{und}\ S\cdot, 2]$

$[S \rightarrow S \ \cdot \textbf{und}\ S, 4]$
$[S \rightarrow S \ \cdot \textbf{oder}\ S, 4]$
$[S' \rightarrow S\cdot, 0]$
$[S \rightarrow S \ \cdot \textbf{und}\ S, 0]$
$[S \rightarrow S \ \cdot \textbf{oder}\ S, 0]$
$[S \rightarrow S \ \cdot \textbf{und}\ S, 2]$
$[S \rightarrow S \ \cdot \textbf{oder}\ S, 2]$