Der Stadiumautomat: Beispiel

Beispiel 1.1:
Sei $G' = (\{Z, E, T, F\},\ \{+, *, (,), a\},\ P,\ Z)$
die (um Z als neues Startsymbol erweiterte) Grammatik von G_4 mit
$P = \{Z \to E,\ E \to E{+}T \mid T,\ T \to T{*}F \mid F,\ F \to (E) \mid a\}$.

(Ein Teil der) Übergangsrelation Δ des Stadiumautomaten K_{G_4} ist in folgender Tabelle dargestellt (im Skript vollständig).

altes oberes Kellerende	Eingabe	neues oberes Kellerende	Übergang
$[Z \to \cdot\, E]$	ε	$[Z \to \cdot\, E]\ [E \to \cdot\, E+T]$	Expansion
$[Z \to \cdot\, E]\ [E \to E+T\, \cdot]$	ε	$[Z \to E\, \cdot]$	Reduktion
$[Z \to \cdot\, E]$	ε	$[Z \to \cdot\, E]\ [E \to \cdot\, T]$	Expansion
$[Z \to \cdot\, E]\ [E \to T\, \cdot]$	ε	$[Z \to E\, \cdot]$	Reduktion
$[E \to \cdot\, E+T]$	ε	$[E \to \cdot\, E+T]\ [E \to \cdot\, E+T]$	Expansion
$[E \to \cdot\, E+T]\ [E \to E+T\, \cdot]$	ε	$[E \to E\, \cdot +T]$	Reduktion
$[E \to \cdot\, E+T]$	ε	$[E \to \cdot\, E+T]\ [E \to \cdot\, T]$	Expansion
$[E \to \cdot\, E+T]\ [E \to T\, \cdot]$	ε	$[E \to E\, \cdot +T]$	Reduktion
$[E \to E\, \cdot +T]$	$+$	$[E \to E+\cdot\, T]$	Lesen
$[E \to E+\cdot\, T]$	ε	$[E \to E+\cdot\, T]\ [T \to \cdot\, T*F]$	Expansion
$[E \to E+\cdot\, T]\ [T \to T*F\, \cdot]$	ε	$[E \to E+T\, \cdot]$	Reduktion
$[E \to E+\cdot\, T]$	ε	$[E \to E+\cdot\, T]\ [T \to \cdot\, F]$	Expansion
$[E \to E+\cdot\, T]\ [T \to F\, \cdot]$	ε	$[E \to E+T\, \cdot]$	Reduktion
$[E \to \cdot\, T]$	ε	$[E \to \cdot\, T]\ [T \to \cdot\, T*F]$	Expansion
$[E \to \cdot\, T]\ [T \to T*F\, \cdot]$	ε	$[E \to T\, \cdot]$	Reduktion
$[E \to \cdot\, T]$	ε	$[E \to \cdot\, T]\ [T \to \cdot\, F]$	Expansion
$[E \to \cdot\, T]\ [T \to F\, \cdot]$	ε	$[E \to T\, \cdot]$	Reduktion
$[T \to \cdot\, T*F]$	ε	$[T \to \cdot\, T*F]\ [T \to \cdot\, T*F]$	Expansion
$[T \to \cdot\, T*F]\ [T \to T*F\, \cdot]$	ε	$[T \to T\, \cdot *F]$	Reduktion
$[T \to \cdot\, T*F]$	ε	$[T \to \cdot\, T*F]\ [T \to \cdot\, F]$	Expansion
$[T \to \cdot\, T*F]\ [T \to F\, \cdot]$	ε	$[T \to T\, \cdot *F]$	Reduktion

Kellerautomaten mit Ausgabe

Ein Kellerautomat ist ein Akzeptor:
er entscheidet nur, ob ein Wort zur Sprache gehört oder nicht.

Die Syntaxanalyse soll aber auch die syntaktische Struktur des akzeptierten Wortes liefern, z.B. als Syntaxbaum oder als Folge der in einer Rechts- bzw. Linksableitung benutzten Produktionen.

Ein **Kellerautomat mit Ausgabe** ist ein Tupel $K = (Q, T, O, \Delta, q_0, F)$, also ein Kellerautomat mit zusätzlichem endlichen **Ausgabealphabet** O.

Bei jedem Übergang *kann* der Automat ein Symbol aus O ausgeben:
Δ wird endliche Relation zwischen $Q^+ \times (T \cup \{\varepsilon\})$ und $Q^* \times (O \cup \{\varepsilon\})$.

Eine **Konfiguration** ist ein Element aus $Q^+ \times T^* \times O^*$ und beschreibt den aktuellen Kellerinhalt, die restliche Eingabe und die bisherige Ausgabe.

Kellerautomaten als Parser

Ausgabealphabet eines Parsers:
Produktionen der kontextfreien Grammatik (oder ihre Nummern).

Ein Stadiumautomat soll die benutzte Produktion ausgeben bei jedem

- *Expansion*sübergang: der aktuelle Zustand ist ein *initiales* Stadium. Für jede akzeptierende Berechnung ist dann die Ausgabe eine *Linksableitung* für das akzeptierte Wort: **Linksparser**.

- *Reduktion*sübergang: der aktuelle Zustand ist ein *finales* Stadium. Für jede akzeptierende Berechnung ist die Ausgabe eine *gespiegelte Rechtsableitung* (Linksreduktion) für das akzeptierte Wort: **Rechtsparser**.

Beispiel: Linkspartei und Rechtsparser

Beispiel 1.2:
Sei $G' = (\{Z, E, T, F\},\ \{+, *, (,), a\},\ P,\ Z)$ die erweiterte Grammatik zu G_4 aus Beispiel 1.1 mit
$P = \{Z \to E,\ E \to E{+}T \mid T,\ T \to T{*}F \mid F,\ F \to (E) \mid a\}$.

Wir geben die (einzige) Konfigurationsfolge des Stadiumautomaten K_{G_4}, die zur Akzeptanz des Wortes $a + a * a$ führt, an zusammen mit der Ausgabe eines Links- bzw. eines Rechtsparsers:

Kellerinhalt	Eingabe (Rest)	Ausgabe Links-parser	Ausgabe Rechts-parser
$[Z \to \cdot\, E]$	$a+a*a$	$Z \to E$	
$[Z \to \cdot\, E]\ [E \to \cdot\, E{+}T]$	$a+a*a$	$E \to E{+}T$	
$[Z \to \cdot\, E]\ [E \to \cdot\, E{+}T]\ [E \to \cdot\, T]$	$a+a*a$	$E \to T$	
$[Z \to \cdot\, E]\ [E \to \cdot\, E{+}T]\ [E \to \cdot\, T]\ [T \to \cdot\, F]$	$a+a*a$	$T \to F$	
$[Z \to \cdot\, E]\ [E \to \cdot\, E{+}T]\ [E \to \cdot\, T]\ [T \to \cdot\, F]\ [F \to \cdot\, a]$	$a+a*a$	$F \to a$	
$[Z \to \cdot\, E]\ [E \to \cdot\, E{+}T]\ [E \to \cdot\, T]\ [T \to \cdot\, F]\ [F \to a\, \cdot]$	$+a*a$		$F \to a$
$[Z \to \cdot\, E]\ [E \to \cdot\, E{+}T]\ [E \to \cdot\, T]\ [T \to F\, \cdot]$	$+a*a$		$T \to F$
$[Z \to \cdot\, E]\ [E \to \cdot\, E{+}T]\ [E \to T\, \cdot]$	$+a*a$		$E \to T$
$[Z \to \cdot\, E]\ [E \to E \cdot +T]$	$+a*a$		
$[Z \to \cdot\, E]\ [E \to E{+} \cdot\, T]$	$a*a$		
$[Z \to \cdot\, E]\ [E \to E{+} \cdot\, T]\ [T \to \cdot\, T{*}F]$	$a*a$	$T \to T{*}F$	
$[Z \to \cdot\, E]\ [E \to E{+} \cdot\, T]\ [T \to \cdot\, T{*}F]\ [T \to \cdot\, F]$	$a*a$	$T \to F$	
$[Z \to \cdot\, E]\ [E \to E{+} \cdot\, T]\ [T \to \cdot\, T{*}F]\ [T \to \cdot\, F]\ [F \to \cdot\, a]$	$a*a$	$F \to a$	
$[Z \to \cdot\, E]\ [E \to E{+} \cdot\, T]\ [T \to \cdot\, T{*}F]\ [T \to \cdot\, F]\ [F \to a\, \cdot]$	$*a$		$F \to a$
$[Z \to \cdot\, E]\ [E \to E{+} \cdot\, T]\ [T \to \cdot\, T{*}F]\ [T \to F\, \cdot]$	$*a$		$T \to F$
$[Z \to \cdot\, E]\ [E \to E{+} \cdot\, T]\ [T \to T \cdot *F]$	$*a$		
$[Z \to \cdot\, E]\ [E \to E{+} \cdot\, T]\ [T \to T{*} \cdot\, F]$	a		
$[Z \to \cdot\, E]\ [E \to E{+} \cdot\, T]\ [T \to T{*} \cdot\, F]\ [F \to \cdot\, a]$	a	$F \to a$	
$[Z \to \cdot\, E]\ [E \to E{+} \cdot\, T]\ [T \to T{*} \cdot\, F]\ [F \to a\, \cdot]$			$F \to a$
$[Z \to \cdot\, E]\ [E \to E{+} \cdot\, T]\ [T \to T{*}F\, \cdot]$			$T \to T{*}F$
$[Z \to \cdot\, E]\ [E \to E{+}T\, \cdot]$			$E \to E{+}T$
$[Z \to E\, \cdot]$			$Z \to E$

Der Linksparser erzeugt folgende Linksableitung: $Z \Rightarrow E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow a+T \Rightarrow a+T*F \Rightarrow a+F*F \Rightarrow a+a*F \Rightarrow a+a*a$

Der Rechtsparser erzeugt folgende Linksreduktion: $a+a*a \Leftarrow F+a*a \Leftarrow T+a*a \Leftarrow E+a*a \Leftarrow E+F*a \Leftarrow E+T*a \Leftarrow E+T*F \Leftarrow E+T \Leftarrow E \Leftarrow Z$

Der charakteristische endliche Automat eines Stadiumautomaten

Statt durch die Übergangsrelation Δ (wie in Beispiel 2.1) kann man den Stadiumautomaten K_G weitgehend durch einen nichtdeterministischen endlichen Automaten beschreiben, seinen **charakteristischen endlichen Automaten** $c(G)$.

Mit einigen Übergängen von $c(G)$ muss man allerdings Kelleroperationen verbinden.

Der charakteristische endliche Automat eines Stadiumautomaten

Definition:
Sei G eine reduzierte, kontextfreie Grammatik und G^+ ihre Erweiterung.
Der nichtdeterministische endl. Automat $c(G) = (Q_c, V_c, \Delta_c, q_c, F_c)$ heißt **charakteristischer endlicher Automat** zu G, wenn:

$Q_c = St_{G^+}$ Zustände: Stadien der Grammatik

$V_c = N \cup T$ Eingabealphabet: Nichtterminal- und Terminalsymbole

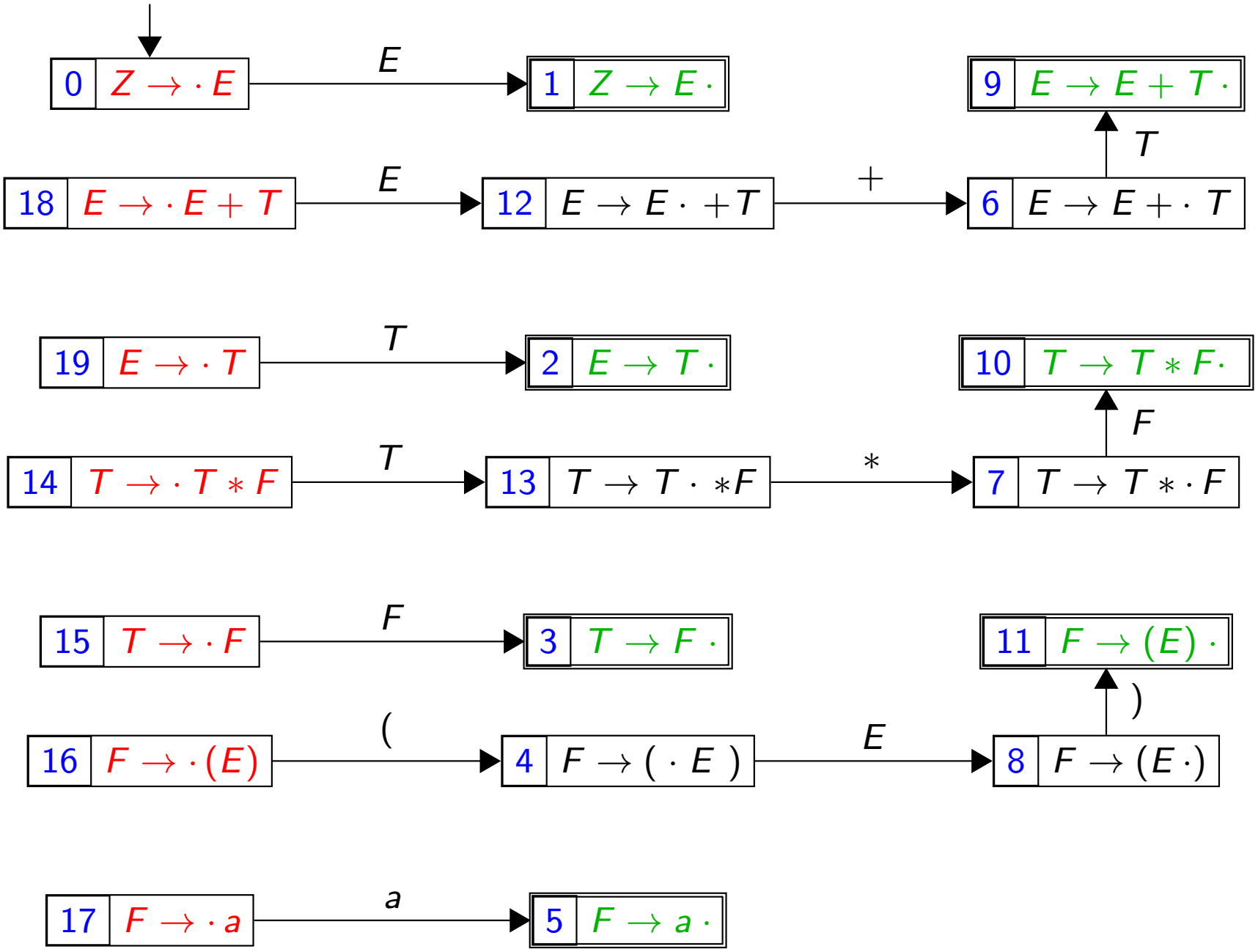
$q_c = [Z \to \cdot S]$ Startzustand: initiales Stadium der neuen Produktion

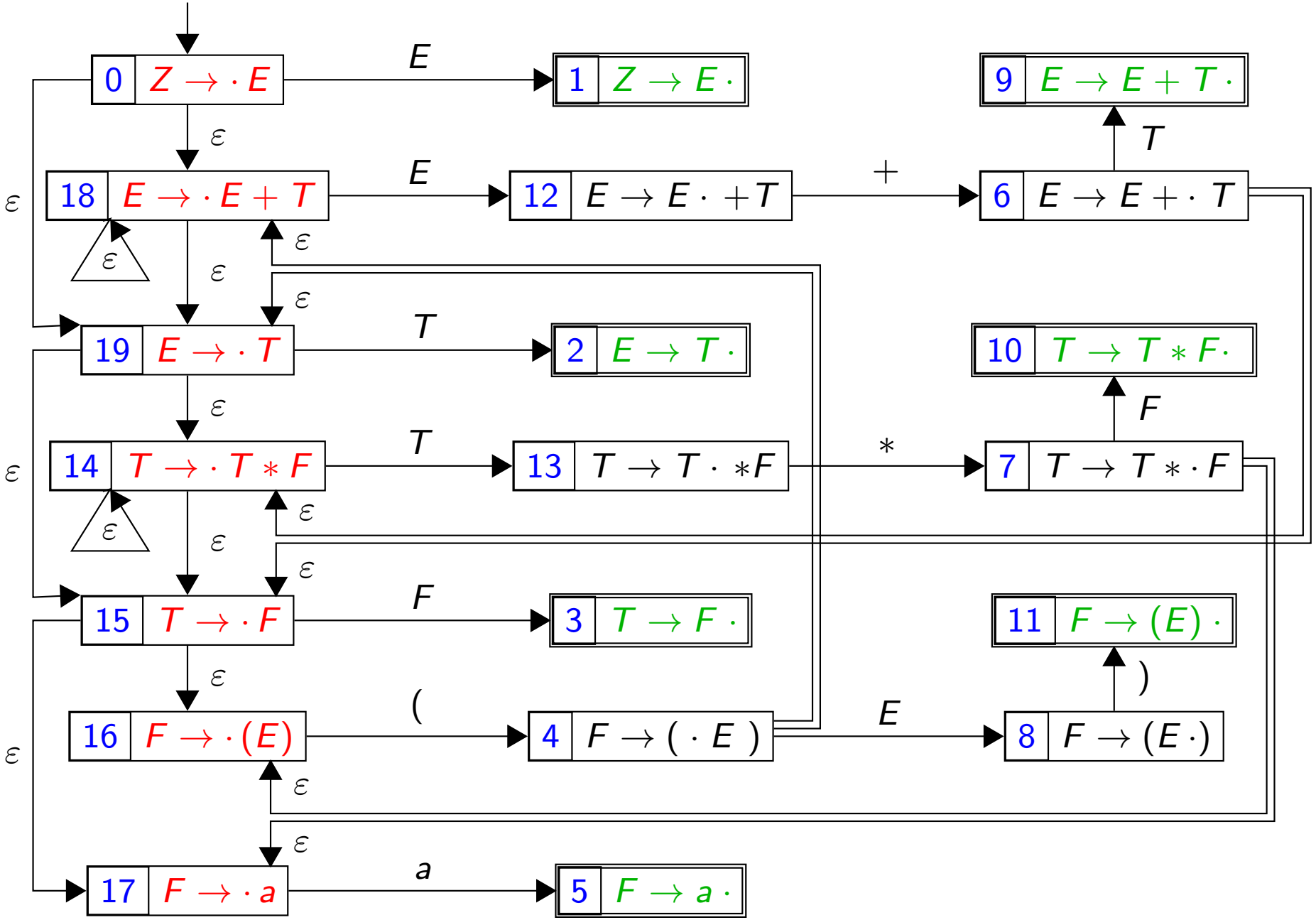
$F_c = \{[X \to \alpha \cdot] \mid X \to \alpha \in P\}$ Endzustände: finale Stadien

$$\Delta_c = \{([X \to \alpha \cdot Y\beta], Y, [X \to \alpha Y \cdot \beta]) \mid X \to \alpha Y \beta \in P \wedge Y \in N \cup T\}$$
$$\bigcup \{([X \to \alpha \cdot Y\beta], \varepsilon, [Y \to \cdot \gamma]) \mid X \to \alpha Y \beta \in P \wedge Y \to \gamma \in P\}$$

Beispiel: charakteristischer endlicher Automat

Beispiel 2.1:
Der charakteristische endliche Automat $c(G_4)$ zur Grammatik G_4 mit Produktionen
(0) $Z \to E$,
(1) $E \to E + T$,
(2) $E \to T$,
(3) $T \to T * F$,
(4) $T \to F$,
(5) $F \to (E)$,
(6) $F \to a$.


0 $Z \to \cdot E$
E
1 $Z \to E \cdot$
9 $E \to E + T \cdot$
T
18 $E \to \cdot E + T$
E
12 $E \to E \cdot + T$
+
6 $E \to E + \cdot T$
19 $E \to \cdot T$
T
2 $E \to T \cdot$
10 $T \to T * F \cdot$
F
14 $T \to \cdot T * F$
T
13 $T \to T \cdot * F$
*
7 $T \to T * \cdot F$
15 $T \to \cdot F$
F
3 $T \to F \cdot$
11 $F \to (E) \cdot$
)
16 $F \to \cdot (E)$
(
4 $F \to (\cdot E)$
E
8 $F \to (E \cdot)$
17 $F \to \cdot a$
a
5 $F \to a \cdot$


0 Z → · E
1 Z → E ·
9 E → E + T ·
18 E → · E + T
12 E → E · + T
6 E → E + · T
19 E → · T
2 E → T ·
10 T → T ∗ F ·
14 T → · T ∗ F
13 T → T · ∗F
7 T → T ∗ · F
15 T → · F
3 T → F ·
11 F → (E) ·
16 F → · (E)
4 F → (· E)
8 F → (E ·)
17 F → · a
5 F → a ·
E
T
F
a
(
)
+
∗
ε

Der charakteristische endliche Automat eines Stadiumautomaten

- $c(G)$ soll genau dann in den Zustand $[X \to \alpha \cdot \beta]$ übergehen, wenn der Stadiumautomat einen Keller ρ erreichen kann, dessen *Vergangenheit* mit dem von $c(G)$ gelesenen Wort übereinstimmt.
  $c(G)$ liest also die bereits abgearbeiteten Präfixe von rechten Seiten von Produktionen ein, deren Abarbeitung im Stadiumautomaten zum aktuellen Stadium führten.
- Die Menge der Eingabesymbole von $c(G)$ ist $N \cup T$, weil die Produktionen Nichtterminal- und Terminalsymbole enthalten können.
- Endzustände von $c(G)$ sind die finalen Stadien $[A \to \alpha \cdot]$.
  In einem Endzustand entspricht das gelesene Wort einem Kellerinhalt von K_G, in dem eine Reduktion mit der Produktion $A \to \alpha$ durchgeführt werden *kann*.

Der charakteristische endliche Automat eines Stadiumautomaten

- Das Lesen eines Terminalsymbols entspricht im Stadiumautomaten einem Leseübergang, das Lesen eines Nichtterminalsymbols dem Verschieben des Punkts nach einem Reduktionsübergang und die ε-Übergänge entsprechen den Expansionsübergängen.
- Einige Übergänge in $c(G)$ verbinden wir mit Keller-Operationen in K_G:
  - Bei jedem ε-Übergang in $c(G)$ legen wir den neuen Zustand von $c(G)$ oben auf den Keller von K_G (Expansionsübergang).
  - Gelangt $c(G)$ in einen Endzustand $[X \to \alpha \cdot]$, so wird in K_G das oberste Stadium $[X \to \alpha \cdot]$ vom Keller entfernt und das neue oberste Kellerstadium macht einen Übergang unter X (Reduktionsübergang).

Der kanonische LR(0)-Automat

Wir wollen den charakteristischen endlichen Automaten $c(G)$ jetzt deterministisch machen.

Bekanntlich kann man zu jedem nichtdeterminstischen endlichen Automaten einen deterministischen endlichen Automaten konstruieren, der die gleiche Sprache erkennt.

Dieser Algorithmus arbeitet in zwei Schritten:

Endlichen Automat deterministisch machen

1. ε-Übergänge im nichtdeterministischen endl. Automaten entfernen:
   - Alle ε-Übergänge fallen weg und alle Knoten, die *nur* durch ε-Übergänge erreicht werden.
   - Zusätzliche Kanten für jeden alten Weg, der aus ε-Übergängen und einem anschließenden Nicht-ε-Übergang besteht.
   - Knoten, die durch ε-Übergänge einen alten Endknoten erreichen, werden ebenfalls Endknoten.
2. Deterministischen endlichen Automaten mit Hilfe von Knoten*mengen* konstruieren (Potenzmengenkonstruktion):
   - Die Menge mit dem alten Startknoten wird neuer (Start-)Knoten.
   - Für jeden neuen Knoten wird für jedes Terminalsymbol die Menge der Folgeknoten bestimmt als Vereinigung der Mengen der alten Folgeknoten ihrer Elemente unter diesem Terminalsymbol und diese Menge wird zu einem neuen Knoten.
   - Jeder Knoten, der einen alten Endknoten enthält, wird ein neuer Endknoten.

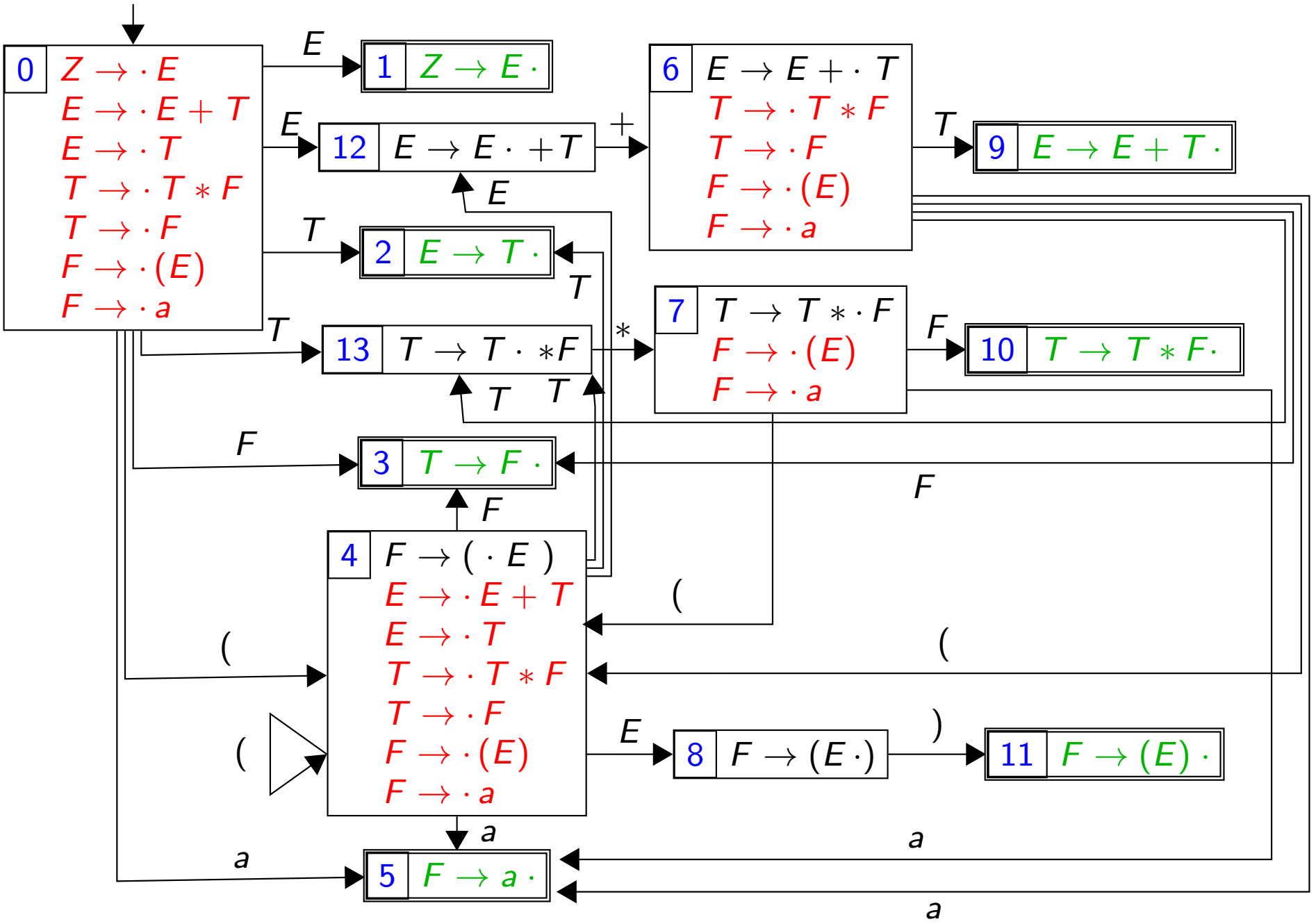
Der kanonische LR(0)-Automat

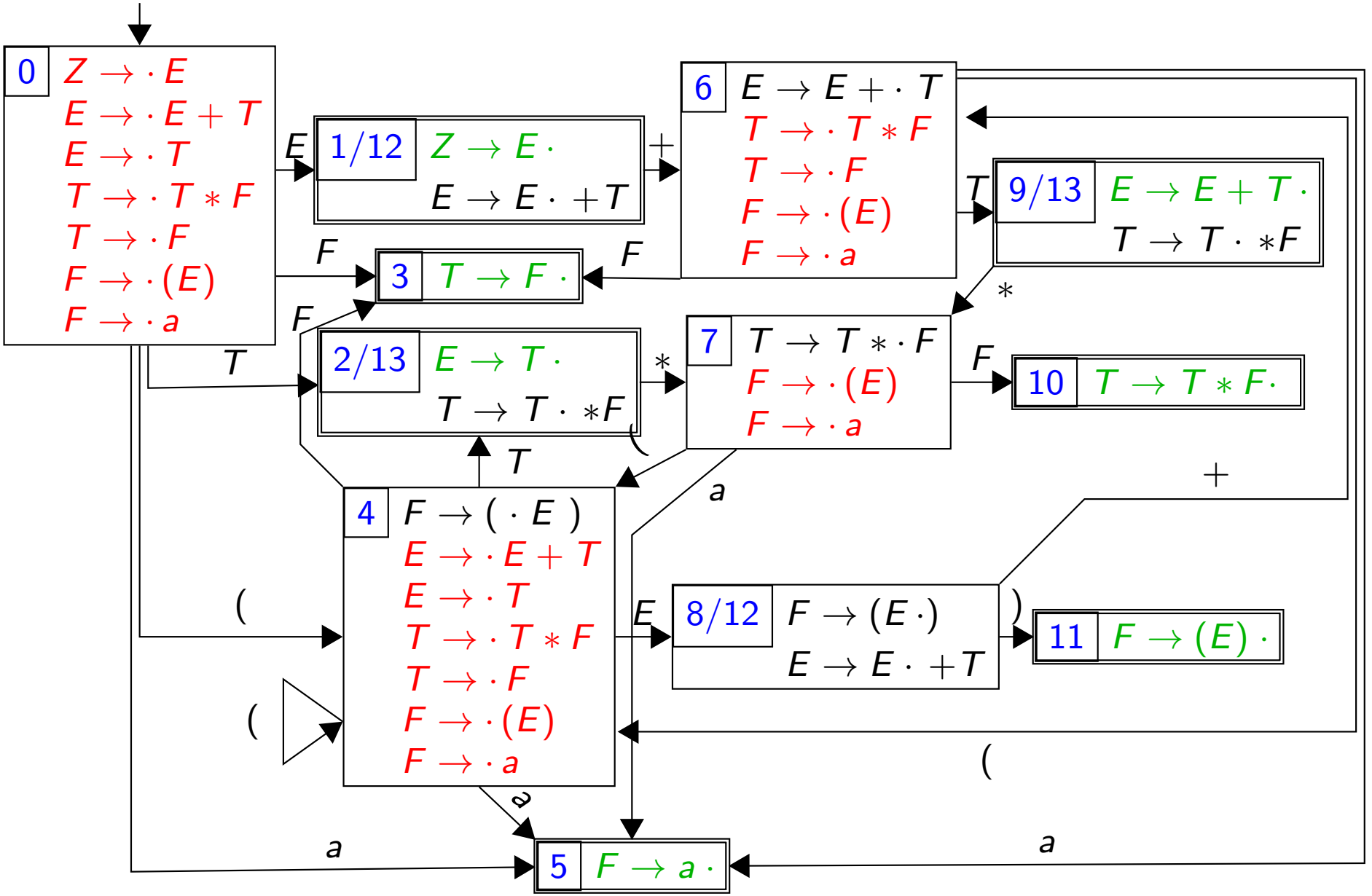
Wendet man diesen Algorithmus auf $c(G)$ an, so erhalten wir den determistischen endlichen Automaten $(Q_d, N \cup T, \Delta_d, q_d, F_d)$, den **kanonischen LR(0)-Automaten** für G, kurz $LR_0(G)$.

Beispiel 3.1:
Der Automat $c(G_4)$ aus Beispiel 3.1

- nach Entfernung der ε-Übergänge und
- nach der Potenzmengenkonstruktion.

Den Fehlerzustand (mit der leeren Menge von Stadien) und alle Zustandsübergänge in den Fehlerzustand lassen wir weg.


0
Z → · E
E → · E + T
E → · T
T → · T ∗ F
T → · F
F → · (E)
F → · a
1
Z → E ·
12
E → E · + T
2
E → T ·
13
T → T · ∗F
3
T → F ·
4
F → (· E)
E → · E + T
E → · T
T → · T ∗ F
T → · F
F → · (E)
F → · a
5
F → a ·
6
E → E + · T
T → · T ∗ F
T → · F
F → · (E)
F → · a
7
T → T ∗ · F
F → · (E)
F → · a
8
F → (E ·)
9
E → E + T ·
10
T → T ∗ F·
11
F → (E) ·
E
T
F
(
a
+
∗
)


0
Z → · E
E → · E + T
E → · T
T → · T ∗ F
T → · F
F → · (E)
F → · a
1/12
Z → E ·
E → E · + T
3
T → F ·
6
E → E + · T
T → · T ∗ F
T → · F
F → · (E)
F → · a
9/13
E → E + T ·
T → T · ∗F
2/13
E → T ·
T → T · ∗F
7
T → T ∗ · F
F → · (E)
F → · a
10
T → T ∗ F·
4
F → (· E)
E → · E + T
E → · T
T → · T ∗ F
T → · F
F → · (E)
F → · a
8/12
F → (E ·)
E → E · + T
11
F → (E) ·
5
F → a ·
E
T
F
+
*
(
)
a

Kombinierte Zustände notieren wir durch die kleinste enthaltene Nummer.

Wir fassen die wichtigsten Informationen des Übergangsdiagramms in einer Tabelle zusammen, indem wir für jeden Zustand die Nummern der Folgezustände und die Produktionen ihrer finalen Stadien notieren.

Zustand	a	$+$	$*$	$($	$)$	\$	E	T	F	
0	5			4			1	2	3	
1		6								$Z \to E$
2			7							$E \to T$
3										$T \to F$
4	5			4			8	2	3	
5										$F \to a$
6	5			4				9	3	
7	5			4					10	
8		6			11					
9			7							$E \to E+T$
10										$T \to T*F$
11										$F \to (E)$

Auf dem Weg zur Parser-Tabelle

Wir gehen also vom Zustand 0, wenn wir ein a einlesen, in den Zustand 5 und im Zustand 11 wenden wir die Produktion $F \to (E)$ an.

In welchen Zustand gehen wir, nachdem wir die Produktion $F \to (E)$ angewendet haben?
Dadurch wurde ja das Nonterminal F im vorletzten Stadium bearbeitet!
Von dessen Zustand gucken wir also in die Spalte dieses Nonterminals.

Aber was machen wir im Zustand 2, falls ein $*$ einzulesen ist:
Die Produktion $E \to T$ anwenden oder lesend in den Zustand 7 gehen?
Und welche Produktion wenden wir an, falls ein Zustand mehr als ein finales Stadium enthält?

Ohne Zusatzinformation könnte es *für jedes Eingabesymbol* zu einer Reduktion mit jeder in dieser Zeile angegebenen Produktion kommen.
Damit erhalten wir folgende LR(0)-Syntaxanalyse-Tabelle:

LR(0)-Syntaxanalyse-Tabelle (Langform)

LR(0)-Syntaxanalyse-Tabelle in ausführlicher Schreibweise

Zust.	Aktionstabelle						Sprungtabelle							
	a	$+$	$*$	$($	$)$	\$	a	$+$	$*$	$($	$)$	E	T	F
0	shift	error	error	shift	error	error	5			4		1	2	3
1	error	shift	error	error	error	accept		6						
2	$E \to T$	$E \to T$	$E \to T$/shift	$E \to T$	$E \to T$	$E \to T$			−/7					
3	$T \to F$	$T \to F$	$T \to F$	$T \to F$	$T \to F$	$T \to F$								
4	shift	error	error	shift	error	error	5			4		8	2	3
5	$F \to a$	$F \to a$	$F \to a$	$F \to a$	$F \to a$	$F \to a$								
6	shift	error	error	shift	error	error	5			4			9	3
7	shift	error	error	shift	error	error	5			4				10
8	error	shift	error	error	shift	error		6			11			
9	$E \to E+T$	$E \to E+T$	$E \to E+T$/shift	$E \to E+T$	$E \to E+T$	$E \to E+T$			−/7					
10	$T \to T*F$	$T \to T*F$	$T \to T*F$	$T \to T*F$	$T \to T*F$	$T \to T*F$								
11	$F \to (E)$	$F \to (E)$	$F \to (E)$	$F \to (E)$	$F \to (E)$	$F \to (E)$								

Diese Tabelle enthält zwei Konflikte (in Zustand 2 und 9; Spalte $*$);
G_4 ist also keine LR(0)-Grammatik.

Üblicherweise
integriert man die Folgezustände der shift-Operation in die Aktionstabelle und gibt bei reduce-Operationen nur die Nummer der Produktion an:

LR(0)-Syntaxanalyse-Tabelle (Kurzform)

LR(0)-Syntaxanalyse-Tabelle

Zust.	Aktionstabelle						Sprungtab.		
	a	$+$	$*$	$($	$)$	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2	r2	r2	r2/s7	r2	r2	r2			
3	r4	r4	r4	r4	r4	r4			
4	s5			s4			8	2	3
5	r6	r6	r6	r6	r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9	r1	r1	r1/s7	r1	r1	r1			
10	r3	r3	r3	r3	r3	r3			
11	r5	r5	r5	r5	r5	r5			

SLR(1)-Syntaxanalyse

Aber was machen wir im Zustand 2, falls ein $*$ einzulesen ist:
Die Produktion $E \to T$ anwenden oder lesend in den Zustand 7 gehen?
Entscheidungshilfe erhalten wir durch die Vorschausymbole.

Wir wissen für jeden Zustand, ob und falls ja, mit welcher Produktion zu reduzieren ist. Sei $A \to \alpha$ diese Produktion.
Und wir wissen bereits, welche Symbole auf das Nichtterminalsymbol A folgen können: die aus **Follow**(A)!

Ein **SLR(1)-Parser** („S“ steht für „simple“) setzt seine reduce-Einträge daher *in die Spalten der Vorschausymbole, die in der FOLLOW-Menge des Nichtterminalsymbols auf der linken Seite der Produktion enthalten sind*.

Für unsere Beispiel-Grammatik gibt es keine Konflikte, sie ist eine SLR(1)-Grammatik.

SLR(1)-Syntaxanalyse-Tabelle

A	Zust.	Aktionstabelle						Sprungtab.		
		a	$+$	$*$	(	)	\$	E	T	F
	0	s5			s4			1	2	3
Z	1		s6				acc			
E	2		r2	s7		r2	r2			
T	3		r4	r4		r4	r4			
	4	s5			s4			8	2	3
F	5		r6	r6		r6	r6			
	6	s5			s4				9	3
	7	s5			s4					10
	8		s6			s11				
E	9		r1	s7		r1	r1			
T	10		r3	r3		r3	r3			
F	11		r5	r5		r5	r5			

$$\text{Follow}(Z) = \{\$\},\quad \text{Follow}(E) = \{+,), \$\},$$
$$\text{Follow}(T) = \text{Follow}(F) = \{+, *,), \$\}$$

LR(1)-Syntaxanalyse

Dazu erweitern wir jedes Stadium um die im jeweiligen Zustand erwarteten Vorschau-Symbole (**LookAhead**-Menge), also der Menge der Terminalsymbole, die dem Nichtterminalsymbol auf der linken Seite der Produktion *in diesem Zustand* folgen können.

- Dem initialen Stadium $[Z \to \cdot\, S]$ der Startproduktion wird die **LookAhead**-Menge $\{\$\}$ zugeordnet.
- Kommt man von einem erweiterten Stadium $[A \to \alpha \cdot B\beta,\ L]$ (α bzw. β kann auch ε sein) zu einem Stadium $[B \to \cdot\, \gamma]$, so berechnet sich dessen **LookAhead**-Menge zu $\mathrm{FIRST}(\beta L)$ mit $\beta L = \{\beta x \mid x \in L\}$ und $\mathrm{FIRST}(L) = \bigcup_{\alpha \in L} \mathrm{FIRST}(\alpha)$.

Der **kanonische LR(1)-Parser** setzt seine reduce-Einträge dann *in die Spalten der Vorschausymbole, die in der* **LookAhead**-*Menge der Produktion für den jeweiligen Zustand enthalten sind.*

LALR(1)-Syntaxanalyse

Der **LALR(1)-Parser** („LA“ steht für „look ahead“; unglückliche Namensgebung, denn SLR(1) und LR(1) nutzen ebenfalls die Vorschau) kommt mit den Zuständen des LR(0)-Parsers aus, indem die Zustände des LR(1)-Parsers, die sich nur in den **LookAhead**-Mengen unterscheiden, zu einer Gruppe zusammengefasst und dann verschmolzen werden, indem die **LookAhead**-Mengen gleicher Stadien vereinigt werden.

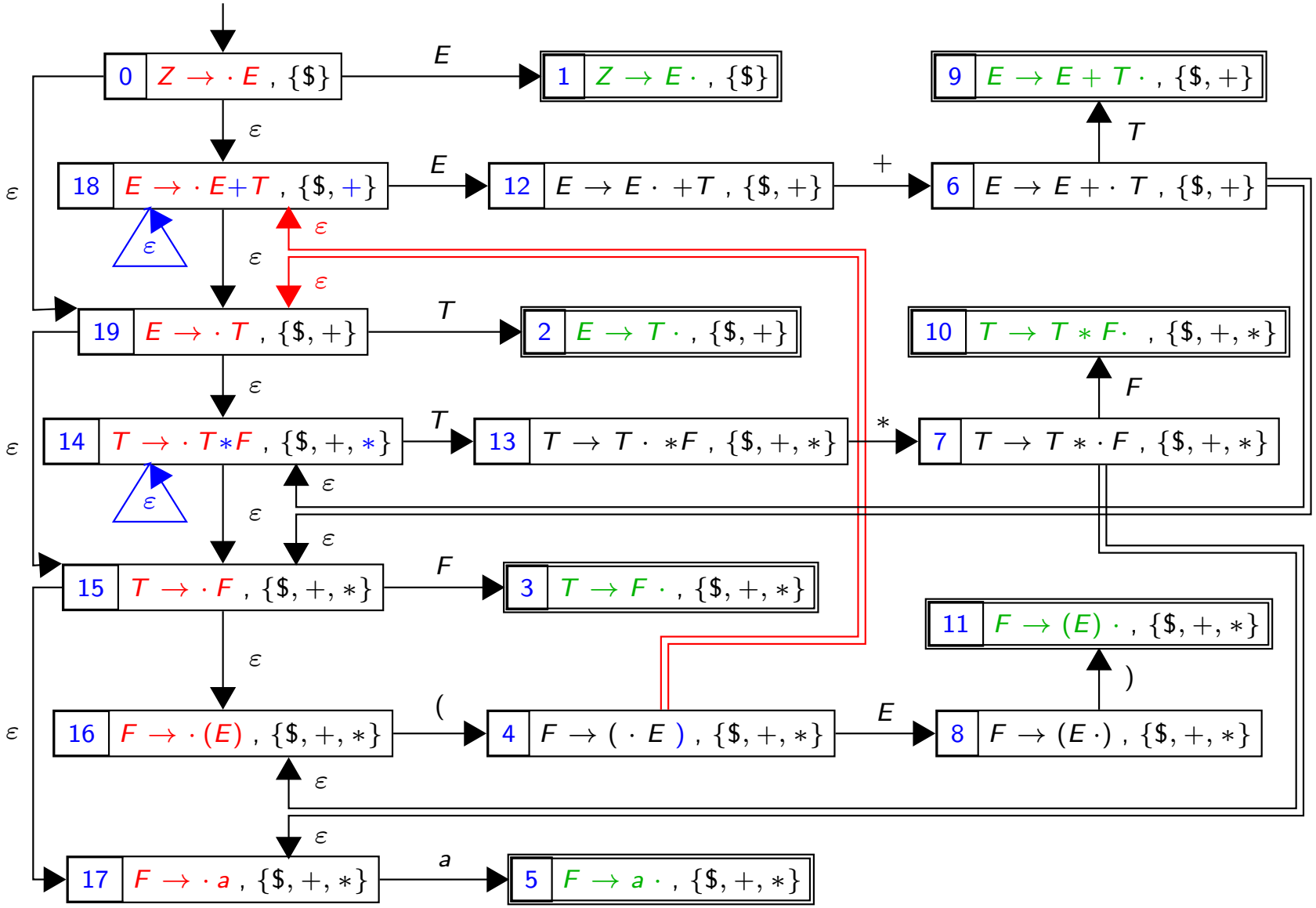
Der LALR(1)-Parser hat reduce-Einträge daher *in den Spalten der Vorschausymbole, an denen die verschmolzenen Zustände dieser Gruppe des LR(1)-Parsers reduce-Einträge hatten*.

Durch die zusätzlichen reduce-Einträge führt ein LALR(1)-Parser (ebenso wie ein SLR(1)- oder LR(0)-Parser) evtl. weitere Reduktionen aus, bevor er einen Fehler bemerkt.

LR(1)-Syntaxanalyse

Beispiel 4.1:
Der charakteristische endliche Automat zur Grammatik G_4 aus Beispiel 4.1 mit erweiterten Stadien.

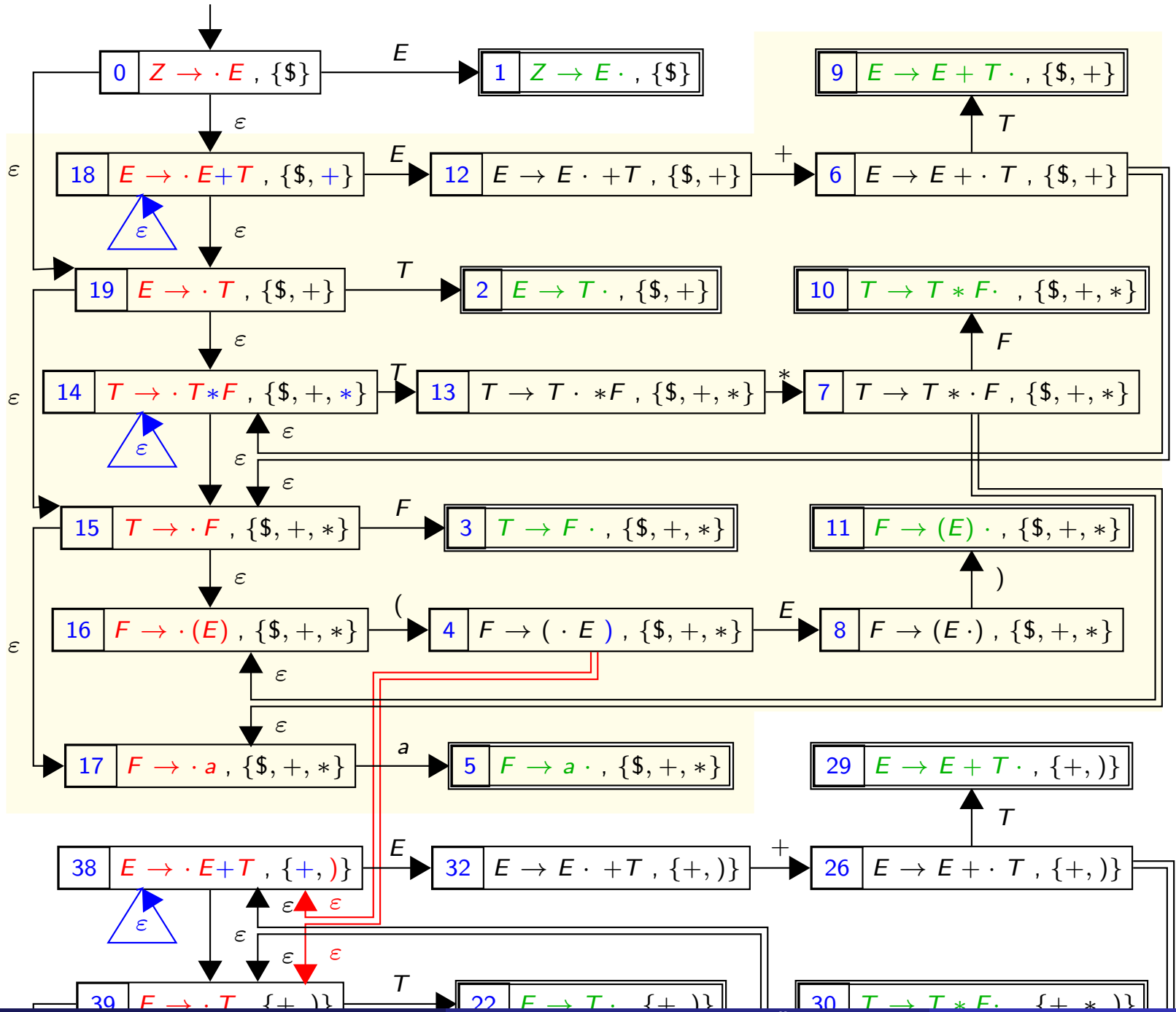
Versuch: Der charakteristische endliche Automat $c(G_4)$



LR(1)-Syntaxanalyse

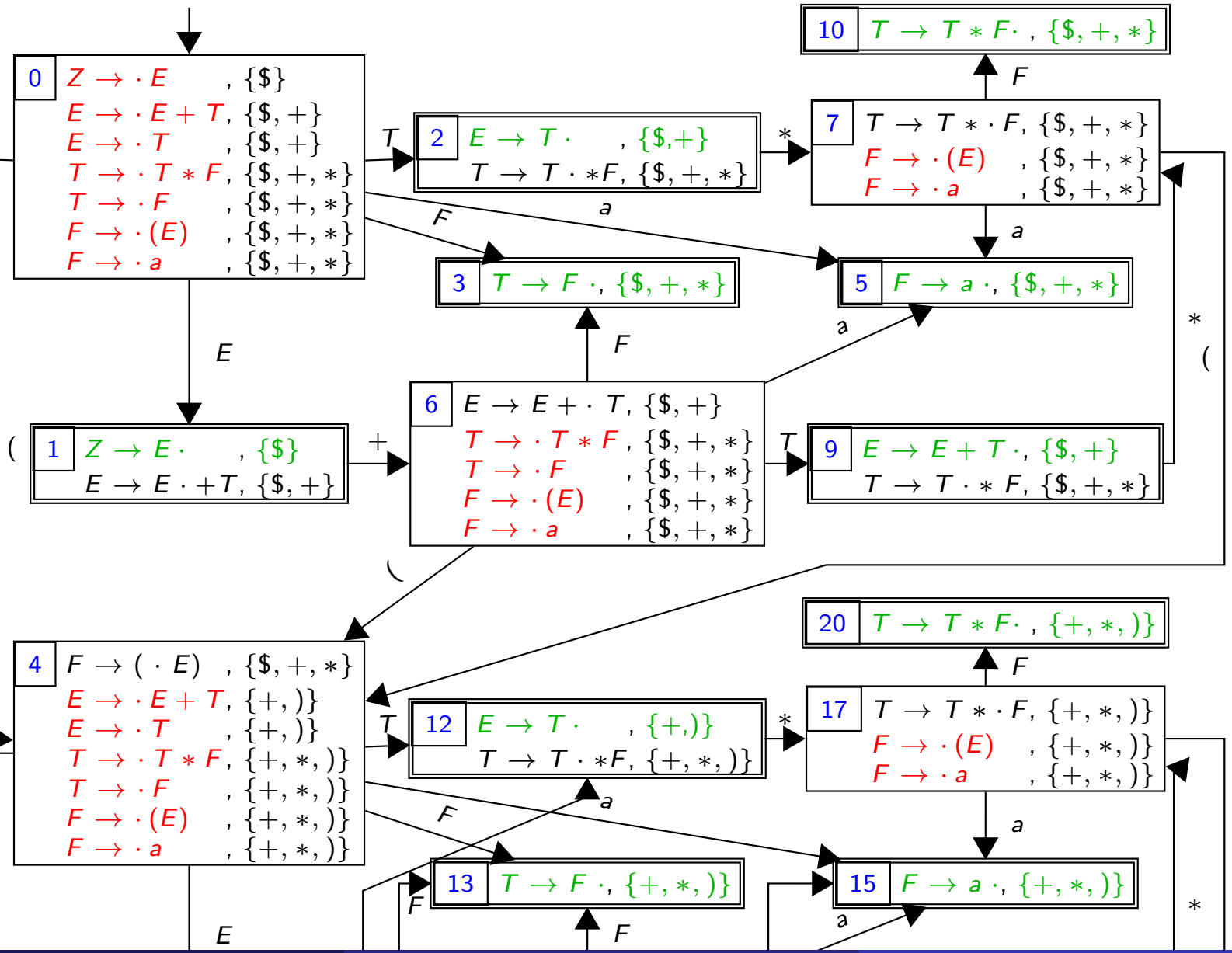
Eine Rückführung von S_4 nach S_{18}/S_{19} ist nicht möglich, da die Vorschau-Menge $\{)\}$ jetzt anders ist!
Denn nach einer „Klammer auf“ ist die Endmarke nicht mehr Vorschausymbol, sondern die „Klammer zu“.

Das führt zu einer Verdoppelung der Zustände S_2 bis S_{19}.

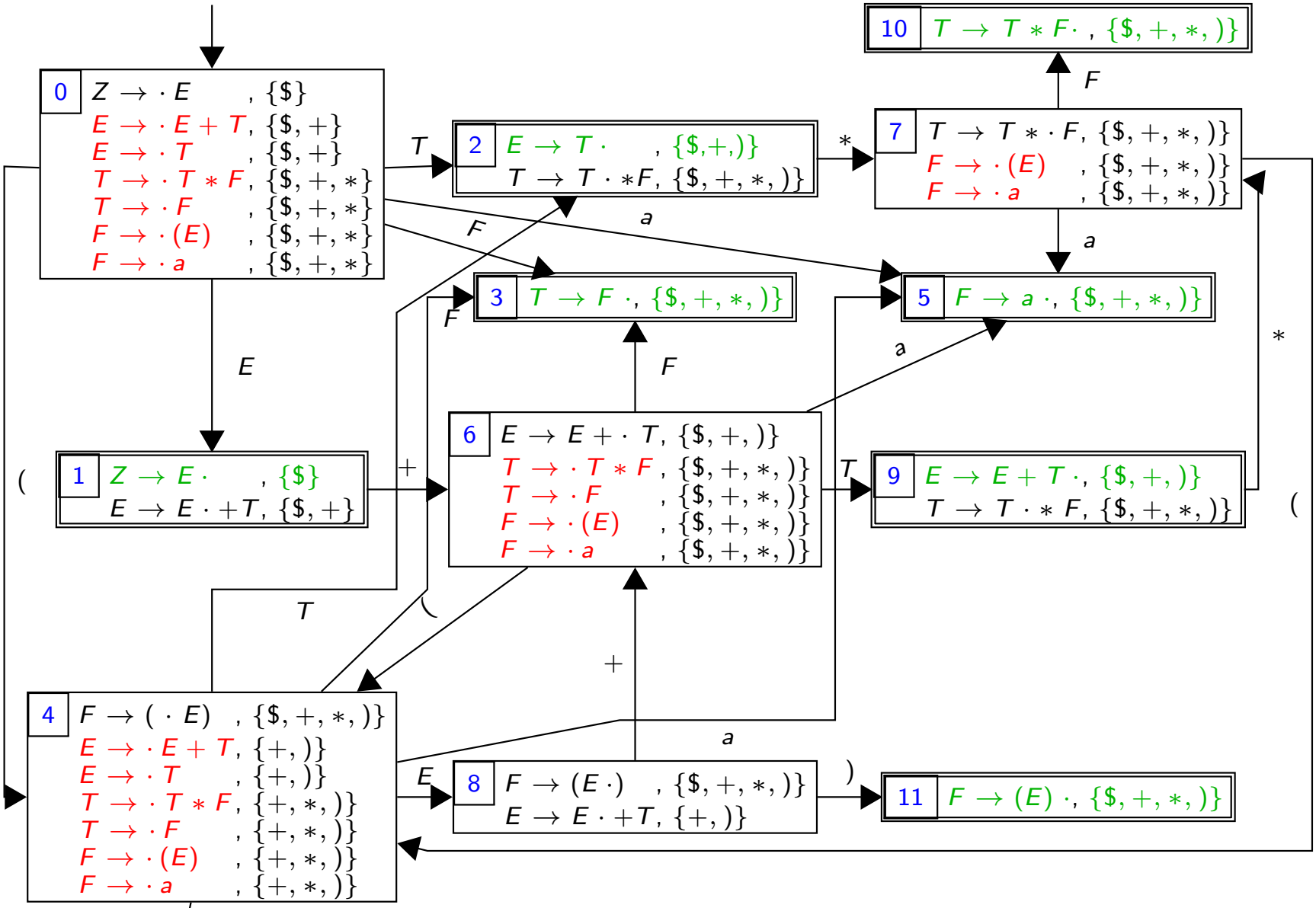

0 Z → · E , {$}
1 Z → E · , {$}
9 E → E + T · , {$, +}
18 E → · E+T , {$, +}
12 E → E · +T , {$, +}
6 E → E + · T , {$, +}
19 E → · T , {$, +}
2 E → T · , {$, +}
10 T → T ∗ F· , {$, +, ∗}
14 T → · T∗F , {$, +, ∗}
13 T → T · ∗F , {$, +, ∗}
7 T → T ∗ · F , {$, +, ∗}
15 T → · F , {$, +, ∗}
3 T → F · , {$, +, ∗}
11 F → (E) · , {$, +, ∗}
16 F → · (E) , {$, +, ∗}
4 F → (· E) , {$, +, ∗}
8 F → (E ·) , {$, +, ∗}
17 F → · a , {$, +, ∗}
5 F → a · , {$, +, ∗}
29 E → E + T · , {+,)}
38 E → · E+T , {+,)}
32 E → E · +T , {+,)}
26 E → E + · T , {+,)}
39 F → · T , {+,)}
22 F → T · , {+,)}
30 T → T ∗ F· , {+, ∗,)}
E
T
F
a
(
)
+
∗
ε

Beispiel 4.2:
deterministischer **LR(1)-Zustandsübergangsgraph** für die Grammatik G_4 aus Beispiel 4.1:

deterministischer LR(1)-Übergangsgraph (Ausschnitt)



deterministischer LALR(1)-Übergangsgraph



LR(1)-Syntaxanalyse-Tabelle

Zust.	Aktionstabelle						Sprungtab.		
	a	$+$	$*$	$($	$)$	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7			r2			
3		r4	r4			r4			
4	s15			s14			8	12	13
5		r6	r6			r6			
6	s5			s4				9	3
7	s5			s4					10
8		s16			s11				
9		r1	s7			r1			
10		r3	r3			r3			
11		r5	r5			r5			
12		r2	s17		r2				
13		r4	r4		r4				
14	s15			s14			18	12	13
15		r6	r6		r6				
16	s15			s14				19	13
17	s15			s14					20
18		s16			s21				
19		r1	s17		r1				
20		r3	r3		r3				
21		r5	r5		r5				

LALR(1)-Syntaxanalyse-Tabelle

Zust.	Aktionstabelle						Sprungtabelle		
	a	$+$	$*$	$($	$)$	\$	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			

Für den LALR(1)-Parser werden die Zustände i und $i + 10$ des LR(1)-Parsers für $i = 2...11$ verschmolzen.

LALR(1)- und SLR(1)-Syntaxanalyse-Tabelle

Auch wenn für die Grammatik G_4 die SLR(1)-Tabelle und die LALR(1)-Tabelle gleich sind:

Beide Tabellen sind ganz unterschiedlich entstanden:

- Die SLR(1)-Tabelle aus der LR(0)-Tabelle durch Weglassen von Reduce-Einträgen und
- die LALR(1)-Tabelle aus der LR(1)-Tabelle durch Verschmelzen von Zuständen!

Deterministische Parser

Der Stadiumautomat K_G einer kontextfreien Grammatik G akzeptiert genau deren Sprache $L(G)$, arbeitet aber leider nicht deterministisch.

Die Quelle für den Nichtdeterminismus liegt in den Expansionsübergängen: K_G muss raten, welche Produktion er für das aktuelle Nichtterminal –das Nichtterminal hinter dem Punkt– auswählen soll.

Bei einer eindeutigen Grammatik ist höchstens eine der Alternativen richtig, um die restliche Eingabe abzuleiten.

Deterministische Parser

Wir haben zwei Möglichkeiten, diesen Nichtdeterminismus zu beseitigen:

- Ein LL-Parser wählt deterministisch eine Produktion für das aktuelle Nichtterminal aus und benutzt dazu eine beschränkte Vorschau auf die restliche Eingabe.
  Für LL(k)-Grammatiken kann *genau ein Expansionsübergang* ausgewählt werden, wenn man das zu expandierende Nichtterminal und die nächsten k Eingabesymbole betrachtet.
  LL-Parser sind Linksparser.
- Ein LR-Parser verfolgt parallel alle Möglichkeiten, die zu einer (Rechts-)Ableitung für das Eingabewort führen können.
  Erst wenn ein *Reduktionsübergang* möglich ist, entscheidet der Parser, ob weiter gelesen oder reduziert werden soll, und genauer, mit welcher Produktion reduziert werden soll.
  Für diese Entscheidung werden der aktuelle Kellerinhalt und k Symbole der Vorschau herangezogen.
  LR-Parser gibt es nur für LR(k)-Grammatiken.
  LR-Parser sind Rechtsparser.

Deterministische Parser

Nur die LL- und die LR-Parser haben folgende Eigenschaften:

- Der Parser liest das Eingabewort *einmal* von **l**inks nach rechts und baut dabei die Ableitung auf (als **L**inks- bzw. als **R**echtsableitung).
- Der Parser erkennt einen Fehler beim *ersten* Zeichen a, das nicht zu einem Wort der Sprache gehören kann:
  Ist $xay \notin L(G)$ und der Parser erkennt den Fehler beim Zeichen a, so gibt es ein Wort z mit $xz \in L(G)$.

Wiederholung: LR(1)-Syntaxanalyse

Dazu erweitern wir jedes Stadium um die im jeweiligen Zustand erwarteten Vorschau-Symbole (**LookAhead**-Menge), also der Menge der Terminalsymbole, die dem Nichtterminalsymbol auf der linken Seite der Produktion *in diesem Zustand* folgen können.

- Dem initialen Stadium $[Z \to \cdot\, S]$ der Startproduktion wird die **LookAhead**-Menge $\{\$\}$ zugeordnet.
- Kommt man von einem erweiterten Stadium $[A \to \alpha \cdot B\beta,\ L]$ (α bzw. β kann auch ε sein) zu einem Stadium $[B \to \cdot\, \gamma]$, so berechnet sich dessen **LookAhead**-Menge zu $\mathrm{FIRST}(\beta L)$ mit $\beta L = \{\beta x \mid x \in L\}$ und $\mathrm{FIRST}(L) = \bigcup_{\alpha \in L} \mathrm{FIRST}(\alpha)$.

Zur Berechnung der erwarteten Vorschausymbole sind nur die *Zukunft* des Stadiums und die bisherige erwartete Vorschaumenge nötig!

Optimierte LR(1)-Parser

Wir berechnen einen **vergangenheitsreduzierten Automaten**, indem wir nur die Zukunft des Stadiums beibehalten.

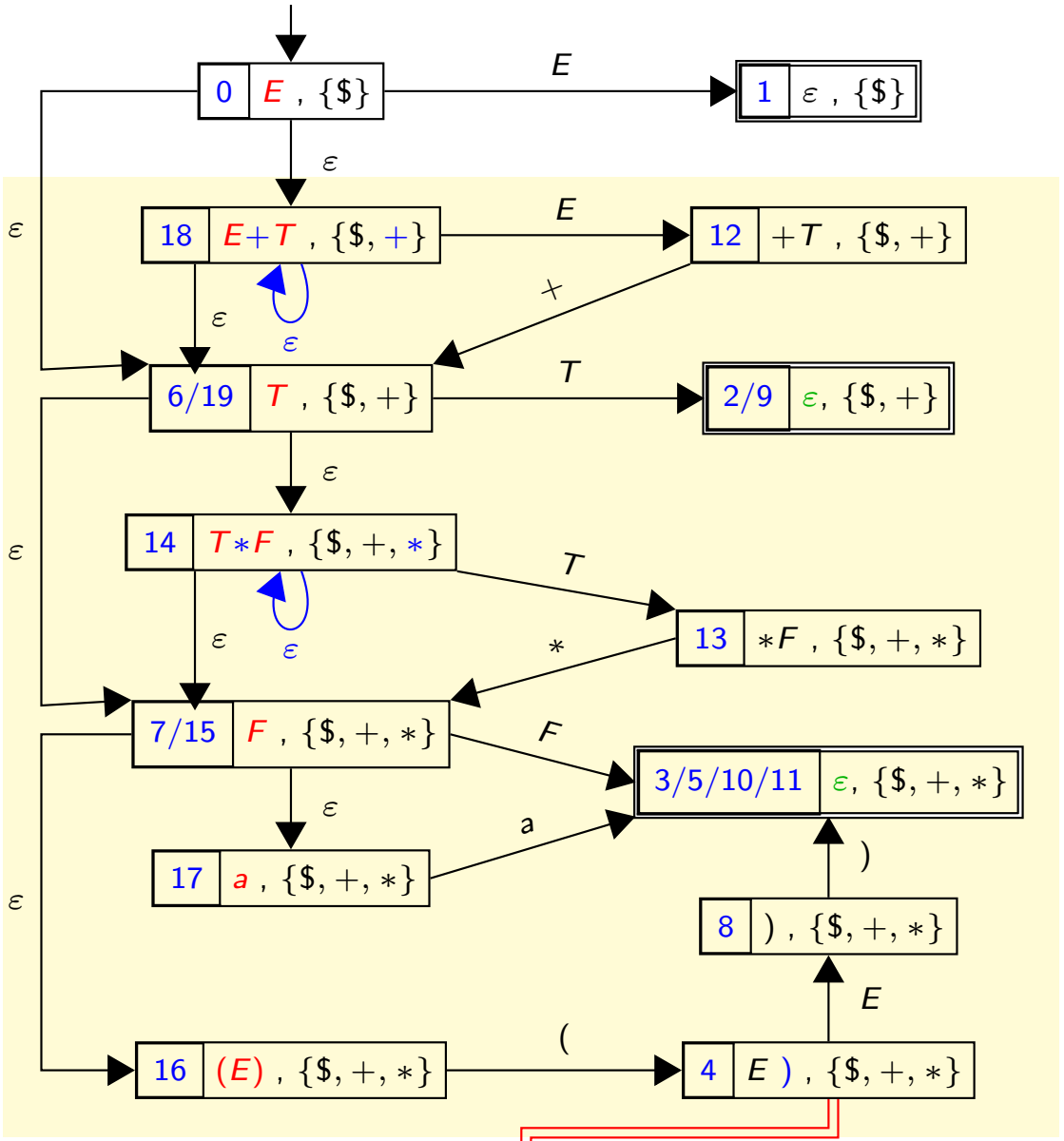
Das **Zukunftsstadium** eines erweiterten Stadiums $[A \to \alpha \cdot B\beta, u]$ ist $[\beta, u]$.

Finale Zukunftsstadien haben als erste Komponente stets das leere Wort ε.

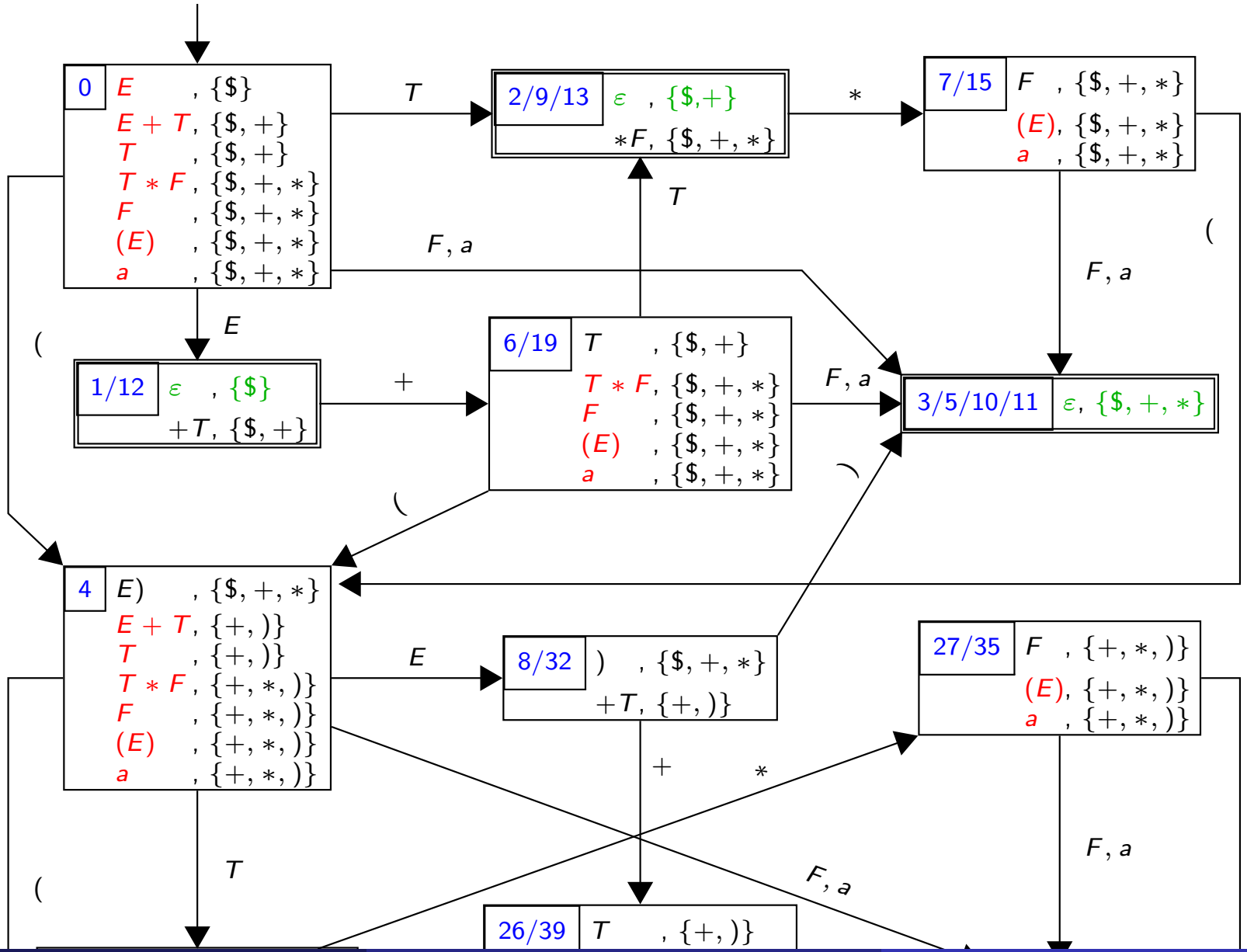
Daher können wir am obersten Zustand immer noch erkennen, ob reduziert werden kann, benötigen aber für die Entscheidung, *mit welcher Produktion* zu reduzieren ist, evtl. mehrere oberste Zustände.

Wir erhalten damit Automaten mit i.Allg. deutlich weniger Zuständen.

vergangenheitsreduzierter nicht-det. Automat



vergangenheitsreduzierter deterministischer Automat



Optimierte LR(1)-Parser

vergangenheitsreduzierte LR(1)-Syntaxanalyse-Tabelle für G_4

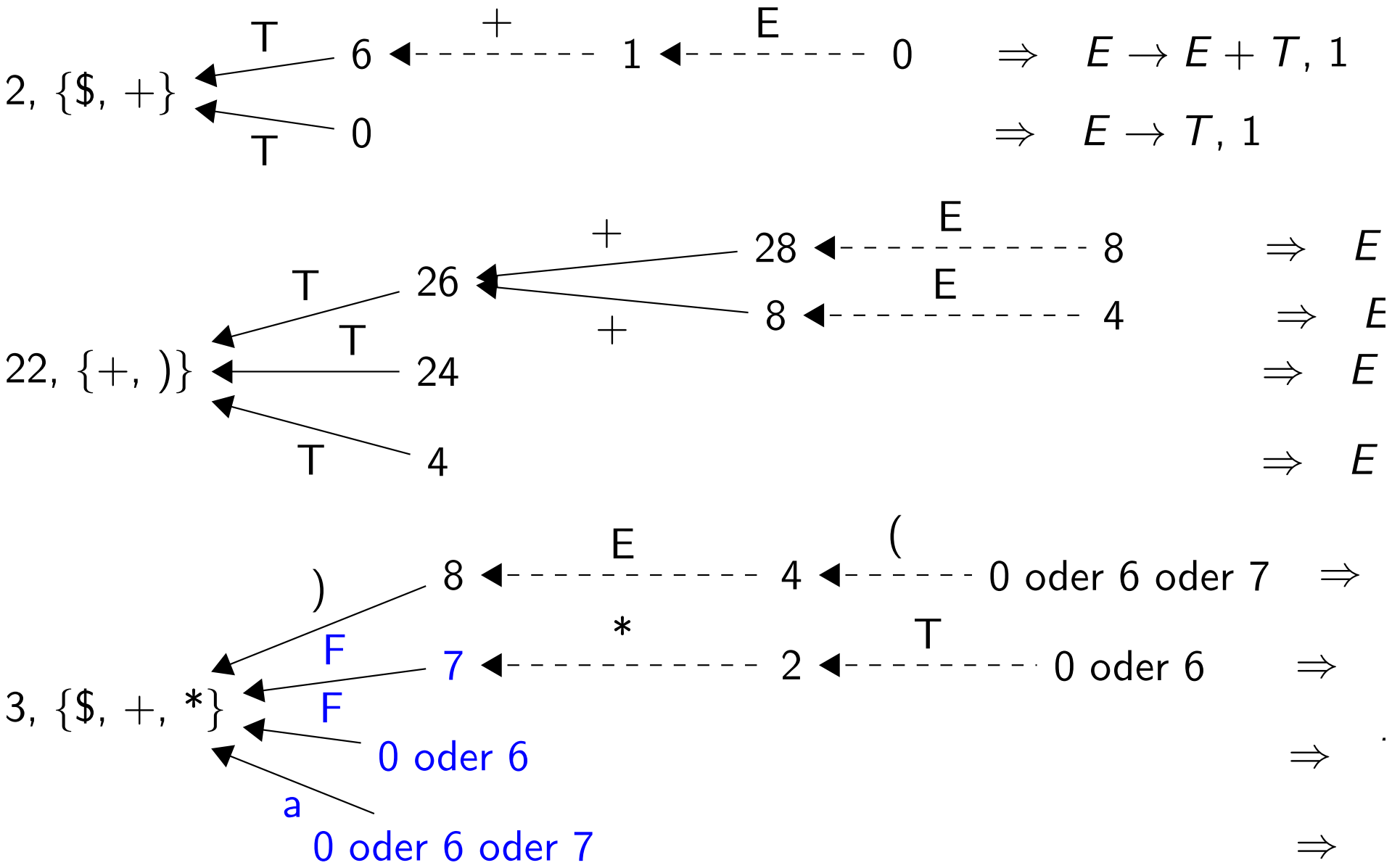
Zust.	Aktionstabelle					
	a	$+$	$*$	$($	$)$	\$
0	s3			s4		
1		s6				acc
2		r	s7			r
3		r	r			r
4	s23			s24		
6	s3			s4		
7	s3			s4		
8		s26			s3	
22		r	s27		r	
23		r	r		r	
24	s23			s24		
26	s23			s24		
27	s23			s24		
28		s26			s23	

Entscheidungsbäume

Entscheidungsbäume zeigen dann an, mit welcher Produktion reduziert wird: Ausgehend vom letzten Zustand (oberstes Kellersymbol) und der Vorschau-Menge werden evtl. die darunterliegenden Kellersymbole (und zugehörige Eingabesymbole) überprüft, bis eine Entscheidung für eine Produktion und den Folgezustand getroffen wird.

Diese Entscheidungsbäume entstehen dadurch, dass man alle Wege des Automaten von den initialen zu den finalen (Zukunfts-)Stadien verfolgt und anschließend ausgehend von den finalen Stadien gruppiert:

Entscheidungsbäume



Optimierte LALR(1)-Parser

vergangenheitsreduzierte LALR(1)-Syntaxanalyse-Tabelle für G_4

Zust.	Aktionstabelle					
	a	$+$	$*$	$($	$)$	\$
0	s3			s4		
1		s6				acc
2		r	s7		r	r
3		r	r		r	r
4	s3			s4		
6	s3			s4		
7	s3			s4		
8		s6			s3	

Optimierte LALR(1)-Parser