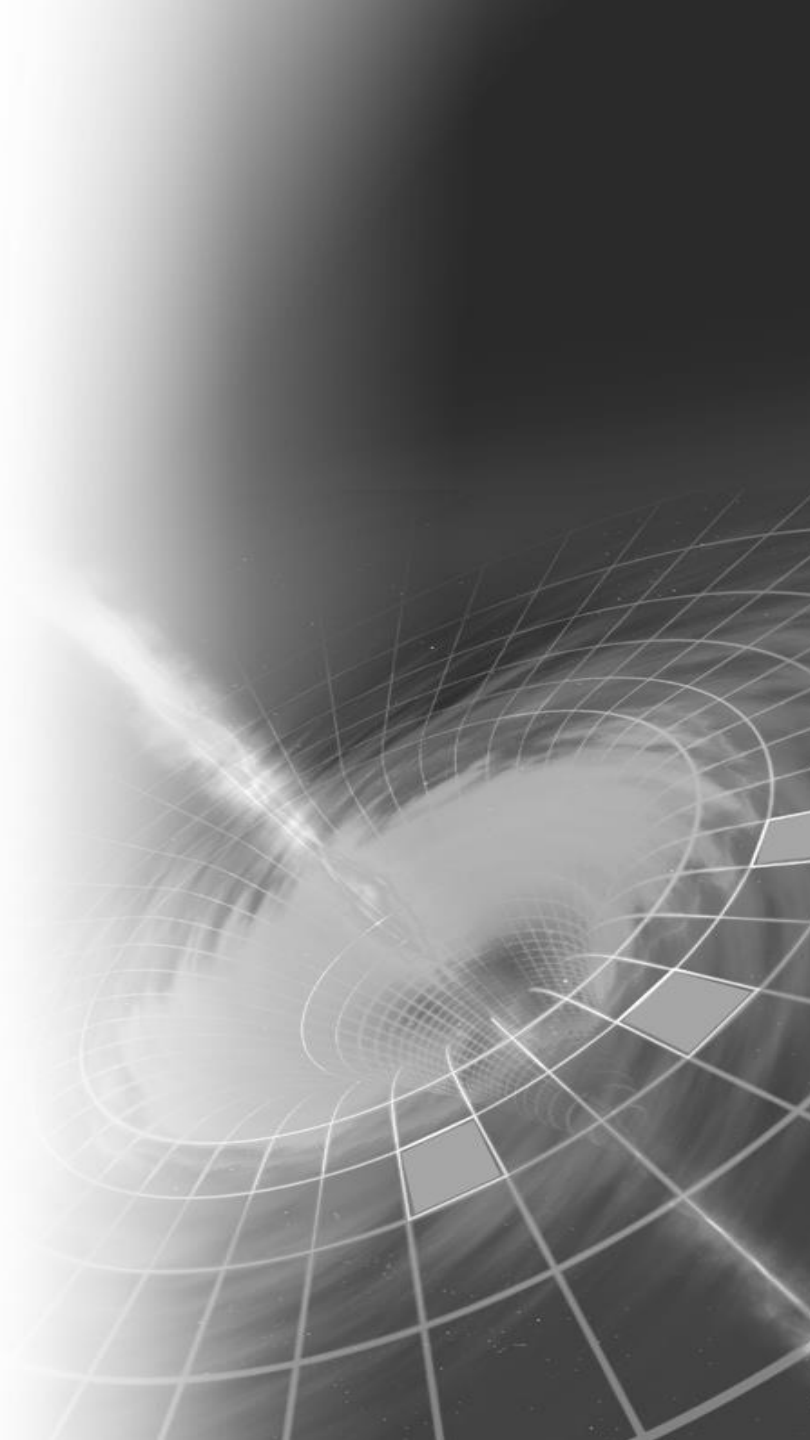
Bachelor-Programm

Compilerbau

im SoSe 2014

Prof. Dr. Joachim Fischer
Dr. Klaus Ahrens
Dipl.-Inf. Ingmar Eveslage

fischer@informatik.hu-berlin.de



Position

- Teil I
  Die Programmie
- Teil II
  Methodische Gru
- Teil III
  Entwicklung ein

- Kapitel 1
  Compilationsprozess
- Kapitel 2
  Formalismen zur Sprachbeschreibung
- Kapitel 3
  Lexikalische Analyse: der Scanner
- Kapitel 4
  Syntaktische Analyse: der Parser
- Kapitel 5
  Parsergeneratoren: Yacc, Bison
- Kapitel 6
  Statische Semantikanalyse
- Kapitel 7
  Laufzeitsysteme
- Kapitel 8
  Ausblick: Codegenerierung

- 4.1
  Einführung in die Syntaxanalyse
- 4.2
  Restrukturierung von Grammatiken
- 4.3
  LL-Parser
- 4.4
  Beispiel: Ein-Pass-Compiler (Parser, Übersetzer)
- 4.5
  Tabellengesteuerter LL-Parser
- 4.6
  Tabellengesteuerter LR-Parser

4.6.1 Allgemeine Betrachtung

- Allgemeines Prinzip von Shift-Reduce-Verfahren
- Klassifikation von LR-Analysemethoden/Grammatiken
- Präzisiertes Automatenmodell eines beliebigen LR-Parsers
- Arbeitsweise an einem Beispiel
- Konstruktionsvarianten von LR-Syntaxanalysetabellen im Überblick

4.6.2 LR(0)-Syntaxanalyse-Verfahren

- LR(0)-Elemente und Idee zur Zustandsbildung
- Die Operatoren Closure0 und Goto0
- Kanonische LR(0)-Kollektion, charakteristischer Automat und Übergangstabellenkonstruktion für einen LR(0)-Parser
- Beispiel: Konstruktion eines LR(0)-Parsers
- LR(0)-Konfliktbeispiel

Beispiel: (Wdh.)

- ad (1): Hinzufügen einer neuen **Startproduktion** zur Ausgangsgrammatik

Grammatik G

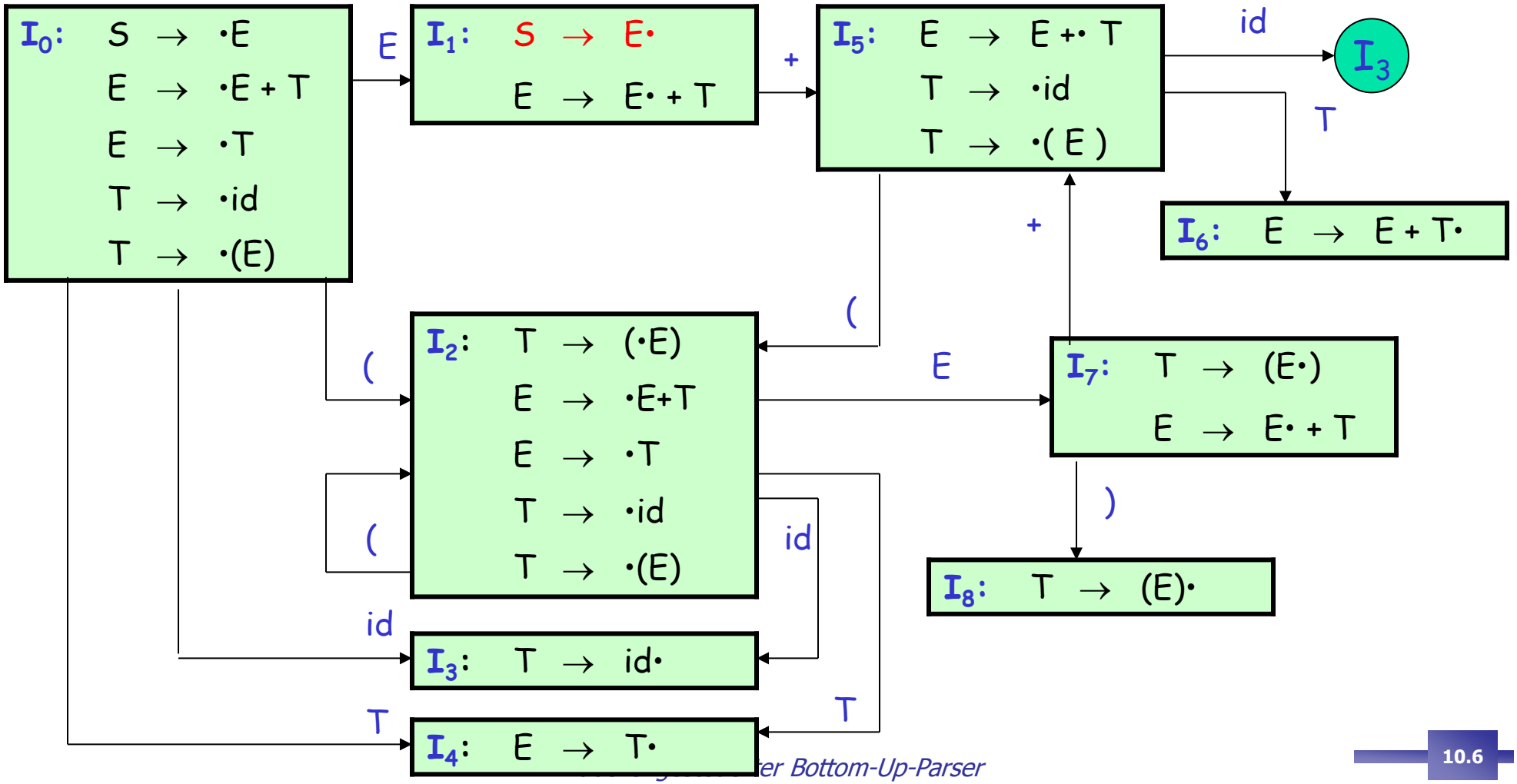
1	E	→	E **+** T
2		|	T
3	T	→	**id**
4		|	**(** E **)**

Grammatik G'

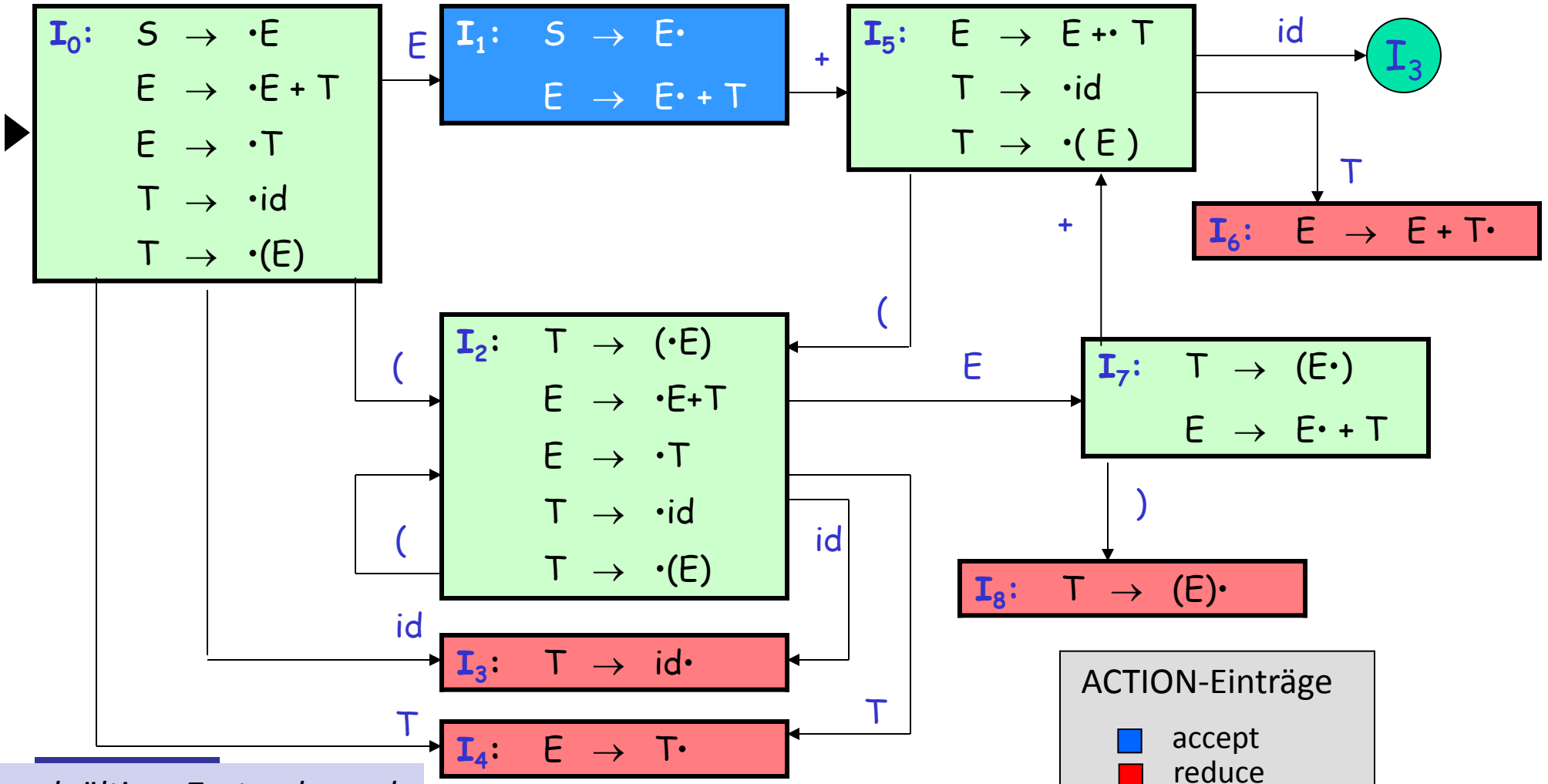
1	S	→	E
2	E	→	E **+** T
3		|	T
4	T	→	**id**
5		|	**(** E **)**

Beispiel: (Wdh.)

ad (2): Konstruktion des **Zustandsübergangsgraphen** des CFSM für Grammatik G‘



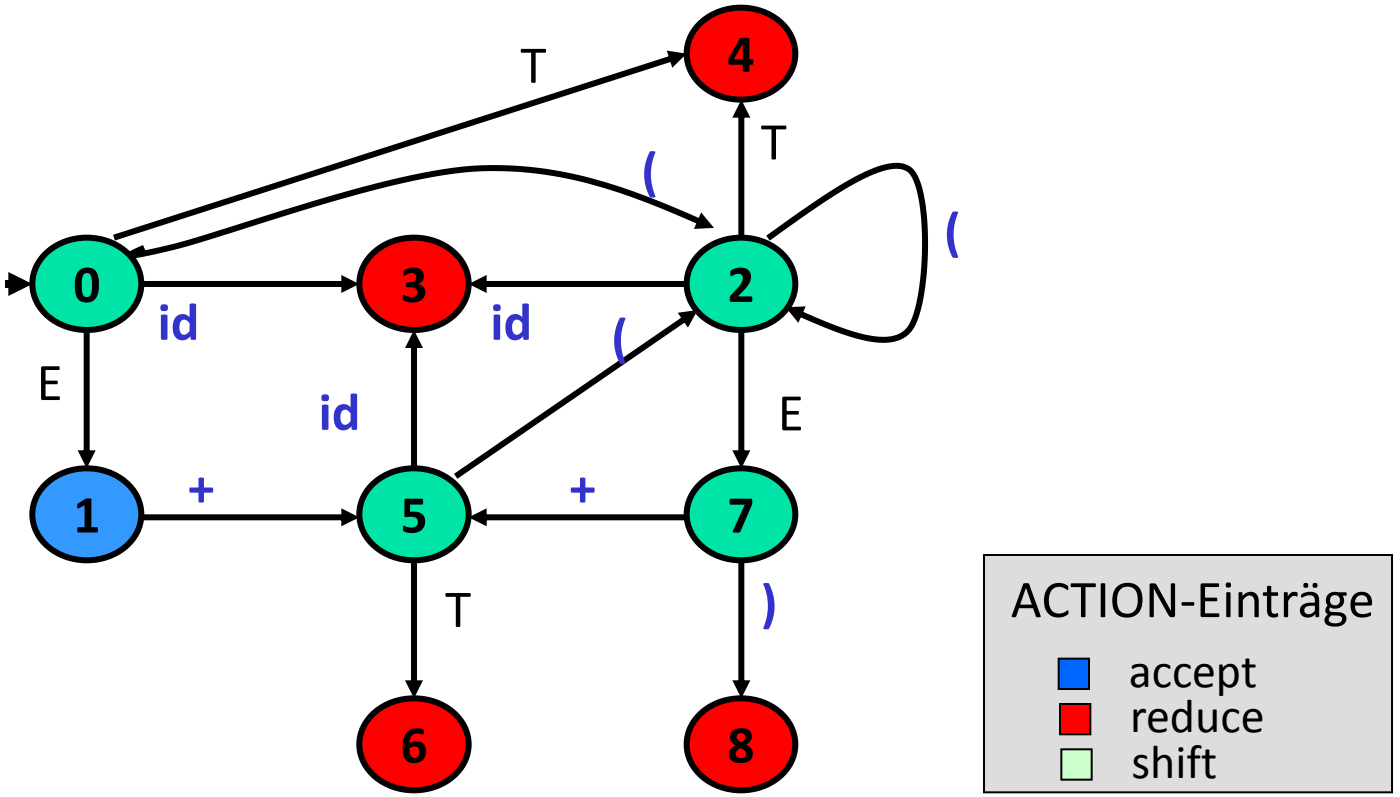
Beispiel: (Wdh.)



endgültiger Zustandsgraph

Beispiel: (Wdh.)

kompakter Zustandsgraph des charakteristischen Automaten (als DFA)



Allgemeiner Tabellenkonstruktionsablauf

unabhängig von konkreter LR-Technik:

(1) Hinzufügen einer neuen **Startproduktion** (S‘ → S) zu G: neue Grammatik G‘

(2) Konstruktion des **Zustandsübergangsgraphen** des CFSM für Grammatik G‘

Zustände entstehen durch Mengenbildung von LR(k)-Elementen

demonstriert bislang für LR(0)_Elemente

LR(0)-Element einer Grammatik G ist eine Produktion von G mit einer Bearbeitungs-Markierung (dargestellt als Punkt) in der rechten Seite: z.B. A → X•YZ

Einsatz: verfahrensspezifisch

(3) Ableitung der **Syntaxanalysetabelle** aus dem Zustandsübergangsgraphen des DFA

noch offen: sowohl allgemein als auch am Beispiel

Vorgehen bei der Konstruktion LR(0)-Parsers

- **ad (3):** Ableitung der **Syntaxanalysetabelle** aus dem konstruierten CFSM-Automaten

- Konstruktion der **Mengenkollektion** M= $\{I_0, I_1, ..., I_n\}$ von LR(0)-Elementen für G´
  bei Identifikation der **Zustände i** des CFSM aus I_i.

- Bestimmung der **Aktionen** für einen **Zustand I_i:**
  a) $[A \rightarrow \alpha \bullet a \beta] \in I_i$ und $goto_0(I_i, a) = I_k$, dann **ACTION**[i, a]:= "**shift** k"
  a muss dabei ein **Terminalsymbol** sein

  b) $[A \rightarrow \alpha \bullet] \in I_i$ und $A \neq S'$, dann **ACTION** [i, a]:= "**reduce** $A \rightarrow \alpha$"
  für **alle Terminalsymbole a** von G´

  c) $[S' \rightarrow S \bullet] \in I_i$, dann **ACTION** [i, **$**]:= "**accept**"

ACHTUNG:
Spezifik des LR(0)-Verfahrens

- wenn $goto_0(I_i, A) = I_k$ dann **GOTO** [i, A]:= k für **alle Nichtterminale** A von G´

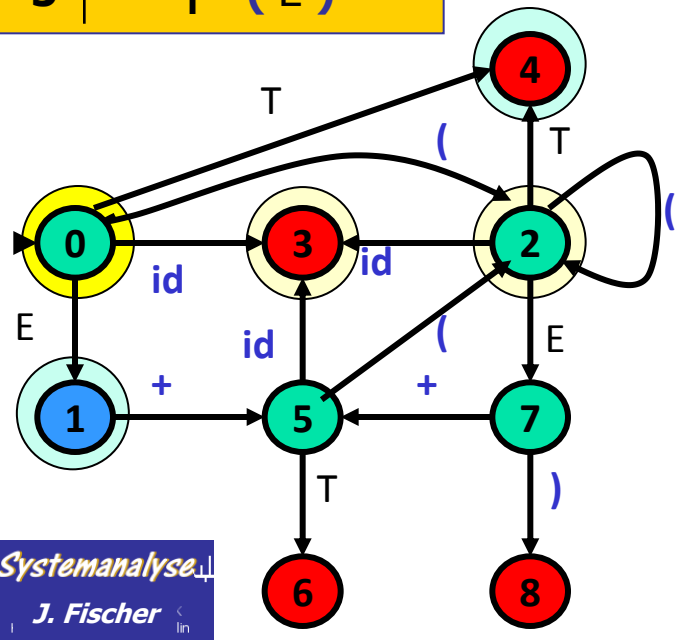
- setze undefinierte Einträge in **ACTION** und **GOTO** auf "error"

- **Anfangszustand** (Zustand 0) des Parsers (Kellerautomat) ist $closure_0([S' \rightarrow \bullet S]$

Beispiel: Konstruktion der LR(0)-Syntaxtabelle (1)

Grammatik

1	S	→	E
2	E	→	E **+** T
3		|	T
4	T	→	**id**
5		|	**(** E **)**

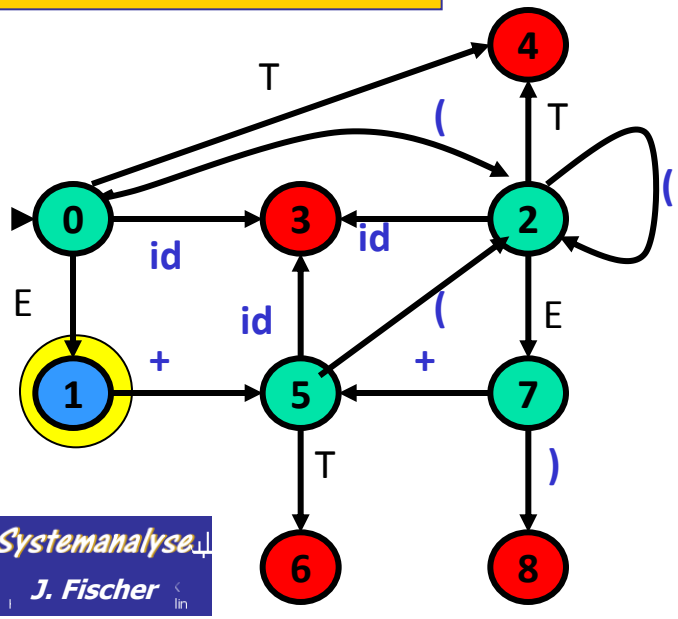


Zust	ACTION					GOTO		
	id	(	)	+	$	S	E	T
0	s3	s2	-	-	-	-	1	4
1	-	-	-	-	-	-	-	-
2	-	-	-	-	-	-	-	-
3	-	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-
5	-	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-	-
7	-	-	-	-	-	-	-	-
8	-	-	-	-	-	-	-	-

Beispiel: Konstruktion der LR(0)-Syntaxtabelle (2)

Grammatik

1	S	→	E
2	E	→	E **+** T
3		|	T
4	T	→	**id**
5		|	**(** E **)**

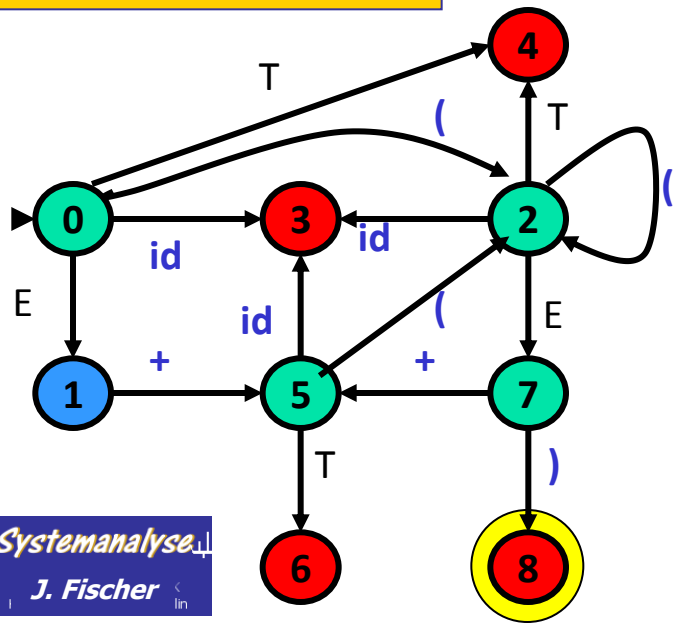


Zust	ACTION					GOTO		
	id	(	)	+	$	S	E	T
0	s3	s2	-	-	-	-	1	4
1	-	-	-	s5	acc	-	-	-
2	-	-	-	-	-	-	-	-
3	-	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-
5	-	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-	-
7	-	-	-	-	-	-	-	-
8	-	-	-	-	-	-	-	-

Beispiel: Konstruktion der LR(0)-Syntaxtabelle (9)

Grammatik

1	S	→	E
2	E	→	E **+** T
3		|	T
4	T	→	**id**
5		|	**(** E **)**



Zust	ACTION					GOTO		
	id	(	)	+	$	S	E	T
0	s3	s2	-	-	-	-	1	4
1	-	-	-	s5	acc	-	-	-
2	s3	s2	-	-	-	-	7	4
3	r4	r4	r4	r4	r4	-	-	-
4	r3	r3	r3	r3	r3	-	-	-
5	s3	s2	-	-	-	-	-	6
6	r2	r2	r2	r2	r2	-	-	-
7	-	-	s8	s5	-	-	-	-
8	r5	r5	r5	r5	r5	-	-	-

LR(0)-Grammatikeigenschaft und mögliche Konflikte in der Syntax-Tabelle

- Falls die LR(0)-Syntaxtabelle **ACTION**-Einträge hat, die **mehrfach** belegt sind, ist die Grammatik **nicht** vom Typ **LR(0)**
- Phänomen wird auch als Existenz **nicht adäquater Zustände** bezeichnet
- zwei Konfliktmöglichkeiten
  - **Shift-Reduce**: beide Operationen (shift und reduce) sind auf der selben Elementmenge möglich
  - **Reduce-Reduce**: mehr als eine Reduktionsmöglichkeit für die selbe Elementmenge
- Konflikte können u.U. durch LookAhead in der Syntax-Tabelle gelöst werden

also partielle Vorausschau

- Falls die LR(0)-Syntaxtabelle **nur einfache** Einträge in der Syntax-Tabelle hat, ist die Grammatik vom Typ **LR(0),** d.h., es wird kein LookAhead benötigt
- der so konstruierte Parser ist ein **LR(0)-Parser**
- eine Grammatik ist vom Typ **LR(0)**, wenn der charakteristische Automat für diese Grammatik **keine** nicht-adäquaten Zustände besitzt

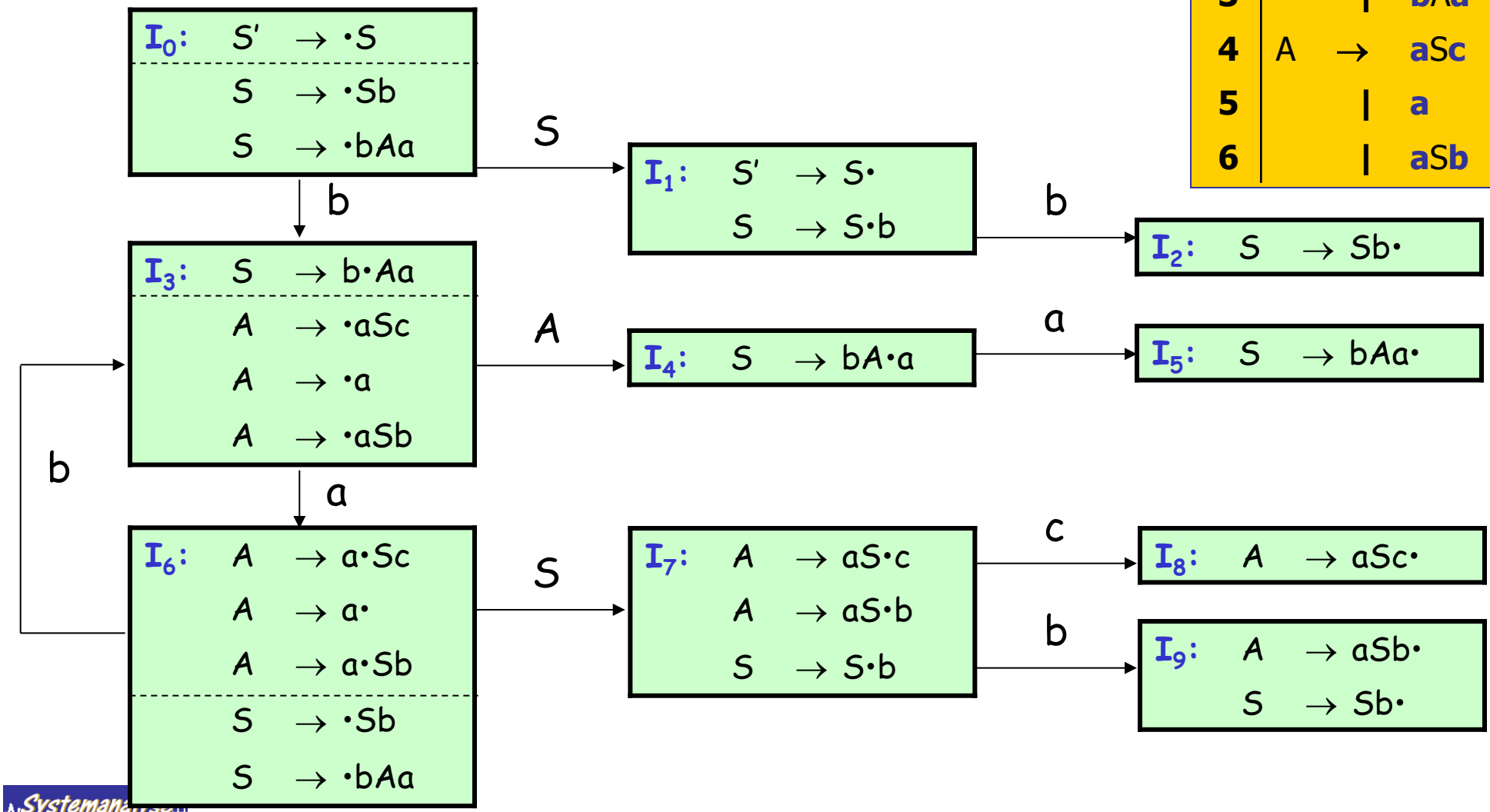
4.6.2 LR(0)- Syntaxanalyse

- LR(0)-Elemente und Idee zur Zustandsbildung
- Die Operatoren Closure0 und Goto0
- Kanonische LR(0)-Kollektion, charakteristischer Automat und Übergangstabellenkonstruktion für einen LR(0)-Parser
- Beispiel: Konstruktion eines LR(0)-Parsers
- LR(0)-Konfliktbeispiel

Konflikt-Beispiel: Kanonische LR(0)- Kollektion

Grammatik

1	S′	→	S
2	S	→	**S**b
3		|	**b**A**a**
4	A	→	**a**S**c**
5		|	**a**
6		|	**a**S**b**

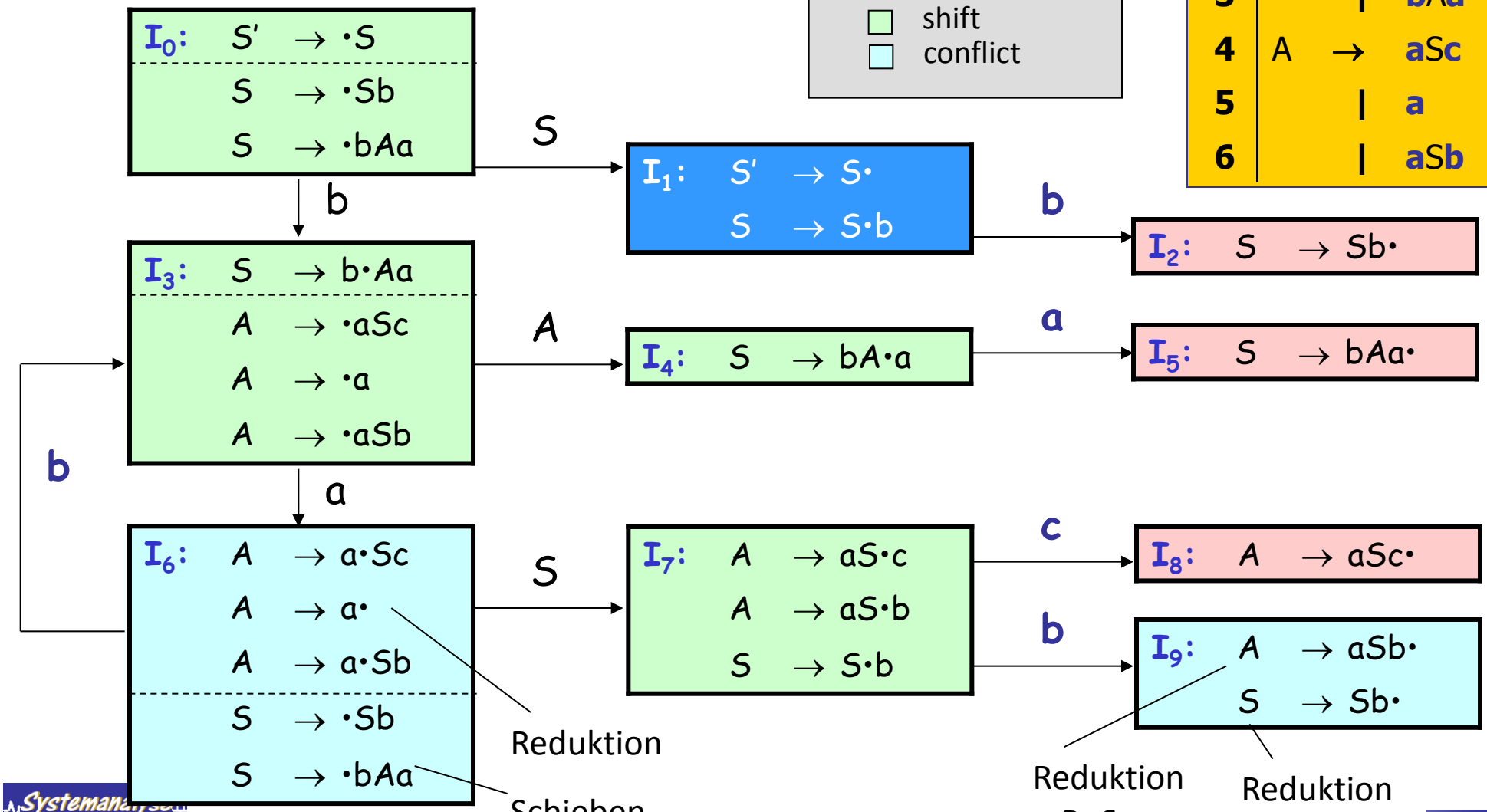


Konflikt-Beispiel: Nicht-adäquate

ACTION-Einträge
- accept
- reduce
- shift
- conflict

Grammatik

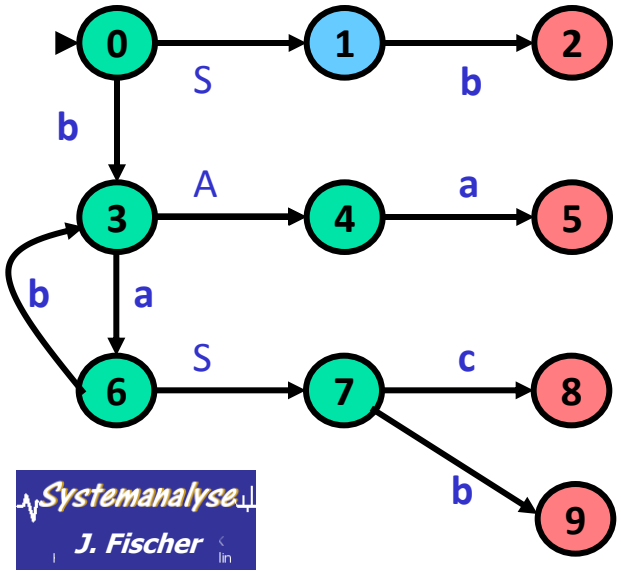
1	S'	→	S
2	S	→	Sb
3		|	bAa
4	A	→	aSc
5		|	a
6		|	aSb



Konflikt-Beispiel: Konstruktion der LR(0)-Syntaxtabelle

Grammatik

1	S'	→	S
2	S	→	Sb
3	S	|	bAa
4	A	→	aSc
5		|	a
6		|	aSb



Zust	ACTION				GOTO		
	a	b	c	$	A	S	S'
0	-	s3	-	-	-	1	-
1	-	s2	-	acc	-	-	-
2	-	r2	r2	r2	-	-	-
3	s6	-	-	-	4	-	-
4	s5	-	-	-	-	-	-
5	r3	r3	r3	r3	-	-	-
6	r5	s3/r5	r5	r5	-	7	-
7	-	s9	s8	-	-	-	-
8	r4	r4	r4	r4	-	-	-
9	r2/r6	r2/r6	r2/r6	r2/r6	-	-	-

Konflikte nicht behandelbar von LR(0)-Parsern

Unser Ausweg: **LookAhead** in kritischen Situationen einsetzen

4.6.3 SLR(1) - Syntaxanalyse

- Konstruktion von SLR(1)-Syntaxtabellen
- Beispiel: SLR(1)-Parser für LR(0)-Konfliktgrammatik
- weiteres Beispiel: SLR(1)-Konstruktion
- SLR(1)-Konfliktbeispiel

Zur Erinnerung (1): Allgemeiner Tabellenkonstruktionsablauf

unabhängig von konkreter LR-Technik:

(1) Hinzufügen einer neuen **Startproduktion** (S' → S) zu G: neue Grammatik G'

(2) Konstruktion des **Zustandsübergangsgraphen** des CFSM für Grammatik G'

Zustände entstehen durch Mengenbildung von LR(k)-Elementen

demonstriert bislang für LR(0)_Elemente

LR(0)-Element einer Grammatik G ist eine Produktion von G mit einer Bearbeitungs-Markierung (dargestellt als Punkt) in der rechten Seite: z.B. A → X•YZ

Einsatz: verfahrensspezifisch

(3) Ableitung der **Syntaxanalysetabelle** aus dem Zustandsübergangsgraphen des DFA

Zur Erinnerung (2): Zustandsabstraktion

konzeptionelle Basis

Zustände des jeweils gesuchten DPDA als LR(k)-Parser
entstehen durch geeignete Zusammenfassungen von **LR(k)-Elementen** als Paare $[\alpha, \beta]$

LR(1)-Parser
LALR(1)-Parser

LR(1)-Element $[\alpha, \beta]$

$[A \rightarrow X \bullet YZ]$

LookAhead-Zeichenkette

Symbol "•" zeigt Verarbeitungs-/Erkennungs-zustand der rechten Seite an: (X liegt bereits auf dem Stack)

$$LR(0) \subset SLR(1) \subset LALR(1) \subset LR(1) = LR(k)$$

LR(0)-Element $[\alpha]$

LR(0)-Parser
SLR(1)-Parser

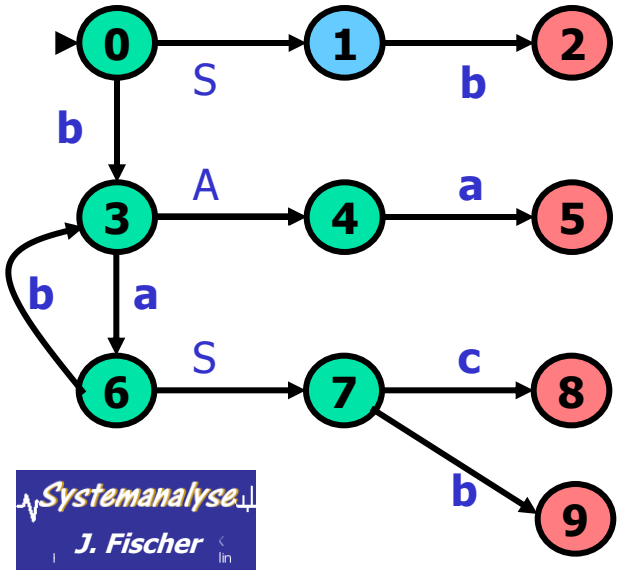
LR(k)-Element $[\alpha, \beta]$

LR(k)-Parser

Konflikt-Beispiel: Konstruktion der LR(0)-Syntaxtabelle

Grammatik

1	S′	→	S
2	S	→	Sb
3	S	|	bAa
4	A	→	aSc
5		|	a
6		|	aSb



Zust	ACTION				GOTO		
	a	b	c	$	A	S	S′
0	-	s3	-	-	-	1	-
1	-	s2	-	acc	-	-	-
2	-	r2	r2	r2	-	-	-
3	s6	-	-	-	4	-	-
4	s5	-	-	-	-	-	-
5	r3	r3	r3	r3	-	-	-
6	r5	s3/r5	r5	r5	-	7	-
7	-	s9	s8	-	-	-	-
8	r4	r4	r4	r4	-	-	-
9	r2/r6	r2/r6	r2/r6	r2/r6	-	-	-

LookAhead in kritischen Situationen:

Hier zunächst nur FOLLOW-Mengenbestimmung → SLR(1)-Parser

Konstruktion von SLR(1)-Syntaxtabellen

… als Modifikation des LR(0)-Verfahrens

- Konstruktion der **Mengenkollektion** M= {I_0, I_1, ..., I_n} von LR(0)-Elementen für G´
  bei Identifikation der **Zustände i** des CFSM aus I_i.

- Bestimmung der **Aktionen** für einen Zustand I_i:
  a) $[A \rightarrow \alpha \bullet \mathbf{a} \beta] \in I_i$ und $goto0(I_i, \mathbf{a}) = I_k$, dann **ACTION**[i, **a**]:= "**shift** k"
     a muss dabei ein **Terminalsymbol** sein

  b) $[A \rightarrow \alpha \bullet] \in I_i$ und $A \neq S'$, dann **ACTION** [i, **a**]:= "**reduce** $A \rightarrow \alpha$"
     für **alle Terminalsymbole a** von G´ mit **a** ∈ FOLLOW(A)

  c) $[S' \rightarrow S \bullet] \in I_i$, dann **ACTION** [i, **$**]:= "**accept**"

ACHTUNG:
Besonderheit des SLR(1)-Verfahrens

- wenn $goto0(I_i, A) = I_k$ dann **GOTO** [i,A]:= k für **alle Nichtterminale** A von G´

- setze undefinierte Einträge in **ACTION** und **GOTO** auf "error"

- **Anfangszustand** des Parsers (Zustand 0) ist $closure0([S' \rightarrow \bullet S]$

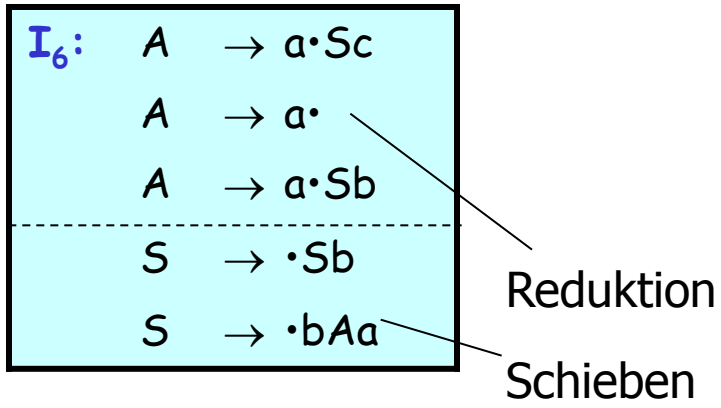
4.6.3 SLR(1) - Syntaxanalyse

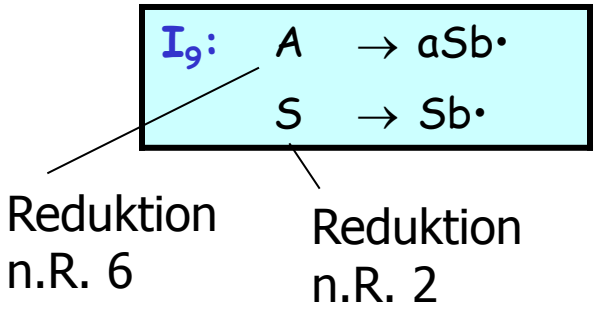
- Konstruktion von SLR(1)-Syntaxtabellen
- Beispiel: SLR(1)-Parser für LR(0)-Konfliktgrammatik
- weiteres Beispiel: SLR(1)-Konstruktion
- SLR(1)-Konfliktbeispiel

"altes" Konflikt-Beispiel des LR(0)-Parsers

Grammatik

1	S'	→	S
2	S	→	**S**b
3	S	|	**b**A**a**
4	A	→	**a**S**c**
5		|	**a**
6		|	**a**S**b**





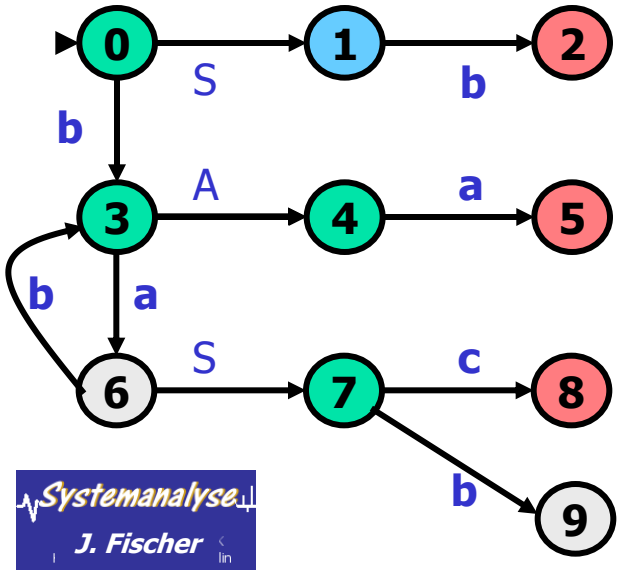
- Aufbau einer LR(0)-Syntaxtabelle scheiterte
- jetzt Aufbau einer SLR(1)-Syntaxtabelle

  (LR(0)-Kollektion wird dabei übernommen,
  d.h. auch der Zustandsübergangsgraph)

LR(0)-Konfliktgrammatik: *Konstruktion der SLR(1)-Syntaxtabelle (1)*

Grammatik

1	S′	→	S
2	S	→	Sb
3	S	|	bAa
4	A	→	aSc
5		|	a
6		|	aSb



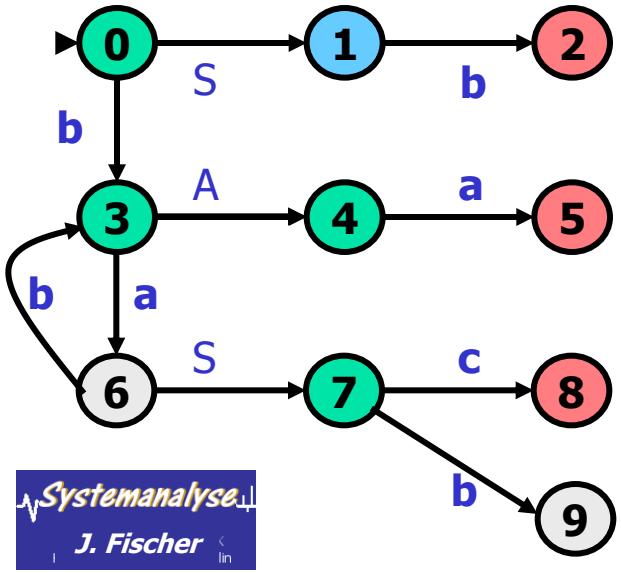
	FOLLOW
S′	{$}
S	{b, c, $ }
A	{a}

Zust	ACTION				GOTO		
	a	b	c	$	A	S	S′
0							
1							
2							
3							
4							
5							
6							
7							
8							
9							

"altes" Beispiel: Konstruktion der SLR(1)-Syntaxtabelle (2)

Grammatik

1	S'	→	S
2	S	→	Sb
3	S	|	bAa
4	A	→	aSc
5		|	a
6		|	aSb



	FOLLOW
S'	{$}
S	{b, c, $ }
A	{a}

Zust	ACTION				GOTO		
	a	b	c	$	A	S	S'
0	-	s3	-	-	-	1	-
1	-	s2	-	acc	-	-	-
2							
3							
4							
5							
6							
7							
8							
9							

"altes" Beispiel: Konstruktion der SLR(1)-Syntaxtabelle (3)

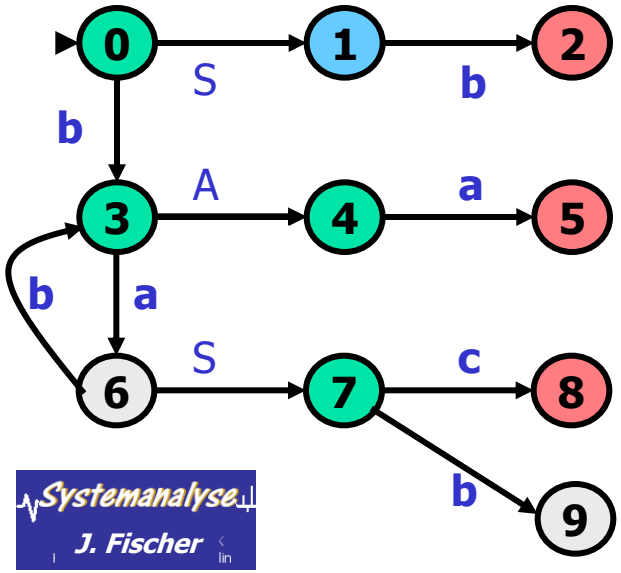
Grammatik

1	S'	→	S
2	S	→	Sb
3	S	|	bAa
4	A	→	aSc
5		|	a
6		|	aSb

I_2: S → Sb•

	FOLLOW
S'	{$}
S	{b, c, $ }
A	{a}

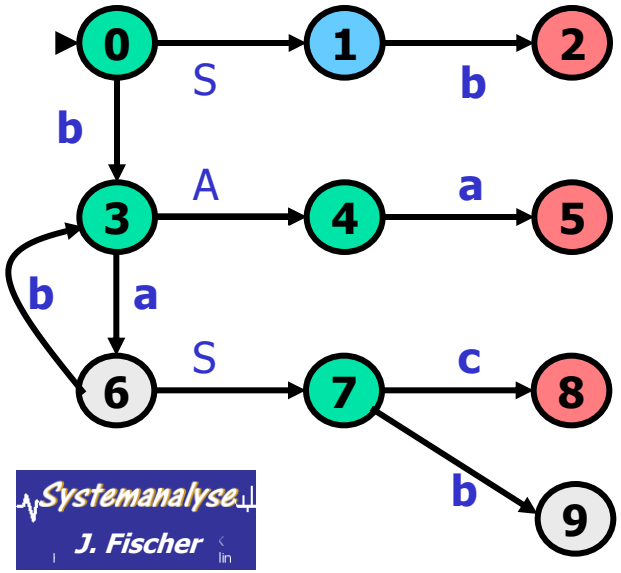
Zust	ACTION				GOTO		
	a	b	c	$	A	S	S'
0	-	s3	-	-	-	1	-
1	-	s2	-	acc	-	-	-
2	-	r2	r2	r2	-	-	-
3							
4							
5							
6							
7							
8							
9							



"altes" Beispiel: Konstruktion der SLR(1)-Syntaxtabelle (4)

Grammatik

1	S'	→	S
2	S	→	Sb
3	S	|	bAa
4	A	→	aSc
5		|	a
6		|	aSb



	FOLLOW
S'	{$}
S	{b, c, $ }
A	{a}

Zust	ACTION				GOTO		
	a	b	c	$	A	S	S'
0	-	s3	-	-	-	1	-
1	-	s2	-	acc	-	-	-
2	-	r2	r2	r2	-	-	-
3	s6	-	-	-	4	-	-
4	s5	-	-	-	-	-	-
5							
6							
7							
8							
9							

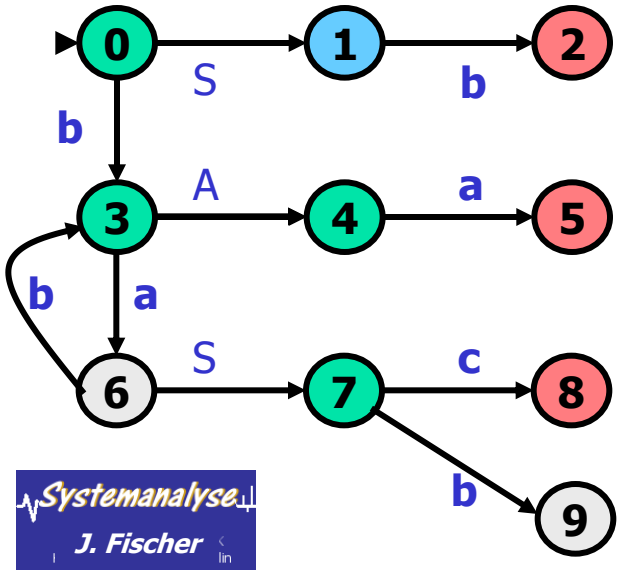
"altes" Beispiel: Konstruktion der SLR(1)-Syntaxtabelle (5)

Grammatik

1	S'	→	S
2	S	→	Sb
3	S	|	bAa
4	A	→	aSc
5		|	a
6		|	aSb

I_5: S → bAa•

	FOLLOW
S'	{$}
S	{b, c, $ }
A	{a}



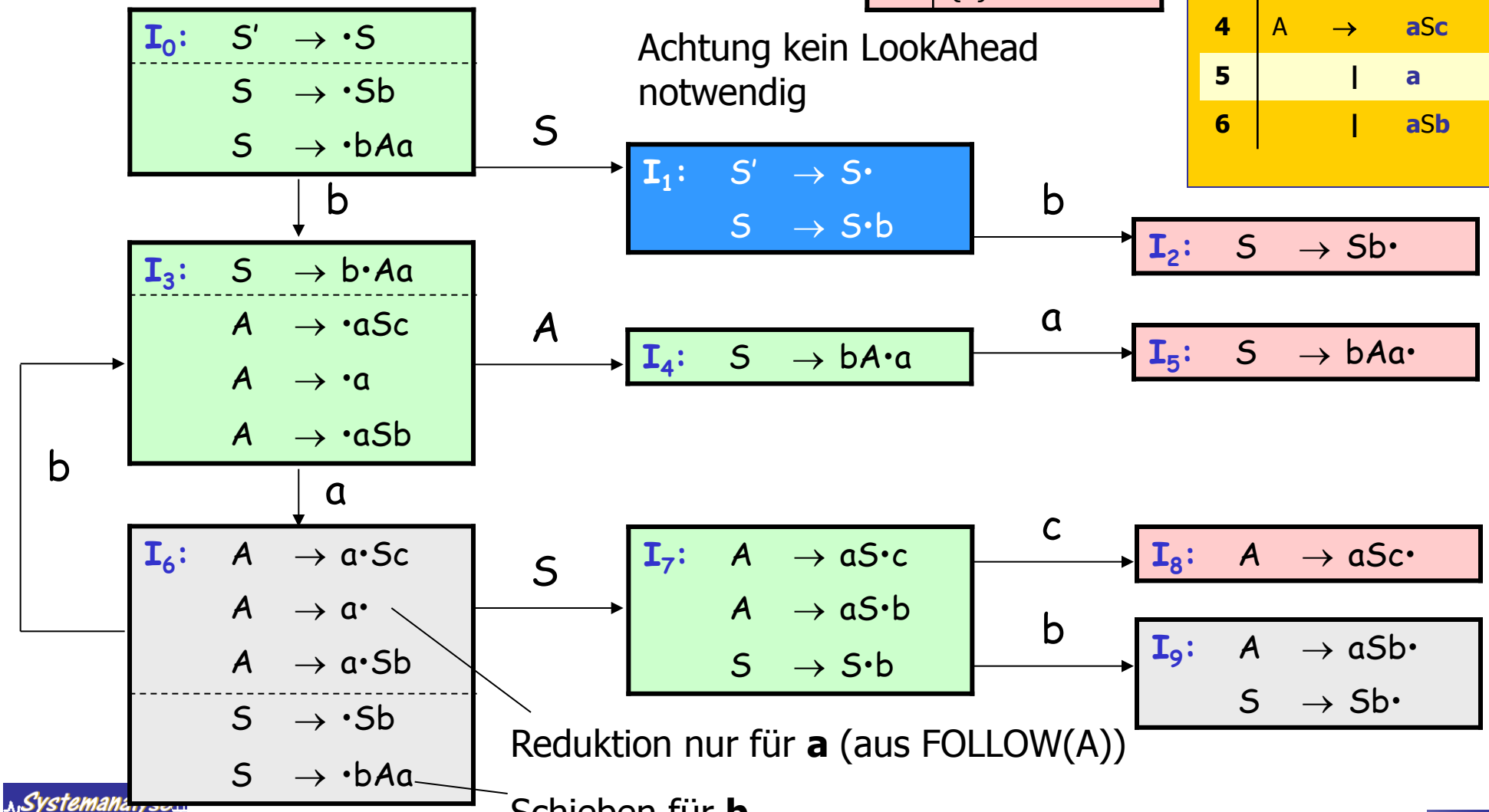
Zust	ACTION				GOTO		
	a	b	c	$	A	S	S'
0	-	s3	-	-	-	1	-
1	-	s2	-	acc	-	-	-
2	-	r2	r2	r2	-	-	-
3	s6	-	-	-	4	-	-
4	s5	-	-	-	-	-	-
5	-	r3	r3	r3	-	-	-
6							
7							
8							
9							

Konflikt-Beispiel: Nicht-adäquate Zu

	FOLLOW
S′	{$}
S	{b, c, $ }
A	{a}

Grammatik

1	S′	→	S
2	S	→	Sb
3		|	bAa
4	A	→	aSc
5		|	a
6		|	aSb



Achtung kein LookAhead notwendig

Reduktion nur für **a** (aus FOLLOW(A))

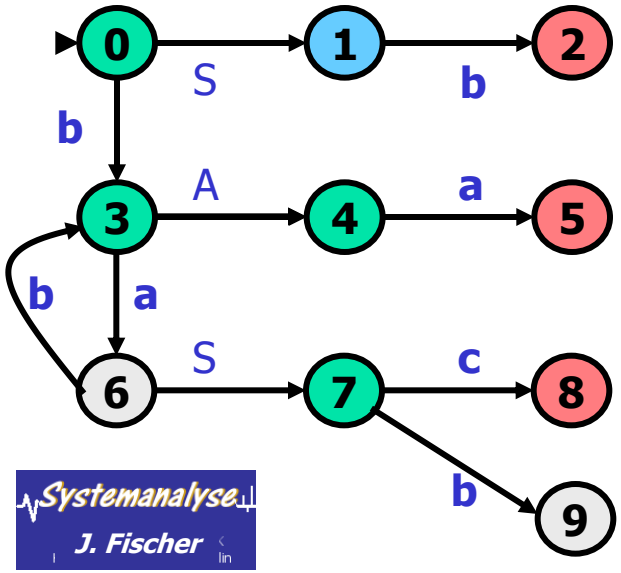
Schieben für **b**

"altes" Beispiel: Konstruktion der SLR(1)-Syntaxtabelle (6)

	FOLLOW
S'	{$}
S	{b, c, $ }
A	{a}

Grammatik

1	S'	→	S
2	S	→	Sb
3	S	|	bAa
4	A	→	aSc
5		|	a
6		|	aSb



Zust	ACTION				GOTO		
	a	b	c	$	A	S	S'
0	-	s3	-	-	-	1	-
1	-	s2	-	acc	-	-	-
2	-	r2	r2	r2	-	-	-
3	s6	-	-	-	4	-	-
4	s5	-	-	-	-	-	-
5	-	r3	r3	r3	-	-	-
6	r5	s3			-	7	-
7							
8							
9							

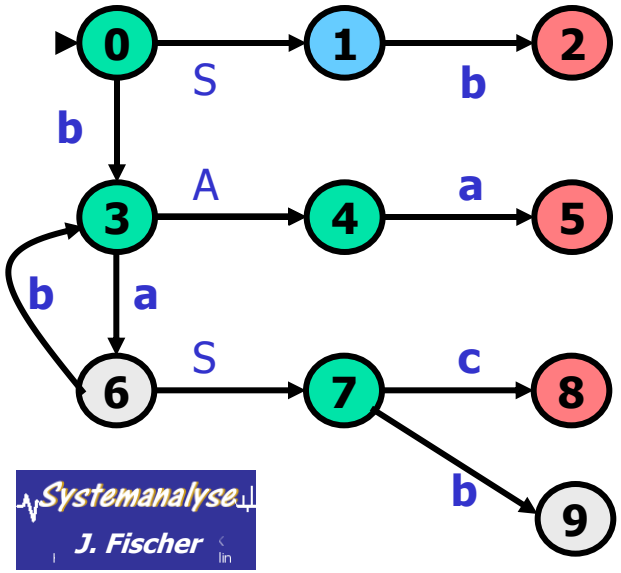
"altes" Beispiel: Konstruktion der SLR(1)-Syntaxtabelle (7)

Grammatik

1	S′	→	S
2	S	→	Sb
3	S	|	bAa
4	A	→	aSc
5		|	a
6		|	aSb

I_8: $A \rightarrow aSc\cdot$

	FOLLOW
S′	{$}
S	{b, c, $ }
A	{a}



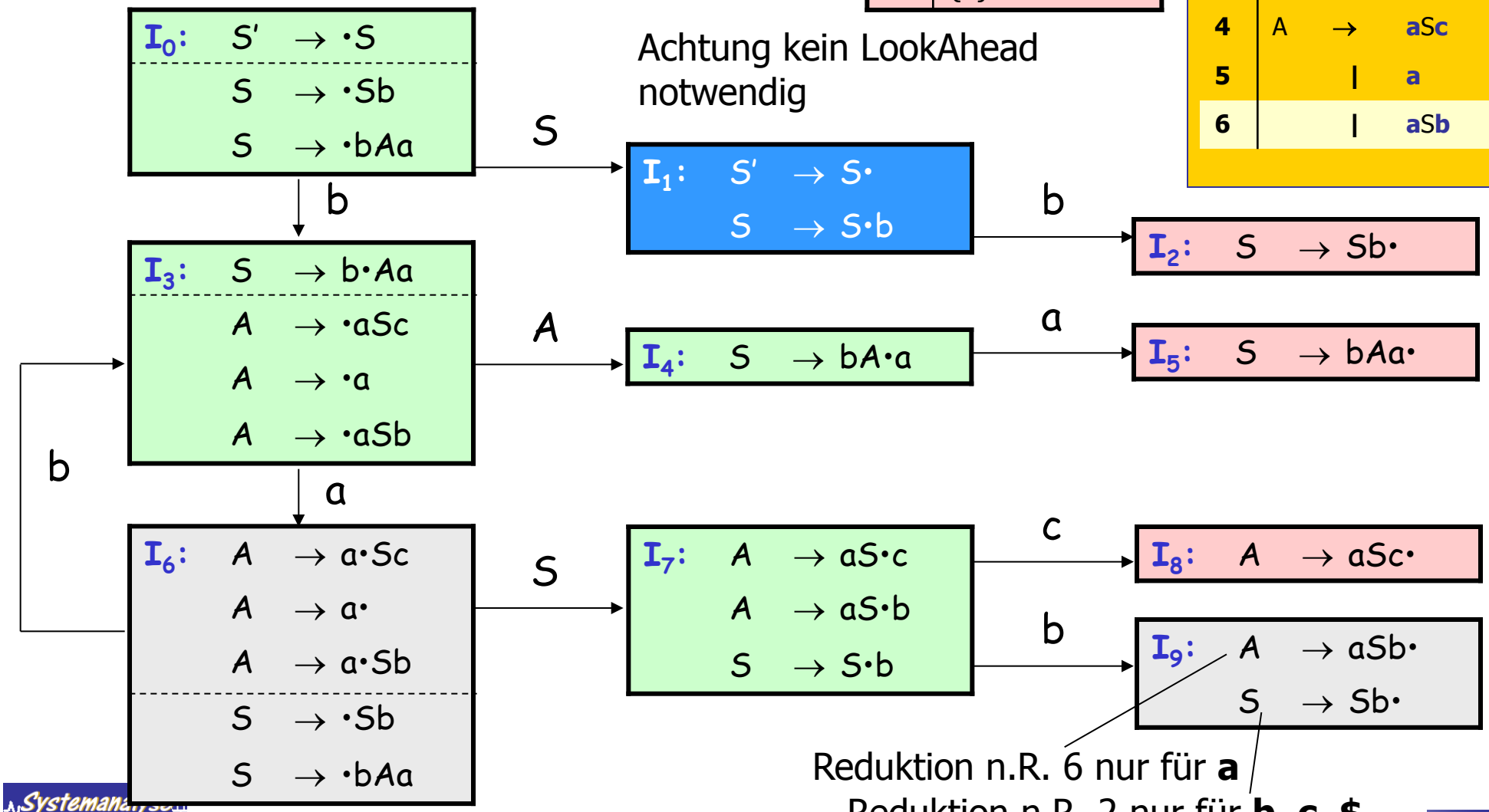
Zust	ACTION				GOTO		
	a	b	c	$	A	S	S′
0	-	s3	-	-	-	1	-
1	-	s2	-	acc	-	-	-
2	-	r2	r2	r2	-	-	-
3	s6	-	-	-	4	-	-
4	s5	-	-	-	-	-	-
5	-	r3	r3	r3	-	-	-
6	r5	s3			-	7	-
7	-	s9	s8	-	-	-	-
8	r4	-	-	-	-	-	-
9							

Konflikt-Beispiel: Nicht-adäquate Zu

	FOLLOW
S′	{$}
S	{b, c, $ }
A	{a}

Grammatik

1	S′	→	S
2	S	→	Sb
3		|	bAa
4	A	→	aSc
5		|	a
6		|	aSb



Achtung kein LookAhead notwendig

Reduktion n.R. 6 nur für **a**

Reduktion n.R. 2 nur für **b, c, $**

"altes" Beispiel: Konstruktion der SLR(1)-Syntaxtabelle (8)

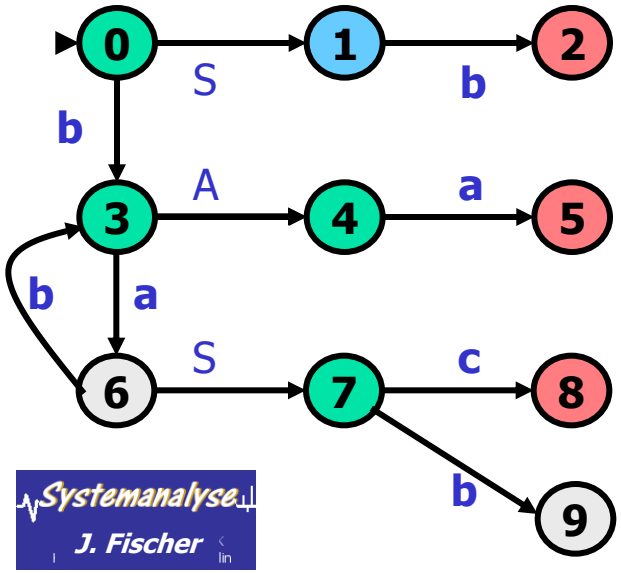
Grammatik

1	S′	→	S
2	S	→	Sb
3	S	|	bAa
4	A	→	aSc
5		|	a
6		|	aSb

I_9: $A \rightarrow aSb\cdot$

$S \rightarrow Sb\cdot$

	FOLLOW
S′	{$}
S	{b, c, $ }
A	{a}



Zust	ACTION				GOTO		
	a	b	c	$	A	S	S′
0	-	s3	-	-	-	1	-
1	-	s2	-	acc	-	-	-
2	-	r2	r2	r2	-	-	-
3	s6	-	-	-	4	-	-
4	s5	-	-	-	-	-	-
5	-	r3	r3	r3	-	-	-
6	r5	s3			-	7	-
7	-	s9	s8	-	-	-	-
8	r4	-	-	-	-	-	-
9	r6						

"altes" Beispiel: Konstruktion der SLR(1)-Syntaxtabelle (9)

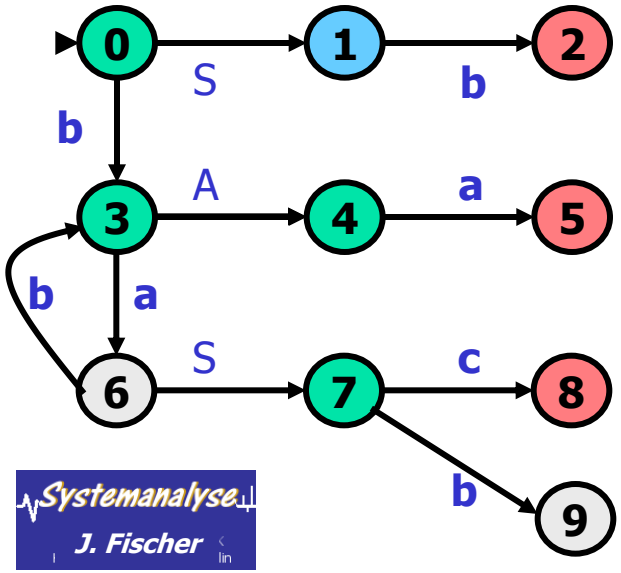
Grammatik

1	S′	→	S
2	S	→	Sb
3	S	|	bAa
4	A	→	aSc
5		|	a
6		|	aSb

I_9: $A \rightarrow aSb\cdot$

$S \rightarrow Sb\cdot$

	FOLLOW
S′	{$}
S	{b, c, $ }
A	{a}



Zust	ACTION				GOTO		
	a	b	c	$	A	S	S′
0	-	s3	-	-	-	1	-
1	-	s2	-	acc	-	-	-
2	-	r2	r2	r2	-	-	-
3	s6	-	-	-	4	-	-
4	s5	-	-	-	-	-	-
5	-	r3	r3	r3	-	-	-
6	r5	s3			-	7	-
7	-	s9	s8	-	-	-	-
8	r4	-	-	-	-	-	-
9	r6	r2	r2	r2	-	-	-

4.6.3 SLR(1) - Syntaxanalyse

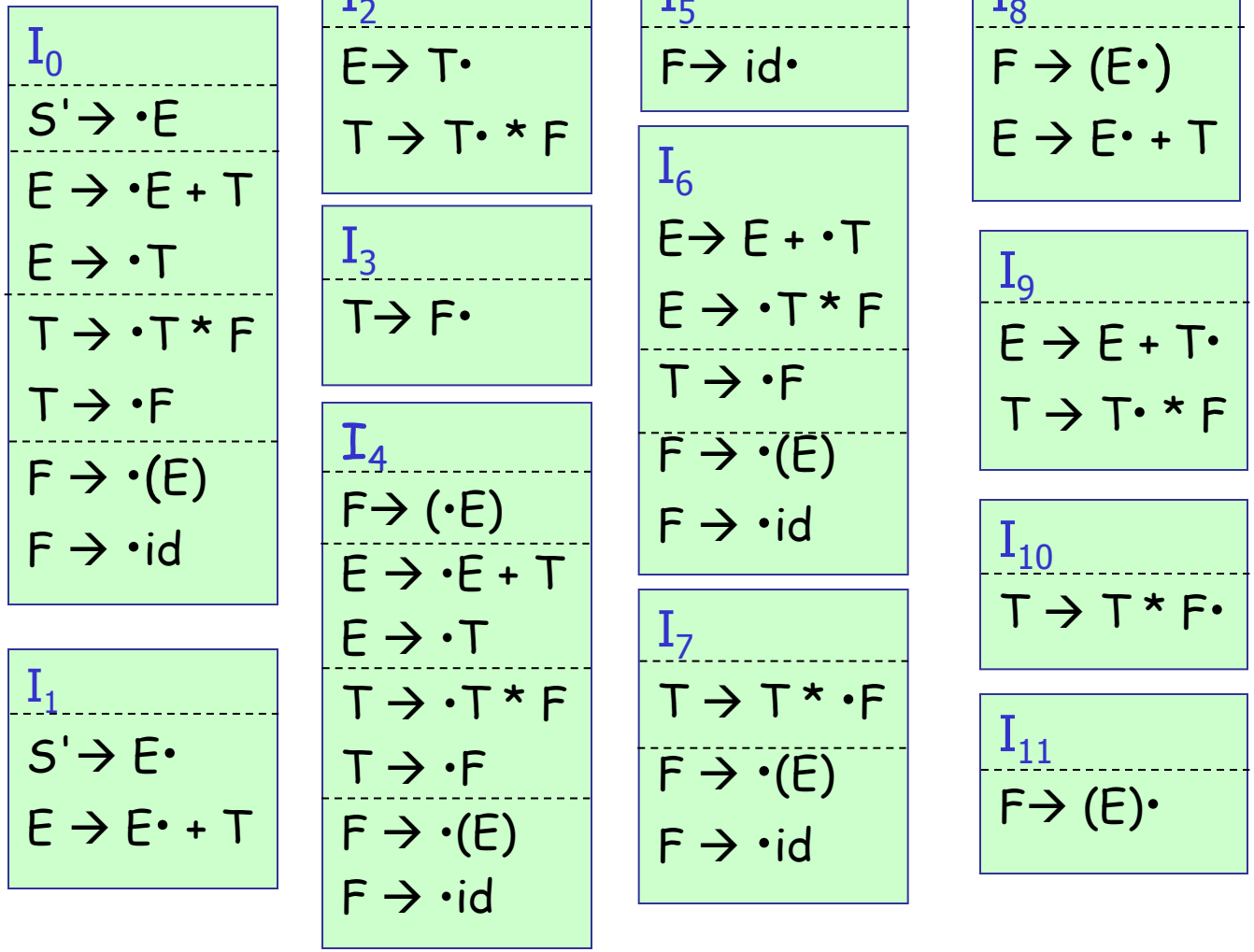
- Konstruktion von SLR(1)-Syntaxtabellen
- Beispiel: SLR(1)-Parser für LR(0)-Konfliktgrammatik
- weiteres Beispiel: SLR(1)-Konstruktion
- SLR(1)-Konfliktbeispiel

Beispiel: Kanonische SLR(1)-Kollektion (1)

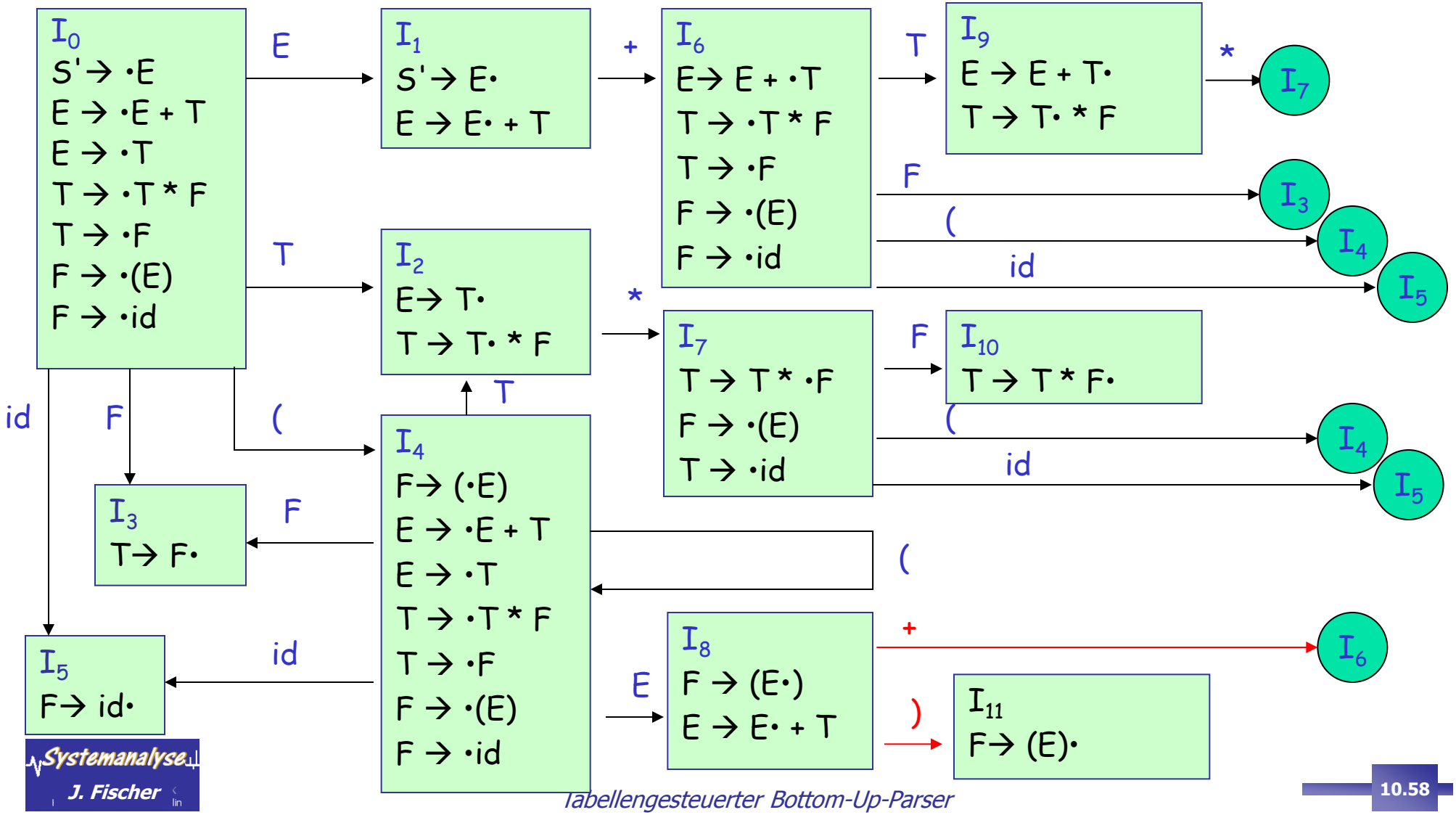
Grammatik

1) S'→ E
2) E → E + T
3) | T
4) T → T * F
5) | F
6) F → (E)
7) | id

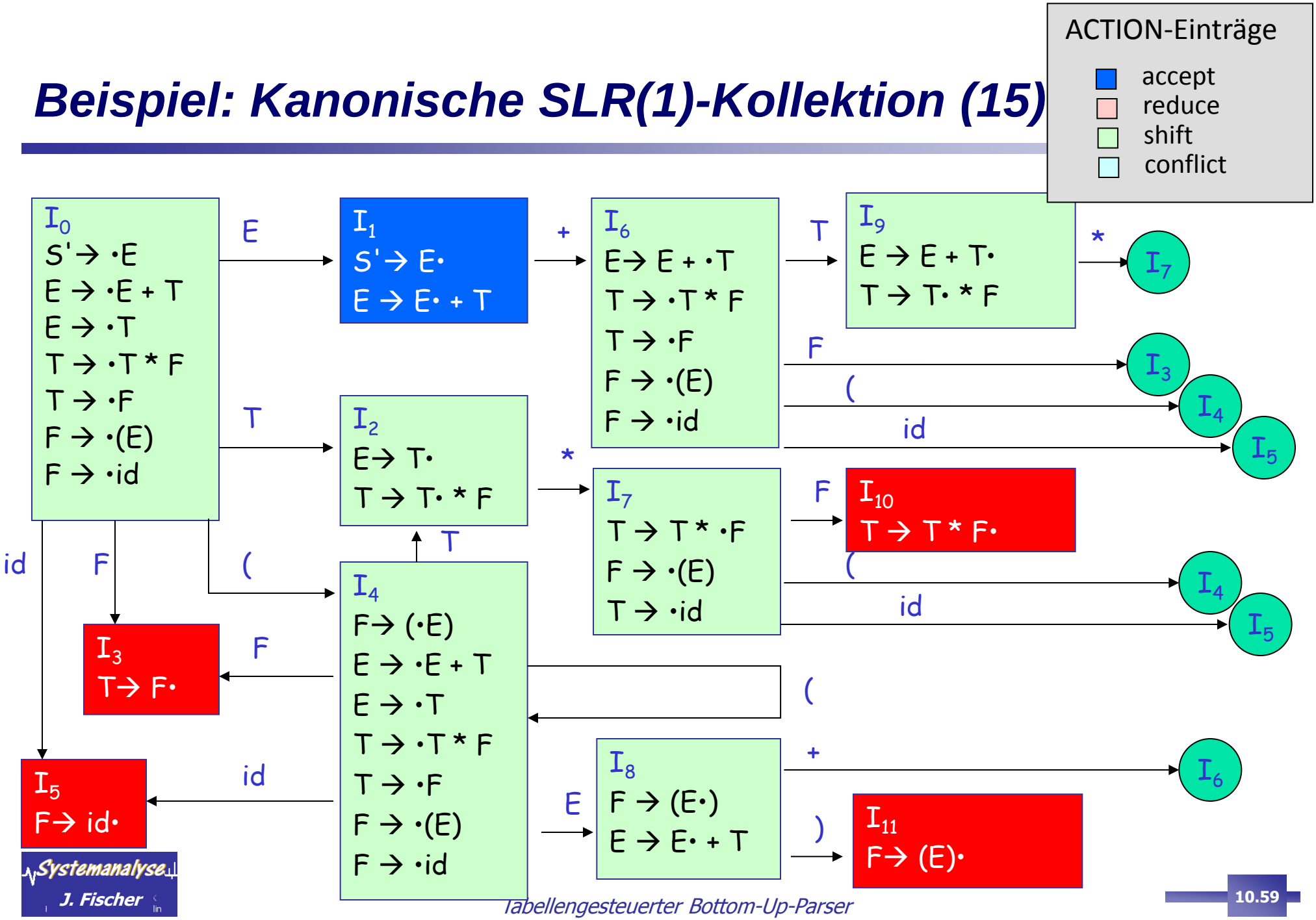
	FOLLOW
E	{+,) }
T	{+, *,) }
F	{+, *,) }



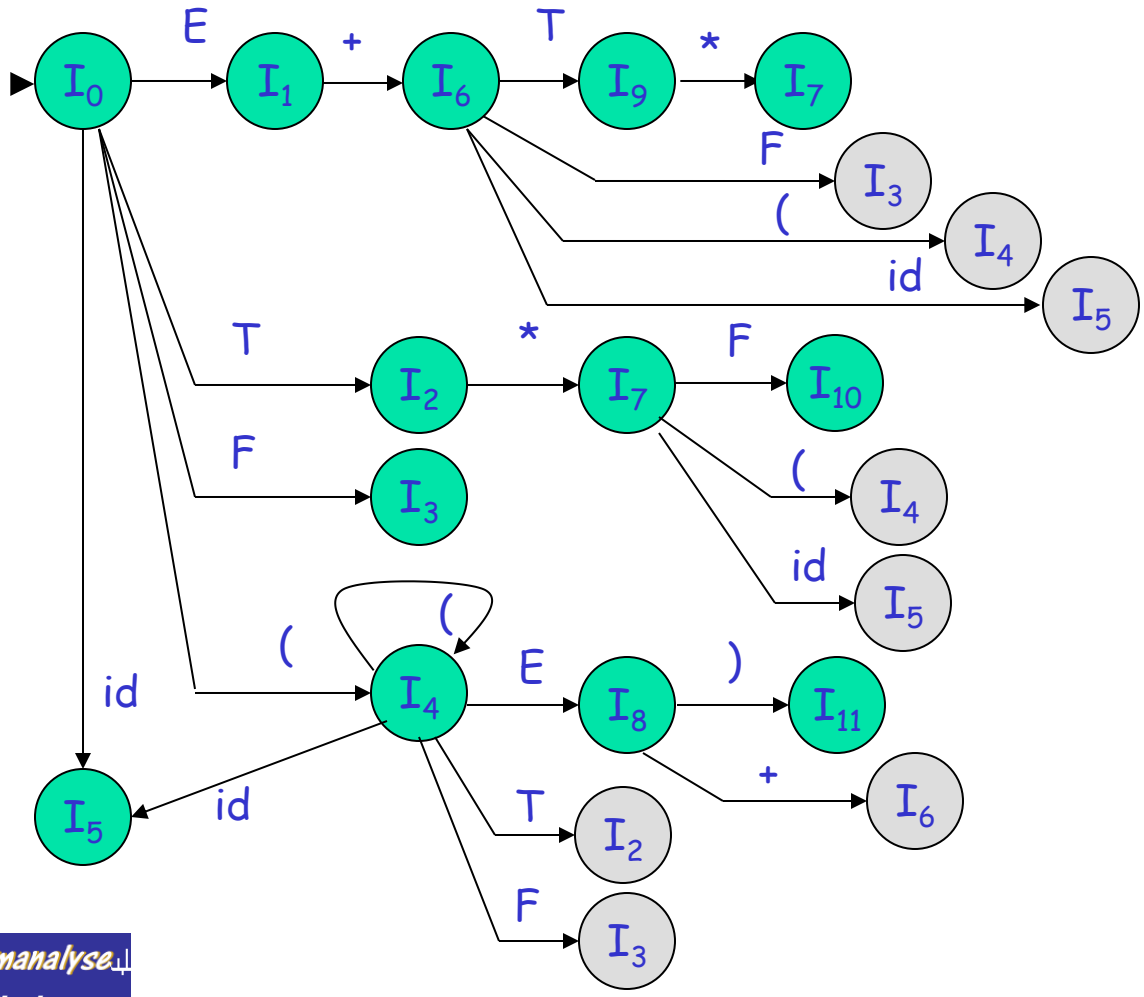
Beispiel: Kanonische SLR(1)-Kollektion (14)

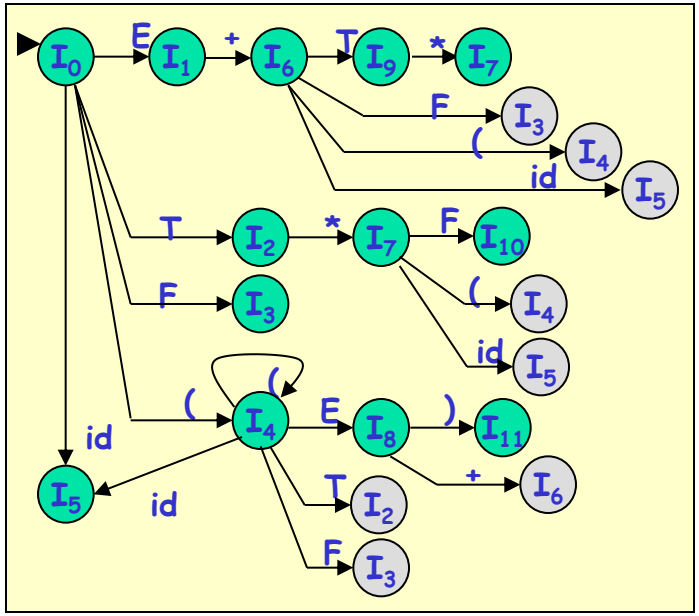


Beispiel: Kanonische SLR(1)-Kollektion (15)



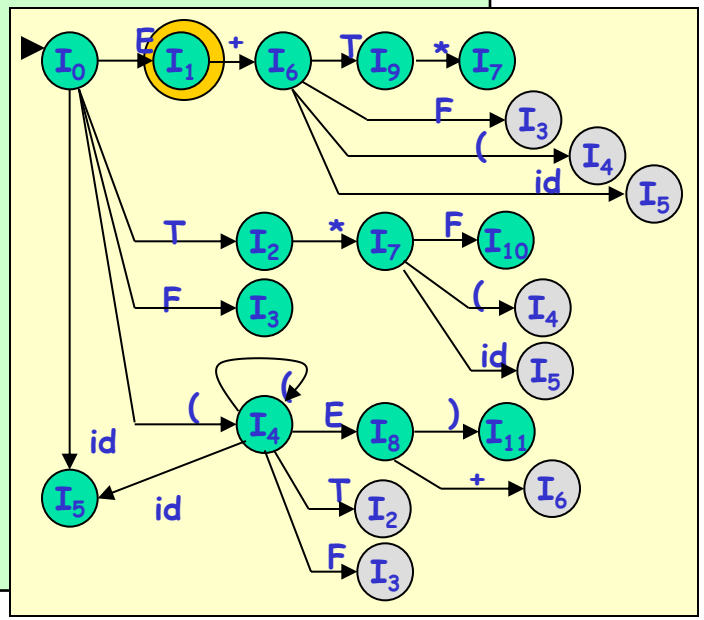
Beispiel: Kanonische SLR(1)-Kollektion (16)





Beispiel: Aufbau einer SLR(1)-Syntaxtabelle (2)

Zust	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5	-	-	s4	-	-	1	2	3
1	-	s6	-	-	-	acc	-	-	-
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									



Beispiel: Aufbau einer SLR(1)-Syntaxtabelle (3)

ACTION[2,a]:= reduce 3
für alle a ∈ FOLLOW(E)

	ACTION					GOTO		
	id	+	*	(	)	E		
0	s5	-	-	s4	-	1		
1	-	s6	-	-	-	-		
2	-	r3	s7	-	r3	-	-	-
3						-	-	-
4								
5								
6								
7								

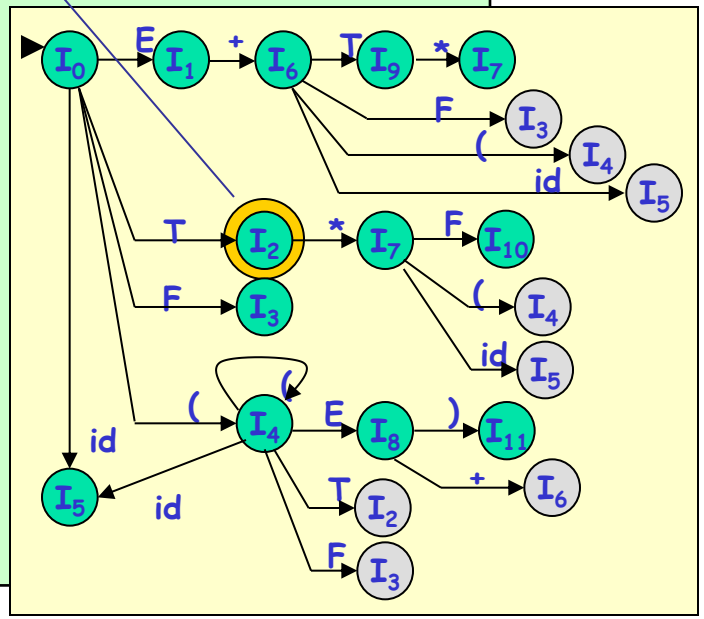
I2
E→ T•
T → T• * F

ACTION[2,*]:= shift 7

	FOLLOW
E	+,)
T	+, *,)
F	+, *,)

Grammatik

1) S'→ E
2) E → E + T
3) | T
4) T → T * F
5) | F
6) F → (E)
7) | id



Beispiel: Aufbau einer SLR(1)-Syntaxtabelle (5)

ACTION[3,a]:= reduce 5
für alle a ∈ FOLLOW(T)

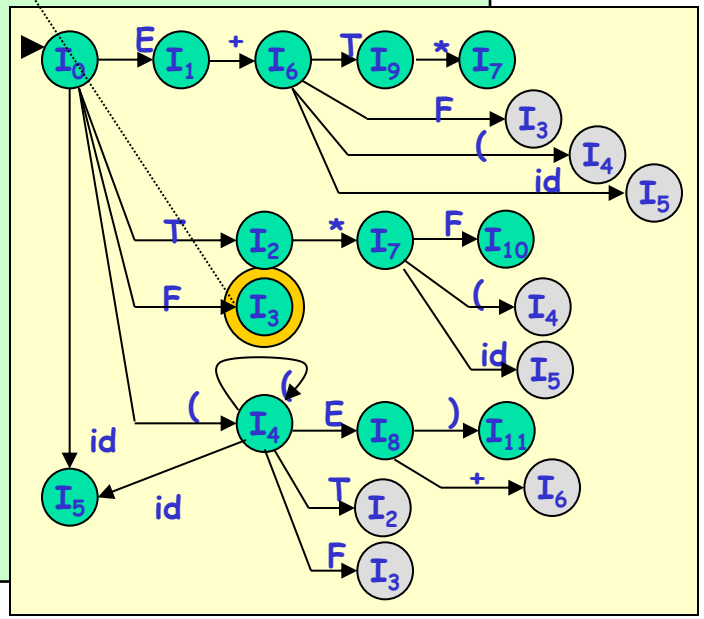
Z	ACTION					GOTO		
	id	+	*	(	)	E	T	F
0	s5	-	-	s4	-	1	2	3
1	-	s6	-	-	-	-	-	-
2	-	r3	s7	-	r3	-	-	-
3	-	r5	r5	-	r5	-	-	-
4								
5								
6								
7								

I_3
T → F•

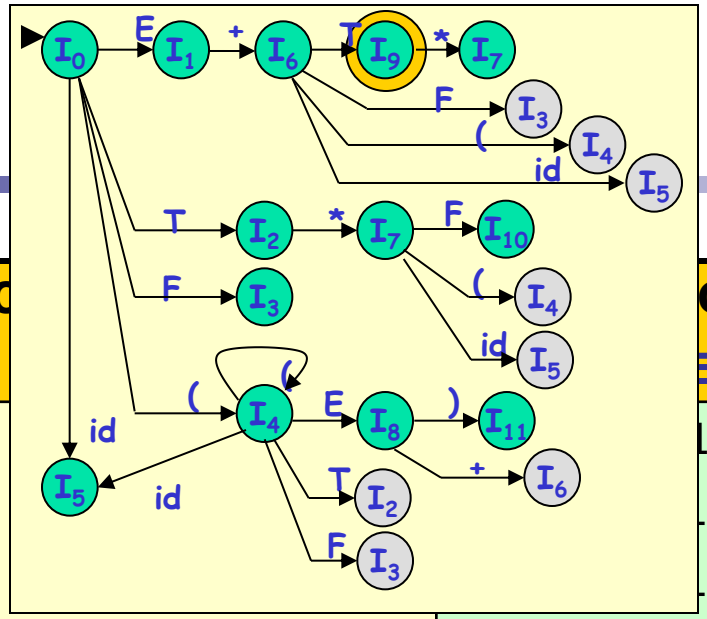
	FOLLOW
E	+,)
T	+, *,)
F	+, *,)

Grammatik

1) S'→ E
2) E → E + T
3) | T
4) T → T * F
5) | F
6) F → (E)
7) | id



	FOLLOW
E	+,)
T	+, *,)
F	+, *,)



I_9

E → E + T• reduce

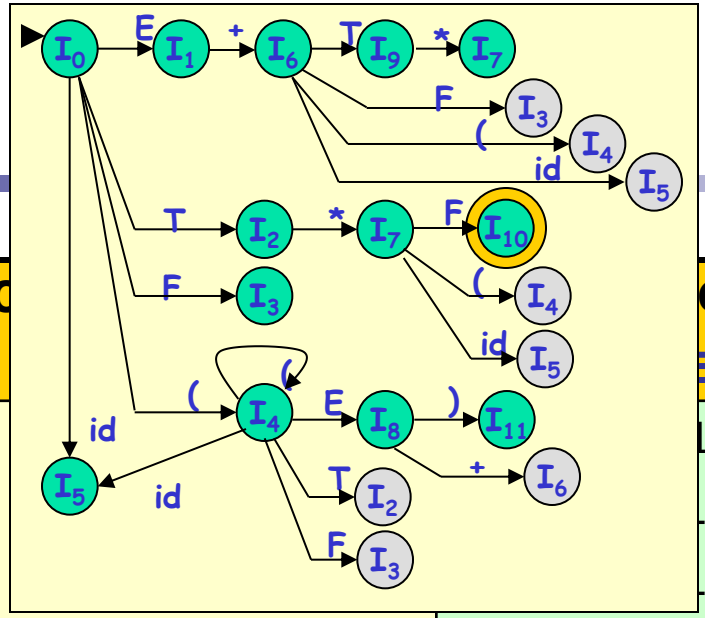
T → T• * F shift

	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5	-	-	s4	-	-	1	2	3
1	-	s6	-	-	-	acc	-	-	-
2	-	r3	s7	-	r3	-	-	-	-
3	-	r5	r5	-	r5	-	-	-	-
4	s5	-	-	s4	-	-	8	2	3
5		r7	r7	-	r7	-	-	-	-
6	s5	-	-	s4	-	-	-	9	3
7	s5	-	-	s4	-	-	-	-	10
8	-	s6	-	-	s11	-	-	-	-
9	-	r2	s7	-	r2	-	-	-	-
10									
11									

Grammatik

1) S'→ E
2) **E → E + T**
3) | T
4) T → T * F
5) | F
6) F → (E)
7) | id

	FOLLOW
E	+,)
T	+, *,)
F	+, *,)



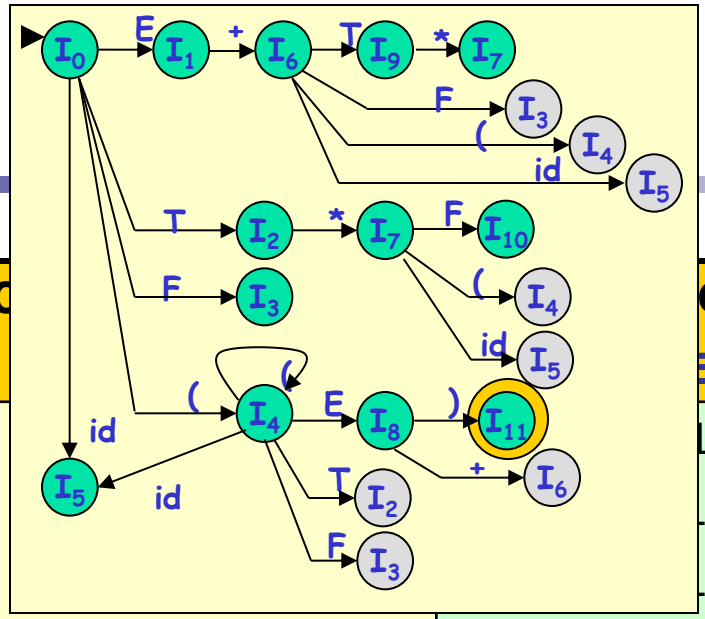
I_{10}
T → T * F•

Grammatik

1) S'→ E
2) E → E + T
3) | T
4) **T → T * F**
5) | F
6) F → (E)
7) | id

	ACTIO						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5	-	-				1		
1	-	s6	-				-		
2	-	r3	s7				-		
3	-	r5	r5	-	r5	-	-		
4	s5	-	-	s4	-	-	8		
5		r7	r7	-	r7	-	-	-	-
6	s5	-	-	s4	-	-	-	9	3
7	s5	-	-	s4	-	-	-	-	10
8	-	s6	-	-	s11	-	-	-	-
9	-	r2	s7	-	r2	-	-	-	-
10	-	r4	r4	-	r4	-	-	-	-
11									

	FOLLOW
E	+,)
T	+, *,)
F	+, *,)



I_{11}
F→ (E)·

Grammatik

1) S'→ E
2) E → E + T
3) | T
4) T → T * F
5) | F
6) **F → (E)**
7) | id

	ACTIO						GOTO		
	id	+	*				E		
0	s5	-	-				1		
1	-	s6	-				-		
2	-	r3	s7				-		
3	-	r5	r5	-	r5	-	-		
4	s5	-	-	s4	-	-	8		
5		r7	r7	-	r7	-	-	-	-
6	s5	-	-	s4	-	-	-	9	3
7	s5	-	-	s4	-	-	-	-	10
8	-	s6	-	-	s11	-	-	-	-
9	-	r2	s7	-	r2	-	-	-	-
10	-	r4	r4	-	r4	-	-	-	-
11	-	r6	r6	-	r6	-	-	-	-

LR(0)- und SLR(1)-Parser-Konstruktion

(Zusammenfassung)

LR(0)-Parserkonstruktion

- LR(0)-Elemente
- Startproduktion → 1.LR(0)-Element
- Hüllenbildung (1.LR(0)-Element → Startzustand vom CFSM)
  mit closure0
- Anwendung von goto0 für alle Grammatiksymbole und Zustände
- LR(0)-Verfahren zur Konstruktion der Syntaxanalysetabelle

$[A \rightarrow \alpha\bullet] \in I_i$ und $A \neq S'$,
dann **ACTION** [i,**a**]:= "**reduce** $A \rightarrow \alpha$"
für **alle Terminale a** von G

SLR(1)-Parserkonstruktion

- LR(0)-Elemente
- Startproduktion → 1.LR(0)-Element
- Hüllenbildung (1.LR(0)-Element → Startzustand vom CFSM)
  mit closure0
- Anwendung von goto0 für alle Grammatiksymbole und Zustände
- SLR(1)-Verfahren zur Konstruktion der Syntaxanalysetabelle

$[A \rightarrow \alpha\bullet] \in I_i$ und $A \neq S'$,
dann **ACTION** [i,**a**]:= "**reduce** $A \rightarrow \alpha$"
für **alle Terminale a** $\in$ FOLLOW(A)

4.6.3 SLR(1) - Syntaxanalyse

- Konstruktion von SLR(1)-Syntaxtabellen
- Beispiel: SLR(1)-Parser für LR(0)-Konfliktgrammatik
- weiteres Beispiel: SLR(1)-Konstruktion
- SLR(1)-Konfliktbeispiel

SLR(1)-Grammatik

- Jede SLR(1)-Grammatik ist eindeutig
- aber nicht jede eindeutige Grammatik ist vom SLR(1)-Typ

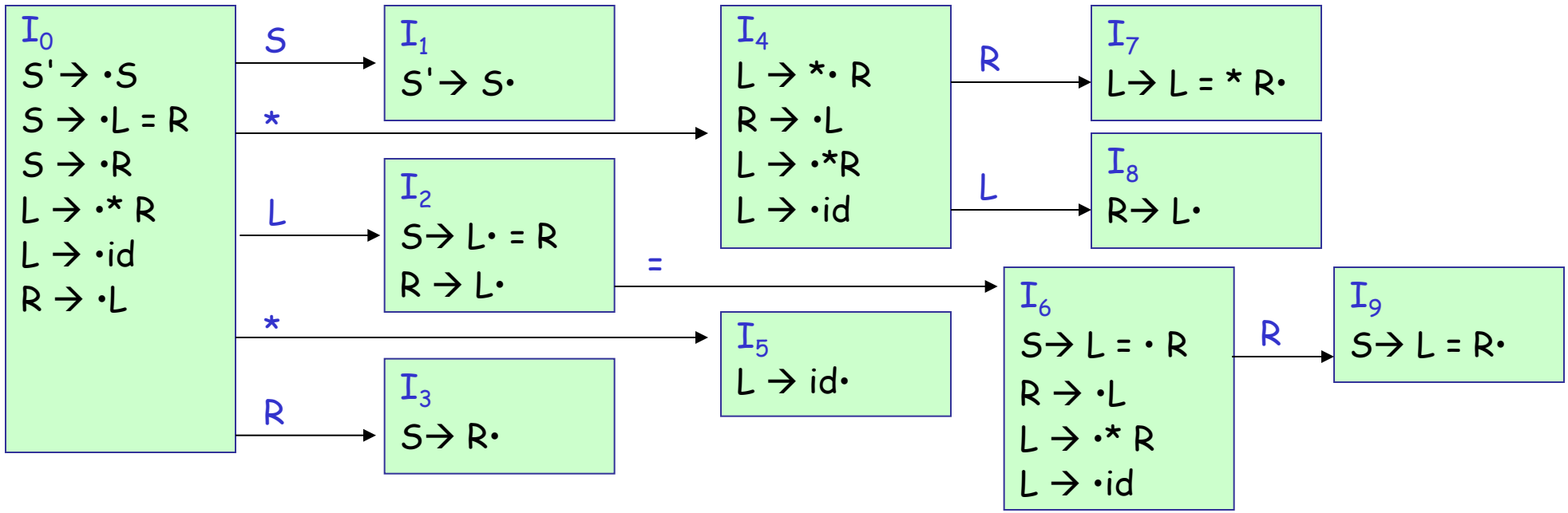
- **Beispiel**

Grammatik

1) S → L = R
2) S → R
3) L → *R
4) L → id
5) R → L

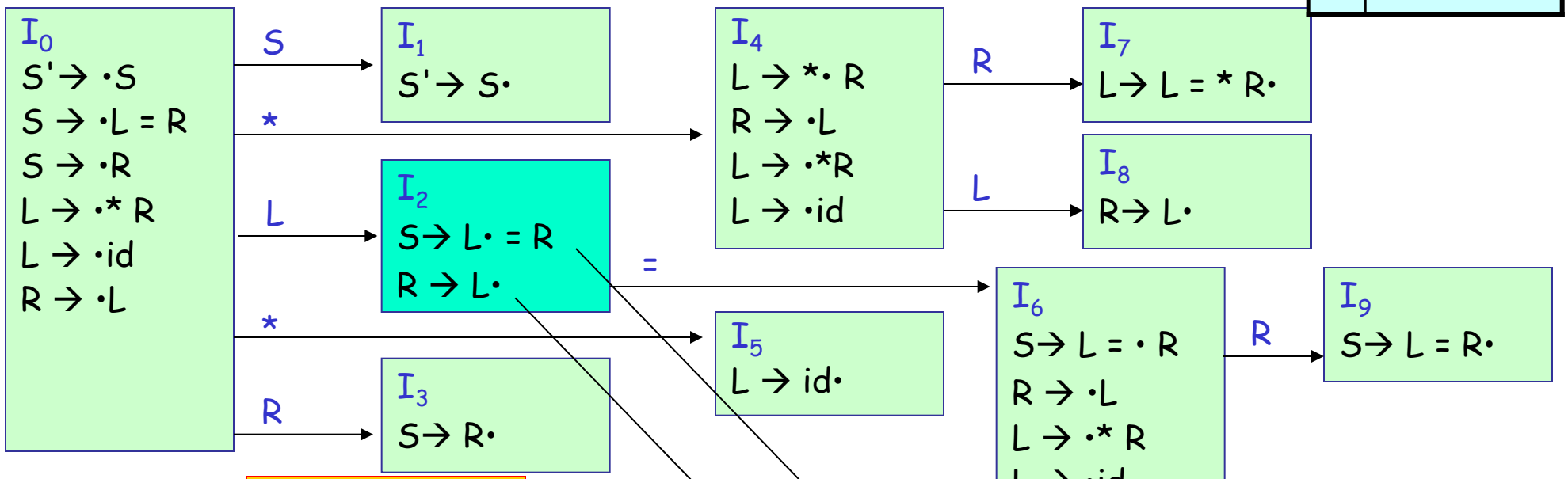
	FOLLOW
S	…
L	…
R	…, =

Eindeutige, aber dennoch Konflikt-Grammatik für SLR(1)



CFS für Konflikt-Grammatik

	FOLLOW
S	…
L	…
R	…, =



Grammatik

1) S → L = R
2) S → R
3) L → *R
4) L → id
5) R → L

ACTION[2, =]:= s6

ACTION[2, =]:= r5

weil '=' ∈ FOLLOW(R)

Zwischenfazit

Grammatik

1) S → L = R
2) S → R
3) L → *R
4) L → id
5) R → L

- Grammatik ist **nicht** vom Typ SLR(1)

➔ SLR-Parser (d.h. deren Grammatiken sind nicht mächtig genug, um gängige Programmier-sprachkonstrukte behandeln zu können)

FOLLOW-Menge ist eine zu grobe Orientierung

4.6.4 LR(1) - Syntaxanalyse

Das LR(1)-Verfahren wird gelegentlich auch als kanonisches LR-Verfahren bezeichnet

- Motivation für LR(1)-Elemente
- Von FOLLOW-Mengen zu Look-Ahead-Betrachtungen
- Konstruktion von LR(1)-Elementen mittels closure1 und goto1
- Beispiel: LR(1)-Parser
- Beispiel: Behandlung der SLR(1)-Konfliktgrammatik mit LR(1)

LookAhead-Mengen statt FOLLOW-Mengen

- FOLLOW-Menge:
  enthält alle Symbole, die einem Nichtterminalsymbol potentiell in jedem möglichen Kontext folgen können

  Zustandsmaschine muss aber sensibler eingestellt werden

- man müsste sich nur auf die Terminalsymbole konzentrieren, die dem Nichtterminal in Abhängigkeit von der Position im Zustandsübergangsdiagramm folgen können

- Menge relevanter LookAhead-Symbole ist eine Teilmenge der FOLLOW-Menge

Erweiterungsprinzip

Ziel: Erweiterung des Zustandes um LookAhead-Informationen, die Konflikte vorab ausschließen:
jeder Zustand eines LR-Parsers soll genau anzeigen,
welche **Eingabesymbole** einem Handle α folgen dürfen,
für den es eine Reduktion $A \rightarrow \alpha$ gibt

Weg: LR(k)-Elemente mit k>=1

Erinnerung:

Ein **LR(k)-Element** ist ein Paar $[\alpha, \beta]$, wobei

- α eine Produktion der Grammatik G ist mit einer Markierung "•" in der RS der Regel,
  die anzeigt, wie viel von der RS einer Produktion schon erkannt worden ist
- β ist die LookAhead-Zeichenkette, die k Symbole (Terminalsymbole oder "$") umfasst

Was sind LR(1)-Elemente?

- LR(1)-Elemente haben die Form [A→ X•YZ, a]
- alle LookAhead-Zeichenketten a haben die Länge 1

LR(1)-Elemente

dienen als

- Basis für Entscheidung: Schieben oder Reduzieren

  sie werden dabei nur für **End-Situationen** gebraucht:
  - für [A→ X•YZ, **a**] hat **a** keine Bedeutung für die Entscheidung
  - für [A→ XYZ•, **a**] ist **a** dagegen wichtig

- Basis für Entscheidung, wonach reduziert werden soll:

  für [A→ α•, **a**] und [B→ α•, **b**] kann entschieden werden,
  ob nach A oder B zu reduzieren ist,
  und zwar abhängig von einem rechten, begrenzten Kontext

erlauben den Einsatz von Grammatiken zur Sprachdefinition, die nicht »eindeutig invertierbar« sind

- es gibt mindestens zwei Regeln mit derselben RS

LR(1)-Elemente (Forts.)

Grammatik

1	S'	→	S
2	S	→	S**b**
3		|	**b**A**a**
4	A	→	**a**S**c**
5		|	**a**
6		|	**a**S**b**

- [S'→ •S, $]
- [S'→ •S, …]

LR(0)-Element (**Kern**)

gültige LookAhead-Symbole eines Kerns bilden die so genannte LookAhead-Menge (Berechnung und Angabe erfolgt **pro Regel** und nicht mehr generell als FOLLOW-Menge des Meta-Symbols)

Lesart:

Bei der Bearbeitung der Regel S'→ S befinden wir uns noch **vor** dem S.

Sollten wir irgendwann die gesamte rechte Seite (also S) verarbeitet haben, so wird nach dieser Regel reduziert, **aber** nur unter der Voraussetzung, dass das nächste Eingabezeichen ein Element der LookAhead-Menge ist

LR(1)-Elemente (Forts.)

zwei LR(1)-Elemente

- $[\alpha \rightarrow \bullet\beta,\ \mathbf{a}]$
- $[\alpha \rightarrow \bullet\beta,\ \mathbf{b}]$

sind verschieden,
auch wenn sie sich nur in der LookAhead-Menge unterscheiden sollten

wichtig für die Zustandskonstruktion des charakteristischen endlichen Automaten

FAZIT im Vergleich: SLR(1) – LR(1)

Frage: Wann wird reduziert?

- **SLR(1)-Analyse:**
  FOLLOW-Menge entscheidet,
  ob das nächste Terminalsymbol dem Nichtterminalsymbol,
  zu dem reduziert werden soll, überhaupt folgen darf:
  gehört es zur FOLLOW-Menge, wird reduziert.

- **LR(1)-Analyse:**
  LookAhead-Menge entscheidet,
  ob das nächste Terminalsymbol dem Nichtterminalsymbol, zu dem reduziert werden soll,
  nach Anwendung der Reduktionsregel folgen darf:
  gehört es zur LookAhead-Menge, wird reduziert.

Unterschied:
Man betrachtet bei LR(1) nicht mehr Mengen von Terminalsymbolen, die einem Nichtterminal generell folgen dürfen, sondern nur solche, die diesem Nichtterminal nach Anwendung einer bestimmten **Reduktionsregel** folgen dürfen.