

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing



Vorlesung Compilerbau: Syntaktische Analyse – Parsing VII LR(k)-Parsing

Vorlesung des BA-Studiums
Prof. Johann Christoph Freytag, Ph.D.
Institut für Informatik, Humboldt-Universität zu Berlin
SoSe 2018

© Prof. J.C. Freytag, Ph.D.

10.1

Ziele

- Darstellung der Grundlagen der Syntaxtheorie aus der Theorie der Automaten und formalen Sprachen
- Kontextfreie Grammatiken mit Ableitungsbegriff und Kellerautomaten
- Konstruktion von Top-Down- und Bottom-Up-Parsern
- LL(k)- und LR(k)-Grammatiken für Teilmengen kontextfreier Sprachen
- LR(1)-Analyseverfahren
 - LR(0), SLR(1)
 - kanonische LR(1)
 - LALR(1)

© Prof. J.C. Freytag, Ph.D.

10.2



Erweiterungsprinzip


Ziel

- Erweiterung des Zustandes um Lookahead-Informationen, die Konflikte ausschließen:

jeder Zustand eines LR-Parsers soll genau anzeigen, welche Eingabesymbole einer Handle α folgen können, für den es eine Reduktion $A \rightarrow \alpha$ gibt

Weg

- LR(k)-Elemente



LR(1)-Elemente

Erinnerung:

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

- α eine Produktion der Grammatik G ist und einer Markierung " \bullet " 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 $[A \rightarrow X \bullet YZ, a]$
- alle *Lookahead*-Zeichenketten a haben die Länge 1

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing



LR(1)-Elemente (Forts.)

Motivation für *Lookahead*-Symbole

- Basis für Entscheidung: Schieben oder Reduzieren
wird dabei nur für das »Ende« gebraucht:
 - für $[A \rightarrow X \bullet YZ, a]$ hat a keine Bedeutung
 - für $[A \rightarrow XYZ \bullet, a]$ ist a nützlich und wichtig
- Basis für Entscheidung: wonach reduziert werden soll
Für $[A \rightarrow \alpha \bullet, a]$ und $[B \rightarrow \alpha \bullet, b]$ kann entschieden werden, ob nach A oder B zu reduzieren ist, und zwar abhängig von einem rechten, begrenzten Kontext
- erlaubt den Einsatz von Grammatiken zur Sprachdefinition, die nicht »eindeutig invertierbar« sind
 - eindeutig invertierbar: keine zwei Regeln haben dieselbe RS

© Prof. J.C. Freytag, Ph.D.

10.5



Konstruktion von LR(1)-Elementen

- ähnelt der Konstruktion von LR(0)-Elementen
- dabei Modifikation der Operatoren
Hülle und Sprung
 - $\text{closure0} \rightarrow \text{closure1}$
 - $\text{goto0} \rightarrow \text{goto1}$

© Prof. J.C. Freytag, Ph.D.

10.6

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing

closure1

Gegeben sei ein Element $[A \rightarrow \alpha \cdot B\beta, a]$,
dann enthält die **Hülle**

- das Element selbst und
 - alle weiteren Elemente, die aus einer Teilzeichenkette gebildet werden können, die α folgt
- D.h., falls der Parser einen brauchbaren Präfix α im Keller gespeichert hat, dann sollte die Eingabe zu $B\beta$ reduzieren
oder zu γ für ein anderes Element $[B \rightarrow \cdot \gamma, b]$

```
function closure1 (I)
repeat
  if  $[A \rightarrow \alpha \cdot B\beta, a] \in I$  then add  $[B \rightarrow \cdot \gamma, b]$  to I where  $b \in \text{FIRST}(\beta a)$ 
until (no more items can be added to I)
return I
```

© Prof. J.C. Freytag, Ph.D.

10.7

Beispiel: closure1

Bildung von **closure1** ($\{[S' \rightarrow \cdot S, \$]\}$)

- $[S' \rightarrow \cdot S, \$] \in \text{closure1}$ (Anfangsschritt)

- Match mit $[A \rightarrow \alpha \cdot B\beta, a]$ wobei

- $A = S'$
- $\alpha = \epsilon$
- $B = S$
- $\beta = \epsilon$
- $a = \$$

fordert die Hinzunahme von

$[B \rightarrow \cdot \gamma, b]$ für jede Produktion $B \rightarrow \gamma$
und jedes Terminal b in $\text{FIRST}(\beta a)$

Grammatik

1	$S' \rightarrow S$
2	$S \rightarrow CC$
3	$C \rightarrow cC$
4	$C \rightarrow d$

weiterer Kandidat: $S \rightarrow CC$ mit $\text{FIRST}(\epsilon \$) = \{ \$ \}$

$[S \rightarrow \cdot CC, \$] \in \text{closure1}$

© Prof. J.C. Freytag, Ph.D.

10.8

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing

Beispiel: closure1 (Forts.)

Bildung von **closure1** ($\{[S' \rightarrow \cdot S, \$]\}$)

- da $[S \rightarrow \cdot CC, \$] \in \text{closure1}$

Match mit $[A \rightarrow \alpha \cdot B \beta, a]$ wobei

$A = S, \alpha = \epsilon, B = C, \beta = C, a = \$$

fordert die Hinzunahme von

$[C \rightarrow \cdot cC, c/d]$ mit $\{c, d\} = \text{FIRST}(C\$) = \text{FIRST}(\beta a)$

$[C \rightarrow \cdot d, c/d]$ mit $\{c, d\} = \text{FIRST}(C\$) = \text{FIRST}(\beta a)$

- keines der neuen Elemente hat ein Nichtterminal rechts vom Punkt (damit ist closure1 komplett)

kompakte Schreibweise für 6 Elemente

$I_0:$	$S' \rightarrow \cdot S,$	$\$$
	$S \rightarrow \cdot CC,$	$\$$
	$C \rightarrow \cdot cC,$	c/d
	$C \rightarrow \cdot d,$	c/d

Grammatik

1	$S' \rightarrow S$
2	$S \rightarrow CC$
3	$C \rightarrow cC$
4	$C \rightarrow d$

© Prof. J.C. Freytag, Ph.D.

10.9

goto1

Sei I eine Menge an LR(1)-Elementen und X ein Grammatiksymbol,

- dann ist **GOTO(I, X)** die Hülle der Menge aller Elemente $[A \rightarrow \alpha X \beta, a]$ mit $[A \rightarrow \alpha \cdot X \beta, a] \in I$

Intuition:

- Falls I die Menge aller brauchbaren Elemente für einen brauchbaren Präfix γ ist,
- dann ist **GOTO(I, X)** die Menge aller erlaubten Elemente für den brauchbaren Präfix γX

GOTO(I, X) repräsentiert den (Folge-)Zustand, nachdem X im Zustand I erkannt worden ist

function goto1 (I, X)

sei L die Menge aller Elemente $[A \rightarrow \alpha X \beta, a]$ mit $[A \rightarrow \alpha \cdot X \beta, a] \in I$;

return (closure1 (L))

© Prof. J.C. Freytag, Ph.D.

10.10

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing

Beispiel: goto1

I_0 :	$S' \rightarrow \cdot S,$	\$
	$S \rightarrow \cdot CC,$	\$
	$C \rightarrow \cdot cC,$	c/d
	$C \rightarrow \cdot d,$	c/d

Grammatik	
1	$S' \rightarrow S$
2	$S \rightarrow CC$
3	$C \rightarrow cC$
4	$C \rightarrow d$

Bildung von **goto1** (I_0, X) für alle möglichen X

- **X=S** : Bildung von closure1 ($[S' \rightarrow S \cdot, \$]$)

Punkt steht am Ende:

I_1 :	$S' \rightarrow S \cdot,$	\$
---------	---------------------------	----

© Prof. J.C. Freytag, Ph.D.

10.11

Beispiel: goto1 (Forts.)

I_0 :	$S' \rightarrow \cdot S,$	\$
	$S \rightarrow \cdot CC,$	\$
	$C \rightarrow \cdot cC,$	c/d
	$C \rightarrow \cdot d,$	c/d

Grammatik	
1	$S' \rightarrow S$
2	$S \rightarrow CC$
3	$C \rightarrow cC$
4	$C \rightarrow d$

Bildung von **goto1** (I_0, X) für alle möglichen X

- **X=C** : Bildung von closure1 ($[S \rightarrow C \cdot C, \$]$)

I_2 :	$S \rightarrow C \cdot C,$	\$
	$C \rightarrow \cdot cC,$	\$
	$C \rightarrow \cdot d,$	\$

© Prof. J.C. Freytag, Ph.D.

10.12

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing

Beispiel: goto1 (Forts.)

I_0 :	$S' \rightarrow \cdot S,$	$\$$
	$S \rightarrow \cdot CC,$	$\$$
	$C \rightarrow \cdot cC,$	c/d
	$C \rightarrow \cdot d,$	c/d

Grammatik	
1	$S' \rightarrow S$
2	$S \rightarrow CC$
3	$C \rightarrow cC$
4	$C \rightarrow d$

Bildung von **goto1** (I_0, X) für alle möglichen X

- $X=c$: Bildung von $\text{closure1} (\{ [C \rightarrow c \bullet C, c/d] \})$

I_3 :	$C \rightarrow c \cdot C,$	c/d
	$C \rightarrow \cdot cC,$	c/d
	$C \rightarrow \cdot d,$	c/d

© Prof. J.C. Freytag, Ph.D.

10.13

Beispiel: goto1 (Forts.)

I_0 :	$S' \rightarrow \cdot S,$	$\$$
	$S \rightarrow \cdot CC,$	$\$$
	$C \rightarrow \cdot cC,$	c/d
	$C \rightarrow \cdot d,$	c/d

Grammatik	
1	$S' \rightarrow S$
2	$S \rightarrow CC$
3	$C \rightarrow cC$
4	$C \rightarrow d$

Bildung von **goto1** (I_0, X) für alle möglichen X

- $X=d$: Bildung von $\text{closure1} (\{ [C \rightarrow d \bullet, c/d] \})$

I_4 :	$C \rightarrow d \cdot,$	c/d
---------	--------------------------	-------

© Prof. J.C. Freytag, Ph.D.

10.14

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing



Erzeugen der LR(1)-Elemente-Sammlung

- Beginn der Erzeugung mit dem Element $[S' \rightarrow \bullet S, \$]$ mit
 - S' als (neuem) Startsymbol der erweiterten Grammatik G'
 - S ist das Startsymbol der Grammatik G
 - $\$$ repräsentiert das Ende der Eingabe
- Erzeugen der Sammlung aller LR(1)-Element-Mengen

```

function items (G')
     $s_0 \leftarrow \text{closure1}(\{[S' \rightarrow \bullet S, \$]\})$ ;
     $\mathbf{S} \leftarrow \{s_0\}$ ;
    repeat
        for each set of item  $s \in \mathbf{S}$  do
            for each grammar symbol  $X$  do
                if  $\text{goto1}(s, X) \neq \emptyset$  and  $\text{goto1}(s, X) \notin \mathbf{S}$  then add  $\text{goto1}(s, X)$  to  $\mathbf{S}$  ;
    until (no more item sets can be added to  $\mathbf{S}$  )
    return ( $\mathbf{S}$  )
    
```

© Prof. J.C. Freytag, Ph.D.

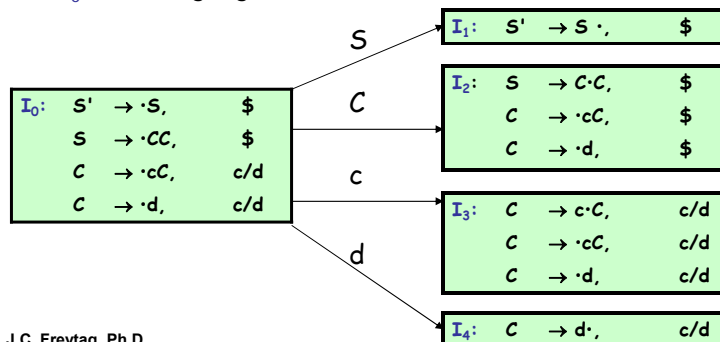
10.15



Beispiel: Komplettierung einer LR(1)-Elemente-Sammlung

Bildung von **goto1 (I, x)** für alle Kombinationen möglicher Zustände I und möglicher Eingaben x

- I_0 hat Übergänge bei $x = S, C, c$ und d



Grammatik

1	$S' \rightarrow S$
2	$S \rightarrow CC$
3	$C \rightarrow cC$
4	$C \rightarrow d$

© Prof. J.C. Freytag, Ph.D.

10.16

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing



Beispiel: Komplettierung einer LR(1)-Elemente-Sammlung

Bildung von **goto1 (I, x)** für alle Kombinationen möglicher Zustände I und möglicher Eingaben X

- I_1 ist Endzustand

$I_1: S' \rightarrow S \cdot, \$$

© Prof. J.C. Freytag, Ph.D.

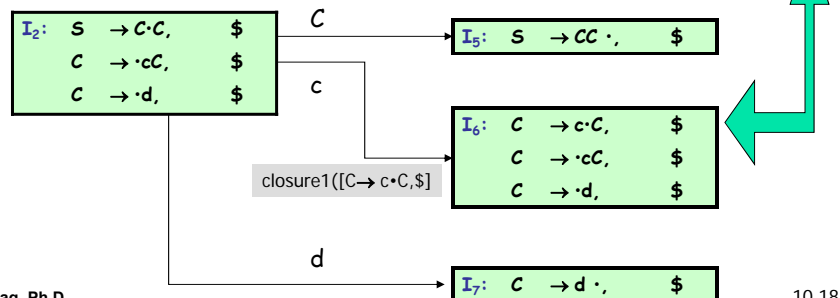
10.17



Beispiel: Komplettierung einer LR(1)-Elemente-Sammlung

Bildung von **goto1 (I, x)** für alle Kombinationen möglicher Zustände I und möglicher Eingaben X

- I_2 hat Übergänge bei $X = C, c$ und d



© Prof. J.C. Freytag, Ph.D.

10.18

Vorlesung Compilerbau (SoSe 2018)

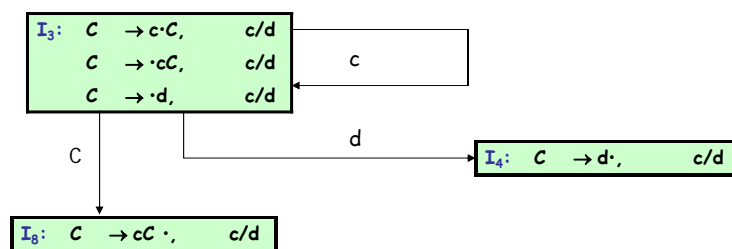
Teil 10: LR(k)-Parsing



Beispiel: Komplettierung einer LR(1)-Elemente-Sammlung

Bildung von **goto1 (I, X)** für alle Kombinationen möglicher Zustände I und möglicher Eingaben X

- I_3 hat Übergänge bei $X = c$ und d



© Prof. J.C. Freytag, Ph.D.

10.19



Beispiel: Komplettierung einer LR(1)-Elemente-Sammlung

Bildung von **goto1 (I, X)** für alle Kombinationen möglicher Zustände I und möglicher Eingaben X

- I_4 und I_5 haben keine Übergänge



© Prof. J.C. Freytag, Ph.D.

10.20

Vorlesung Compilerbau (SoSe 2018)

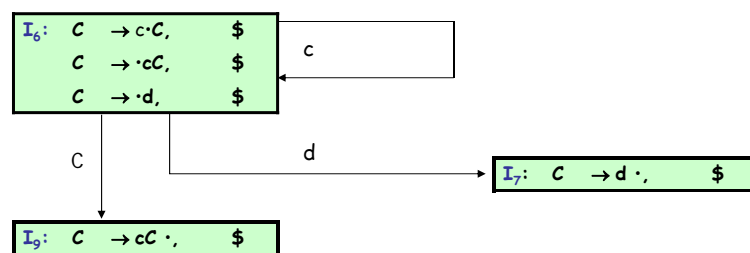
Teil 10: LR(k)-Parsing



Beispiel: Komplettierung einer LR(1)-Elemente-Sammlung

Bildung von **goto1 (I, X)** für alle Kombinationen möglicher Zustände I und möglicher Eingaben X

- I_6 hat Übergänge für c, d und C



© Prof. J.C. Freytag, Ph.D.

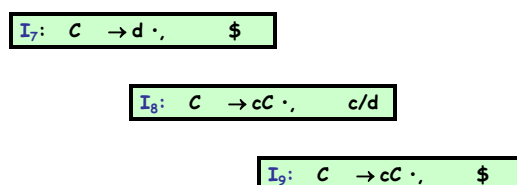
10.21



Beispiel: Komplettierung einer LR(1)-Elemente-Sammlung

Bildung von **goto1 (I, X)** für alle Kombinationen möglicher Zustände I und möglicher Eingaben X

- I_7 , I_8 und I_9 haben keine Übergänge (fertig)

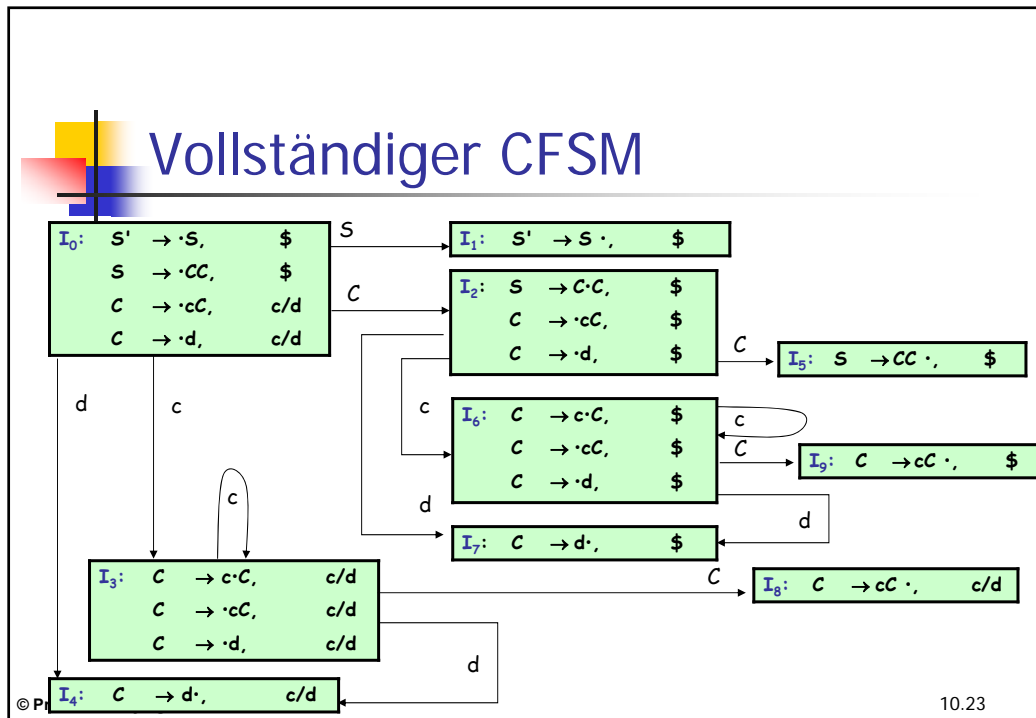


© Prof. J.C. Freytag, Ph.D.

10.22

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing



Konstr. der LR(1)-Syntaxanalysetabelle

- (1) Konstruktion der Sammlung $\{I_0, I_1, \dots, I_n\}$ von LR(1)-Elemente-Mengen für G' (Erweiterung von G)
- (2) Aufbau der ACTION-Tabelle:
 - (a) $[A \rightarrow \alpha \cdot a \beta, b] \in I_i$ und $\text{goto1}(I_i, a) = I_j$ dann $\text{ACTION}[i, a] \leftarrow \text{shift } j$
 - (b) $[A \rightarrow \alpha \cdot, a] \in I_i$ und $A \neq S'$, dann $\text{ACTION}[i, a] \leftarrow \text{reduce } A \rightarrow \alpha$
 - (c) $[S' \rightarrow S \cdot, \$] \in I_i$ dann $\text{ACTION}[i, \$] \leftarrow \text{accept}$
- (3) $\text{goto1}(I_i, A) = I_j$ dann $\text{GOTO}[i, A] \leftarrow j$
- (4) setze undefinierte Einträge in ACTION und GOTO auf »error«
- (5) Anfangszustand des Parsers s_0 ist $\text{closure1}([S' \rightarrow \cdot S, \$])$

Bem.: "Lookahead" wurde von vorn herein in den DFA eingebaut.

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing

Beispiel: Konstruktion der LR(1)- Syntaxanalysetabelle

Zust.	ACTION			GOTO	
	c	d	\$	S	C
0	s3	s4	-	1	2
1	-	-	acc	-	-
2	s6	s7	-	-	5
3	s3	s4	-	-	8
4	r4	r4	-	-	-
5	-	-	r2	-	-
6	s6	s7	-	-	9
7	-	-	r4	-	-
8	r3	r3	-	-	-
9	-	-	r3	-	-

Grammatik

- 1 $S' \rightarrow S$
- 2 $S \rightarrow CC$
- 3 $C \rightarrow cC$
- 4 $\quad \quad \quad | d$

© Prof. J.C. Freytag, Ph.D.

10.25

Beziehungen....

- Jede SLR(1)-Grammatik ist eine LR(1)-Grammatik
- aber: für eine SLR(1)-Grammatik kann der kanonische LR-Parser mehr Zustände haben als der SLR-Parser
 - letztens Beispiel im Foliensatz 9 (mit $L=R$, u.a.....)
 - SLR(1)-Parser: 10 Zustände
 - LR(1)-Parser: 14 Zustände

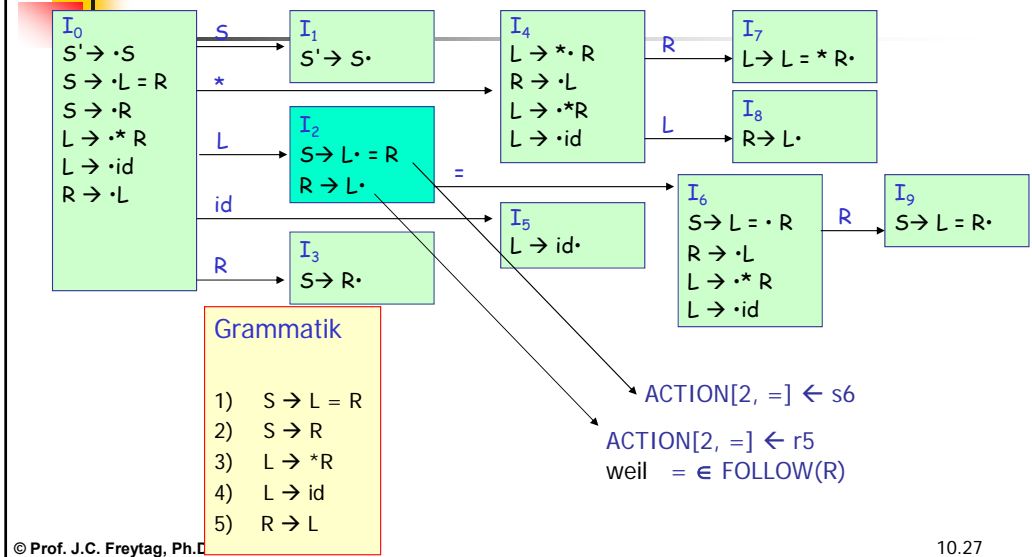
© Prof. J.C. Freytag, Ph.D.

10.26

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing

Beispiel mit Shift-Reduce-Konflikt



© Prof. J.C. Freytag, Ph.D.

10.27

Beispiel mit kanonischer LR(1)-Analyse

Grammatik	
1	$S \rightarrow L = R$
2	$ R$
3	$L \rightarrow *R$
4	$ id$
5	$R \rightarrow L$

Für I_2 gibt es **keinen** Shift-Reduce-Konflikt mehr:

- reduce auf \$ und
- shift auf =

I_0 :	$S' \rightarrow \cdot S, \$$	
	$S \rightarrow \cdot L = R, \$$	
	$S \rightarrow \cdot R, \$$	
	$R \rightarrow \cdot L, \$$	
	$L \rightarrow \cdot * R, \neq \$$	
	$L \rightarrow \cdot id, \neq \$$	
I_1 :	$S' \rightarrow S \cdot, \$$	
I_2 :	$S \rightarrow L \cdot = R, \$$	
	$R \rightarrow L \cdot, \$$	
I_3 :	$S \rightarrow R \cdot, \$$	
I_4 :	$L \rightarrow * \cdot R, \neq \$$	
	$R \rightarrow \cdot L, \neq \$$	
	$L \rightarrow \cdot * R, \neq \$$	
	$L \rightarrow \cdot id, \neq \$$	
I_5 :	$L \rightarrow id \cdot, \neq \$$	
I_6 :	$S \rightarrow L = \cdot R, \$$	
	$R \rightarrow \cdot L, \$$	
	$L \rightarrow \cdot * R, \$$	
	$L \rightarrow \cdot id, \$$	
I_7 :	$L \rightarrow * R \cdot, \neq \$$	
I_8 :	$R \rightarrow L \cdot, \neq \$$	
I_9 :	$S \rightarrow L = R \cdot, \$$	
I_{10} :	$R \rightarrow L \cdot, \$$	
I_{11} :	$L \rightarrow * \cdot R, \$$	
	$R \rightarrow \cdot L, \$$	
	$L \rightarrow \cdot * R, \$$	
	$L \rightarrow \cdot id, \$$	
I_{12} :	$L \rightarrow id \cdot, \$$	
I_{13} :	$L \rightarrow * R \cdot, \$$	

© Prof. J.C. Freytag, Ph.D.

10.28

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing

Zurück zur SLR(1)-Grammatik

Generell hat LR(1) mehr Zustände als LR(0)/SLR(1)

1	$S \rightarrow E$	4	$T \rightarrow T * F$
2	$E \rightarrow E + T$	5	$\mid F$
3	$\mid T$	6	$F \rightarrow id$
		7	$\mid (E)$

LR(1)-Elemente-Menge (Auszug)

I_0 :	$S \rightarrow \cdot E,$	$\$$
	$E \rightarrow \cdot E + T,$	$+\$$
	$E \rightarrow \cdot T,$	$+\$$
	$T \rightarrow \cdot T * F,$	$*+\$$
	$T \rightarrow \cdot F,$	$*+\$$
	$T \rightarrow \cdot id,$	$*+\$$
	$F \rightarrow \cdot (E),$	$*+\$$

I_0' :	$F \rightarrow (\cdot E),$	$*+\$$
	$E \rightarrow \cdot E + T,$	$+\)$
	$E \rightarrow \cdot T,$	$+\)$
	$T \rightarrow \cdot T * F,$	$*+\)$
	$T \rightarrow \cdot F,$	$*+\)$
	$F \rightarrow \cdot id,$	$*+\)$
	$F \rightarrow \cdot (E),$	$*+\)$

I_0'' :	$F \rightarrow (\cdot E),$	$*+\)$
	$E \rightarrow \cdot E + T,$	$+\)$
	$E \rightarrow \cdot T,$	$+\)$
	$T \rightarrow \cdot T * F,$	$*+\)$
	$T \rightarrow \cdot F,$	$*+\)$
	$F \rightarrow \cdot id,$	$*+\)$
	$F \rightarrow \cdot (E),$	$*+\)$

© Prof. J.C. Freytag, Ph.D.

10.29

LALR-Syntaxanalyse

(LookAhead-**LR**-Syntaxanalyse)

© Prof. J.C. Freytag, Ph.D.

10.30

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing



Zustandsraumdimensionen

- LALR-Technik wird in der Praxis benutzt, da Tabellen **kleiner** sind als die der kanonische LR-Technik
- LALR- und SLR-Parser haben die gleiche Zustandsanzahl (z.B. einige hundert für Pascal) Zustände eines LALR-Parsers sind aber komplexer
- ein kanonischer LR-Parser benötigt für Pascal einige tausend Zustände

© Prof. J.C. Freytag, Ph.D.

10.31



Idee einer Zustandsreduktion

betrachten noch einmal LR(1)-Elemente

$I_4: C \rightarrow d \cdot, \quad c/d$

$I_7: C \rightarrow d \cdot, \quad \$$

versuchen dabei

die Rollen der ähnlichen Zustände beim Parser zu erkennen

Zustand: I_4

1. Parser schiebt erste **c**-Gruppe und das folgende **d** auf den Stack und erreicht nach Lesen von **d** den Zustand I_4
2. Reduktion $C \rightarrow d$
(Voraussetzung: nächstes Eingabezeichen ist **c** oder **d**, sonst Fehler; folgt also **\$** dem ersten **d**, wird Fehler erkannt)

Grammatik

1	$S' \rightarrow S$
2	$S \rightarrow CC$
3	$C \rightarrow cC$
4	$C \rightarrow d$

reguläre
Menge


c^*dc^*d

© Prof. J.C. Freytag, Ph.D.

10.32

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing



Idee einer Zustandsreduktion (Forts.)

betrachten noch einmal LR(1)-Elemente

$I_4: C \rightarrow d \cdot, \quad c/d$

$I_7: C \rightarrow d \cdot, \quad \$$

versuchen dabei
die Rollen der ähnlichen Zustände beim Parser zu erkennen

Zustand: I_7

- I_7 wird nach Lesen des zweiten d erreicht
- Reduktion $C \rightarrow d$
(Vor. nächstes Eingabezeichen ist $\$$, sonst Fehler)


Grammatik

1	S'	$\rightarrow S$
2	S	$\rightarrow CC$
3	C	$\rightarrow cC$
4		$ d$

↓ reguläre Menge

c^*dc^*d

© Prof. J.C. Freytag, Ph.D.
10.33



Idee einer Zustandsreduktion (Forts.)

ersetzen zwei Zustände (I_4 und I_7) durch einen Zustand I_{47}

$I_4: C \rightarrow d \cdot, \quad c/d$

$I_7: C \rightarrow d \cdot, \quad \$$

$I_{47}: C \rightarrow d \cdot, \quad c/d/\$$

Zustand: I_{47}

- in jedem Fall: Reduktion $C \rightarrow d$
(Vor. nächstes Eingabezeichen ist $\$, c$ oder d sonst Fehler)

Problem:
bei Fehlereingaben (wie ccd oder $cdcdc$) wird d zu C reduziert
Fehler wird nicht gleich, aber dennoch erkannt

Grammatik

1	S'	$\rightarrow S$
2	S	$\rightarrow CC$
3	C	$\rightarrow cC$
4		$ d$

↓ reguläre Menge

c^*dc^*d

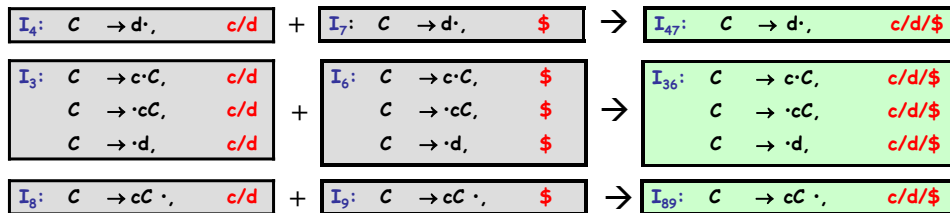
© Prof. J.C. Freytag, Ph.D.
10.34

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing

Verallgemeinerung der Idee zur Zustandsreduktion

Suchen nach LR(1)-Elementen mit **gleichem Kern**
(erste Komponenten sind gleich)



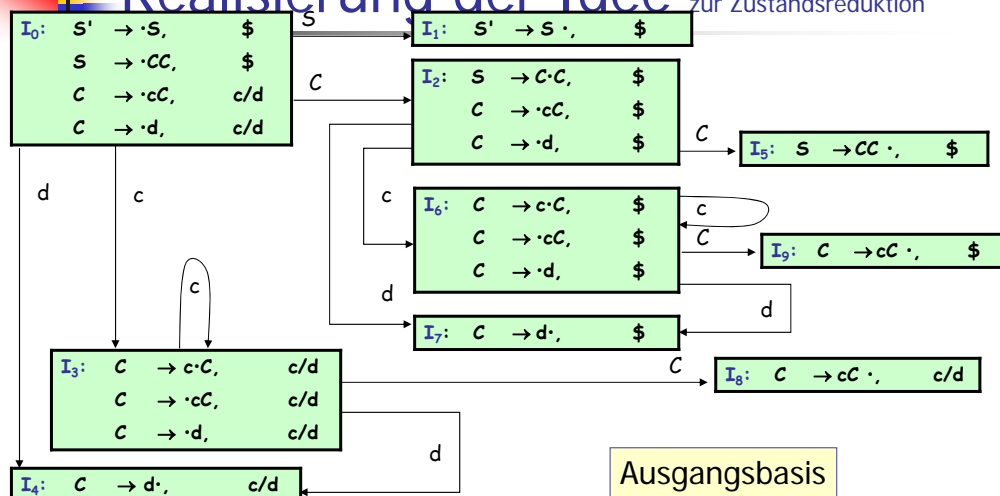
Bem.: im Beispiel haben (nur) Elementpaare einen gemeinsamen Kern
(i.allg. sind mehr Elemente möglich)

Feststellung: Der Kern von $\text{goto1}(I, X)$ hängt nur von I ab, nicht von X

© Prof. J.C. Freytag, Ph.D.

10.35

Realisierung der Idee zur Zustandsreduktion



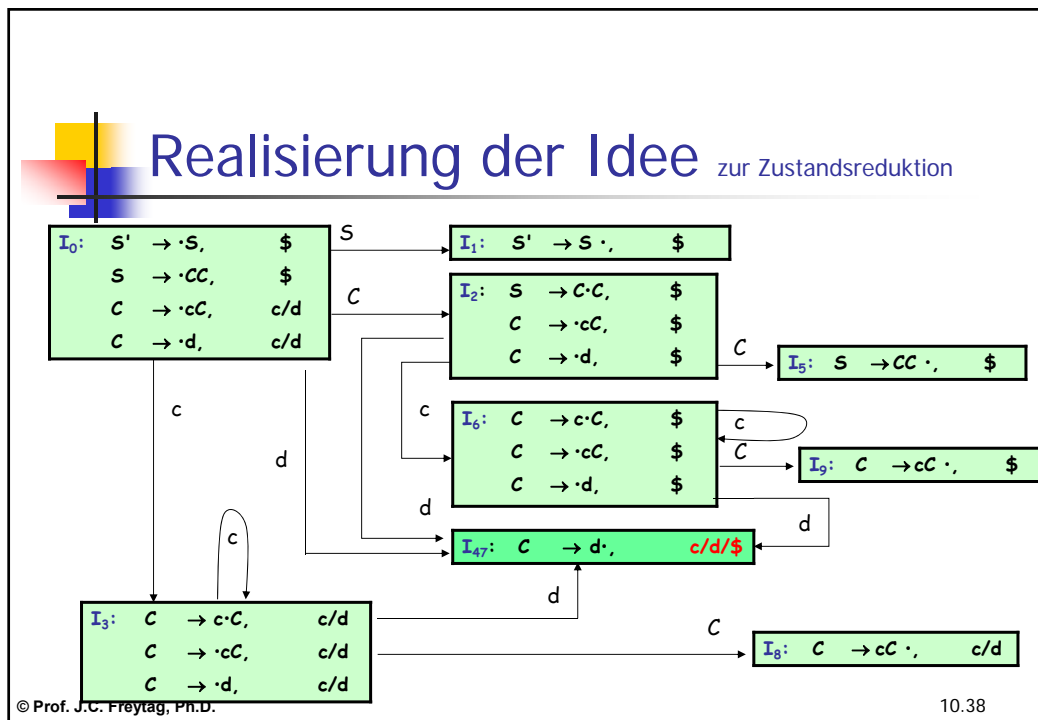
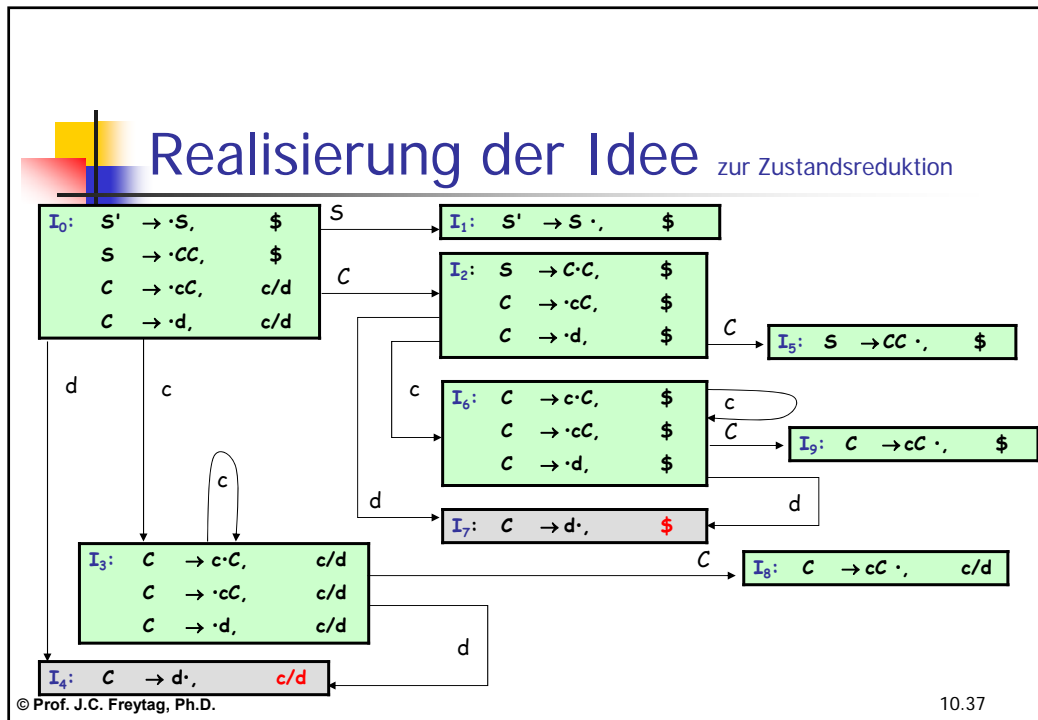
Ausgangsbasis

© Prof. J.C. Freytag, Ph.D.

10.36

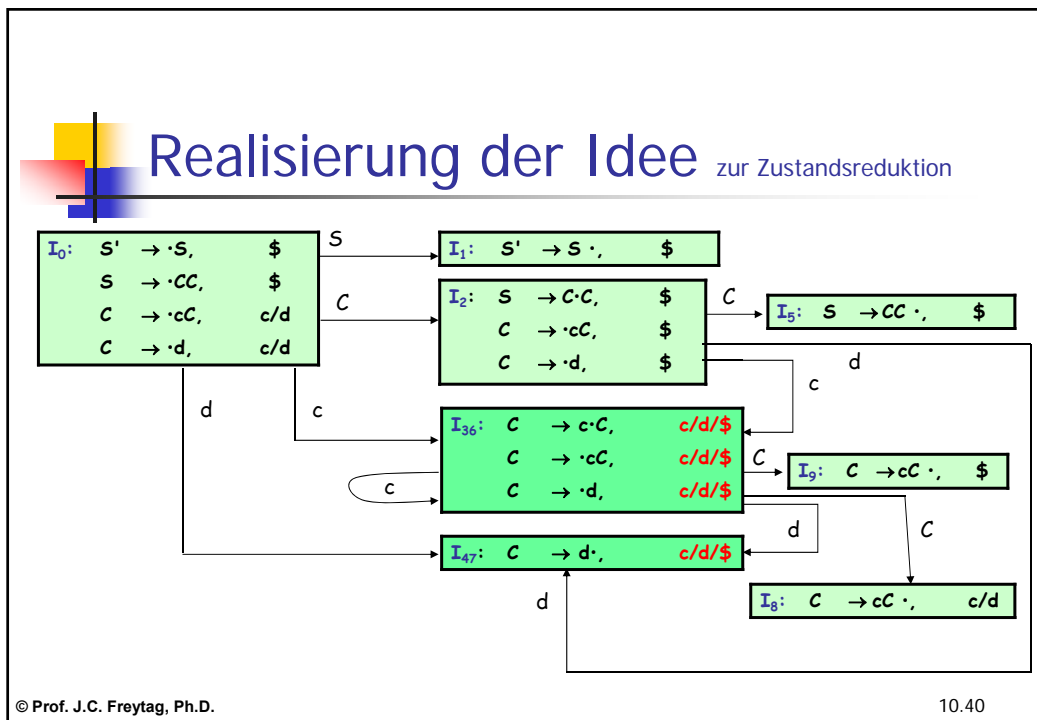
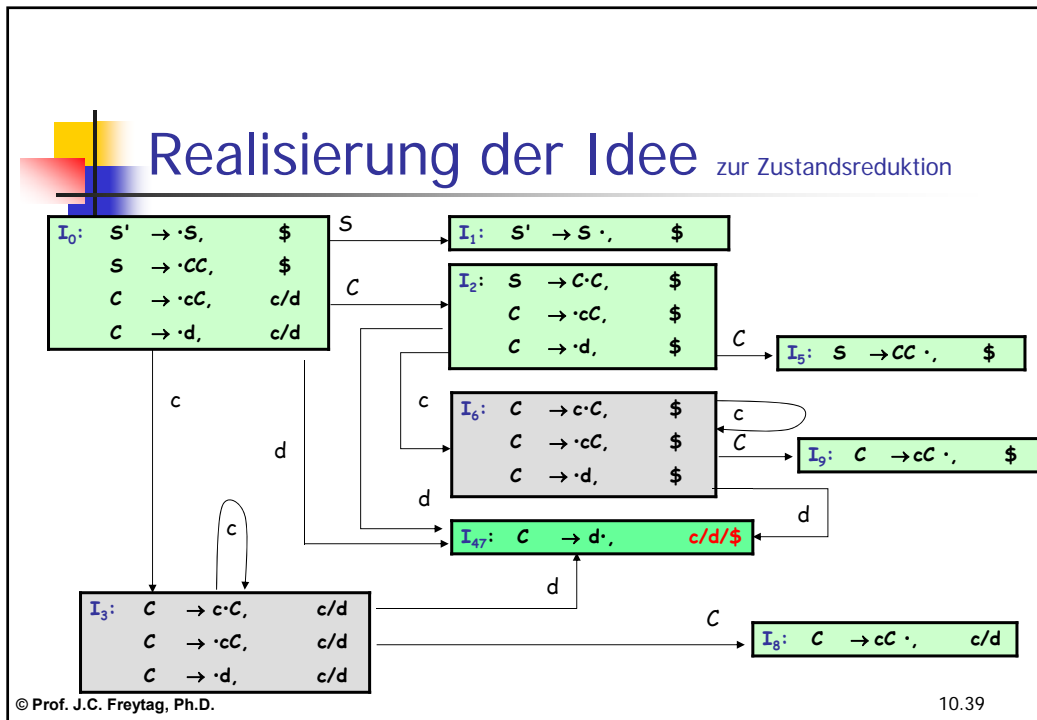
Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing



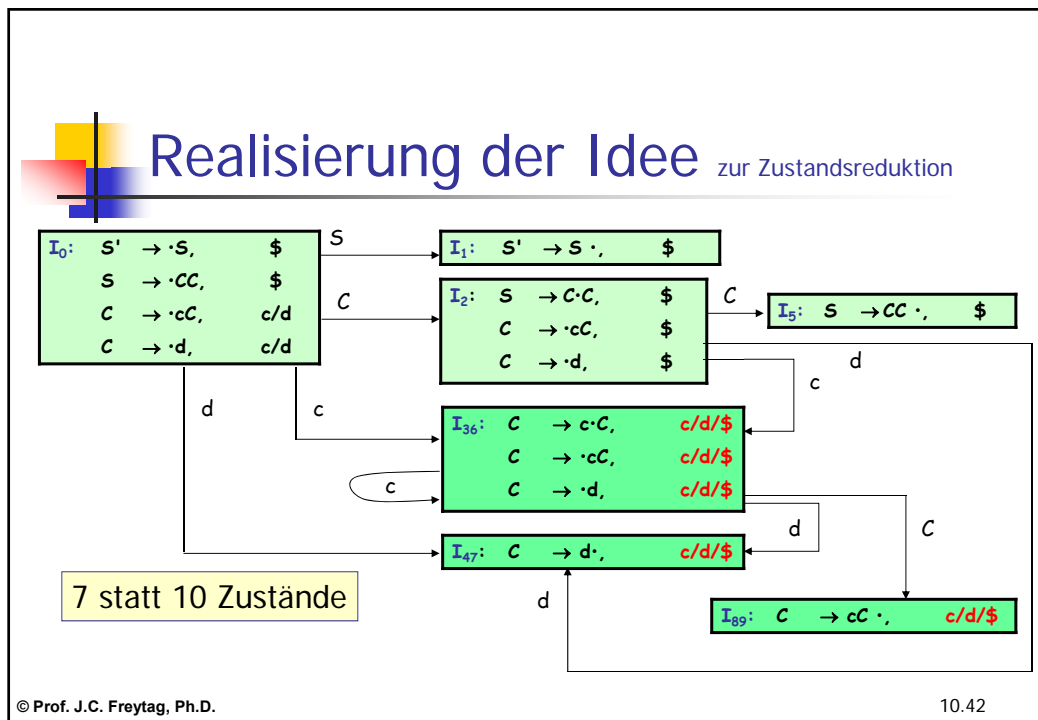
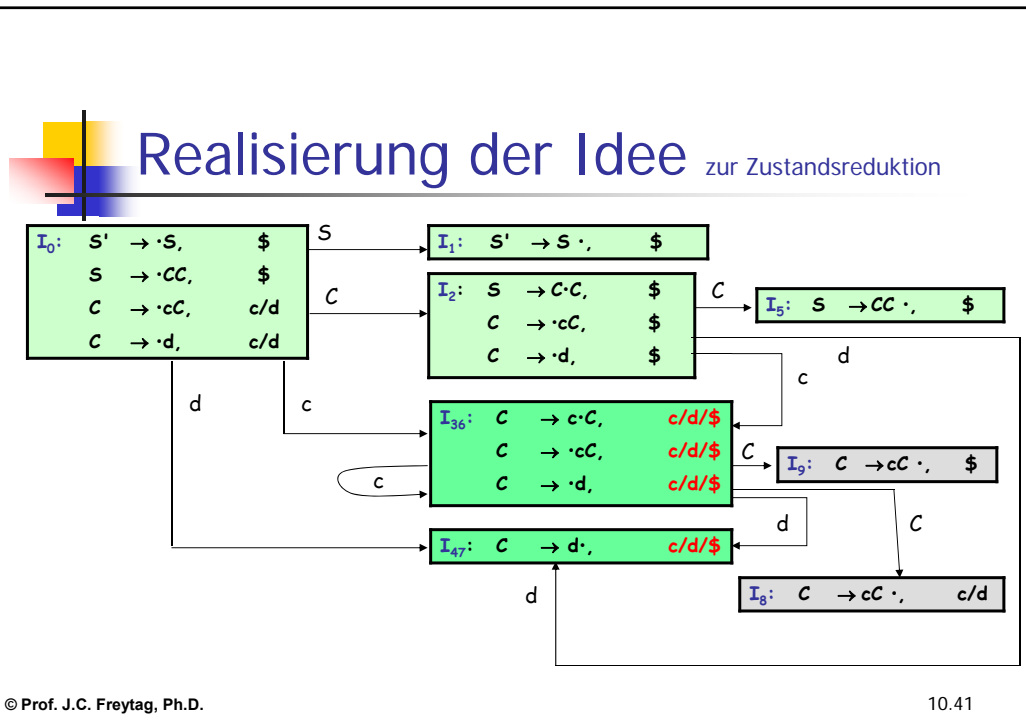
Vorlesung Compilerbau (SoSe 2018)


Teil 10: LR(k)-Parsing



Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing





LALR(1)- Syntaxanalyse

Definition

Der **Kern** (engl. core) einer Menge von LR(1)-Elementen ist die Menge der LR(0)-Elemente, die dadurch abgeleitet werden, dass die Lookahead-Symbole ignoriert werden

Beispiel:

die beiden Mengen

- $\{[A \rightarrow \alpha \cdot \beta, a], [A \rightarrow \alpha \cdot \beta, b]\}$ und
- $\{[A \rightarrow \alpha \cdot \beta, c], [A \rightarrow \alpha \cdot \beta, d]\}$

haben denselben Kern

Schlüsselidee:

Falls zwei Mengen von LR(1)-Elementen I_i und I_k denselben Kern haben, dann werden die Zustände, die sie repräsentieren, in der ACTION- und GOTO-Tabelle zusammengelegt



LALR(1)- Syntaxanalysetabelle

- Um die LALR(1)-Syntaxanalysetabelle zu konstruieren, braucht nur ein einziger zusätzlicher Schritt in den LR(1)-Algorithmus eingefügt zu werden
 - (1) für jeden Kern, der unter den Mengen an LR(1)-Elementen existiert, finde alle Mengen, die diesen Kern besitzen und ersetze diese Mengen durch ihre Vereinigung
 - (2) die **goto1**-Funktion muss verändert werden, um die neue Zustandsmenge zu reflektieren

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing



LALR(1)- Syntaxanalysetabelle (Forts.)

geänderter Algorithmus

- (1) Konstruiere die **LR(1)**-Elementmengen von **G'**
- (2) Für **jeden Kern**, der unter den Mengen der LR(1)-Items existiert, finde alle Menge, die denselben Kern haben, und ersetze diese Mengen durch ihre **Vereinigung** (Verändere die **goto1**-Funktion inkrementell)
- (3) **Zustand i** des LALR(1)-Automaten wird von I_i wie folgt konstruiert:
 - (a) $[A \rightarrow \alpha \cdot a\beta, b] \in I_i$ und $\text{goto1}(I_i, a) = I_j$ dann **ACTION** $[i, a] \leftarrow \text{shift } j$
 - (b) $[A \rightarrow \alpha \cdot, a] \in I_i$ und $A \neq S'$, dann **ACTION** $[i, a] \leftarrow \text{reduce } A \rightarrow \alpha$
 - (c) $[S' \rightarrow S \cdot, \$] \in I_i$ dann **ACTION** $[i, \$] \leftarrow \text{accept}$

© Prof. J.C. Freytag, Ph.D.

10.45



LALR(1)- Syntaxanalysetabelle (Forts.)

geänderter Algorithmus

...

- (4) Sei J die Vereinigung einer oder mehrere Mengen von LR(1)-Elementen:
$$J = I_1 \cup I_2 \cup \dots \cup I_k,$$
 - dann sind die Kerne von $\text{goto1}(I_1, X)$, $\text{goto1}(I_2, X)$, ..., $\text{goto1}(I_k, X)$ gleich
 - sei nun K die Vereinigung aller Elementmengen, die den gleichen Kern wie $\text{goto1}(I_1, X)$ haben, dann ist **goto1** $[J, X] \leftarrow K$
- (5) Setze undefinierte Einträge in **ACTION** und **GOTO** auf error
- (6) **Anfangszustand** des Parsers s_0 ist $\text{closure1}([S' \rightarrow \cdot S, \$])$

Problem: Der Algorithmus erfordert viel Speicherplatz

© Prof. J.C. Freytag, Ph.D.

10.46

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing

Zusammenfassung

- Tabelle heißt LALR-Syntaxtabelle für G
- gibt es keine Syntaxanalysekonflikte, dann ist die gegebene Grammatik G eine LALR(1)-Grammatik
- LR- und LALR-Parser verhalten sich bei fehlerfreien Eingaben gleich
- bei fehlerhaften Eingaben kann der LALR-Parser noch weitere Reduktionen ausführen, nachdem der LR-Parser bereits einen Fehler erkannt hat
ABER: er kann niemals schieben, nachdem der LR-Parser einen Fehler erkannt hat.

© Prof. J.C. Freytag, Ph.D.

10.47

Beispiel wie zuvor ...

Grammatik

1	$S' \rightarrow S$
2	$S \rightarrow CC$
3	$C \rightarrow cC$
4	$C \rightarrow d$

vereinige
folgende Zustände

$I_0:$	$S' \rightarrow \cdot S, \$$
	$S \rightarrow \cdot CC, \$$
	$C \rightarrow \cdot cC, cd$
	$C \rightarrow \cdot d, cd$
$I_1:$	$S' \rightarrow S \cdot, \$$
$I_2:$	$S \rightarrow C \cdot C, \$$
	$C \rightarrow \cdot cC, \$$
	$C \rightarrow \cdot d, \$$

$I_3:$	$C \rightarrow c \cdot C, cd$
	$C \rightarrow \cdot cC, cd$
	$C \rightarrow \cdot d, cd$
$I_4:$	$C \rightarrow d \cdot, cd$
$I_5:$	$S \rightarrow CC \cdot, \$$
$I_6:$	$C \rightarrow c \cdot C, \$$
	$C \rightarrow \cdot cC, \$$
	$C \rightarrow \cdot d, \$$
$I_7:$	$C \rightarrow d \cdot, \$$
$I_8:$	$C \rightarrow cC \cdot, cd$
$I_9:$	$C \rightarrow cC \cdot, \$$

$I_{36}:$	$C \rightarrow c \cdot C, cd\$$
	$C \rightarrow \cdot cC, cd\$$
	$C \rightarrow \cdot d, cd\$$
$I_{47}:$	$C \rightarrow d \cdot, cd\$$
$I_{89}:$	$C \rightarrow cC \cdot, cd\$$

Zust.	ACTION			GOTO	
	c	d	\$	S	C
0	s36	s47	-	1	2
1	-	-	acc	-	-
2	s36	s47	-	-	5
36	s36	s47	-	-	89
47	r4	r4	r4	-	-
5	-	-	r2	-	-
89	r3	r3	r3	-	-

© Prof. J.C. Freytag, Ph.D.

10.48



Syntaxanalysekonflikte

- **Annahme:** unsere Ausgangsgrammatik sei vom Typ LR(1) (d.h., die kanonische LR(1)-Analyse verursacht keine Syntaxanalysekonflikte)
- **Frage:** können Syntaxanalysekonflikte nach Zustandsreduktion bei einer LALR-Analyse entstehen?
- **Antwort:** Shift-Reduce-Konflikte sind nicht möglich, **aber** Reduce-Reduce-Konflikte



Shift-Reduce

- **Ann.:** Shift-Reduce tritt für Lookahead **a** in der Vereinigung auf d.h., es gibt zwei Elemente
 - $[A \rightarrow \alpha \bullet, a]$ ruft **Reduktion** für $A \rightarrow \alpha$
 - $[B \rightarrow \beta \bullet a \gamma, b]$ ruft **Schieben**
- dann muss es separate Zustände gegeben haben
 - $[A \rightarrow \alpha \bullet, a]$ muss dann zu einer Menge I gehört haben
 - da alle Kerne für die Vereinigung gleich sein mussten, muss die Menge I auch ein Element $[B \rightarrow \beta \bullet a \gamma, c]$ für irgend ein c enthalten haben
 - dann hat aber dieser Zustand bereits ein Shift-Reduce-Konflikt bei a (Widerspruch zur Annahme Grammatik sei LR(1))

Fazit:

- Schiebe-Aktionen sind nur vom Kern abhängig (nicht vom Lookahead)
- Zustandsvereinigung bringt keine Shift-Reduce-Konflikte hervor

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing

Reduce-Reduce

LR(1)- Grammatik

1	$S' \rightarrow S$
2 3 4 5	$S \rightarrow a A d \mid b B d \mid a B e \mid b A e$
6	$A \rightarrow c$
7	$B \rightarrow c$

generiert die Zeichenketten:

acd
ace
bcd
bce

gültige Vorsilbe **ac**: Zustand $I_x = \{[A \rightarrow c\bullet, d], [B \rightarrow c\bullet, e]\}$

gültige Vorsilbe **bc**: Zustand $I_y = \{[A \rightarrow c\bullet, e], [B \rightarrow c\bullet, d]\}$

→ keine der Mengen ruft Konflikt hervor, auch ihre Kerne sind gleich

→ Vereinigung

I_{xy} :	$A \rightarrow c\bullet$	e/d
	$B \rightarrow c\bullet$	e/d

→ Konflikt: Reduktionen sind sowohl mit $A \rightarrow c$ als auch mit $B \rightarrow c$ möglich

© Prof. J.C. Freytag, Ph.D.

10.51

Effizientere Konstruktionen von LALR(1)-Syntaxanalysetabellen

Frage: kann der Aufbau der vollständigen kanonischen Sammlung an Mengen von LR(1)-Elementen vermieden werden?

Antwort: ja, bei Ausnutzung folgender Feststellungen

- Repräsentiere I_i durch seinen Kern:
Item-Mengen werden entweder durch das Startelement $[S' \rightarrow \bullet S, \$]$ dargestellt oder haben keinen " \bullet " links einer RS
- Berechnung von shift-, reduce- und goto-Aktionen für Zustände, kann für I_i direkt aus dem Kern abgeleitet werden

© Prof. J.C. Freytag, Ph.D.

10.52

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing



Verdichtung von LR-Analysetabellen

- typische Programmiersprachen (50..100 Terminale, 100 Produktionen)
 - ➔ LALR
 - 100 Zustände
 - goto1 20.000 Einträge (8 Bit pro Eintrag)
 - effiziente Kodierung 2-dimensionaler Felder
- Verwendung von Zeigern bei identischen Zeilen
(Zeiger für Zustände mit den gleichen Aktionen zeigen auf den gleichen Platz)

© Prof. J.C. Freytag, Ph.D.

10.53



Fehlererholung in SR-Parsern

- Folgende Probleme können auftreten:
 - Behandlung eines nicht-erlaubten Tokens
 - inkorrektter Baum, der im Keller gespeichert ist
 - inkorrekte Einträge in der Symboltabelle
- Ziel: Rest der Eingabe soll »geparst« werden
- Wiederanlauf (Restart) der Parsers:
 - finde einen Zustand im Keller, von dem aus die Korrektheit der Restes der Eingabe weiter überprüft (reduziert) werden kann
 - rücke auf einen konsistenten Punkt in der Eingabe vor
 - schreibe informative (Fehler-) Nachrichten aus

© Prof. J.C. Freytag, Ph.D.

10.54

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing

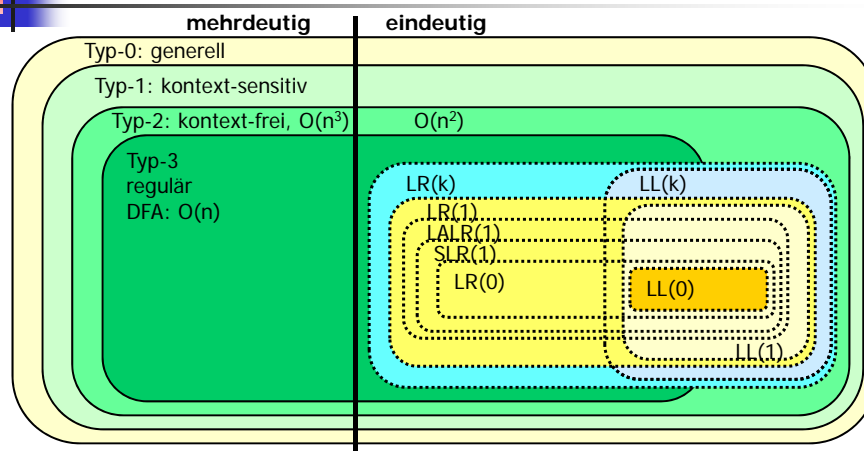
Zusammenfassung Parsing

- Rekursiv absteigend
 - Ein handkodierter rekursiv absteigender Parser kodiert eine Grammatik (typischerweise eine LL(1)-Grammatik) durch eine Folge von Prozeduren, die sich gegenseitig aufrufen.
- LL(k)
 - Ein LL(k)-Parser muss in der Lage sein, die Anwendung einer Regel zu erkennen, nachdem er nur die ersten k Symbole der rechten Seite einer Regel erkannt hat
- LR(k)
 - Ein LR(k)-Parser muss in der Lage sein, die rechte Seite einer Regel zu erkennen, nachdem er alles gesehen hat, was von dieser rechten Seite abgeleitet werden kann mit einem Lookahead von k Symbolen

© Prof. J.C. Freytag, Ph.D.

10.55

Komplexität: Hierarchie



Achtung: dieses Bild ist eine Hierarchie von Grammatiken und nicht von Sprachen

© Prof. J.C. Freytag, Ph.D.

10.56

Vorlesung Compilerbau (SoSe 2018)

Teil 10: LR(k)-Parsing

Sprache - Grammatik

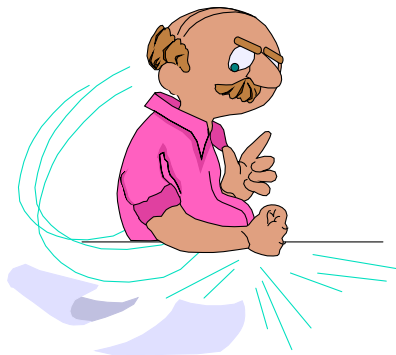
Beobachtung:

- Jede reguläre Sprache hat eine Grammatik, die LL(1) ist, aber nicht alle regulären Grammatiken dieser Sprache sind vom Typ LL(1)
 - Beispiel: $S \rightarrow ab$, $S \rightarrow ac$
 - Ohne Linksfaktorisierung ist diese Grammatik nicht LL(1)
- $S \rightarrow aX$, $X \rightarrow b$, $X \rightarrow c$ (diese ist LL(1))

© Prof. J.C. Freytag, Ph.D.

10.57

Fragen???



© Prof. J.C. Freytag, Ph.D.

10.58