Syntaktische Analyse – Teil 3

Übersicht

- Bottom-up-Syntaxanalyse
- LR(k)-Analysatoren
  - Charakteristischer endlicher Automat char(K_G)
  - Item-Kellerautomat K_G und char(K_G)
  - Zuverlässige Präfixe, gültige Items
- LR-DEA(G)
- Kellerautomat K_0
- LR(k)
  - Definition, Eigenschaften, Beispiele
  - LR(k)-Items
  - Alternative Definition für LR(k)
- LR(k)-Parser
- Kanonische LR(1)-Zustandsmengen
- Andere LR-Verfahren
  - SLR(1)
  - LALR(1)
- Fehlerbehandlung in LR-Parsern

Lernziele

- Das Grundprinzip der Bottom-Up-Syntaxanalyse erklären können
- Grundidee und wichtigste theoretische Aspekte von LR(k) benennen können
- Beschreiben können, wie ein LR(k)-Parser im Prinzip arbeitet
- Verschiedene Verfahren zur Berechnung von LR(1)-Zustandsmengen und ihre Zusammenhänge gegenüberstellen können
- Die Grundideen der Fehlerbehandlung in LR-Parsern beschreiben können

© H. Partsch 2014

Bottom-up-Syntaxanalyse (1/2)

Bottom-up-Parser

- Konstruiert Syntaxbaum von den Blättern her ——— *Ist Rechtsparser*
- Anfangssituation

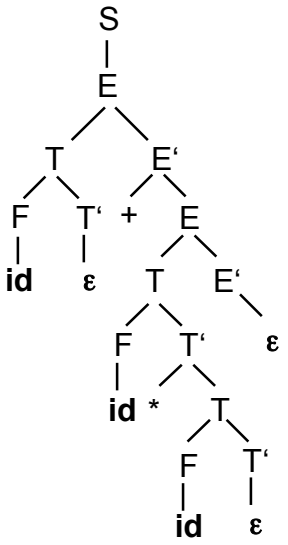
Keller	Resteingabe
ε	**id + id * id**

- Aktivitäten („shift-reduce-Parser“)
  - Terminale lesen = Kellern des nächsten Eingabesymbols („shift“)
  - Am Kellerende Produktionen „von rechts nach links“ anwenden („reduce“)

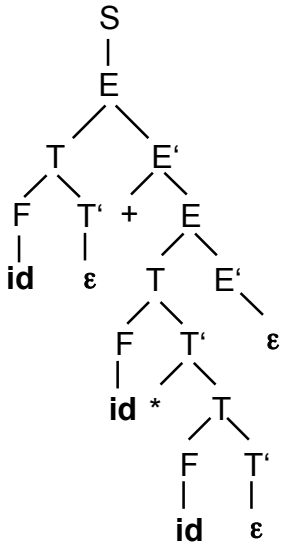
Beispiel (Rechtsableitung und shift-reduce-Verhalten)

- Geg.: kfG G_2 = ({S,E,T,F}, {+, *,(,),**id**}, P, S) mit
  P = {S → E, E → TE‘, E‘ → ε, E‘ → +E, T → FT‘,
  T‘ → ε, T‘ → *T, F → (E), F → **id**}

Eingabewort **id + id * id**



reduktive Sicht



Rechtsableitung

© H. Partsch 2014

Bottom-up-Syntaxanalyse (2/2)

Schwierigkeiten

- Eine Reduktion die gemacht werden muss (*handle* ☛), um eine Rechtsableitung zu erhalten, darf nicht verpasst werden (d.h. im Keller durch ein gelesenes Symbol „zugedeckt“ werden)
- Aber: Nicht jede rechte Seite einer Produktion am oberen Kellerende ist ein „handle“ (⇒ mögliche Sackgassen bei der Analyse)

Beispiel

- Sei G_0‘ = ({S,E,T,F}, {+,*,(,),**id**}, {S → E, E → E+T | T, T →T*F | F, F→ (E) | **id**}, S)
- Bottom-up-Analyse des Satzes **id** * **id**

Keller	Eingabe	Kommentar
	id * **id**	
id	* **id**	
F	* **id**	
T	* **id**	Reduktion von T zu E führte in Sackgasse
T *	**id**	
T * **id**		
T * F		Reduktion von F zu T führte in Sackgasse
T		
E		
S		akzeptiert

© H. Partsch 2014

LR(k)-Analysatoren

Grundlage

- Item-Kellerautomat K_G
- Bei LL(k)
  - Durch Vorausschau: Eindeutige Auswahl von Alternativen bei Expansionsübergängen
- Bei LR(k)
  - Durch Vorausschau: Entscheidung über Weiterlesen oder Reduzieren
    - L: von Links nach rechts lesen
    - R: Rechtsparser
    - k: Länge der Vorausschau

Ziel
Schrittweiser Übergang zu deterministischem Kellerautomaten, der bottom-up arbeitet

Charakteristischer endlicher Automat char(K_G)

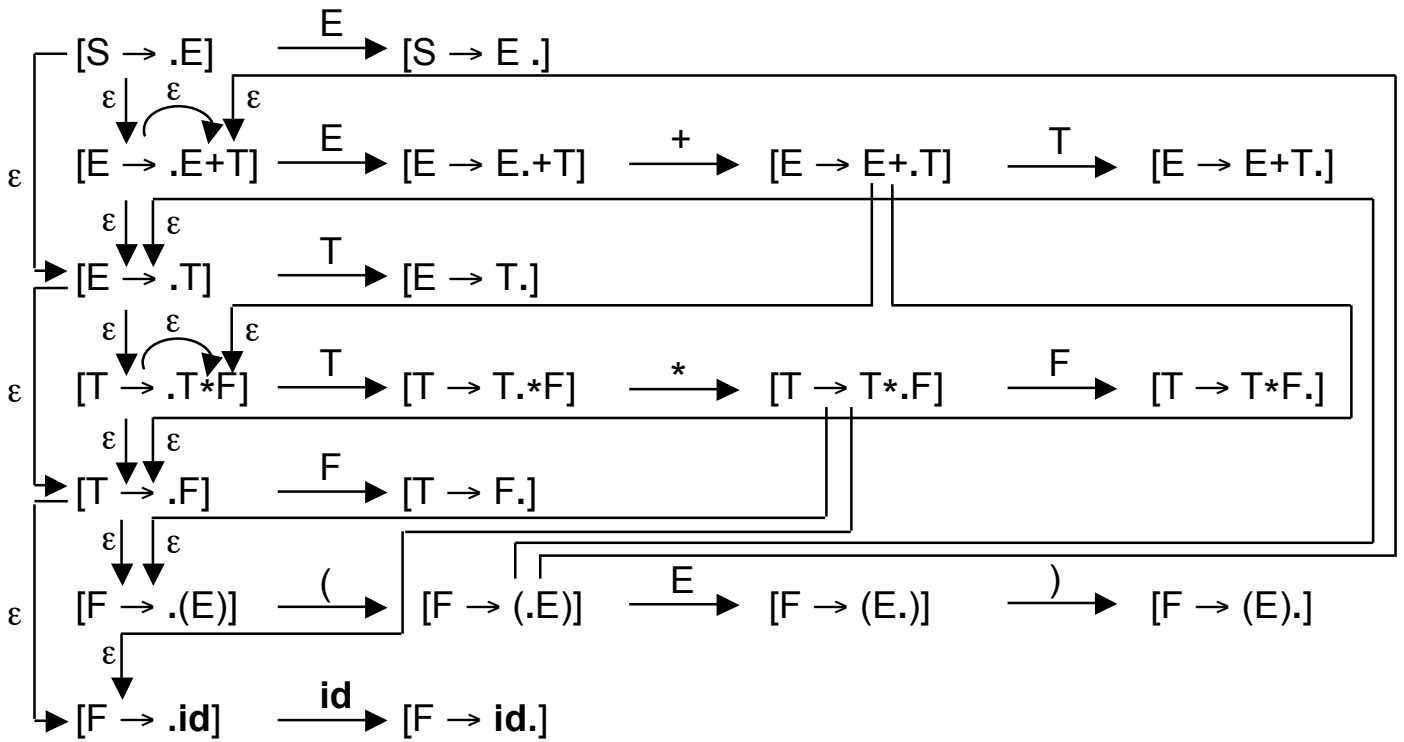
- Alternative Darstellung des Item-Kellerautomaten K_G für eine kfG G
- **char(K_G)** = $(V_T \cup V_N, It_G, \Delta_c, [S' \rightarrow .S], \{[X \rightarrow \alpha.] \mid X \rightarrow \alpha \in P\})$, wobei
  - $\Delta_c =_{def} \{([X \rightarrow \alpha.Y\beta], Y, [X \rightarrow \alpha Y.\beta]) \mid X \rightarrow \alpha Y\beta \in P \wedge Y \in V_T \cup V_N\} \cup$
    $\{([X \rightarrow \alpha.Y\beta], \varepsilon, [Y \rightarrow .\gamma]) \mid X \rightarrow \alpha Y\beta \in P \wedge Y \rightarrow \gamma \in P\}$
  - D.h. Δ_c hat
    - Übergänge innerhalb einer Produktion (unter den Symbolen aus $V_T \cup V_N$)
    - Übergänge, die den (E)-Übergängen von K_G entsprechen

© H. Partsch 2014

Charakteristischer endlicher Automat char(K_G)

Beispiel

- Sei G_0‘ = ({S,E,T,F}, {+,*,(,),**id**}, {S → E, E → E+T | T, T →T*F | F, F→ (E) | **id**}, S)
- Charakteristischer endlicher Automat char($K_{G_0‘}$)



© H. Partsch 2014

Item-Kellerautomat K_G und char(K_G)

Korrespondenz (zwischen Item-Kellerautomat K_G und char(K_G))

- Item-Kellerautomat K_G — Charakteristischer endlicher Automat char(K_G)
  - Expansion — ε-Übergang
  - Lesen — Lese-Übergang unter Terminalsymbol
  - Reduktion — Lese-Übergang unter Nichtterminalsymbol

Wichtige Eigenschaft

- Betrachtet man alle Zustände als Endzustände, ist die von char(K_G) akzeptierte Sprache für LR-Analyse interessant („reguläre Sprache der *zuverlässigen Präfixe* ☛“)

© H. Partsch 2014

Zuverlässige Präfixe, gültige Items (1/2)

Begriffe

- Sei $S \Rightarrow^*_{rm} \gamma X w \Rightarrow_{rm} \gamma\beta\alpha w$ eine Rechtsableitung zu einer kfG G, $[X \rightarrow \beta.\alpha]$ Item von G
  - **Griff** (der Rechtssatzform $\gamma\beta\alpha w$): $\beta\alpha$
  - **Zuverlässiges Präfix** (von G): jedes Präfix von $\gamma\beta\alpha$
  - Item $[X \rightarrow \beta.\alpha]$ ist **gültig** für zuverlässiges Präfix $\gamma\beta \Leftrightarrow \exists$ Rechtsableitung $S \Rightarrow^*_{rm} \gamma X w \Rightarrow_{rm} \gamma\beta\alpha w$
- Intuition
  - *Griff* (in einer nicht mehrdeutigen Grammatik): eindeutig bestimmtes Teilwort, das bei bottom-up-Analyse beim nächsten Reduktionsschritt durch Nichtterminal zu ersetzen ist
  - *Zuverlässiges Präfix*: Anfang einer Rechtssatzform; Reduktion (höchstens) am Ende möglich
  - *Gültiges Item* $[X \rightarrow \beta.\alpha]$ für zuverlässiges Präfix $\gamma\beta$:
    zugehörige Produktion ist Kandidat für spätere Reduktionen

© H. Partsch 2014

Zuverlässige Präfixe, gültige Items (2/2)

Beispiele

- Sei G_0' = ({S,E,T,F}, {+,*,(,),**id**}, {S → E, E → E+T | T, T →T*F | F, F→ (E) | **id**}, S)
- Es war
  - Für Rechtsableitung $S \Rightarrow^*_{rm} \gamma Xw \Rightarrow_{rm} \gamma\beta\alpha w$: Griff = $\beta\alpha$; zuverl. Präfixe = Präfixe von $\gamma\beta\alpha$
  - [$X \rightarrow \beta.\alpha$] gültig für zuverlässiges Präfix $\gamma\beta$ $\Leftrightarrow$ $\exists$ Rechtsableitung $S \Rightarrow^*_{rm} \gamma Xw \Rightarrow_{rm} \gamma\beta\alpha w$
- Rechtssatzform — Griff — zuverlässiges Präfix — Begründung
  - E + F — F — E, E+, E+F — $S \Rightarrow^*_{rm} E + T \Rightarrow_{rm} E + F$
  - T * **id** — **id** — T, T *, T * **id** — $S \Rightarrow^*_{rm} T * F \Rightarrow_{rm} T * \mathbf{id}$
- Zuverl. Präfix — gültige Items — Begründung — γ — β — α — X — w
  - E + — [E → E+.T] — $S \Rightarrow_{rm} E \Rightarrow_{rm} E + T$ — ε — E+ — T — E — ε
  - — [T →.F] — $S \Rightarrow^*_{rm} E + T \Rightarrow_{rm} E + F$ — E+ — ε — F — T — ε
  - — [F →.**id**] — $S \Rightarrow^*_{rm} E + F \Rightarrow_{rm} E + \mathbf{id}$ — E+ — ε — **id** — F — ε
  - (E + (— [F → (.E)] — $S \Rightarrow^*_{rm} (E+F) \Rightarrow_{rm} (E+(E))$ — (E+ — (— E) — F —)

Einige wichtige Fakten

- Zu jedem zuverlässigen Präfix gibt es mindestens ein gültiges Item
- char(K_G) akzeptiert $\gamma \in (V_T \cup V_N)^*$ in Zustand q $\Leftrightarrow$ γ ist zuverl. Präfix + q ist gültiges Item
- Die Sprache der zuverlässigen Präfixe einer kfG ist regulär

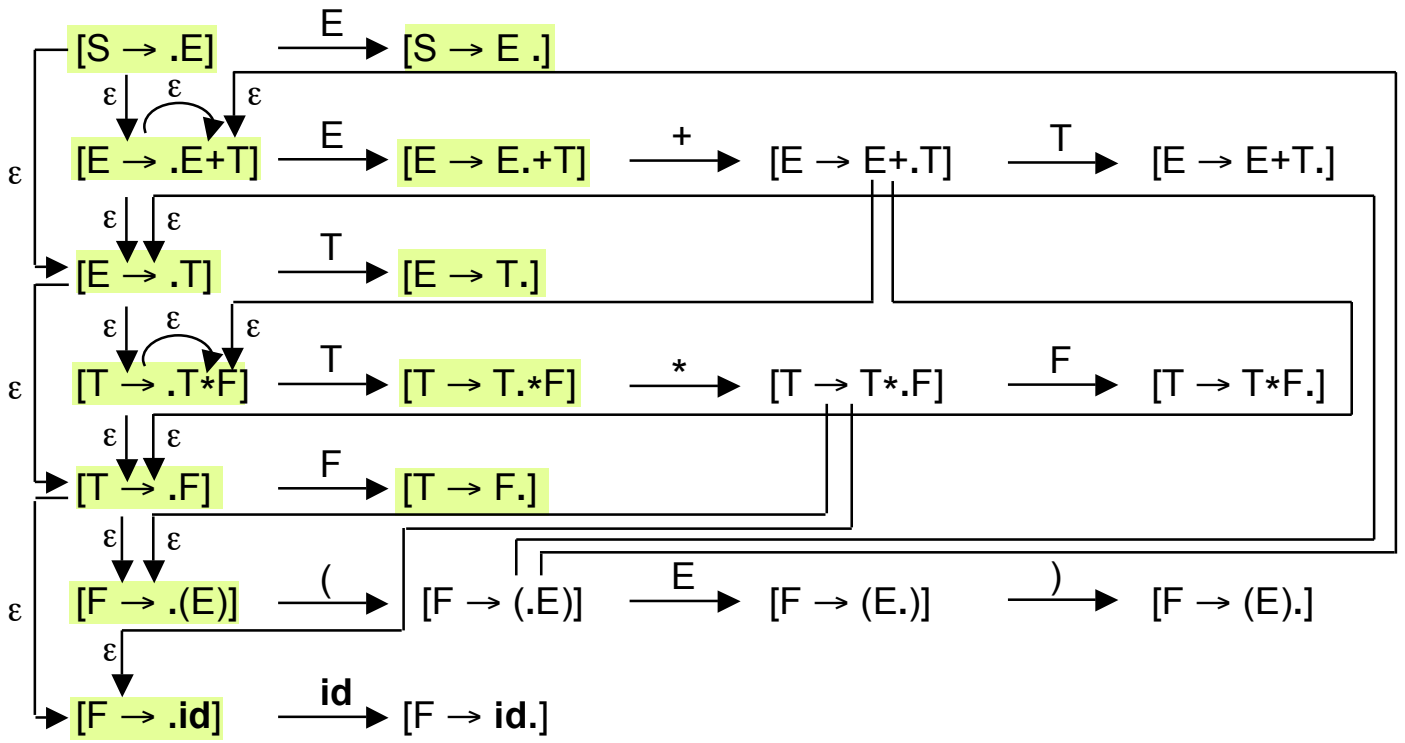
© H. Partsch 2014

LR-DEA(G) (1/3)

LR-DEA(G)

- Zu char(K_G) gehörender deterministischer endlicher Automat
  (Endzustände sind die Zustände, die vollständige Items enthalten)

Beispiel

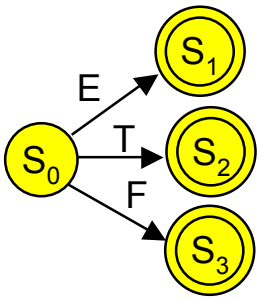


S_0 =
{[S → .E], [E → .E+T],
[E → .T], [T → .T*F],
[T → .F], [F → .(E)],
[F → **.id**]}

S_1 =
{[S → E.], [E → E.+T]}

S_2 =
{[E → T.], [T → T.*F]}

S_3 = {[T → F.]}



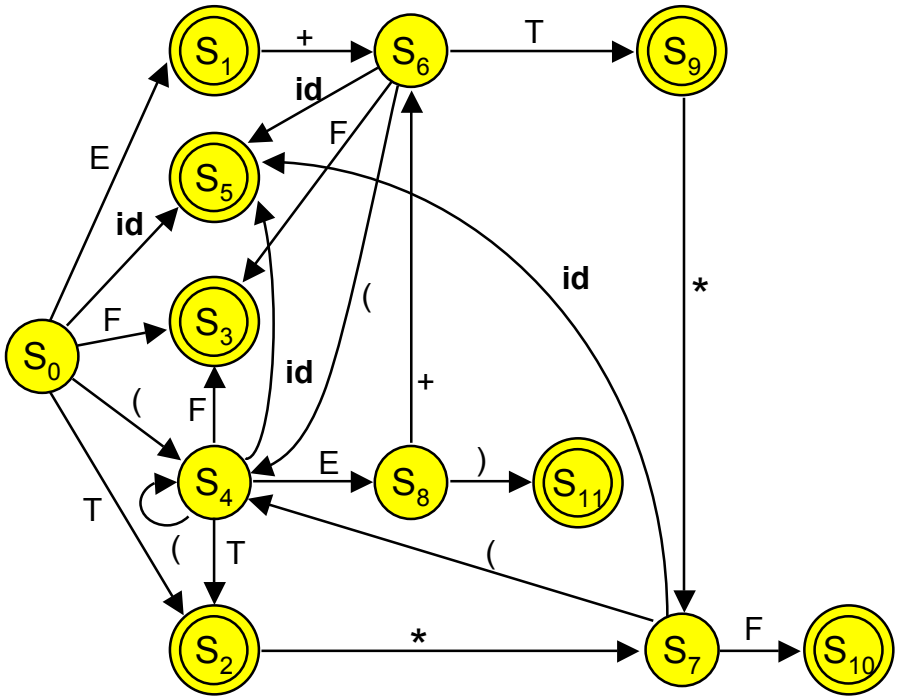
char(K_G)

Konstruktion von LR-DEA(G)

© H. Partsch 2014

LR-DEA(G) (2/3)

Vollständiger LR-DEA(G)



S_0 =
{[S → .E], [E → .E+T],
[E → .T], [T → .T*F],
[T → .F], [F → .(E)],
[F → **.id**]}

S_1 =
{[S → E.], [E → E.+T]}

S_2 =
{[E → T.], [T → T.*F]}

S_3 = {[T → F.]}

S_4 =
{[F → (.E)], [E → .E+T],
[E → .T], [T → .T*F],
[T → .F], [F → .(E)],
[F → **.id**]}

S_5 = {[F → **id.**]}

S_6 =
{[E → E+.T], [T → .T*F],
[T → .F], [F → .(E)],
[F → **.id**]}

S_7 =
{[T → T*.F], [F → .(E)],
[F → **.id**]}

S_8 =
{[F → (E.)], [E → E.+T]}

S_9 =
{[E → E+T.], [T → T.*F]}

S_{10} = {[T → T*F.]}

S_{11} = {[F → (E).]}

© H. Partsch 2014

LR-DEA(G) (3/3)

Direkte Konstruktion des LR-DEA(G)

Algorithmus LR-DEA

```
var q, q‘: set of item;
function Abschluss (s: set of item): set of item;
   begin
      q := s;
      while exist [X → α.Yβ] in q
               and Y → γ in P and Y → .γ not in q
         do q := q ∪ {[Y → .γ]} od;
      return(q)
   end;
function Nachf (s: set of item, Y: V_N ∪ V_T): set of item;
   return ({[X → αY.β] | [X → α.Yβ] ∈ s});
begin
   Q_d := Abschluss({[S →.S‘]}); δ_d := ∅;
   foreach q in Q_d and X in V_N ∪ V_T do
      let q‘ = Abschluss(Nachf(q, X)) in
      if q‘ ≠ ∅ then
         if q‘ not in Q_d then Q_d := Q_d ∪ {q‘} fi;
         δ_d := δ_d ∪ {q -X-> q‘} fi tel od
end
```

Bedeutung von **foreach**:
Alle möglichen Kombinationen (bis sich nichts mehr ändert)

Beispiel

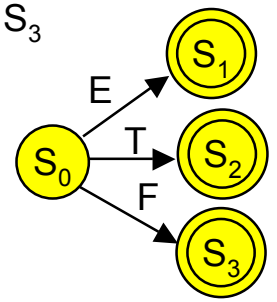
Geg.: {S → E, E → E+T | T, T →T∗F | F, F→ (E) | **id**}

Abschluss({[S → .E]}) = {[S → .E]}
∪ {[E → .E+T], [E → .T]}
∪ {[T → .T∗F], [T → .F]}
∪ {[F → .(E)], [F → **.id**]} = S_0

Nachf(S_0, E) = {[S → E.], [E → E.+T]} = S_1‘
Abschluss(S_1‘) = {[S → E.], [E → E.+T]} = S_1

Nachf(S_0, T) = {[T → T.∗F], [E → T.]} = S_2‘
Abschluss(S_2‘) = {[T → T.∗F], [E → T.]} = S_2

Nachf(S_0, F) = {[T → F.]} = S_3‘
Abschluss(S_3‘) = {[T → F.]} = S_3



© H. Partsch 2014

Eigenschaften des LR-DEA(G) (1/2)

Nach Konstruktion sind die Endzustände diejenigen Zustände die vollständige Items enthalten

Wichtige Eigenschaften

- LR-DEA(G) akzeptiert die Sprache der zuverlässigen Präfixe von G, die mit Griff enden
- LR-DEA(G) ist so konstruiert, dass
  - Die (i.a. unendliche) Menge der zuverlässigen Präfixe in endlich viele disjunkte Teilmengen zerlegt wird, die durch die Zustände von LR-DEA(G) repräsentiert sind
  - Der einem zuverlässigen Präfix γ zugeordnete Zustand aus allen für γ gültigen Items besteht, d.h. allen möglichen Beschreibungen der aktuellen Analysesituation

© H. Partsch 2014

Eigenschaften des LR-DEA(G) (2/2)

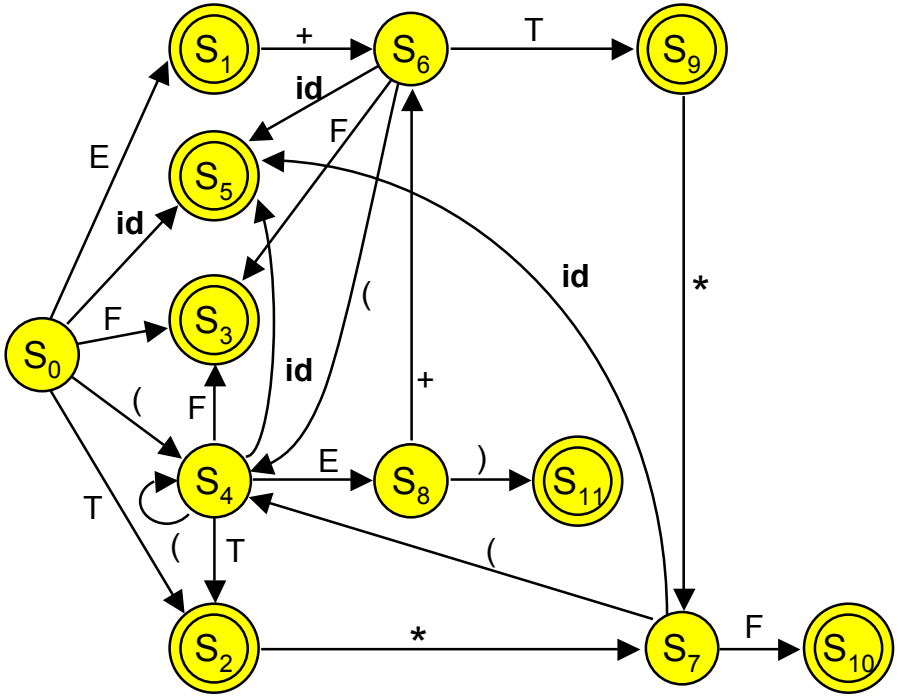
Weitere wichtige Eigenschaft

- LR-DEA(G) lässt sich auch als Beschreibung eines (nicht-deterministischen) Kellerautomaten $K_0 =_{def} (V_T, \Gamma, \Delta, q_0, \{q_f\})$ auffassen
  - Γ: Kelleralphabet = Menge der Zustände von LR-DEA(G)
  - q_0: Anfangszustand = q_d (= Anfangszustand von LR-DEA(G))
  - q_f: Endzustand = Zustand des LR-DEA(G), der das Item [S‘ → S.] enthält
  - $\Delta \subseteq \Gamma^* \times (V_T \cup \{\varepsilon\}) \times \Gamma^*$: Übergangsrelation, definiert durch
    - *Lesen*
      $(q, a, q\delta_d(q, a)) \in \Delta$, falls $\delta_d(q, a)$ definiert ist
      (nur möglich, wenn mindestens 1 Item der Form [X → ··· .a ···] in q)
    - *Reduzieren*
      $(qq_1 \ldots q_n, \varepsilon, q\delta_d(q, X)) \in \Delta$, falls $[X \rightarrow \alpha.] \in q_n$, $X \neq S'$ und $|\alpha| = n$
      $(q_0 q_f, \varepsilon, q_f) \in \Delta$

© H. Partsch 2014

Kellerautomat K_0

Beispiel



Analyse von id * id

S_0	id * id	(1)	$S_0 S_2 S_7 S_5$	ε	(17)
$S_0 S_5$	* id	(18)	$S_0 S_2 S_7 S_{10}$	ε	(19)
$S_0 S_3$	* id	(16)	$S_0 S_2$	ε	(14)
$S_0 S_2$	* id	(4)	$S_0 S_1$	ε	(22)
$S_0 S_2 S_7$	id	(9)	S_1		

Zugehöriger Kellerautomat K_0

$\Gamma = \{S_i \mid 0 \le i \le 11\}$
$q_0 = S_0$
$q_f = S_1$
Elemente der Übergangsrelation

Lesen
(1) $(S_0, \mathbf{id}, S_0S_5)$
(2) $(S_0, (, S_0S_4)$
(3) $(S_1, +, S_1S_6)$
(4) $(S_2, *, S_2S_7)$
(5) $(S_4, \mathbf{id}, S_4S_5)$
(6) $(S_4, (, S_4S_4)$
(7) $(S_6, \mathbf{id}, S_6S_5)$
(8) $(S_6, (, S_6S_4)$
(9) $(S_7, \mathbf{id}, S_7S_5)$
(10) $(S_7, (, S_7S_4)$
(11) $(S_8, +, S_8S_5)$
(12) $(S_8,), S_8S_{11})$
(13) $(S_9, *, S_9S_7)$

Reduzieren
(14) $(S_0 S_2, \varepsilon, S_0S_1)$
(15) $(S_4 S_2, \varepsilon, S_4S_8)$
(16) $(S_0 S_3, \varepsilon, S_0S_2)$, ...
(17) $(S_7 S_5, \varepsilon, S_7S_{10})$, ...
(18) $(S_0 S_5, \varepsilon, S_0S_3)$, ...
(19) $(S_0 S_2 S_7 S_{10}, \varepsilon, S_0S_2)$, ...
(20) $(S_0 S_1 S_6 S_9, \varepsilon, S_0S_1)$, ...
(21) $(S_0 S_4 S_8 S_{11}, \varepsilon, S_0S_3)$, ...
(22) $(S_0 S_1, \varepsilon, S_1)$

Probleme mit K_0

- Shift-reduce-Konflikt
- Reduce-reduce-Konflikt

© H. Partsch 2014

Konflikte

Begriffe

- **Shift-reduce-Konflikt**
  Ein Zustand erlaubt sowohl Lese- wie Reduktionsübergang
  (d.h. enthält vollständiges Item der Form $[X \rightarrow \alpha.]$, sowie Item der Form $[X \rightarrow \alpha.a\beta]$, $a \in V_T$)
- **Reduce-reduce-Konflikt**
  Ein Zustand erlaubt mehr als einen Reduktionsübergang
  (d.h. enthält mindestens 2 vollständige Items der Form $[X \rightarrow \alpha.]$, $[Y \rightarrow \beta.]$)
- **Ungeeignete Zustände** des LR-DEA
  Zustand q ist ungeeignet $\Leftrightarrow$ q enthält shift-reduce- oder reduce-reduce-Konflikt

Geeigneter Zustand enthält also
- entweder nur unvollständige Items
- oder genau ein (shift-Konflikt-freies) vollständiges Item

Beispiel

- Ungeeignete Zustände (shift-reduce-Konflikte)
  - $S_1 = \{[S \rightarrow E.], [E \rightarrow E.+T]\}$
  - $S_2 = \{[E \rightarrow T.], [T \rightarrow T.*F]\}$
  - $S_9 = \{[E \rightarrow E+T.], [T \rightarrow T.*F]\}$

Hier keine reduce-reduce-Konflikte, da jeder Zustand höchstens ein vollständiges Item enthält

© H. Partsch 2014

LR(k): Definition, Eigenschaften

Bedeutet für LR(0): in jeder Rechtssatzform sind Griff und zugehörige Produktion eindeutig festgelegt

Intuitiv

- Eine kfG heißt LR(k)-Grammatik, wenn in jeder Rechtssatzform jeder Rechtsableitung
  - Der Griff lokalisiert und
  - Die zugehörige Produktion durch Vorausschau auf (höchstens) k Symbole der Resteingabe eindeutig bestimmt werden kann

Formal

- Geg.: erweiterte kfG $G' = (V_N \cup \{S'\}, V_T, P \cup \{S' \rightarrow S\}, S')$ mit $S' \notin V_N$.
- G' heißt **LR(k)-Grammatik**, wenn
  - für zwei Rechtsableitungen
    $S' \Rightarrow^*_{rm} \alpha X w \Rightarrow_{rm} \alpha\beta w$
    $S' \Rightarrow^*_{rm} \gamma Y x \Rightarrow_{rm} \alpha\beta y$
  - mit $k : w = k : y$
  - gilt $\alpha = \gamma$ und $X = Y$ und $x = y$

© H. Partsch 2014

LR(k): Beispiele (1/2)

Beispiel

- Geg.: kfG G mit Produktionen {S → A | B, A → aAb | 0, B → aBbb | 1}
- Bekannt: G ist nicht LL(k) für beliebiges k
- Mögliche Rechtsableitungen und Rechtssatzformen
  - $S \Rightarrow_{rm} A \Rightarrow^*_{rm} a^{n-1}aAbb^{n-1} \Rightarrow_{rm} a^n0b^n \quad (n \geq 1)$
  - $S \Rightarrow_{rm} A \Rightarrow_{rm} 0$
  - $S \Rightarrow_{rm} B \Rightarrow^*_{rm} a^{n-1}aBbbb^{2n-1} \Rightarrow_{rm} a^n1b^{2n} \quad (n \geq 1)$
  - $S \Rightarrow_{rm} B \Rightarrow_{rm} 1$
- G ist LR(0): Griff und zugehörige Produktion sind jeweils eindeutig festgelegt

Beispiel

- Geg.: kfG G_6 mit Produktionen {S → aAc, A →Abb | b}

Ist nicht LL(k), da linksrekursiv

- Mögliche Rechtsableitungen und Rechtssatzformen
  - $S \Rightarrow^*_{rm} aAb^{2n}c \Rightarrow_{rm} aAbbb^{2n}c$
  - $S \Rightarrow^*_{rm} aAb^{2n}c \Rightarrow_{rm} abb^{2n}c$
- G_6 ist LR(0): Für die Präfixe aAbb und ab (der Rechtssatzformen $aAb^{2n+2}c$ bzw. $ab^{2n+1}c$) ist der Griff jeweils eindeutig bestimmt (nämlich Abb bzw. b)

© H. Partsch 2014

LR(k): Beispiele (2/2)

Weitere Beispiele

- kfG G_7 mit Produktionen {S → aAc, A → bbA | b}

  Rechtsrekursion ist kein Problem für LR

  - Mögliche Rechtsableitungen und Rechtssatzformen
    - $S \Rightarrow^*_{rm} ab^{2n}Ac \Rightarrow_{rm} ab^{2n}bbAc$
    - $S \Rightarrow^*_{rm} ab^{2n}Ac \Rightarrow_{rm} ab^{2n}bc$
  - G_7 ist LR(1): Kritische Rechtssatzform ist $ab^{2n+1}w$
    - Falls 1 : w = b, dann Griff in w
    - Falls 1 : w = c, dann Griff = b (letztes b in b^{2n+1})

- kfG G_8 mit Produktionen {S → aAc, A → bAb | b} ist nicht LR(k) für beliebiges k

  Übrigens auch nicht LL(k)

  Beweis: Sei k beliebig, aber fest gewählt. Für die Rechtsableitungen (mit n ≥ k)

  - $S \Rightarrow^*_{rm} ab^nAb^nc \Rightarrow_{rm} ab^nbb^nc$ (d.h. $S \Rightarrow^*_{rm} \alpha Xw \Rightarrow_{rm} \alpha\beta w$ mit $\alpha = ab^n$, X =A, β = b, w = b^nc)
  - $S \Rightarrow^*_{rm} ab^{n+1}Ab^{n+1}c \Rightarrow_{rm} ab^{n+1}bb^{n+1}c$
    (d.h. $S \Rightarrow^*_{rm} \gamma Yx \Rightarrow_{rm} \alpha\beta y$ mit $\gamma = ab^{n+1}$,Y = A, β = b, x = $b^{n+1}c$, y = $b^{n+1}c$)
  - gilt k: w = k: y = b^k,X = Y, x = y, aber $\alpha \neq \gamma$

LR(k)-Eigenschaft und LR-DEA

- Eine kfG G ist LR(0)-Grammatik ⇔ LR-DEA(G) hat keine ungeeigneten Zustände

© H. Partsch 2014

LR(k)-Items

LR(k)-Item

- Sei G‘ = (V_N, V_T, P ∪ {S‘ → S}, S‘) erweiterte kfG, X → αβ ∈ P
  - **LR(k)-Item**: [X → α.β, L], L ⊆ $V_{T\#}^{\leq k}$, mit
    - **Kern**: [X → α.β]
    - **Vorausschaumenge**: L

Mengen von Worten über V_T mit Länge ≤ k, evtl. durch # abgeschlossen

  - (für zuverlässiges Präfix γα) **gültiges LR(k)-Item** [X → α.β, L]:
    für alle u ∈ L gibt es Rechtsableitung S‘# $\Rightarrow^*_{rm}$ γXw $\Rightarrow_{rm}$ γαβw mit u = k: w

Beispiel

- Geg.: G_0‘ = ({S,E,T,F}, {+,*,(,),**id**}, {S → E, E → E+T | T, T →T*F | F, F→ (E) | **id**}, S)
- Es gilt
  - [E → E+.T, {), +}] ist gültiges LR(1)-Item für zuverlässiges Präfix (E+ in G_0‘
    - Für u =): S# $\Rightarrow^*_{rm}$ (E)# $\Rightarrow_{rm}$ (E+T)# mit γ = (, α = E+, β = T, w =)#
    - Für u = +: S# $\Rightarrow^*_{rm}$ (E+**id**)# $\Rightarrow_{rm}$ (E+T+**id**)# mit γ = (, α = E+, β = T , w = +**id**)#
  - [E → T., {*}] ist kein gültiges LR(1)-Item für irgend ein zuverlässiges Präfix
    - In keiner Rechtssatzform kann das Teilwort E* auftreten

© H. Partsch 2014

Alternative Definition für LR(k)

Alternative Definition für LR(k) (konstruktive Charakterisierung)

- Eine kfG G ist **LR(k)-Grammatik** $\Leftrightarrow$ für (beliebiges) zuverlässiges Präfix $\alpha\beta$ gilt:
  Ist LR(k)-Item $[X \rightarrow \beta., L_1]$ gültig für $\alpha\beta$, dann gibt es kein anderes für $\alpha\beta$ gültiges LR(k)-Item $[Y \rightarrow \beta_1.\beta_2, L_2]$, so dass $L_1 \cap FIRST_k(\beta_2 L_2) \neq \varnothing$
- Intuitiv: zu jedem zuverlässigen Präfix gibt es nur **ein** gültiges LR(k)-Item
- Auf LR-DEA(G) bezogen
  - LR(k)-Items jedes Zustands erfüllen obige Bedingung
  - k-Vorausschau beseitigt Konflikte

Beispiel

- Geg.: G_0' = ({S,E,T,F}, {+,*,(,),**id**}, {S $\rightarrow$ E, E $\rightarrow$ E+T | T, T $\rightarrow$T*F | F, F$\rightarrow$ (E) | **id**}, S)
- Für die 1-Vorausschaumengen des LR-DEA(G_0') muss gelten
  - S_1 = {[S $\rightarrow$ E., L_1], [E $\rightarrow$ E.+T, L_2]}: G_0' ist LR(1) $\Leftrightarrow$ + $\notin$ L_1
  - S_2 = {[E $\rightarrow$ T., L_1], [T $\rightarrow$ T.*F, L_2]}: G_0' ist LR(1) $\Leftrightarrow$ * $\notin$ L_1
  - S_9 = {[E $\rightarrow$ E+T., L_1], [T $\rightarrow$ T.*F, L_2]}: G_0' ist LR(1) $\Leftrightarrow$ * $\notin$ L_1

© H. Partsch 2014

LR(k)-Parser (1/3)

LR(k) bedeutet (intuitiv)

- Hat man Kandidaten für Reduktion gefunden, so kann man (mithilfe des zugehörigen zuverlässigen Präfix und der k nächsten Symbole) entscheiden
  - Ob Kandidat Griff ist und
  - Wenn ja, wozu er reduziert werden muss
- Mit anderen Worten: Menge der (genau einem Zustand des LR-DEA zugeordneten) zuverlässigen Präfixe ZP und Worte der Länge k bestimmen (eindeutig) Parser-Aktionen

Damit gilt

- Parser-Aktionen darstellbar durch Tabelle („Action-Tabelle“) mit Einträgen
  - shift — lies nächstes Eingabesymbol
  - reduce ($X \rightarrow \alpha$) — reduziere mittels Produktion $X \rightarrow \alpha$
  - error — melde Fehler
  - accept — melde erfolgreiches Ende des Parserlaufs

Zustände	k-Vorausschau-Symbole
	Parser-Aktionen

© H. Partsch 2014

LR(k)-Parser (2/3)

Darstellung der Übergangsfunktion des LR-DEA(G)

- In zweiter Tabelle („Goto-Tabelle“ = Übergangstabelle des LR-DEA(G))
- Wird bei shift und reduce konsultiert, um neuen Zustand zu berechnen
  - shift: Übergang aus aktuellem Zustand unter gelesenem Eingabesymbol
  - reduce($X \rightarrow \alpha$): Übergang unter X aus dem Zustand, der nach Entfernen der zur rechten Seite von $X \rightarrow \alpha$ gehörenden Zustände oben auf dem Keller liegt

Bestandteile eines LR(k)-Parsers (für LR(k)-Grammatik G)

- Action-Tabelle
- Goto-Tabelle
- (von G unabhängiges) Programm, das die Tabellen interpretiert

	$u \in V_{T\#}^{\leq k}$
$q \in Q$	Parser-Aktion für (q, u)

Action-Tabelle

	$X \in (V_N \cup V_T)$
$q \in Q$	$\delta_d(q, X)$

Goto-Tabelle

© H. Partsch 2014

LR(k)-Parser (3/3)

LR(k)-Parser-Hauptprogramm

Erste k Symbole lesen

Keller initialisieren

```
type state = set of item;
var   lookahead: sequ of symbol;
      (* die nächsten k lexikalisch analysierten, aber noch nicht konsumierten Eingabesymbole*)
      S: stack of state;
proc scan;
   (* analysiert ein weiteres Symbol lexikalisch; fügt es hinten an lookahead an *)
proc acc;
   (* meldet erfolgreiches Ende der syntaktischen Analyse; hält an *)
proc err (meldung: string);
   (* meldet Fehler; hält an *)
scan^k;
push(S, q_0);
do
   case action[top(S), lookahead] of
      shift: begin push(S, goto[top(S), hd(lookahead)]); lookahead := tl(lookahead); scan end
      reduce(X → α): begin pop^|α|(S); push(S, goto[top(S), X]); output(„X → α“) end;
      accept: acc;
      error: err(„...“)
   end case
od
```

© H. Partsch 2014

Konstruktion von LR(k)-Parsern

Drei verschiedene Verfahren (zur Bestimmung der Zustandsmengen)

- *Kanonisches LR(k)-Verfahren* ☛
- *SLR(k)-Verfahren* ☛
- *LALR(k)-Verfahren* ☛

Unterschiede

- Zahl der Zustände und/oder Größe der Vorausschaumengen

Im folgenden

- Praktisch relevanter Fall k = 1

  (ausreichend, da es zu jeder LR(k)-Grammatik eine äquivalente LR(1)-Grammatik gibt)

Ohne dabei allzu viel Struktur zu verlieren

Konstruktion eines LR(1)-Parsers

- Sei I eine Menge von LR(1)-Items. I hat
  - **shift-reduce-Konflikt**: I enthält Item $[X \rightarrow \alpha.a\beta, L_1]$, $a \in V_T$ und Item $[Y \rightarrow \gamma., L_2]$ mit $a \in L_2$
  - **reduce-reduce-Konflikt**: I enthält Items $[X \rightarrow \alpha., L_1]$ und $[Y \rightarrow \gamma., L_2]$ mit $L_1 \cap L_2 \neq \emptyset$
- **Ungeeigneter Zustand**: Itemmenge mit einem Konflikt
- Konstruktion ist erfolgreich, wenn keine ungeeigneten Zustände erzeugt werden

© H. Partsch 2014

Kanonische LR(1)-Zustandsmengen (1/2)

Berechnung der kanonischen LR(1)-Zustandsmengen

Algorithmus LR(1)-GEN

Unterschied zu LR-DEA

```
var q, q‘: set of item;
function Abschluss (s: set of item): set of item;
   begin
      q := s;
      foreach [X → α.Yβ, L] in q and Y → γ in P do
      if exist [Y → .γ, L‘] in q
         then ersetze [Y → .γ, L‘] durch [Y → .γ, L‘ ∪ Fi1(βL)]
         else q := q ∪ [Y → .γ, Fi1(βL)] fi od;
      return(q)
   end;
function Nachf (s: set of item, Y: VN ∪ VT): set of item;
   return ({[X → αY.β, L] | [X → α.Yβ, L] ∈ s});
begin
   Qd := Abschluss({[S‘ →.S, {#}]}); δd := ∅;
   foreach q in Qd and X in VN ∪ VT do
      let q‘ = Abschluss(Nachf(q, X)) in
      if q‘ ≠ ∅ then
         if q‘ not in Qd then Qd := Qd ∪ {q‘} fi;
         δd := δd ∪ {q --X--> q‘} fi tel od
end
```

Beispiel

Geg.: {S → E, E → E+T | T, T →T∗F | F, F→ (E) | **id**}

Abschluss(q) mit q = {[S → .E, {#}])}

Ab¹: {[S → .E, {#}]} ∪

{[E → .E+T, {#}], [E → .T, {#}]}

Ab²: {[S → .E, {#}]} ∪

{[E → .E+T, {#, +}], [E → .T, {#, +}]} = q

Ab³: q ∪

{[T → .T∗F, {#, +}], [T → .F, {#, +}]}

Ab⁴: q ∪

{[T → .T∗F, {#, +, ∗}], [T → .F, {#, +, ∗}]} = q‘

Ab⁵: q‘ ∪

{[F → .(E) , {#, +, ∗}], [F → .**id**, {#, +, ∗}]} = S_0‘

Nachf(S_0‘, E) = {[S → E., {#}], [E → E.+T, {#, +}]} = S_1‘

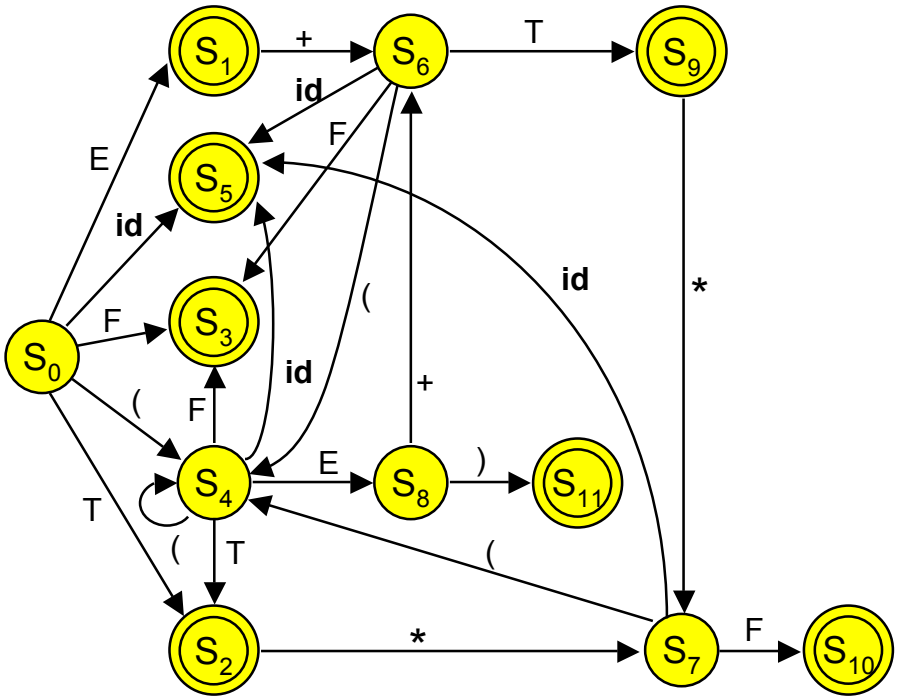
Abschluss(S_1‘) = S_1‘

© H. Partsch 2014

Kanonische LR(1)-Zustandsmengen (2/2)

Beispiel

LR(0)-Automat (= LR-DEA)



Beachte

- S_2“ ≠ S_2‘
- S_4‘ ≠ Abschluss(Nachf(S_4‘, „(“))

Einige LR(1)-Zustände

S_0‘ = Abschluss({[S → .E,{#}]})
= {[S → .E, {#}],
[E → .E+T, {#, +}],
[E → .T, {#, +}],
[T → .T*F, {#, +, *}],
[T → .F, {#, +, *}],
[F → .(E), {#, +, *}],
[F → **.id**, {#, +, *}]}

S_1‘ = Abschluss(Nachf(S_0‘, E))
= {[S → E., {#}],
[E → E.+T, {#, +}]}

S_4‘ = Abschluss(Nachf(S_0‘, „(“))
= {[F → (.E), {#, +, *}],
[E → .E+T, {), +}],
[E → .T, {), +}],
[T → .T*F, {), +, *}],
[T → .F, {), +, *}],
[F → .(E), {), +, *}],
[F → **.id**, {), +, *}]}

S_2‘ = Abschluss(Nachf(S_0‘, T))
= {[E → T., {#, +}],
[T → T.*F, {#, +, *}]}

S_6‘ = Abschluss(Nachf(S_1‘, +))
= {[E → E+.T, {#, +}],
[T → .T*F, {#, +, *}],
[T → .F, {#, +, *}],
[F → .(E) , {#, +, *}],
[F → **.id**, {#, +, *}]}

S_9‘ = Abschluss(Nachf(S_6‘, T))
= {[E → E+T., {#, +}],
[T → T.*F, {#, +, *}]}

S_2“ = Abschluss(Nachf(S_4‘, T))
= {[E → T., {), +}],
[T → T.*F, {), +, *}]}

© H. Partsch 2014

LR(1)-action-Tabelle

Konstruktion der LR(1)-action-Tabelle

- Letzter Schritt bei LR(1)-Parser-Konstruktion (für alle Verfahren gleich)
- Eingabe: LR(1)-Zustandsmenge Q (vom jeweiligen Verfahren abhängig)

```
foreach q ∈ Q do
   foreach LR(1)-Item [K, L] ∈ q do
      if K = [S' → S.] and L = {#} then action[q, #] := „accept“
      elsif K = [X → α.] then foreach a ∈ L do action[q, a] := „reduce (X → α)“ od
      elsif K = [X → α.aβ] then action[q, a] := „shift“ fi
   od
od;
foreach q ∈ Q, a ∈ V_T ∪{#} mit action[q, a] = undef. do action[q, a] := „error“ od
```

S_0' = Abschluss({[S → .E, {#}]})
= {[S → .E, {#}],
[E → .E+T, {#, +}],
[E → .T, {#, +}],
[T → .T*F, {#, +, *}],
[T → .F, {#, +, *}],
[F → .(E), {#, +, *}],
[F → .**id**, {#, +, *}]}

S_1' = Abschluss(Nachf(S_0', E))
= {[S → E., {#}],
[E → E.+T, {#, +}]}

S_2' = Abschluss(Nachf(S_0', T))
= {[E → T., {#, +}],
[T → T.*F, {#, +, *}]}

S_6' = Abschluss(Nachf(S_1', +))
= {[E → E+.T, {#, +}],
[T → T*.F, {#, +, *}],
[T → .F, {#, +, *}],
[F → .(E) , {#, +, *}],
[F → .**id**, {#, +, *}]}

S_9' = Abschluss(Nachf(S_6', T))
= {[E → E+T., {#, +}],
[T → T.*F, {#, +, *}]}

Action-Tabelle
(Ausschnitt)

	id	(	)	*	+	#
S_0'	s	s				
S_1'					s	acc
S_2'				s	r(E → T)	r(E → T)
S_6'	s	s				
S_9'				s	r(E → E+T)	r(E → E+T)

© H. Partsch 2014

Andere LR-Verfahren

SLR(1) und LALR(1)

- SLR(1) („simple LR“) und
- LALR(1) („lookahead LR“)
- sind echte Teilmengen von (kanonischem) LR(1)

Größe realer LALR-Tabellen (Action- und Goto-Tabelle zusammen) für Programmiersprache mit 50-100 Terminalen und ca. 100 Produktionen: Einige hundert Zustände; ≥ 20 000 action-Einträge

Vorteil gegenüber kanonischen LR-Parsern

- Zustandsmenge nur so groß wie die des zugehörigen LR(0)-Parsers

Zustandsmenge des kanonischen LR(1)-Parsers i.a. viel größer

Ausgangspunkt bei der Parser-Konstruktion

- Schon vorhandener LR(0)-Parser (= LR-DEA)

Idee (bei beiden Verfahren)

- Beseitigung von ungeeigneten Zuständen durch Erweiterung der vollständigen Items durch Vorausschaumengen, die mithilfe von LA: $Q_d \times It_G \rightarrow \mathbb{P}(V_T \cup \{\#\})$ berechnet werden
- Unterschied der beiden Verfahren:
  Unterschiedliche Definitionen von LA

d.h. keine Vorausschau für nicht vollständige Items

Partielle Funktion

© H. Partsch 2014

SLR(1)-Parser (1/3)

SLR(1)-Erweiterung

- Jedem vollständigen Item $[X \to \alpha.]$ wird (in allen Zuständen) $FOLLOW_1(X)$ als Vorausschaumenge zugeordnet
- $LA_S(q, [X \to \alpha.]) =_{def} \{a \in V_T \cup \{\#\} \mid S'\# \Rightarrow^* \beta X a \gamma\}$
  $= FOLLOW_1(X)$ für alle q mit $[X \to \alpha.] \in q$

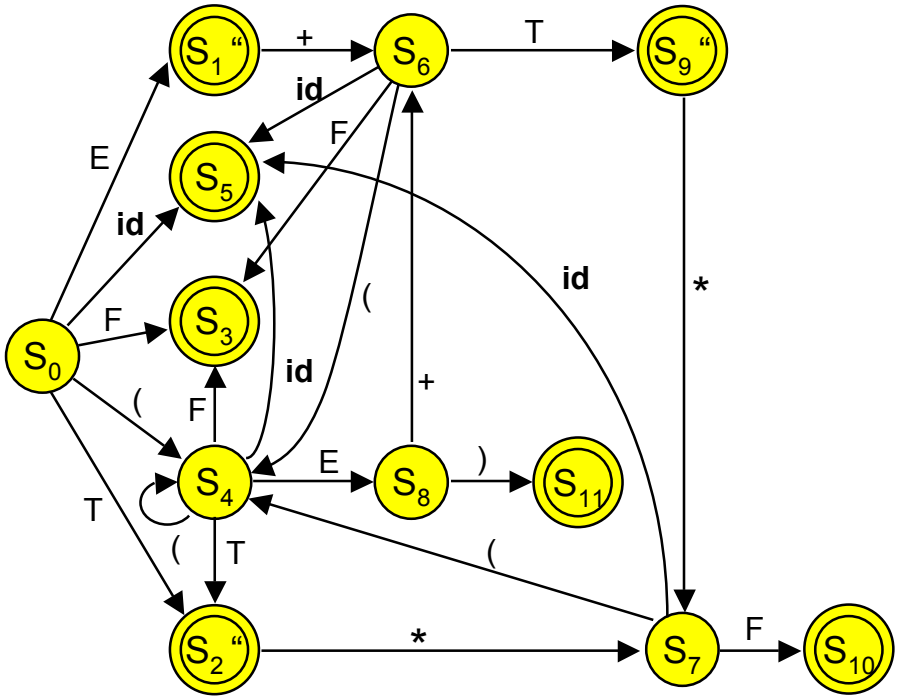
SLR(1)- Grammatik

- Sei G eine kfG und SLR-DEA(G) der SLR-erweiterte LR-DEA(G)
- G ist **SLR(1)-Grammatik** $\Leftrightarrow$ SLR-DEA(G) hat keine SLR-ungeeigneten Zustände
- Zustand q ist **SLR-ungeeignet** $\Leftrightarrow$ q enthält
  - Item $[X \to \alpha.a\beta]$, $a \in V_T$ und Item $[Y \to \gamma., L_2]$ mit $a \in L_2$
  - Items $[X \to \alpha., L_1]$ und $[Y \to \gamma., L_2]$ mit $L_1 \cap L_2 \neq \varnothing$

© H. Partsch 2014

SLR(1)-Parser (2/3)

Beispiel



S_0 =
{[S → .E], [E → .E+T],
[E → .T], [T → .T*F],
[T → .F], [F → .(E)],
[F → **.id**]}

S_1“ =
{[S → E., **{#}**],
[E → E.+T]}

S_2“ =
{[E → T., **{#, +,)}**],
[T → T.*F]}

S_3 = {[T → F.]}

S_4 =
{[F → (.E)], [E → .E+T],
[E → .T], [T → .T*F],
[T → .F], [F → .(E)],
[F → **.id**]}

S_5 = {[F → **id.**]}

S_6 =
{[E → E+.T], [T → T*.F],
[T → .F], [F → .(E)],
[F → **.id**]}

S_7 =
{[T → T*.F], [F → .(E)],
[F → **.id**]}

S_8 =
{[F → (E.)], [E → E.+T]}

S_9“ =
{[E → E+T., **{#, +,)}**],
[T → T.*F]}

S_{10} = {[T → T*F.]}

S_{11} = {[F → (E).]}

Es gilt

- SLR-DEA(G) hat keine SLR-ungeeigneten Zustände
  - S_1“: + ∉ {#}
  - S_2“: * ∉ {#, +,)}
  - S_9“: * ∉ {#, +,)}

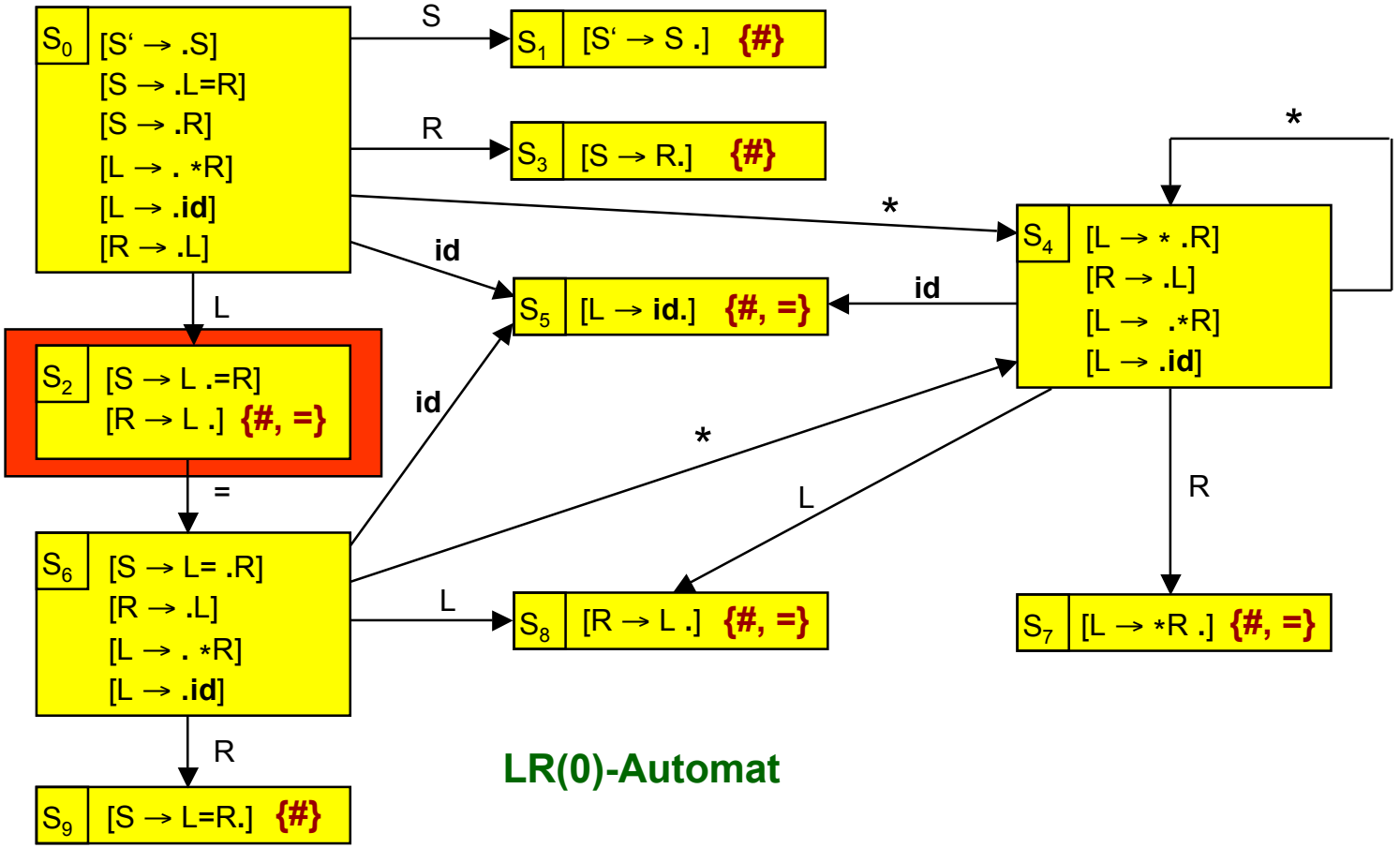
Beachte: *Für die action-Tabelle müssten auch noch die fehlenden vollständigen Items um Vorausschaumengen erweitert werden*

© H. Partsch 2014

SLR(1)-Parser (3/3)

Beispiel (Vereinfachung der C-Wertzuweisung)

- Geg.: kfG G mit Produktionen {S‘ → S, S → L=R | R, L → *R | **id**, R → L}
- G ist **nicht** SLR(1): Zustand S_2 = {[S → L.=R], [R → L.]} ist SLR(1)-ungeeignet



LR(0)-Automat

FOLLOW-Mengen

$FOLLOW_1(S) = \{\#\}$
$FOLLOW_1(L) = \{\#, =\}$
$FOLLOW_1(R) = \{\#, =\}$

© H. Partsch 2014

LALR(1)-Parser (1/3)

LALR(1)-Erweiterung

- Jedem vollständigen Item $[X \to \alpha.]$ wird eine (in FOLLOW oft echt enthaltene) vom Zustand q abhängige Vorausschaumenge zugeordnet
- $LA_L(q, [X \to \alpha.]) =_{def} \{a \in V_T \cup \{\#\} \mid S'\# \Rightarrow_{rm}^* \beta X a w \text{ und } \delta_d^*(q_d, \beta\alpha) = q\}$

Beachte: Definition nicht konstruktiv !!

δ_d = *Übergangsfunktion des LR-DEA(G);*
q_d *sein Startzustand*

LALR(1)- Grammatik

- Sei G eine kfG und LALR-DEA(G) der LALR-erweiterte LR-DEA(G)
- G ist **LALR(1)-Grammatik** $\Leftrightarrow$ LALR-DEA(G) hat keine LALR-ungeeigneten Zustände
- Zustand q ist **LALR-ungeeignet** $\Leftrightarrow$ q enthält
  - Item $[X \to \alpha.a\beta]$, $a \in V_T$ und Item $[Y \to \gamma., L_2]$ mit $a \in L_2$
  - Items $[X \to \alpha., L_1]$ und $[Y \to \gamma., L_2]$ mit $L_1 \cap L_2 \neq \emptyset$

Mögliche Berechungsmethoden

- Kanonischen LR(1)-Parser konstruieren
  - Falls keine Konflikte, Zustände mit gleichen Mengen von Item-Kernen „verschmelzen“, d.h. Vorausschaumengen der jeweiligen Items vereinigen
  - Grammatik ist LALR(1), falls nach Verschmelzung immer noch keine Konflikte
- Verschmelzung in Algorithmus LR(1)-GEN integrieren

© H. Partsch 2014

LALR(1)-Parser (2/3)

Effiziente Berechnung der Vorausschaumengen für LALR(1)-Parser

- Ausgangspunkt: LR(0)-Parser
- Aufgabe

  Berechnung von $LA_L(q, [X \rightarrow \alpha.])$ für jeden Zustand $q \in Q_d$ und jedes vollständige Item $[X \rightarrow \alpha.]$ in q
- Lösung:

  Umrechnung in $LA_L(q, [X \rightarrow \alpha.]) = \bigcup_{\delta_d^*(p, \alpha) = q} LA'(p, X)$ wobei
  - $LA'(q_d, S') =_{def} \{\#\}$
  - $LA'(q, X) =_{def} \bigcup_{[Y \rightarrow \beta.X\gamma] \in q} \bigcup_{\delta_d^*(p, \beta) = q} FIRST_1(\gamma) \oplus_1 LA'(p, Y)$

Idee
Zustände betrachten, von denen aus q unter α erreicht wird

- Beispiel

$LA_L(S_2, [R \rightarrow L.])$ = [Def.]

$\bigcup_{\delta_d^*(p, L) = S_2} LA'(p, R)$ = [siehe Automat]

$LA'(S_0, R)$ = [Def.]

$\bigcup_{[S \rightarrow .R]} \bigcup_{\delta_d^*(p, \varepsilon) = S_0} FIRST_1(\varepsilon) \oplus_1 LA'(p, S)$ = [Vereinf.]

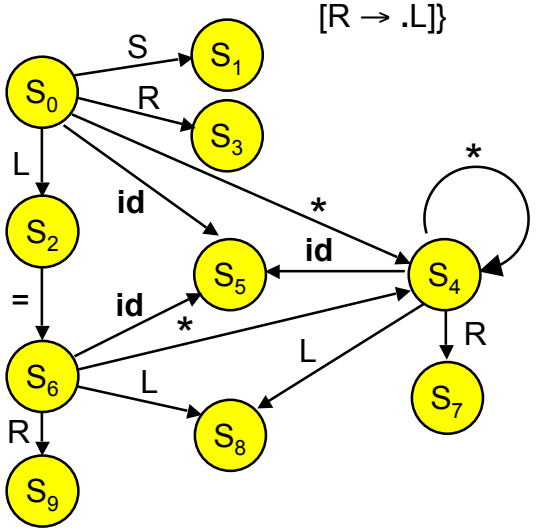
$LA'(S_0, S)$ = [Def.]

$\bigcup_{[S' \rightarrow .S]} \bigcup_{\delta_d^*(p, \varepsilon) = S_0} FIRST_1(\varepsilon) \oplus_1 LA'(p, S')$ = [Vereinf.]

$LA'(S_0, S')$ = [Def.]

$\{\#\}$

S_0 = {[S' → .S],
[S → .L=R],
[S → .R],
[L → . *R],
[L → .**id**],
[R → .L]}

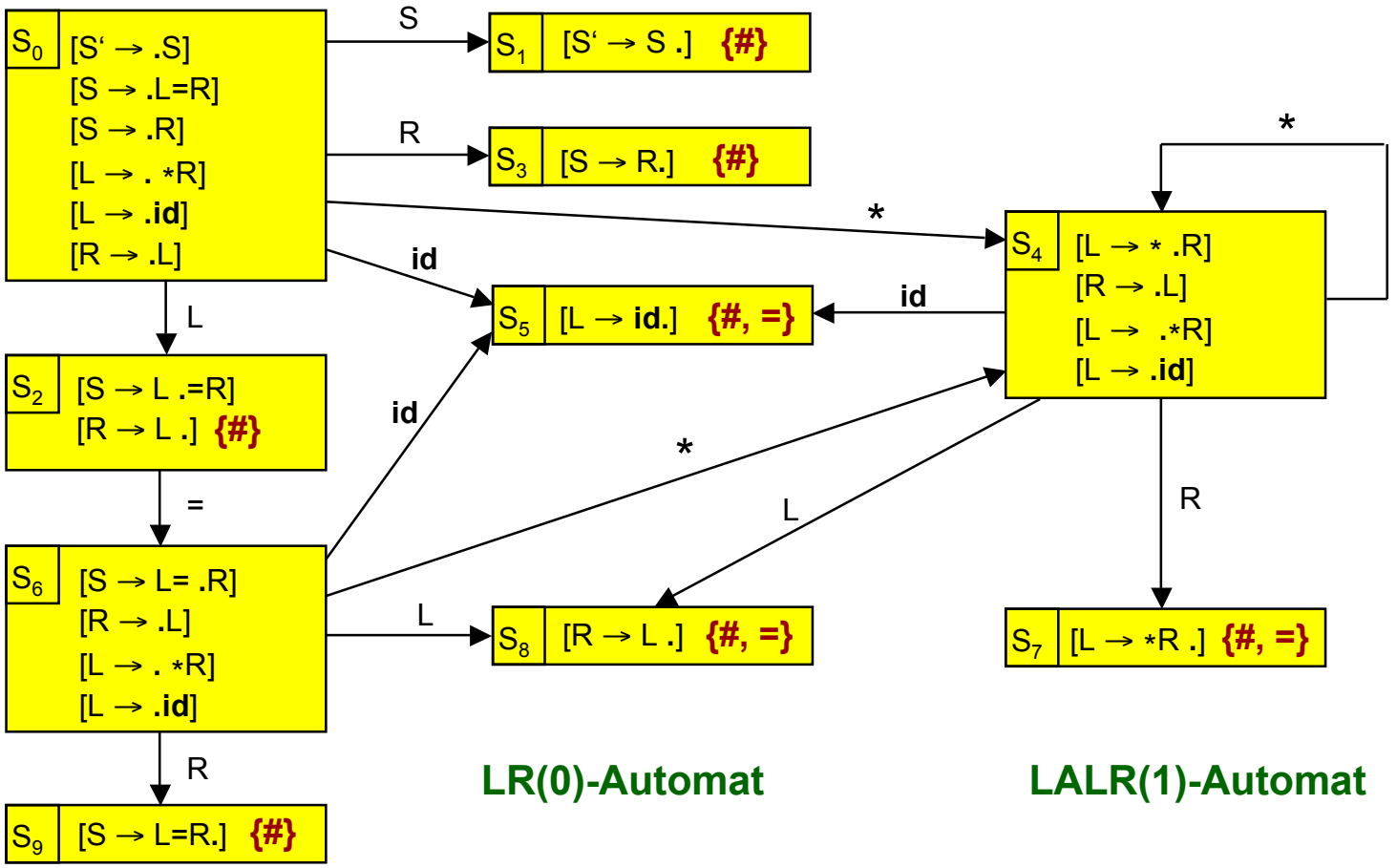


© H. Partsch 2014

LALR(1)-Parser (3/3)

Beispiel (Vereinfachung der C-Wertzuweisung)

- Geg.: kfG G mit Produktionen {S‘ → S, S → L=R | R, L → *R | **id**, R → L}
- G ist LALR(1): kein Konflikt in S_2



© H. Partsch 2014

Fehlerbehandlung in LR-Parsern (1/3)

Grundlage

- Auch LR-Parser haben Eigenschaft des fortsetzungsfähigen Präfix

Frühest mögliche Situation, in der ein Fehler entdeckt werden kann

- **Fehlerkonfiguration**: Konfiguration (φq, $a_i ... a_n$) mit action(q, a_i) = error
- **Fehlerzustand**: Zustand q einer Fehlerkonfiguration

Prinzipiell mögliche Fehlerbehandlungsverfahren

- Fehlerbehandlungsstrategie
  - Vorwärtsfehlerbehandlung (Modifikation in der Resteingabe, Parserkeller unverändert)
  - Rückwärtsfehlerbehandlung (Veränderung des Parserkellers) — *problematisch*
- Art der Realisierung
  - Benutzer-spezifiziert
  - Automatisch generiert

© H. Partsch 2014

Fehlerbehandlung in LR-Parsern (2/3)

Gängiges Verfahren

- Automatisch erzeugte Vorwärtsfehlerbehandlung
  (nur letzte Reduktion kann rückgängig gemacht werden)

Aufgabe des Verfahrens

- Zu Fehlerkonfiguration (φq, $a_i...a_n$) „passende“ Konfiguration finden, in der Fortsetzung der Analyse durch Lesen (mindestens) eines Eingabesymbols möglich ist
- „passend“: geht durch möglichst wenig Veränderungen aus Fehlerkonfiguration hervor

„Lesen“ wichtig für Terminierung der Fehlerbehandlung

Weitere Einschränkung

- „1-Fehlerhypothese“: Fehler wurde durch 1 fehlendes, 1 überflüssiges oder 1 falsches Symbol verursacht

Aus Effizienzgründen

Konsequenz

- 3 Elementaroperationen des Fehlerbehandlungsalgorithmus:
  Einsetzen, Löschen, Ersetzen (jeweils eines Symbols)

© H. Partsch 2014

Fehlerbehandlung in LR-Parsern (3/3)

Aufgabe der Fehlerbehandlung

- Geg.: Fehlerkonfiguration (φq, $a_i...a_n$)
- Ziel einer Fehlerkorrektur
  - Löschen: Finde Kellerinhalte $\varphi' p$ mit
    (φq, $a_{i+1}...a_n$) $\vdash^*$ ($\varphi' p$, $a_{i+1}...a_n$) und action[p, a_{i+1}] = shift
  - Ersetzen: Finde Symbol a und Kellerinhalte $\varphi' p$ mit
    (φq, $aa_{i+1}...a_n$) $\vdash^*$ ($\varphi' p$, $a_{i+1}...a_n$) und action[p, a_{i+1}] = shift
  - Einfügen: Finde Symbol a und Kellerinhalte $\varphi' p$ mit
    (φq, $aa_i...a_n$) $\vdash^*$ ($\varphi' p$, $a_i...a_n$) und action[p, a_i] = shift

Wichtige Eigenschaft

- Terminierung des Fehlerbehandlungsverfahrens ist garantiert
  (Eingabe wird konsumiert)

Allerdings

- Fehlerbehandlung nach obiger Definition zu teuer

Zu viele potentielle Kellerinhalte

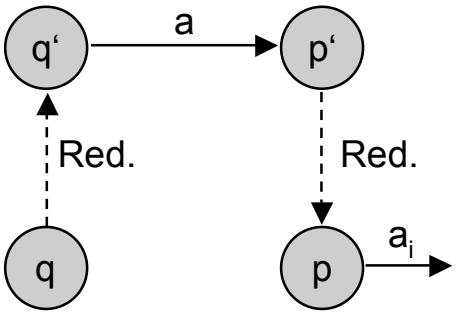
- Deshalb stattdessen: Vorberechnungen („Fehlerbrücke“) zur Generierungszeit für unterschiedliche Fehlerkonfigurationen

© H. Partsch 2014

Fehlerbehandlung durch Einfügen (1/4)

Fehlerbehandlung durch Einfügen eines Symbols a ∈ V_T

- Sei (φq, a_i...a_n) Fehlerkonfiguration
- Vorgehensweise („Schließen der Fehlerbrücke“)
  - (a) Folge von Reduktionen unter Vorausschausymbol a
  - (b) Leseaktion bezüglich a
  - (c) Folge von Reduktionen unter Vorausschausymbol a_i
- Effiziente Durchführung von (a) - (c) durch geeignete Vorausberechnungen



© H. Partsch 2014

Fehlerbehandlung durch Einfügen (2/4)

Zu (b)

- Berechne für a $\in V_T$ die Menge Sh(a) $=_{def}$ {q | action[q, a] = shift}
  (aus Action-Tabelle oder Automat)

Beispiel

- Geg.: kfG G mit Produktionen {S‘ → S, S → L=R | R, L → *R | **id**, R → L}

	=	*	**id**	#
S_0		s	s	
S_1				acc
S_2	s			r(R → L)
S_3				r(S → R)
S_4		s	s	
S_5	r(L → **id**)			r(L → **id**)
S_6		s	s	
S_7	r(L → *R)			r(L → *R)
S_8	r(R → L)			r(R → L)
S_9				r(S → L=R)

Action-Tabelle

Sh(=)	= {S_2}
Sh(*)	= {S_0, S_6, S_4}
Sh(**id**)	= {S_0, S_6, S_4}

Mengen Sh(a)

© H. Partsch 2014

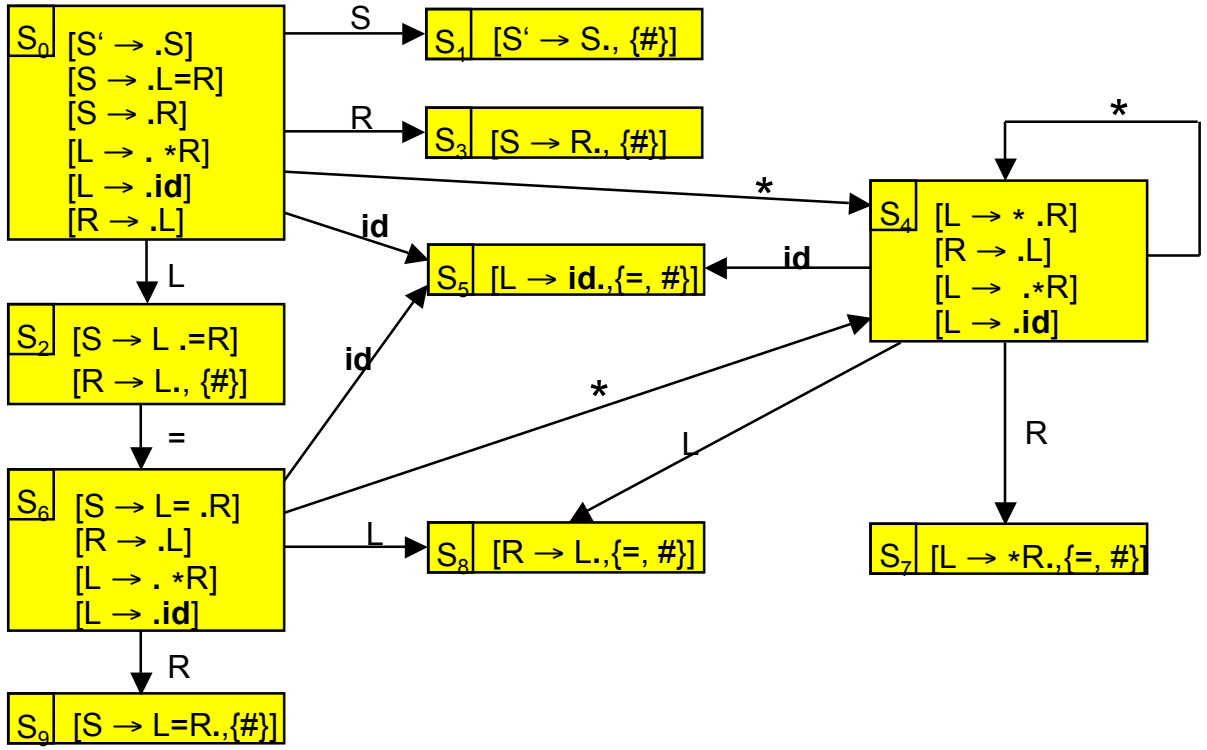
Fehlerbehandlung durch Einfügen (3/4)

Zu (a)

- Berechne für alle $q \in Q_d$ und $a \in V_T$ die Menge $Succ(q)_a$ der **Reduktionsnachfolger**
  - $S'(q)_a =_{def}$ {Zustände, in die der Parser durch 1 Reduktion unter Vorausschau a kommen kann}
  - $S(q)_a =_{def} S'(q)_a \cup \bigcup_{p \in S'(q)_a} S(p)_a$
  - $Succ(q)_a =_{def} (S(q)_a \cap Sh(a)) \cup \{q\}$

Beispiel

- Es gilt
  - $S(S_5)_= = \{S_2,S_8\} \cup S(S_2)_= \cup S(S_8)_=$
    $= \{S_2,S_8\} \cup S(S_8)_=$
  - $S(S_8)_= = \{S_3,S_7,S_9\} \cup$
    $S(S_3)_= \cup S(S_7)_= \cup S(S_9)_=$
    $= \{S_3,S_7,S_9\} \cup S(S_7)_=$
  - $S(S_7)_= = \{S_2,S_8\} \cup S(S_2)_= \cup S(S_8)_=$
    $= S(S_5)_=$
- Damit
  - $S(S_5)_= = \{S_2,S_3,S_7,S_8,S_9\}$
  - $Succ(S_5)_=$
    $= (\{S_2,S_3,S_7,S_8,S_9\} \cap \{S_2\}) \cup \{S_5\}$
    $= \{S_2,S_5\}$



© H. Partsch 2014

Fehlerbehandlung durch Einfügen (4/4)

Berechnung der Brücke

- Berechne für alle $q \in Q_d$ und $a_i \in V_T$ die Menge
  $Bridge(q)_{a_i} =_{def} \{a \in V_T | \exists\ q' \in Succ(q)_a$ und $\delta_d(q',a) = p'$ und $Sh(a_i) \cap Succ(p')_{a_i} \neq \emptyset\}$

Beispiel

- $Bridge(S_0)_=$
  $= \{a \in V_T | \exists\ q' \in Succ(S_0)_a$ und $\delta_d(q',a) = p'$ und
  $Sh(=) \cap Succ(p')_= \neq \emptyset\}$
  Kandidaten
  id, da $\delta_d(S_0, \textbf{id}) = S_5 \Rightarrow Succ(S_5)_= = \{S_2, S_5\}$
  $*$, da $\delta_d(S_0, *) = S_4 \Rightarrow Succ(S_4)_= = \{S_4\}$
  $= \{\textbf{id}\}$, da $\{S_2\} \cap \{S_4\} = \emptyset$
- $Bridge(S_0)_*$
  $= \{a \in V_T | \exists\ q' \in Succ(S_0)_a$ und $\delta_d(q',a) = p'$ und
  $Sh(*) \cap Succ(p')_* \neq \emptyset\}$
  Kandidaten
  id, da $\delta_d(S_0, \textbf{id}) = S_5 \Rightarrow Succ(S_5)_* = \{S_5\}$
  $*$, da $\delta_d(S_0, *) = S_4 \Rightarrow Succ(S_4)_* = \{S_4\}$
  $= \{*\}$, da $\{S_0,S_4,S_6\} \cap \{S_5\} = \emptyset$

$Succ(q)_a$

	=	*	id
S_0	S_0	S_0	S_0
S_1	S_1	S_1	S_1
S_2	S_2	S_2	S_2
S_3	S_3	S_3	S_3
S_4	S_4	S_4	S_4
S_5	S_2,S_5	S_5	S_5
S_6	S_6	S_6	S_6
S_7	S_2,S_7	S_7	S_7
S_8	S_2,S_8	S_8	S_8
S_9	S_9	S_9	S_9

Sh(=)	$= \{S_2\}$
Sh(*)	$= \{S_0, S_6, S_4\}$
Sh(**id**)	$= \{S_0, S_6, S_4\}$

© H. Partsch 2014

Fehlerkorrektur

Beispiel (Korrektur durch Einfügen)

Eingabe	Fehlerkonfiguration	Brücke	Korrektur
* = **id** #	(S_0S_4, = **id** #)	Bridge$(S_4)_=$ = {**id**}	Einfügen von **id**

Bridge$(q)_a$

	=	*	**id**
S_0	{**id**}	{*}	{*}
S_1	∅	∅	∅
S_2	∅	{=}	{=}
S_3	∅	∅	∅
S_4	{**id**}	{*}	{*}
S_5	∅	{=}	{=}
S_6	{**id**}	{*}	{*}
S_7	∅	{=}	{=}
S_8	∅	∅	∅
S_9	∅	∅	∅

Auf der Basis der vorberechneten Mengen

- 1-Symbol-Ersetzungskorrektur
  (Unterschied: Symbole aus Bridge$(q)_{a_{i+1}}$)
- Beispiel

Eingabe	Fehlerkonfiguration	Brücke	Korrektur
id = = **id** #	($S_0S_2S_6$, = **id** #)	Bridge$(S_6)_{\mathbf{id}}$ = {*}	Ersetzen von = durch *

- 1-Symbol-Löschkorrektur
  (Löschung eines Symbols a ist Möglichkeit,
  wenn es Zustand $p \in Sh(a_{i+1}) \cap Succ(q)_{a_{i+1}}$ gibt)
- Beispiel

Eingabe	Fehlerkonfiguration	Brücke	Korrektur
id id = id #	(S_0S_5, **id = id** #)	$Sh(=) \cap Succ(S_5)_= = \{S_2\}$	Löschen von **id** Ersetzen von S_5 durch S_2

© H. Partsch 2014

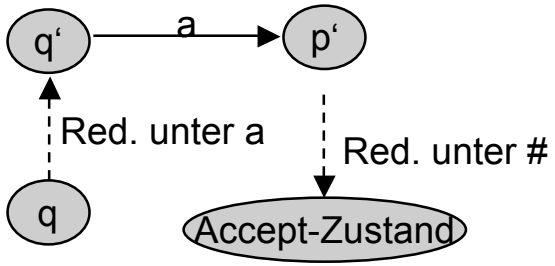
Korrekturen bei erschöpfter Eingabe

Sonderfall

- Korrekturen bei erschöpfter Eingabe
- Hier: Nur Einsetzungskorrekturen sinnvoll

Lösung

- Für jeden Zustand q die Menge Acc(q) berechnen, eine Menge von Terminalsymbolen a, so dass gilt
  - Reduktionen von q unter a führt in Zustand q‘
  - Lesen von a in q‘ ergibt p’
  - Reduktionen von p’ unter # führen in accept-Konfigurationen



- Modifikation von Bridge

  $Bridge(q)_{\#} =_{def}$

  $\{a \in V_T | \exists\ q' \in Succ(q)_a$ und $\delta_d(q',a) = p'$ und $Succ(p')_{\#} \in$ „accept-Zuständen“$\}$

© H. Partsch 2014

Fehlerbehandlung in SLR(1)- / LALR(1)-Parsern

Falsche Reduktionen in SLR(1)- und LALR(1)-Parsern

- Kanonische LR-Parser
  - Lesen weder ein Symbol über Fehlerstelle hinaus
  - Noch reduzieren sie unter falschem Vorausschausymbol
- SLR(1)- und LALR(1)-Parser
  - Machen möglicherweise Reduktionen, bevor in shift-Zustand Fehler entdeckt wird (da weniger differenzierte Vorausschau)

Abhilfe

- Zusätzlicher Keller
  - Zur Speicherung aller seit letztem Lesen durchgeführten Reduktionen
  - Wird bei Lese-Aktion geleert
  - Im Fehlerfall: Gekellerte Reduktionen rückgängig machen

© H. Partsch 2014