Grundlegende Parsingalgorithmen

Tabellen-gesteuertes Parsing: LR Parsing

Kurt Eberle

k.eberle@lingenio.de

(Viele Folien, Teile von Folien, Materialien von **Helmut Schmid**'s Parsing-Kurs WS14 Tübingen, u.a.)

31. Juli, 2018

Überblick

Überblick

Tabellen-gesteuerter Bottom-Up Parser

Problem: "Blinde"/uninformierte Wahl zwischen Shift- und Reduce-Operation des BU-Parsers

Idee: Wähle Shift- oder Reduce-Aktion nur, falls die linke Seite der resultierenden Konfiguration ein *"viable prefix"* (ein *"tragfähiges Präfix* ist.

- Ein *"viable prefix"* ist ein Präfix eines Zwischenergbnisses in einer Rechtsderivation
- Die Menge der viable prefixes ist eine reguläre Sprache die durch einen finiten Automaten erkannt wird.

Viable Prefixes

Was sind *viable prefixes?*

Die Grundidee ist, dass in jedem spezifischen Zustand der Prozessierung es jeweils nur eine Teilmenge von Regeln der Gesamtregelmenge gibt die anwendbar sind:
diejenigen die mit dem jeweils bisher erzeugten Präfix kompatibel sind. Falls der Anwendungsbereich einer Regel sich über dieses Präfix (oder ein Suffix dieses Präfixes) erstreckt und ein Stück weiter, dann is der Präfix-Teil der Sentential Form die sich aus der Anwendung einer solchen Regel ergibt schon erkannt und der noch nicht erkannte Rest stellt Constraints für die Erkennung der jeweils nächsten zu erkennenden Symbole dar. D.h., abhängig davon, ob diese Analyse durchgeht kann dann (bzgl. der gerade betrachteten Regel) zu einem entsprechend neuen Zustand übergegangen werden (der wiederum für eine Reihe teilweise erkannter Regeln steht) oder eben nicht.

Überblick

Basic Operations

Dotted Rule (Item)

S $\rightarrow \cdot$ NP VP	*S is expected next, NP is predicted*
S $\rightarrow$ NP $\cdot$ VP	*S expected, NP recognized, VP predicted*
S $\rightarrow$ NP VP $\cdot$	*S expected and completely recognized*

Closure Operation on a dotted rule set I:
If $A \rightarrow \alpha \cdot B\beta \in I$, then add the dotted rules $B \rightarrow \cdot \gamma$ for all $B \rightarrow \gamma \in P$ to I.
This rule is recursively applied, until no more dotted rules can be added.

Goto Function:
Goto(I,X) = Closure($\{A \rightarrow \alpha X \cdot \beta \mid A \rightarrow \alpha \cdot X\beta \in I\}$)

Konstruktion der Menge der Dotted-Rule-Sets

- Define a **new start symbol** S', i.e. replace $G = (V, \Sigma, P, S)$ by $G' = (V \cup \{S'\}, \Sigma, P \cup \{S' \to S\}, S')$
- Compute the **Set of Dotted Rule Sets**:

  $I_0 := \mathit{Closure}(\{S' \to \cdot\, S\})$;
  $C := \{I_0\}$;
  Repeat
    For all $I \in C$ and all $X \in V \cup \Sigma$ s.t. $\mathit{Goto}(I, X) \neq \{\}$
      Add $\mathit{Goto}(I, X)$ to C
  until nothing can be added to C

- **Kernel dotted rules**
  $\{A \to \alpha \cdot \beta \in I |\ \alpha \neq \varepsilon \text{ or } A = S'\}$
- Non-kernel dotted rules
  $\{A \to \cdot\beta \in I |\ A \neq S'\}$

Überblick

Finite State Automaton

- Each dotted rule set I_k corresponds to a **state** k of the automaton.
- The Goto function defines the possible **state transitions**: If Goto(I_j,X) $= I_k$, then the automaton contains a transition from state j to state k with symbol $X \in V \cup \Sigma$.
- Each set of dotted rules provides information about the possible reduce actions in the corresponding state.

Konstruktion der SLR Kontroll-Tabelle

- Assume that $C = \{I_0, I_1, ..., I_n\}$ is the set of dotted rule sets.
- State i of the parser corresponds to the dotted rule set I_i.
- Computation of the Action and Goto tables:
  1. If $Goto(I_i, a) = I_j$, then $Action[i, a] =$ *'shift j'*.
  2. If $A \rightarrow \alpha\cdot \in I_i$, then $Action[i, a] =$ *'reduce* $A \rightarrow \alpha$*'* for all $a \in Follow(A)$ (SLR parser) or for all $a \in \Sigma$ (LR(0)-Parser).
  3. If $S' \rightarrow \alpha\cdot \in I_i$, then $Action[i, \$] =$ *'accept'*.
  4. If $Goto(I_i, A) = I_j$, then $Goto[i, A] = j$.
  5. All empty boxes of the table are filled with *'error'*.
  6. The start state S_0 of the parser corresponds to the dotted rule set containing $S' \rightarrow \cdot\, S$.

- No multiple entries $\Rightarrow$ LR(0) / SLR Grammar

Beispiel einer SLR-Tabelle

(1) S → if E then S else S
(2) S → if E then S
(3) S → s
(4) E → e

Zustand	Action						Goto	
	if	s	e	then	else	$	S	E
0	s1	s2					9	
1			s3					4
2					r3	r3		
3				r4				
4				s5				
5	s1	s2					6	
6					s7,r2	r2		
7	s1	s2					8	
8					r1	r1		
9						acc		

Wirkungsweise des xLR Erkenners

Start configuration $(S_0, w\$)$
Repeat
  Assume that $(s_0 X_1 s_1 ... X_m s_m, a_i a_{i+1} ... a_n \$)$ is the current configuration
  $A := Action[s_m, a_i]$
  If A = 'shift s' then
    switch to the configuration $(s_0 X_1 s_1 ... X_m s_m \underline{a_i s}, a_{i+1} ... a_n \$)$
  otherwise if A = 'reduce $A \rightarrow \alpha$' then
    switch to the configuration $(s_0 X_1 s_1 ... X_{m-r} s_{m-r} \underline{A s}, a_i a_{i+1} ... a_n \$)$
      where $r = |\alpha|$ and $s = Goto[s_{m-r}, A]$
until A = 'accept' or A = 'error'

Benütze derivationsabhängige Look-Aheads

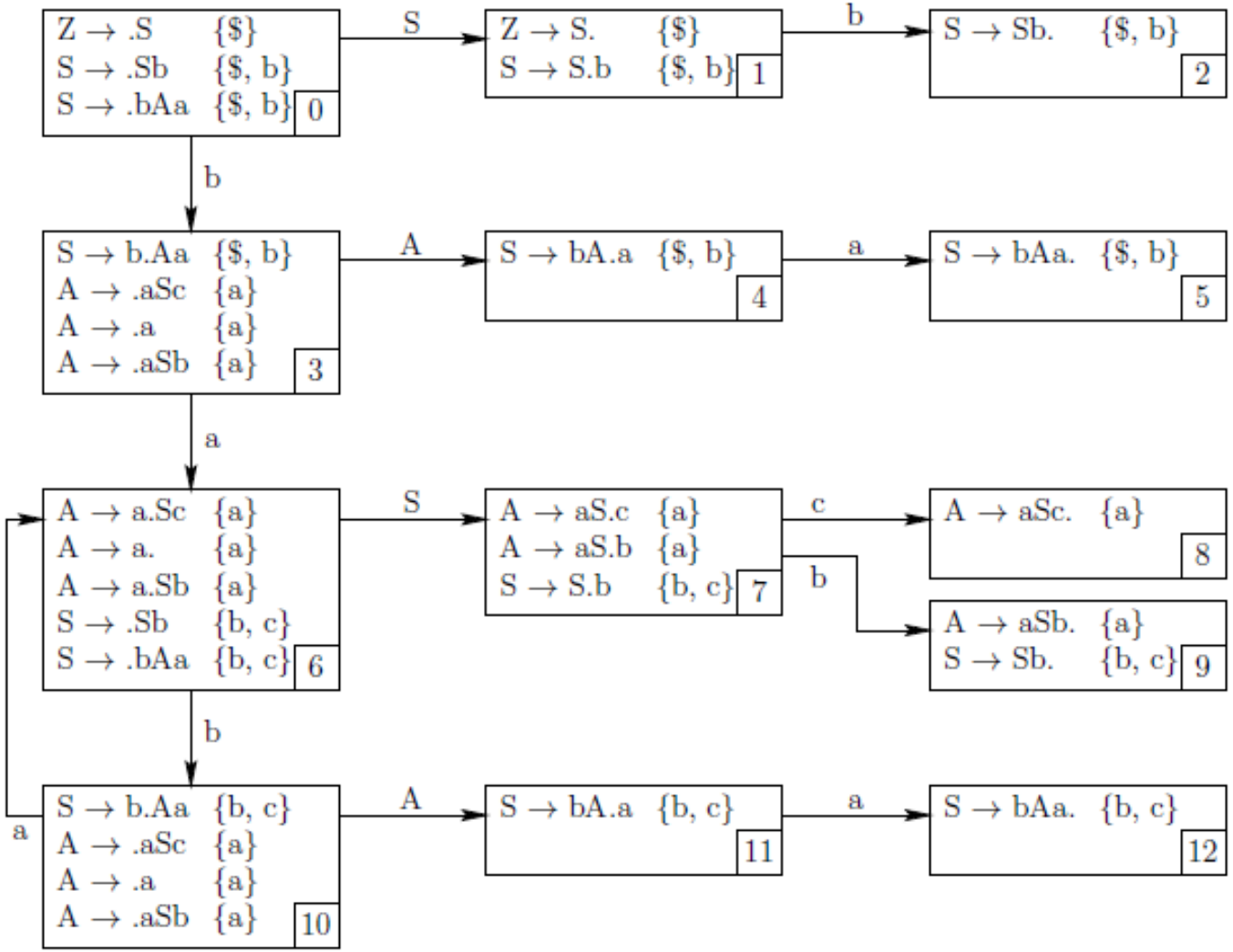
$[Z \to .S \quad , \{\$\}]$
$[S \to .Sb \quad , \{\$\}]$
$[S \to .bAa, \{\$\}]$

$[Z \to .S \quad , \{\$\}]$
$[S \to .Sb \quad , \{\$\}]$
$[S \to .bAa, \{\$\}]$
$[S \to .Sb \quad , \{b\}]$
$[S \to .bAa, \{b\}]$

$[Z \to .S \quad , \{\$\}]$
$[S \to .Sb \quad , \{b, \$\}]$
$[S \to .bAa, \{b, \$\}]$

(aus Kunert: LR(k)-Analyse für Pragmatiker)

LR(1): States and transitions



(aus Kunert: LR(k)-Analyse für Pragmatiker)

LALR(1): Zustände und Übergänge

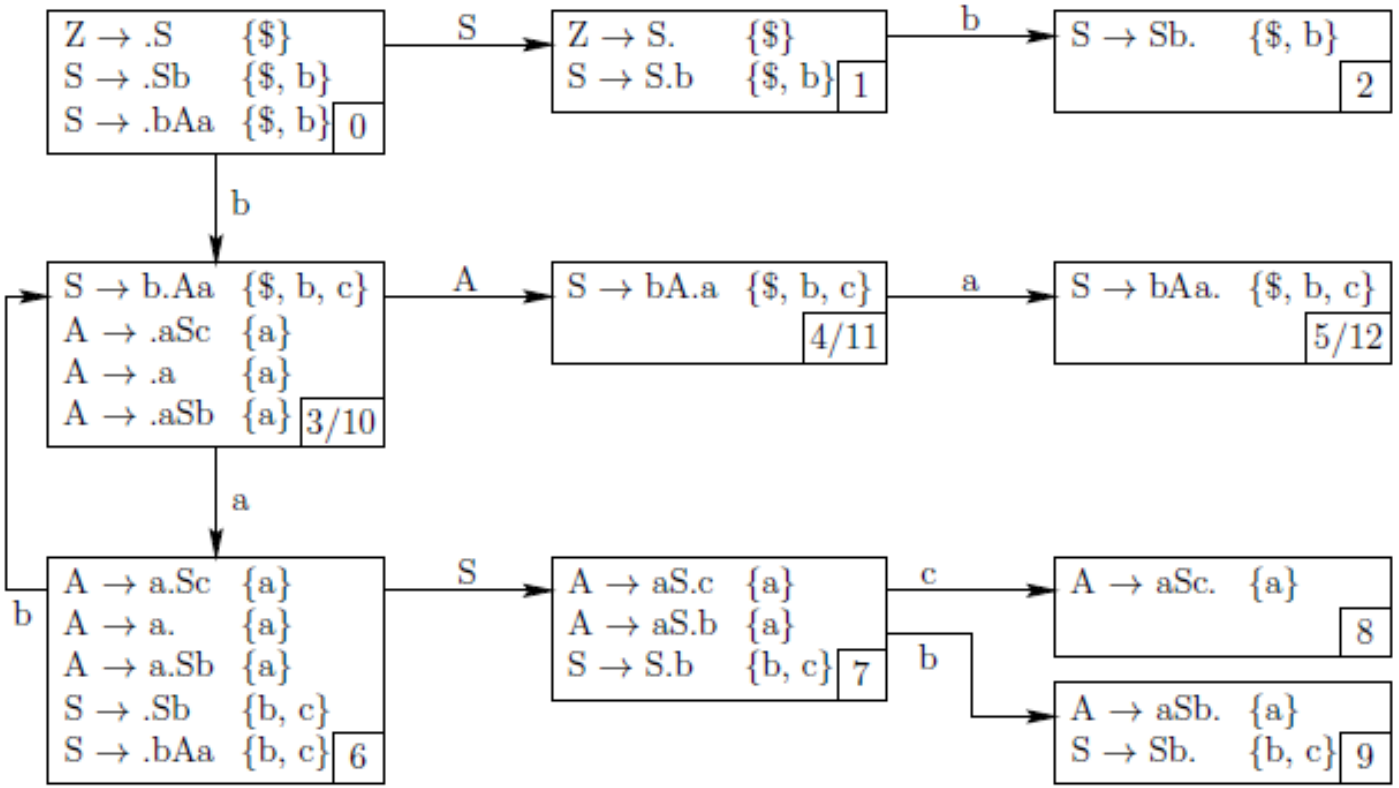


Abbildung 6.1: LALR(1)-Zustandsübergangsgraph für Grammatik G_1

(aus Kunert: LR(k)-Analyse für Pragmatiker)

xLR Parsing-Methoden

- LR(0): Reduce $\langle \alpha\beta, r \rangle$ using $A \to \beta$, if αA is a viable prefix
- SLR(1): Reduce $\langle \alpha\beta, ar \rangle$ using $A \to \beta$, if αA is a viable prefix and $a \in \mathit{Follow}(A)$
- LALR(1): Reduce $\langle \alpha\beta, ar \rangle$ using $A \to \beta$, if αA is a viable prefix and $a \in \mathit{Follow}(\alpha A)$
- $LL(k) \subset LR(k)$

Beispiel-Grammatiken

LR(0)

$E \to aed$
$E \to aec$
$E \to bec$
$E \to bed$

SLR(1)

$E \to aAd$
$E \to aec$
$E \to bec$
$E \to bed$
$A \to e$

LALR(1)

$E \to aAd$
$E \to aec$
$E \to bAc$
$E \to bed$
$A \to e$

LR(1)

$E \to aAd$
$E \to aBc$
$E \to bAc$
$E \to bBd$
$A \to e$
$B \to e$