Syntaxanalyse: LR-Parser (Teil 1)

BC George (FH Bielefeld)

Wiederholung

$S \to \underline{A}BC \mid \underline{D} \mid C \mid \underline{c}A$

$A \to a \mid \varepsilon \mid \ldots$ $\quad First_1(A) = \{a, \ldots\}$

$D \to d \mid \ldots$ $\quad First_1(D) = \{d, \ldots\}$

$B \to b$ $\quad First_1(S) = \{c, a, d\}$

$Follow_1(A) = \{b, \ldots\}$

Top-Down-Analyse

- Baumaufbau von oben nach unten
- die Grammatik muss reduziert sein
- recursive-descent parser
- *First*- und *Follow*-Mengen bestimmen Wahl der Ableitungen
- tabellengesteuert
- nicht mehr rekursiv, sondern mit PDA

Motivation

LL ist nicht alles

Die Menge der *LL*-Sprachen ist eine echte Teilmenge der deterministisch kontextfreien Sprachen. Wir brauchen ein Verfahren, mit dem man alle deterministisch kontextfreien Sprachen parsen kann.

Bottom-Up-Analyse

Von unten nach oben

Bei LL-Sprachen muss man nach den ersten k Eingabezeichen entscheiden, welche Ableitung ganz oben im Baum als erste durchgeführt wird, also eine, die im Syntaxbaum ganz weit weg ist von den Terminalen, die die Entschiedung bestimmen.Es gibt deterministisch kontextfreie Sprachen, die nicht $LL(k)$ sind für irgendein k.

Bei der Bottom-Up-Analyse geht man den umgekehrten Weg. Der Parse Tree wird von unten nach oben aufgebaut, die Entscheidung, welche Produktion angewandt wird, erfolgt "näher" am Terminal. Mit Hilfe der Produktionen und der Vorschautoken werden die Ableitungen "rückwärts" angewandt und "Reduktionen" genannt.

Fehlermeldungen können näher am Programmtext erfolgen.

Baumaufbau von unten

$S' \to E$

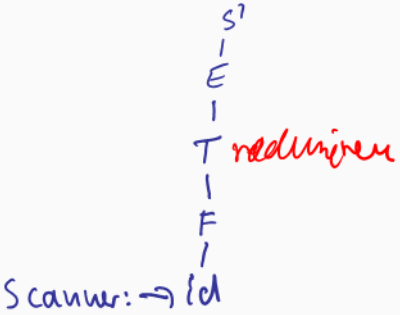
$E \to E + T \mid T$

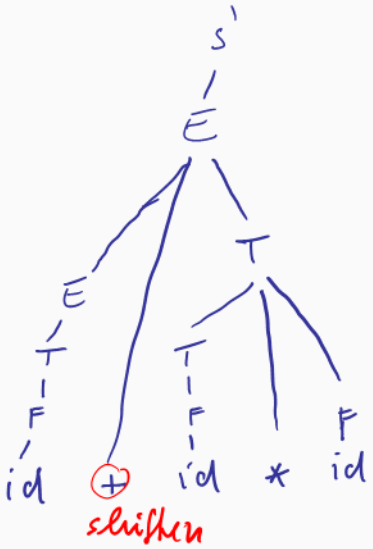
$F \to (E) \mid id$

$T \to T * F \mid F$

zu parsen $id + id * id$

Handle im Stack





Kann ein Stack helfen?

d.h. alle „angefangenen" Unterbäume in einen Stack packen, wenn reduzieren die rechte Seite poppen und das neue Nichtterminal pushen.

S. O.

Wir brauchen Kriterien, die uns helfen, zu entscheiden, ob wir reduzieren, nach welcher Regel, oder schieben.
(= Problem d. Terminals)

Probleme damit?

S. O.

Konfliktfälle

Mehrdeutigkeiten = Konflikte beim Parsen

Es gibt Grammatiken, bei denen nicht aus dem Inhalt des Stacks und dem Eingabezeichen entschieden werden kann, wie fortgefahren wird, auch nicht, wenn man, wie auch schon im Fall LL, eine feste Zahl k von Vorschautoken berücksichtigt. Diese Grammatiken können mehrdeutig sein.

Folgen von falschen Entscheidungen:

- falscher Baum, falsche Bäume
- kein Baum

Mögliche Konflikte

- Reduce-Reduce-Konflikt: Es sind zwei oder mehr verschiedene Reduktionen möglich
- Shift-Reduce-Konflikt: Es kann nicht entschieden werden, ob eine Reduktion oder ein Shift durchgeführt werden soll.

Shiften bedeutet, das nächste Eingabesymbol miteinbeziehen.

LR-Parsing

Da wollen wir hin

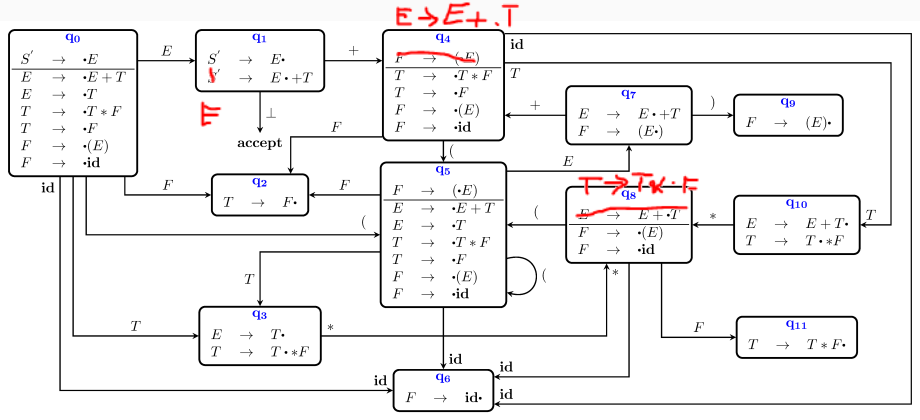


Abbildung 1: Parser-Automat

So geht es

$S \to E$
$E \to E + T \mid T$
$T \to T * F \mid F \quad F \to (E) \mid id$

Der Stack enthält Zustände, keine Terminals oder Nonterminals.

Der Top-of-Stack ist immer der aktuelle Zustand, am Anfang I_0. Im Stack steht I_0 $\perp$.

Vorgehen: Im aktuellen Zustand nachschauen, ob das Eingabezeichen auf einem Pfeil steht.

- ja: Shiften, d. h. dem Pfeil folgen und den Zustand am Ende des Pfeils pushen. Dort weiter.
- nein: Reduzieren nach der Regel aus dem aktuellen Zustand mit dem Punkt hinten, d. h. so viele Zustände poppen, wie die Regel Elemente auf der rechten Seite hat. Der Zustand darunter wird aktuell, dem Pfeil mit dem zu reduzierenden Nonterminal der linken Seite der Regel folgen und pushen.

Am Schluss kann nur noch mit $\perp$ akzeptiert werden.

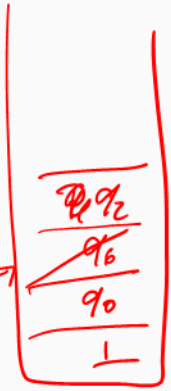
id * id

Beispiel

1) Zustand q_0

2) Scanner liefert "id" $\Rightarrow$ push

3) Scanner liefert "+"
in q_6 gibt es keinen entsprechenden Pfeil mit + $\Rightarrow$ reduzieren,
d.h. E $\Rightarrow$ id anwenden, 3 q_6 poppen,
q_0 wird aktueller Zustand, dann dem Pfeil mit E folgen, ergibt q_2,
pushen



Definitionen

Def.: Bei einer kontextfreien Grammatik G ist die *Rechtsableitung* von $\alpha \in (N \cup T)^*$ die Ableitung, die man erhält, wenn das am weitesten rechts stehende Nichtterminal in α abgeleitet wird. Man schreibt $\alpha \overset{*}{\Rightarrow}_r \beta$.

Def.: Eine *Rechtssatzform* α einer Grammatik G ist ein Element aus $(N \cup T)^*$ mit $S \overset{*}{\Rightarrow}_r \alpha$.

Def.: In dem Syntaxbaum von $S \overset{*}{\Rightarrow}_r \alpha\ A\ w \Rightarrow_r \alpha\ \beta\ w$ einer kontextfreien Grammatik ist β ein *Handle* von der Produktion $A \to \beta$.

LR-Parsing

Bei der *LR*-Analyse eines Wortes w wird w von links nach rechts gelesen, dabei wird die Rechtsableitung von w in G von unten nach oben aufgebaut. Man spricht nicht nicht mehr von Ableitungen, sondern von Reduktionen.

Mehrdeutige Grammatiken können nicht *LR* sein.

- Vor der Konstruktion des Automaten wird die Grammatik um eine neues Nonterminal S' und die neue Produktion $S' \to S$ erweitert. S' ist dann Startsymbol.
- Es wird ein Automat erstellt (s.o.)
- Es wird eine Parse Table aus dem Automaten erstellt, die den Parse-Vorgang steuert, mit Aktionsteil und Sprungteil.

Steuerung des Parsens mittels der Parse Table

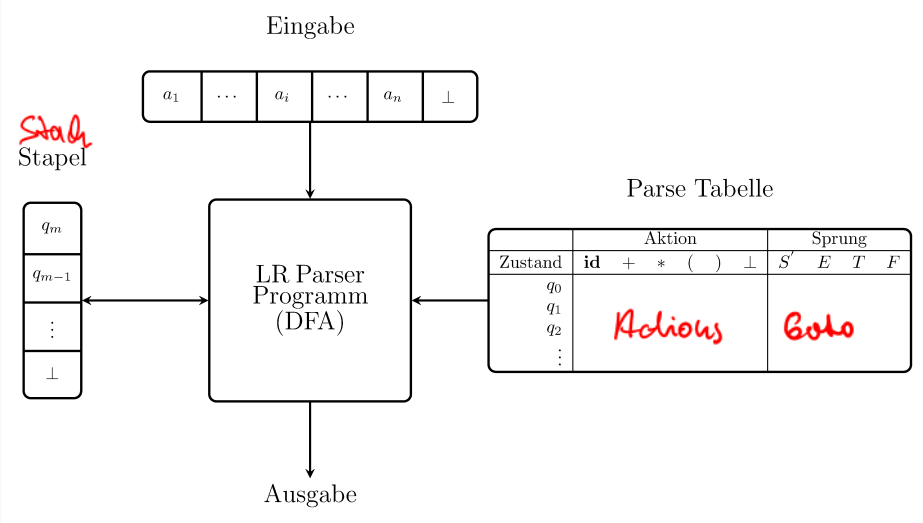


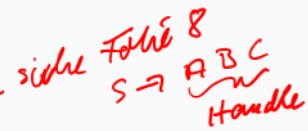
Abbildung 2: Parser Schema

Arbeitsweise

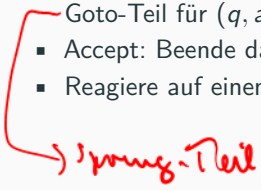
Im Stack stehen nur Zustandsnummern, am Anfang die Nummer des Startzustandes (+ Bottomzeichen, oft auch \$). Es ist nicht nötig, Symbole zu stacken.

- Lesen des obersten Stackelements ergibt Zustand q
- Lesen des nächsten Eingabezeichens ergibt Zeichen a
- Nachschlagen der Reaktion auf (q, a) in der Parse Table
- Durchführung der Reaktion

Mögliche “Actions” ohne Berücksichtigung von Vorschautoken



- Shift: Schiebe logisch das nächste Eingabesymbol auf den Stack (in Wirklichkeit Zustandsnummern)
- Reduce: (Identifiziere ein Handle oben auf dem Stack und ersetze es durch das Nichtterminal der dazugehörigen Produktion.) Das ist gleichbedeutend mit: Entferne so viele Zustände vom Stack wie die rechte Seite der zu reduzierenden Regel Elemente hat, und schreibe den Zustand, der im Goto-Teil für (q, a) steht, auf den Stack.
- Accept: Beende das Parsen erfolgreich
- Reagiere auf einen Syntaxfehler



0 Vorschautoken = LR(0)-Parsing

LR-Parsing ohne Vorschautoken

Wichtig: Das Handle, d. h. die rechte Seite einer zu reduzierenden Regel, erscheint oben auf dem Stack, nie weiter unten.

Je nach Anwendungsfall müssen beim Reduzieren von Handles weitere Aktionen ausgeführt werden: z. B. Syntaxbäume aufgebaut, Werte in Tabellen geschrieben werden, usw. Nicht alle rechten Seiten von Produktionen, die oben auf dem Stack stehen, sind auch Handles, manchmal muss nur geshiftet werden.

Bsp: Steht bei der Beispielgrammatik von Folie 8 oben auf dem Stack ein T mit dem nächsten Eingabezeichen $*$, darf T nicht zu E reduziert werden.

Lösung: Der Parser merkt sich, wo er steht in noch nicht komplett reduzierten Regeln. Dazu benutzt er sogenannte *Items* oder $LR(0) - Items$, auch *dotted Items* oder (*kanonische*) $LR(0) - Elemente$.

// entsprechen den Zuständen des PDAs.

Items

Def.: Ein *Item* einer Grammatik G ist eine Produktion von G mit einem Punkt auf der rechten Seite der Regel vor, zwischen oder nach den Elementen.

Bsp.:

Zu der Produktion $A \to BC$ gehören die Items:

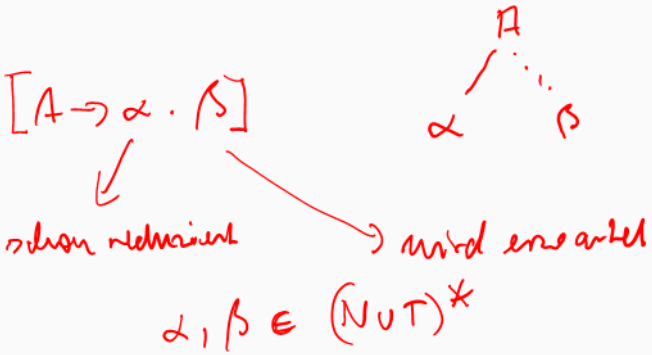
$[A \to \cdot BC]$

$[A \to B \cdot C]$

$[A \to BC\cdot]$

Das zu $A \to \epsilon$ gehörende Item ist $[A \to \cdot]$

Was bedeuten die Items?



Berechnung der *Closure_0* von einer Menge *I* von Items

$[A \to \cdot B]$
$B \to \cdot CD$
$B \to \cdot E$
⋮

$A \to B$
$B \to CD \mid E$

1. füge *I* zu $CLOSURE_0(I)$ hinzu
2. gibt es ein Item $[A \to \alpha \cdot B\beta]$ aus $CLOSURE_0(I)$ und eine Produktion $(B \to \gamma)$, füge $[B \to \cdot\gamma]$ zu $CLOSURE_0(I)$ hinzu

eigentliche Zustände des Automaten

Berechnung der *GOTO_0*-Sprungmarken

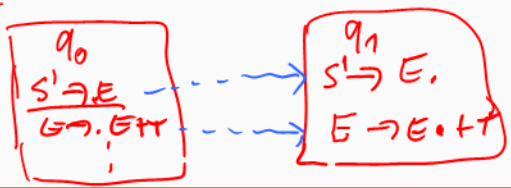
$\forall I, X$:

Item $\in N \cup T$ — neuer Zustand — „alter" Zustand

$$GOTO_0(I, X) = CLOSURE_0(\{[A \to \alpha X \cdot \beta] \mid [A \to \alpha \cdot X\beta] \in I\})$$

für eine Itemmenge I und $X \in N \cup T, A \in N, \alpha, \beta \in (N \cup T)^*$.

berechnet den Folgezustand, wenn wir ein Zeichen I nach X reduzieren wollen



← Das Item war vorher noch nicht drin in q_1.

Konstruktion des $LR(0)$ - Automaten

1. Bilde die Hülle von $S' \to S$ und mache sie zum ersten Zustand.
2. Für jedes noch nicht betrachtete $\cdot X, X \in (N \cup T)$ in einem Zustand q des Automaten berechne $GOTO_0(q, X)$ und mache $GOTO_0(q, X)$ zu einem neuen Zustand r. Verbinde q mit einem Pfeil mit r und schreibe X an den Pfeil. Ist ein zu r identischer Zustand schon vorhanden, wird r mit diesem verbunden und kein neuer erzeugt.

Konstruktion der Parse Table

1. Erstelle eine leere Tabelle mit den Zuständen als Zeilenüberschriften. Für den Aktionstabellenteil überschreibe die Spalten mit den Terminalen, für den Sprungtabellenteil mit den Nonterminals.
2. Shift: Für jeden mit einem Terminal beschrifteten Pfeil aus einem Zustand erstelle in der Aktionstabelle die Aktion shift mit der Nummer des Zustands, auf den der Pfeil zeigt. Für Pfeile mit Nonterminals schreibe in die Sprungtabelle nur die Nummer des Folgezustands.
3. Schreibe beim Zustand $[S' \to S\cdot]$ ein *accept* bei dem Symbol $\perp$.
4. Für jedes Item mit $[A \to \beta\cdot]$ aus allen Zuständen schreibe für alle Terminals *reduce* und die Nummer der entsprechenden Grammatikregel in die Tabelle.

Und wenn in einer Zelle schon ein Eintrag ist?

Die Beispielgrammatik G1

(0) $S' \rightarrow S$

(1) $S \rightarrow aAbScS$

(2) $S \rightarrow aAbS$

(3) $S \rightarrow d$

(4) $A \rightarrow e$

Der LR(0)-Automat zu G1

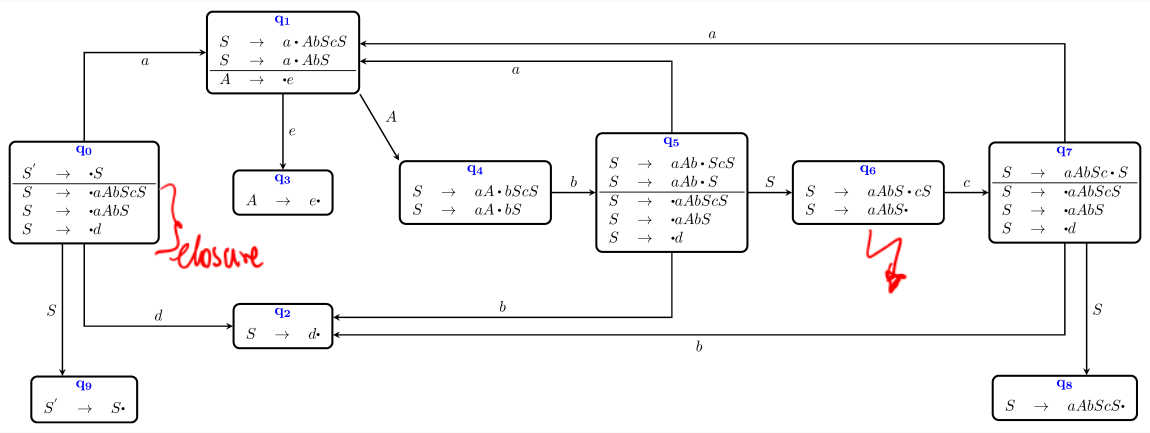


Abbildung 3: LR(0)-Automat

Die LR(0)-Parsertabelle zu G1

	Aktion						Sprung	
Zustand	a	b	c	d	e	$\perp$	S	A
q_0	s1			s2			q_9	
q_1					s3			q_4
q_2	r3	r3	r3	r3	r3	r3		
q_3	r4	r4	r4	r4	r4	r4		
q_4		s5						
q_5	s1	s2		s2			q_6	
q_6	r2	r2	s7,r2	r2	r2	r2		
q_7	s1	s2					q_8	
q_8	r1	r1	r1	r1	r1	r1		
q_9						acc		

Abbildung 4: LR(0)-Parsertabelle

Wrap-Up

Wrap-Up

- LR-Analyse baut den Ableitungbaum von unten nach oben auf.
- Es wird ein DFA benutzt zusammen mit einem Stack, der Zustände speichert.
- Eine Parse-Tabelle steuert über Aktions- und Sprungbefehle das Verhalten des Parsers.
- Die Tabelle wird mit Items und Closures konstruiert.

LICENSE