Compilerbau

LR(K)-Parser

Prof. Dr. Franz-Karl Schmatzer
schmatzf@dhbw-loerrach.de

Literatur

- C.Wagenknecht, M.Hielscher; Formale Sprachen, abstrakte Automaten und Compiler; 3.Aufl. Springer Vieweg 2022;
- A.V.Aho, M.S.Lam,R.Savi,J.D.Ullman, *Compiler – Prinzipien,Techniken und Werkzeuge*. 2. Aufl., Pearson Studium, 2008.
- Güting, Erwin; *Übersetzerbau –Techniken, Werkzeuge, Anwendungen*, Springer Verlag 1999

Agenda

- **LR(K) Parser**
  - **Einführung**
  - **Prinzip der Bottom-Up-Analyse**
  - **Aufbau LR(K) Parser**

LR(K)-Sprachen

- LR(k)-Sprachen stellen die umfassendste Klasse deterministisch analysierbarer kontextfreier Sprachen dar.
- Aus der Theorie formaler Sprachen ist bekannt, dass genau diese Klasse durch deterministische Kellerautomaten beschrieben wird.
- Da es sich um ein Bottom-up-Verfahren handelt, wird die jeweils betrachtete Satz- Bottom-upform, die einer rechten Regelseite entspricht, durch das zugehörige Nichtterminal Verfahren auf der linken Seite dieser Regel ersetzt

Bottom-Up-Analyse

Einführung

- Man beginnt bei den Blättern und baut den Baum auf
- Idee:
  - Man liest so lange Token von der Eingabe ein, bis eine vollständige rechte Seite einer Grammatikregel erreicht wird.
  - Dann werden diese Tokens durch die linke Seite dieser Grammatikregel ersetzt.
  - D.h. es werden Liste von Teilbäume erstellt , bis am Schluss der vollständige Baum entsteht.
- Beispiel:
  - Beim Lesen von id[1] erhält man den ersten Teilbaum

numexpr

if id

if id[1] cop[>] const[2] then id[2] := const[1] end

Bottom-Up-Analyse
Prinzip – Beispiel Grammatik

- Betrachte folgende einfache Grammatik für arithmetische Ausdrücke:

  E → E+T

  E → T

  T → T*F

  T → F

  F → (E)

  F → id

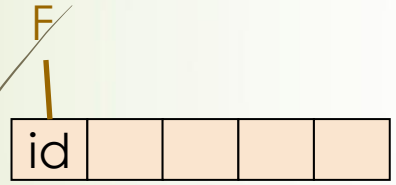
- Gegeben die Tokenfolge: id+id*id

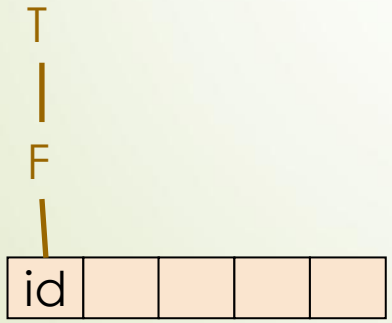
Bottom-Up-Analyse

Prinzip – Erstellen des Syntaxbaums

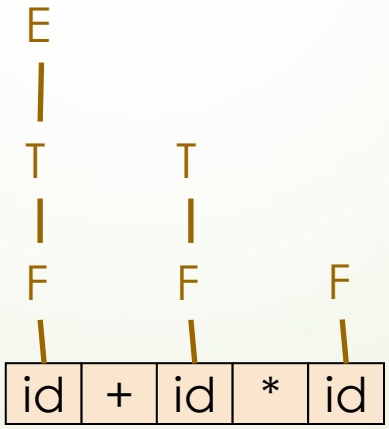
- Aufbau des Baums zu id+id*id

E	→	E+T
E	→	T
T	→	T*F
T	→	F
F	→	(E)
F	→	id

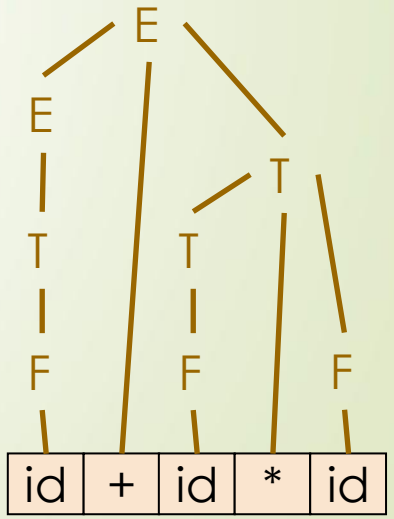




Nach einigen weiteren Schritten



Am Ende der Ableitung



Aufgabe
Ableitung

- Implementieren Sie die Grammatik in FLACI

  E -> E + T

  E -> T

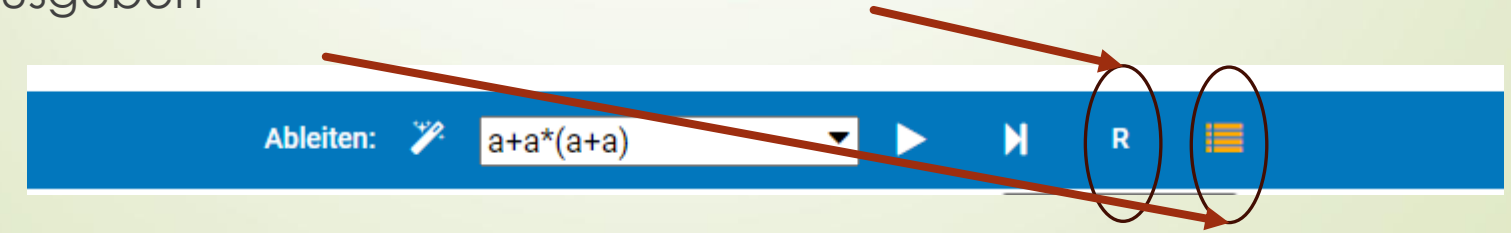
  T -> T * F

  T -> F

  F -> (E)

  F -> a

- Bauen Sie den Ableitungsbaum In Bottom-Up Manier auf.
- Leiten Sie das Wort w= a+a*(a+a) ab indem Sie eine Rechtsableitung durchführen.
- In FLACI können sie die Ableitung L/R steuern und die Satzform ausgeben



Bottom-Up-Analyse

Prinzip

- Wir kehren nun die Reihenfolge der Rechtsableitung
  - Wir notieren die einzelnen Satzformen als Konfigurationensfolge eines DKA, der so etwas wie eine Linksreduktion (vom Wort zum Spitzensymbol hin) simuliert.
  - Um das Ende des Eingabewortes zu kennzeichnen, verwenden wir ein Dollarzeichen $.
  - Als Aktion vermerken wir
    - **shift** bedeutet, dass das nächste Token aus dem Eingabepuffer (Restwort) entfernen und auf den Stapel legen.
    - **reduce**: $X \rightarrow \beta$ bedeutet, dass der Stapelinhalt gemäß der Regel ¨ $X \rightarrow \beta$ reduziert wird. Die rechte Regelseite β stimmt genau mit dem obersten Stapel(teil)wort uberein. Genau dieser Stapelinhalt wird durch die linke Regelseite, also X, ersetzt.
    - **accepted** steht ganz am Ende, wenn das Startsymbol der Grammatik als einziges accepted Zeichen im Keller steht und der Puffer für das Eingabewort leer ist

Bottom-Up-Analyse

Prinzip – Beispiel

- Analysieren der Tokenfolge: id+id*id

E	→	E+T
E	→	T
T	→	T*F
T	→	F
F	→	(E)
F	→	id

Ableitung	Tokenfolge	Schritt
id	id+id*id	Shift
F	+id*id	Reduce
T	+id*id	Reduce
E	+id*id	Reduce
E+	id*id	Shift
E+id	*id	Shift
E+F	*id	Reduce
E+T	*id	Reduce
E	*id	Reduce

Ableitung	Tokenfolge	Schritt
E+T*	id	Shift
E+T*id		Shift
E+T*F		Reduce
E+T		Reduce
E		Reduce

Der letzte Reduce-Schritt hätte man nicht ausführen dürfen, sondern weiter Zeichen lesen ⇒ **Problem der Reduce-Shift-Technik**

Bottom-Up-Analyse

Prinzip – Beispiel

- Schreiben in umgekehrter Reihenfolge mit Rechtsableitung

  $E \Rightarrow E+T \Rightarrow E + T*F \Rightarrow E + T*id \Rightarrow E + F*id \Rightarrow E + id*id \Rightarrow T + id*id \Rightarrow F + id*id \Rightarrow id + id*id$

- Problem: Wann soll eine vollständige rechte Seite reduziert werden und wann nicht?
- Def: Sei G eine kontextfreie Grammatik G und sei

  $$S \Rightarrow^* \alpha Aw \Rightarrow^* \alpha\beta w$$

  eine Rechtsableitung in G. Dann heißt β ein Handle der Rechtssatzform $\alpha\beta w$.
- Frage: Wie findet man Handels?
- Def: Sei G eine kontextfreie Grammatik G und sei

  $$S \Rightarrow^* \alpha Aw \Rightarrow^* \alpha\beta w$$

  eine Rechtsableitung in G. Jedes Anfangsstück $\alpha\beta$ heißt geeignetes Präfix von G.

Bottom-Up-Analyse

Prinzip – Probleme

- Bei der Ableitung ist ein wesentliches Problem aufgetreten
- Wann liegt ein geeignetes Handle vor?
- Dazu betrachten wir die Klasse der LR(K)-Parser
  - Notation wie bei den LL(K) Parser
  - L: Lesen von links nach rechts
  - R: Rechtsableitung
  - K: K Token wird vorausschauend gelesen.
- Vorteile der LR(K)-Parser
  - Praktisch alle Programmiersprachen lassen sich damit analysieren
  - Allgemeinste Shift-Reduce-Technik, die ohne Backtracking auskommt
  - LR-Verfahren sind mächtiger als LL-Verfahren
  - LR-Parser erkennen mögliche Fehler sehr früh bei der Eingabe.
- Nachteile
  - Die Analysetabellen lassen sich von Hand kaum erstellen.
  - Aber es gibt Tools wie yacc, die solche Parser implementieren.

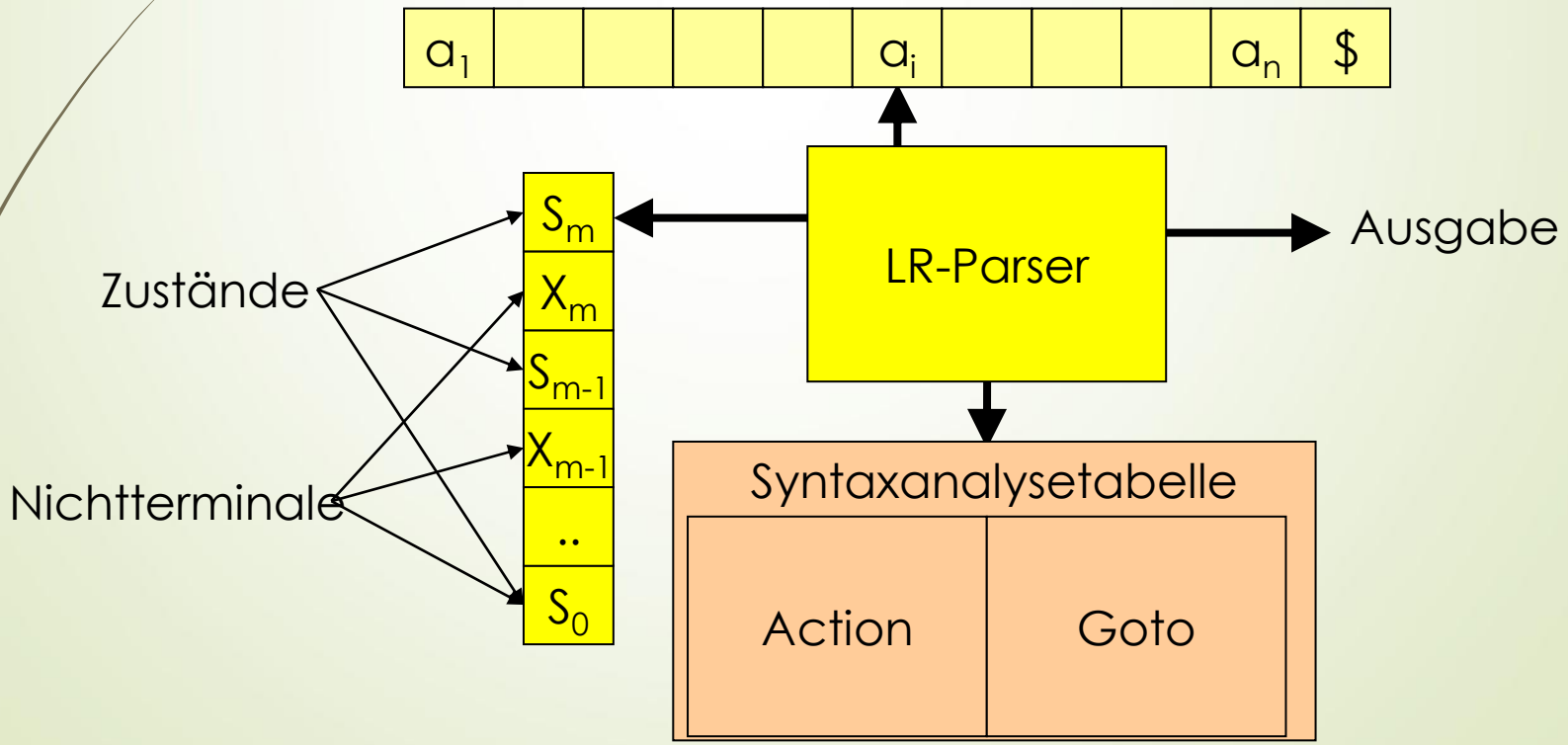
Aufgabe LR-Parser
Shift, reduce, accepted

- Führen Sie den Parservorgang mit dem Wort w = a+a*(a+a)

Bottom-Up-Analyse

Aufbau LR(K)-Parser

- LR-Parser als abstrakte Maschine
  - Auf dem Stack werden die Zustände und Grammatiksymbole der Maschine verwaltet. (Zustände allein reichen, aber zum besseren Verständnis werden die Grammatiksymbole mit auf den Stack gelegt)
  - Die Syntaxanalysetabelle (Action, Goto) steuert die Maschine



Bottom-Up-Analyse

Aufbau der Syntaxanalysetabelle

- Der **Actionteil** der Tabelle enthält für jeden Zustand und jedem Terminalsymbol der Grammatik einen Eintrag.
  - Man unterscheidet 4 verschiedene Einträge
    - shift s, wobei s der neue Zustand ist
    - reduce $A \rightarrow \beta$
    - accept
    - error
- Der **Gototeil** der Tabelle enthält für jeden Zustand s und jedes Nichtterminal einen Zustand s‘ als Eintrag und überführt den Automaten von seinem alten Zustand s in den neuen Zustand s‘.

Bottom-Up-Analyse

Beispiel Syntaxanalysetabelle

Für die Grammatik G ergibt sich folgende Syntaxanalyse-Tabelle:

E → E+T (1)
E → T (2)
T → T*F (3)
T → F (4)
F → (E) (5)
F → id (6)

Notation:
s:shift, r:reduce, acc:accept
r1: reduce mit Produktion 1
s5: Shift in den Zustand 5

	Action						Goto		
Zustand	**id**	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Bottom-Up-Analyse

Syntaxanalysetabelle Funktionsweise

- Zu Beginn
  - Eingabezeiger steht auf dem ersten Zeichen.
  - Der Stack ist mit dem Startzustand s_0 initialisiert.
- In jedem Schritt betrachtet der Parser den oberen Wert s_m des Stacks und das Eingabezeichen a_i.
  - Falls action[s_m,a_i] = shift s: So wird das Eingabesymbol a_i und der Zustand s auf den Stack gelegt.
  - Falls action[s_m,a_i] = reduce $A \rightarrow \beta$: Die Symbole von β und die zugehörigen Zustände werden vom Stack entfernt (d.h. 2* | β | Einträge). Sei nun s‘ der oberste Zustand. Das Nichterminal A und der Zustand s_n, der mit der Gototeil der Tabelle Goto[s‘,A] = s_n bestimmt wird, werden auf den Stack gelegt. Die Produktion $A \rightarrow \beta$ wird ausgegeben.
  - Falls action[s_m,a_i] = accept: Das Parsen ist erfolgreich beendet
  - Falls action[s_m,a_i] = error: Eine Fehlermeldung wird ausgegeben.

Bottom-Up-Analyse

Syntaxanalysetabelle Funktionsweise

- Analysieren der Tokenfolge: id+id*id

Startzustand 0

Speichern von Symbol id und wechseln in Zustand 5

E	→	E+T	(1)
E	→	T	(2)
T	→	T*F	(3)
T	→	F	(4)
F	→	(E)	(5)
F	→	id	(6)

Stack	Eingabe	Action
0	id+id*id$	s5
0id5	+id*id$	r6 (F→id)
0F3	+id*id$	r4 (T→F)
0T2	+id*id$	r2 (E→T)
0E1	+id*id$	s6
0E1+6	id*id$	s5
0E1+6id5	*id$	r6 (F→id)
0E1+6F3	*id$	r4 (T→F)
0E1+6T9	*id$	s7
0E1+6T9*7	id$	s5
0E1+6T9*7id5	$	r6 (F→id)
0E1+6T9*7F10	$	r3 (T→T*F)
0E1+6T9	$	r1 (E→E+T)
0E1	$	acc