Proseminar

Effizientes Parsen und Generieren mit Baumadjunktionsgrammatiken

- Deterministisches Parsen -

Jens Haase

21. Januar 1998

Diese Ausarbeitung stellt das Papier
„*Deterministic left to right parsing of Tree Adjoining Languages*“
von Yves Schabes und K. Vijay-Shanker vor.

Inhaltsverzeichnis

EINLEITUNG **1**

AUTOMATENMODELLE **1**

PDA Pushdown Automaton (Kellerautomat) 1
EPDA Embedded Pushdown Automaton (Einbettender Kellerautomat) 3
BEPDA Bottom-Up Embedded Pushdown Automaton (Aufwärtiger Einbettender Kellerautomat) 4

LR PARSING **5**

Ein LR Parser 5
Ein Parsing Beispiel 8
Eine LR(0) Parsingtabelle 10
Konstruktion eines Zustandsdiagramms *11*
Konstruktion einer Parsingtabelle *13*
Korrekte Präfixeingenschaft *14*

ABKÜRZUNGSVERZEICHNIS ..... **I**

INDEX ..... **I**

Einleitung

Im folgenden Vortrag wird ein Verfahren vorgestellt, das aus einer gegebenen *Baumadjunktionsgrammatik (TAG)* einen Automaten erstellt, der die zugehörige Sprache deterministisch parsen kann.

Wir werden einen *Kellerautomaten (PDA)* derart zu einem *Einbettenden Kellerautomaten (EPDA)* erweitern, daß er *Baumadjunktionssprachen (TALs) top-down* verarbeiten kann. Anschließend wird der *Einbettende Kellerautomat* in seiner Funktionsweise zu einem *Aufwärtigen Einbettenden Kellerautomaten (BEPDA)* sozusagen umgekehrt, um Baumadjunktionssprachen bottom-up zu verarbeiten.

Im zweiten Teil werden wir einen *LR Parser* für TAGs kennenlernen und uns seine Arbeitsweise an einem Beispiel verdeutlichen.

Automatenmodelle

PDA PUSHDOWN AUTOMATON (KELLERAUTOMAT)

Ich gehe davon aus, daß der *Kellerautomat* bekannt ist. Da jedoch die weiteren Automaten direkt auf diesen aufbauen, wird er zur Erinnerung in aller Kürze wiederholt.

Ein PDA besteht aus einem schreibgeschützten Eingabeband, einem Keller (stack) und einer Zustandskontrolle mit einer Übergangstabelle (siehe Abbildung 1).

Der Automat liest das Eingabeband zeichenweise von links nach rechts. Dabei darf er ein Zeichen auf dem Eingabeband lesen, das oberste Zeichen auf dem Keller lesen und ablegen und seinen Zustand ändern. Bei jedem Schritt wandert der Zeiger auf dem Eingabeband ein Zeichen weiter, und das oberste Kellerzeichen wird ersetzt (ggf. durch das gleiche Zeichen oder das leere Wort).

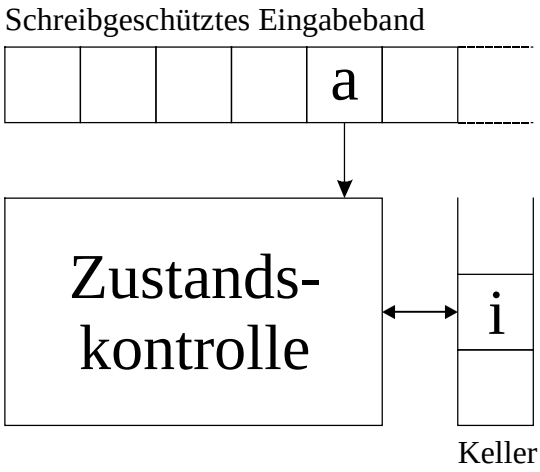

Abbildung 1: Kellerautomat

Kellerautomaten sind in der Lage, kontextfreie Sprachen zu verarbeiten.

Zum Beispiel verarbeitet ein Kellerautomat die Sprache $a^n b^n$, indem er für jedes a ein Zeichen auf den Keller schreibt, bis das erste b gelesen wird. Anschließend geht er in einen anderen Zustand über und nimmt für jedes gelesene b ein a vom Keller. Ist der Keller gleichzeitig mit dem Wortende leer, wird das Wort akzeptiert.

Baumadjunktionssprachen sind mit einem einfachen Kellerautomaten nicht zu verarbeiten. Ein einfaches Beispiel ist schon die Sprache $a^n b^n c^n d^n$. Nachdem der Kellerautomat die Anzahl der Buchstaben a mit der Anzahl der Buchstaben b nach der obigen Methode verglichen hat, ist die Information über die Anzahl der Zeichen verloren, so daß der Vergleich mit den Buchstaben c und d nicht mehr möglich ist.

Baumadjunktionsgrammatiken unterscheiden sich von kontextfreien Grammatiken hauptsächlich in der Möglichkeit, bei der Adjunktion ganze Teilbäume (Teilzeichenketten) einzuschließen, oder anders ausgedrückt, *einzubetten*. Genau diese Fähigkeit fehlt dem Kellerautomaten. Er läßt sich allerdings leicht dahingehend erweitern.

EPDA EMBEDDED PUSHDOWN AUTOMATON (EINBETTENDER KELLERAUTOMAT)

Die Erweiterung des *Einbettenden Kellerautomaten* besteht darin, daß der Keller durch einen Keller aus Kellern ersetzt wird. Zusätzlich zu den Funktionen des PDA darf der EPDA nicht nur Zeichen auf den obersten Keller schreiben, sondern gleichzeitig jeweils einen neuen Keller über und unter dem obersten Keller anlegen, ihn also *einbetten*.

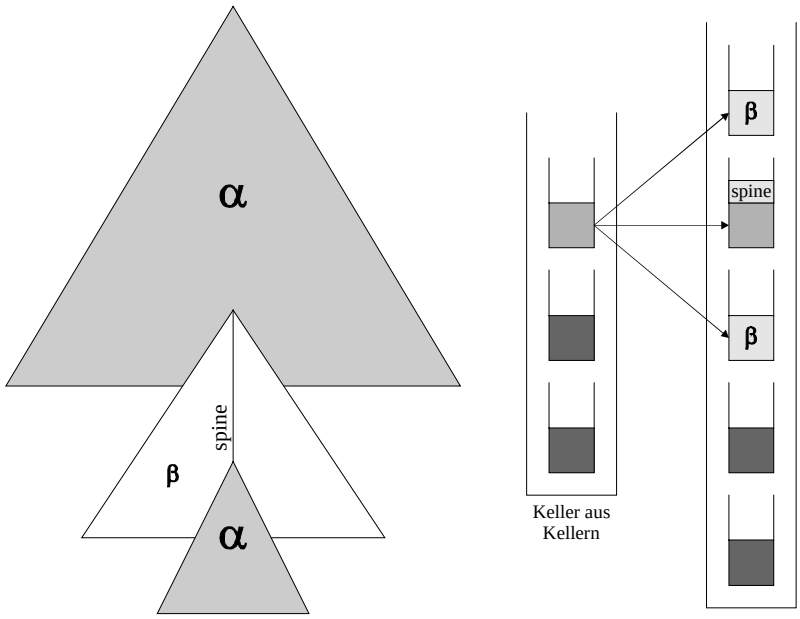

Abbildung 2: Keller des EPDA

Konkret wird eine Adjunktion umgesetzt, indem das Rückrad (spine) des eingesetzten Baumes auf den obersten Keller geschrieben wird. Die linke bzw. rechte Seite des Baumes wird auf einen neuen Keller über bzw. unter dem ehemals obersten Keller abgelegt (siehe Abbildung 2). Für jede Baumadjunktionsgrammatik kann ein EPDA erstellt werden, der jeden Ableitungsschritt auf seinem Keller aufbauen, und somit alle Worte der Sprache erzeugen kann.

Versucht man jetzt ein gegebenes Wort einer Baumadjunktionssprache zu parsen, müßte man in der Lage sein, das Wort nicht auf- sondern abzubauen. Es bietet sich daher als ein weiterer Schritt in diese Richtung an, den EPDA gewissermaßen umzukehren.

BEPDA Bottom-Up Embedded Pushdown Automaton (Aufwärtiger Einbettender Kellerautomat)

Der *Aufwärtige Einbettende Kellerautomat* ist ähnlich aufgebaut wie der EPDA, nur sind seine Funktionen entgegengesetzt. Anstelle der oben beschriebenen Einbettung besitzt des BEPDA die Funktion, mehrere Keller über und unter einem Keller in einem Schritt zu löschen (unwrap) und den erhaltenen Keller dabei zu reduzieren. Genaugenommen wäre in der inversen Funktion der Einbettung nur der Zugriff auf die obersten drei Keller erlaubt. Diese Einschränkung würde allerdings die Verarbeitung auf Normalformen reduzieren. Die inverse Funktion zum Löschen eines Kellers, aus dem das letzte Zeichen entfernt wurde, ist das Anlegen eines neuen, obersten Kellers mit einem Zeichen (push).

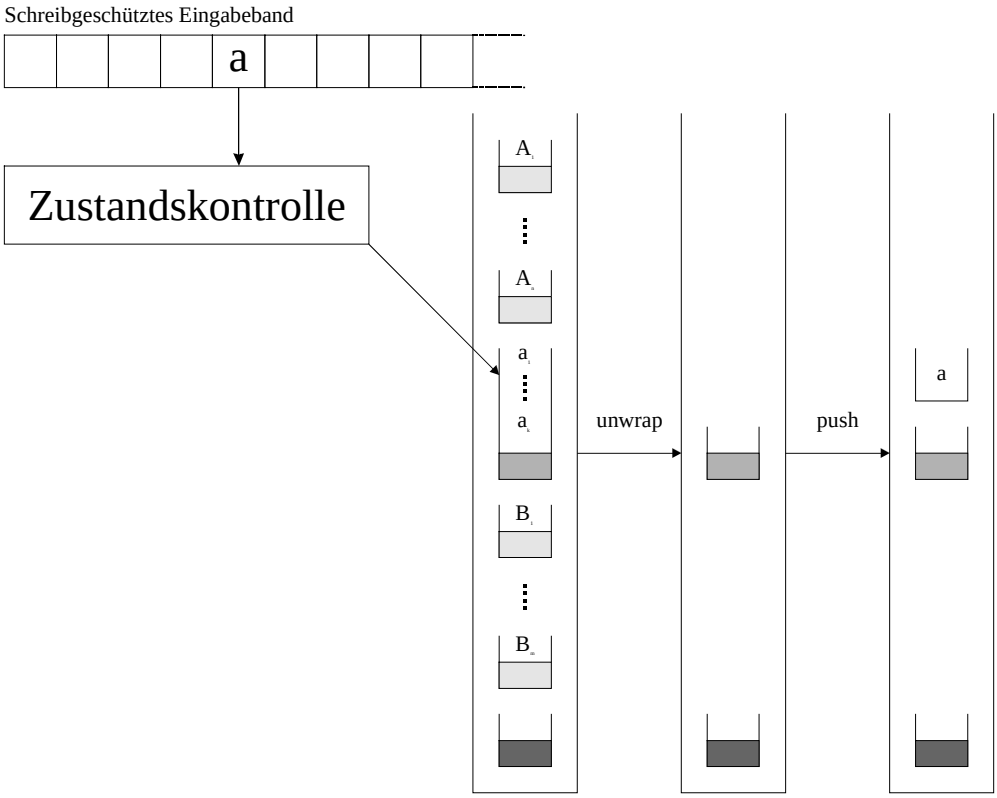

Abbildung 3: BEPDA mit Kellerübergängen

Diese Funktionen ermöglichen es anstelle des Aufbaus eines Wortes nach einer TAG (top-down), ein gegebenes Wort nach den Regeln einer TAG abzubauen (bottm-up).

Der BEPDA ist also in seiner Funktionalität das Gegenstück zum EPDA.

LR[1] Parsing

Die Idee ist relativ einfach. Der Parser arbeitet das Eingabewort zeichenweise ab. Man stelle sich vor, daß eine Markierung, die durch einen Punkt[2] gekennzeichnet wird, außen am abgeleiteten Baum herumwandert. Der Parser schreibt die Information der bisher gelesenen Zeichen als Zustände auf die Keller, wobei unter einem Zustand alle möglichen Markierungen in einem Baum zusammengefaßt werden (siehe Abbildung 7 aus dem Beispiel). Dabei versucht der Parser das passende Vorkommen aller Terminale eines auxiliaren Baumes zu finden (nicht notwendigerweise hintereinander), bis er daraus eine letzte Adjunktion schließen kann, welche er aus dem Baum entfernt. So wird der Baum sozusagen von innen nach außen aufgelöst.

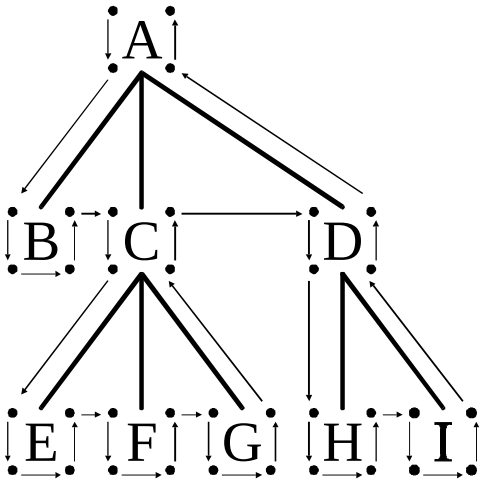

Abbildung 4: dotted Tree

> ***Anmerkung:*** *Es folgt eine reine Beschreibung eines LR Parsers. Bevor ihr verzweifelt, schlage ich vor, diese zu überfliegen und am nachfolgenden Beispiel genauer nachzuvollziehen.*

EIN LR PARSER

Ein LR Parser besteht aus einer Eingabe, einer Ausgabe, einem Keller aus Kellern, einer Übergangsfunktion und einer dreiteiligen Parsingtabelle. Die ersten vier Komponenten sind die gleichen für alle LR Parser, nur die Parsingtabelle variiert abhängig von der TAG. Um eine Konfiguration des Parsers anzugeben, werden zuerst die Keller von unten nach oben, dann das verbleibende Eingabewort notiert. Das Zeichen || markiert den Boden eines Kellers, das Zeichen $ das Eingabeende:

$$(||t_1...t_m||...||s_1...s_w,a_ra_{r+1}...a_n.\$)$$

[1] LR ist lediglich die Abkürzung für „left-right“.

[2] Zum Begriff „punktierter Baum“ siehe Vortrag „Earley-basiertes Parsing“ von Dominik Heckmann.

Der Parser wird mit dem Zustand 0 und dem Eingabewort initialisiert:

$(||0,a_1...a_n\$)$

Die Übergangsfunktion liest in jedem Schritt ein Zeichen vom Eingabeband und legt neue Zustände auf dem Keller aus Kellern ab. Abhängig vom gelesenen Zeichen und dem obersten Kellerzeichen (also dem Zustand) sucht die Übergangsfunktion ihre nächste Aktion aus dem ersten Teil der Parsingtabelle, der Aktionstabelle. Im zweiten und dritten Teil der Aktionstabelle stehen zwei Sprungaktionen ($GOTO_{right}$ und $GOTO_{foot}$), um im Falle einer gefundenen oder abgeschlossenen Adjunktion an einem anderen Baum fortfahren zu können. Die Sprünge bekommen einen Zustand und einen auxiliaren Baum, um einen neuen Zustand zu erzeugen.

Aus folgenden fünf Aktionen bestehen die Einträge der Aktionstabelle:

- **Shift j** (sj): legt einen neuen Keller mit dem Zustand j an (push).

  In diesem Schritt werden nur Möglichkeiten auf den Kellern vermerkt.

- **Resume Right** vom Baum δ an Knoten *dot* (rsδ@*dot*).

  Der Parser hat die rechte untere Seite eines Knotens η erreicht, an dem eine Adjunktion stattgefunden hat (siehe Zustand 4 und 11 in Abbildung 7). Die Information, welcher Baum hier adjungiert wurde, muß aus den Kellern entnommen werden.

  Erster Fall: η subsumiert keinen Fußknoten. Sei k die Anzahl der Terminale, die von η subsumiert werden, und die aktuelle Konfiguration sei:

  $(||...||i_k||...||i_1||i,a_r...a_n\$)$

  Dann werde die ersten k Keller zu einem Keller zusammengefaßt, und der Keller ||m wird angelegt, mit $m=GOTO_{foot}[i_k,\beta]$, wobei β einen auxiliaren Baum kennzeichnet, der an Knoten η subsumiert werden kann:

  $(||...||i_k||i_{k-1}... i_1i||m,a_r...a_n\$)$

  Zweiter Fall: η subsumiert den Fußknoten von δ. Sei k die Anzahl der Terminale, die rechts vom Fußknoten von η subsumiert werden, und k‘ die Anzahl zur Linken. Die aktuelle Konfiguration sei:

  $(||...||n_{k'+1}||...||n_1||s_1...s_l||i_k...||i_1||i,a_r...a_n\$)$

Die k‘ Keller unter dem (k+2)-ten Keller und die (k+1)-sten Keller werden auf den (k+2)-ten Keller geschrieben, und der Keller ||m wird angelegt, mit $m=GOTO_{foot}[n_{k'+1},\beta]$:

$(||...||n_{k'+1}||s_1...s_ln_{k'}...n_1i_k...i_1i||m,a_r...a_n\$)$

- **Reduce Root** vom Baum β, in dem die letzte Adjuktion am Knoten *star* stattgefunden hat (rdβ@*star*).

  Der Parser hat einem auxiliaren Baum vollständig umrundet, löscht dessen Informationen aus den Kellern und fährt in dem Baum fort, in den dieser adjungiert wurde (siehe Zustand 8 und 13 in Abbildung 7).

  Sei k bzw. k‘ die Anzahl der Terminale links bzw. rechts der Fußnote von β. Sei ξ der Knoten, der von *star* markiert wird. Sei p die Anzahl der Terminale, die links vom Fußknoten von ξ subsumiert werden. Dann werden p+k‘+1 Zeichen von den Kellern genommen. Anschließend werden k-p Zeichen unter dem neuen, obersten Keller aus den Kellern genommen (siehe „unwrap“, Abbildung 3). Sei j der neue oberste Zustand. Dann wird ein neuer Keller ||m angelegt, mit $m=GOTO_{right}[j,\beta]$.

- **Accept**

  Der Parser akzeptiert das Eingabewort.

- **Error**

  Der Parser lehnt das Eingabewort ab. Diese Aktion wird auch durch einen fehlenden Eintrag in der Parsingtabelle ausgedrückt.

Verfolgt man die *reduce root* Aktion, kann man sogar den Weg des Parsens angeben, anstatt ihn nur zu akzeptieren oder zu verwerfen.

EIN PARSING BEISPIEL

Der eben beschriebene LR Parser soll im folgenden Abschnitt am Beispiel des Parsens eines konkreten Wortes verdeutlicht werden. Als Beispiel dient das Eingabewort *aabbeccdd*, das mit der TAG in Abbildung 6 erzeugt wurde.

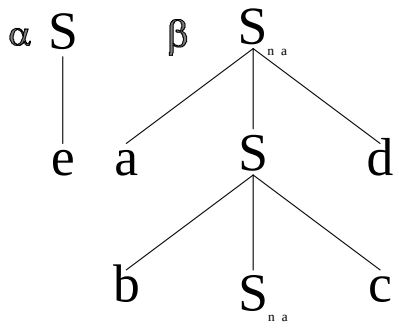

Abbildung 6: TAG $a^n b^n e c^n d^n$

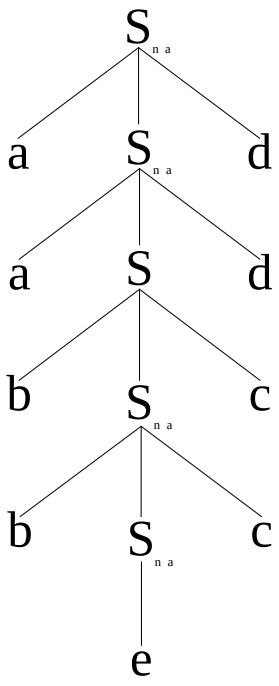

Abbildung 5: Abgeleiteter Baum

Tabelle 1: Beispiel einer Parsingtabelle

	PARSING ACTION						GOTO	
							foot	*right*
	a	b	c	d	e	$	β	β
0	s2				s1			
1						acc		
2	s2	s3						
3		s9			s4		6	
4	rsα@0	rsα@0	rsα@0	rsα@0	rsα@0	rsα@0		5
5						acc		
6			s7					
7				s8				
8	rdβ@-	rdβ@-	rdβ@-	rdβ@-	rdβ@-	rdβ@-		
9		s9			s4		10	
10			s11					
11	rsβ@2	rsβ@2	rsβ@2	rsβ@2	rsβ@2	rsβ@2		12
12				s13				
13	rdβ@2	rdβ@2	rdβ@2	rdβ@2	rdβ@2	rdβ@2		

Wie man auf den ersten Blick sieht, besteht das Beispielwort *aabbeccdd* aus einer Adjunktion des auxiliaren Baumes β in den elementaren Baum α und einer zweiten Adjunktion des Baumes β in den adjungierten Baum β.

Tabelle 2: Parsingbeispiel

Step	Parser configuration	Next move
1	(‖0,aabbeccdd$)	s2
2	(‖0‖2,abbeccdd$)	s2
3	(‖0‖2‖2,bbeccdd$)	s3
4	(‖0‖2‖2‖3,beccdd$)	s9
5	(‖0‖2‖2‖3‖9,eccdd$)	s4
6	(‖0‖2‖2‖3‖9‖4,ccdd$)	rsα@0
7	(‖0‖2‖2‖3‖9‖4‖10,ccdd$)	s11
8	(‖0‖2‖2‖3‖9‖4‖10‖11,cdd$)	rsβ@2
9	(‖0‖2‖2‖3‖4 9 10 11‖6,cdd$)	s7
10	(‖0‖2‖2‖3‖4 9 10 11‖6‖7,dd$)	s8
11	(‖0‖2‖2‖3‖4 9 10 11‖6‖7‖8,d$)	rdβ@-
12	(‖0‖2‖4 9 10‖12,d$)	s13
13	(‖0‖2‖4 9 10‖12‖13,$)	rdβ@2
14	(‖0‖5,$)	acc

Die Schritte 1-5 lesen die linke Seite des abgeleiteten Baumes ein und ordnen diesen Möglichkeiten in Form von Zuständen zu. In Schritt 6 wird der rechte untere Knoten einer Adjunktion im Baum α erreicht. Der Parser führt die Aktion *resume right* in Baum α an Knoten 0 aus, liest dann in Schritt 7 ein weiteres Zeichen und gelangt in Schritt 8 wieder an eine Adjunktion im Baum β. Diesmal führt er die Aktion *resume right* in Baum β an Knoten 2 aus. Nach dem Lesen von zwei weiteren Zeichen gelangt der Parser in Schritt 11 ans Ende der zweiten Adjunktion des Baumes β in dem keine weitere Adjunktion mehr stattgefunden hat. Er löscht also mit der Aktion *reduce root* den Baum β aus den Kellern ohne Angabe einer weiteren Adjunktion. Nach dem Lesen eines weiteren Zeichens gelangt er in Schritt 13 an das Ende der ersten Adjunktion und löscht diese mit der Aktion *reduce root*, wobei er hier angibt, daß in Knoten 2 eine weitere Adjunktion stattgefunden hatte. Schritt 14 erreicht das Ende des elementaren Baumes α und akzeptiert das Eingabewort.

EINE LR(0)[3] PARSINGTABELLE

An dieser Stelle drängt sich die Frage auf, woher man eine geeignete Parsingtabelle bekommt. Zugegebenermaßen erscheint so eine Parsingtabelle nicht gerade trivial. Glücklicherweise gibt es zu diesem Problem einen Algorithmus, der aus einigen (leider nicht allen) TAGs eine LR(0) Parsingtabelle für einem endlichen Automaten erzeugt.

In einem Zustand des Automaten werden alle Bäume zusammengefaßt, in denen wir uns zu einem Verarbeitungszeitpunkt befinden könnten. Die Stelle, an der wir uns befinden könnten, wird durch einen Punkt markiert, eine Adjunktion im Baum durch einen Stern. Dabei sind alle Zustände unter folgenden Operationen abgeschlossen:

- **Voraussage einer Adjunktion**
  Ist ein Baum mit einem Punkt vor einem Nichtterminal in einem Zustand, in das ein weiterer Baum adjungiert werden kann, so gehört dieser weitere Baum auch zu dem Zustand mit einem Punkt vor der Wurzel.

- **Linker Abschluß**
  Ist ein auxiliarer Baum mit einem Punkt vor dem Fußknoten in einem Zustand, so gehören auch alle Bäume zu dem Zustand, in die dieser Baum hätte adjungiert werden können, mit einem Punkt hinter dem passenden Nichtterminal.

- **Kantenüberschreitung**
  Gehört ein Baum mit einem Punkt „hinter“ einem Knoten zu einem Zustand, gehört auch der gleiche Baum mit einem Punkt „vor“ dem nächsten Knoten zu dem Zustand. In diesem Fall wird nur eine Kante überschritten.

- **Nichtterminalüberschreitung**
  Gehört ein Baum mit einem Punkt rechts unter einem Nichtterminal zu einem Zustand, an dem keine Adjunktion stattgefunden hat, gehört der gleiche Baum mit dem Punkt rechts über dem Nichtterminal zu dem Zustand.

[3] In diesem Zusammenhang steht die geklammerte Zahl 0 für die Anzahl der Zeichen der Vorausschau (lookahead), in diesem Fall also „ohne Vorausschau“.

Konstruktion eines Zustandsdiagramms

Der Zustand 0 enthält alle initialen Bäume mit einem Punkt links oben. Dann wird der Zustand 0 unter den obigen Operationen abgeschlossen. Die weiteren Zustände bildet man rekursiv mit folgenden Übergängen:

- Ein Übergang mit einem Terminal wird genau dann gebildet, wenn der Zustand einen Baum enthält mit dem Punkt links über diesem Terminal. Der neue Zustand enthält den Baum mit einem Punkt rechts über dem Terminal und alle Abschlüsse.
- Ein Übergang β_{right} wird genau dann gebildet, wenn der Zustand einen Baum enthält mit einem Punkt rechts unter einem Nichtterminal, in das der Baum β adjungiert werden kann. Man beachte, daß dieses Nichtterminal vorher mit einem Stern markiert wurde. Der neue Zustand enthält den Baum mit einem Punkt rechts über dem Nichtterminal und alle Abschlüsse.
- Ein Übergang β_{foot} wird genau dann gebildet, wenn der Zustand einen auxiliaren Baum β enthält mit einem Punkt links unter einem Fußknoten. Der neue Zustand enthält den Baum β mit einem Punkt rechts unter dem Fußknoten und alle Abschlüsse.

Manchmal erhält man beim Abschluß eines neuen Zustands einen schon vorhandenen Zustand. Hier verweist der Übergang natürlich auf den vorhandenen Zustand, sonst würden sich bei den meisten TAGs unendlich große Diagramme ergeben.

> ***Anmerkung:*** *Zum besseren Verständnis empfehle ich, einige Übergänge in der Abbildung 7 nachzuvollziehen.*

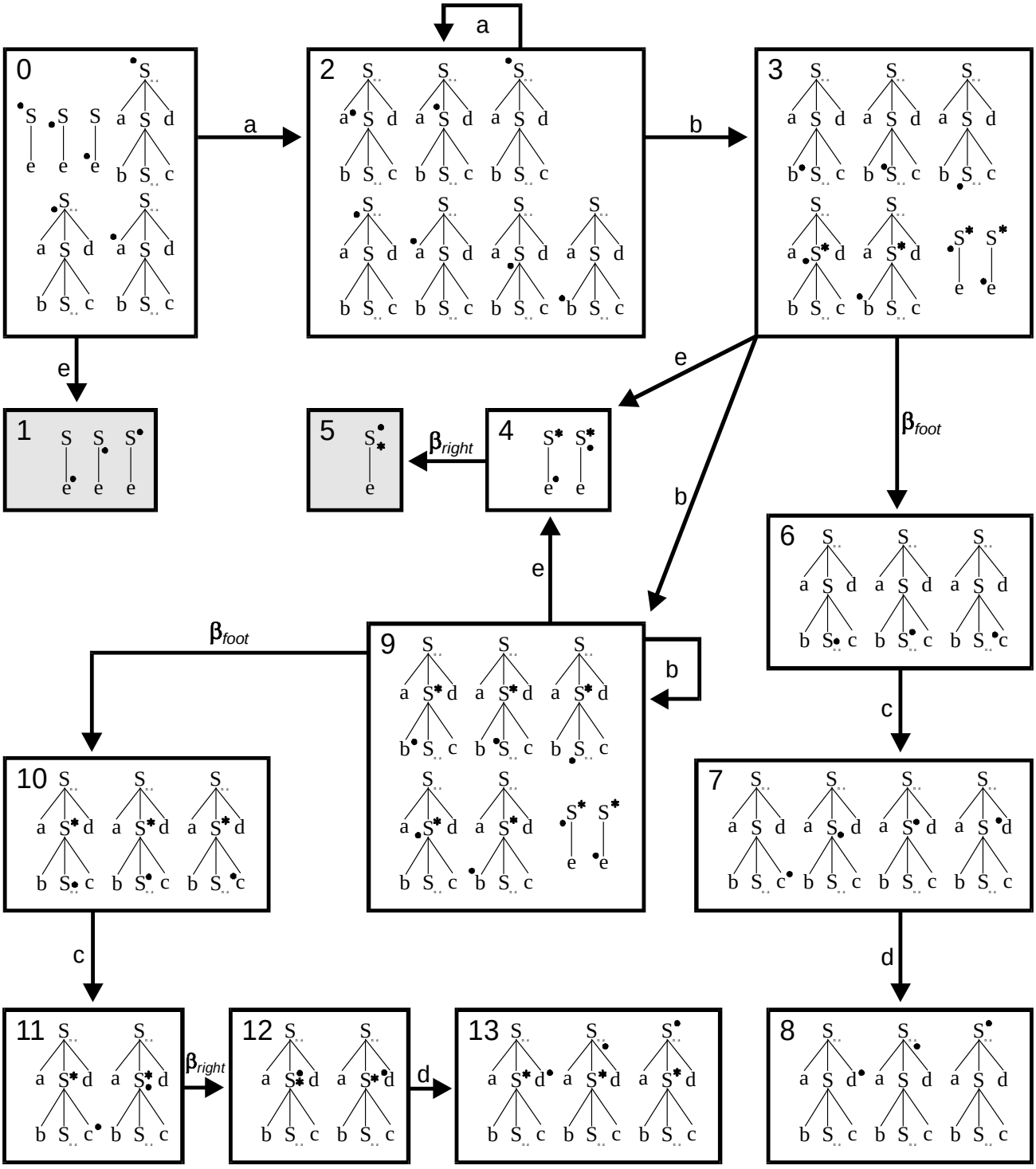

Abbildung 7: Zustandsdiagramm

Konstruktion einer Parsingtabelle

Aus dem so erstellten Zustandsdiagramm läßt sich wie folgt eine Parsingtabelle erstellen. Das Feld (i,a) mit einem Zustand i und einem Terminal a (oder $) in der Aktionstabelle enthält folgenden Eintrag:

- **Shift j**
  wenn es einen Übergang von i nach j mit dem Terminal a gibt.
- **Resume Right (rsδ@dot)**
  wenn der Zustand i einen Baum δ enthält, an dem ein Punkt rechts unter einem Nichtterminal (dot) ist, das mit einem Stern gekennzeichnet ist.
- **Reduce Root (rdβ@star)**
  wenn der Zustand i einen auxiliaren Baum enthält, der mit einem Punkt rechts über der Wurzel gekennzeichnet ist.
- **Accept**
  wenn a=$ und der Zustand i einen elementaren Baum enthält, der mit einem Punkt rechts über der Wurzel gekennzeichnet ist.
- **Error** sonst.

Das Feld (i,β_{foot}) in der Sprungtabelle enthält den Eintrag j, wenn es einen Übergang, von i nach j gibt, der mit β_{foot} gekennzeichnet ist., für das Feld (i,β_{right}) analog.

An dieser Stelle zeigt sich die Einschränkung dieser Konstruktion. Gibt es mehr als genau eine Möglichkeit, ein Feld der Parsingtabelle auszufüllen, ist die Grammatik nicht ohne Vorausschau zu parsen. Die TAGs, die deterministisch ohne Vorausschau geparst werden können, bilden also leider nur eine Teilmenge der TAGs. Immerhin ist dieser Algorithmus in endlich vielen Schritten in der Lage zu entscheiden, ob eine TAG mit LR(0) verarbeitet werden kann oder nicht.

Korrekte Präfixeingenschaft

Der von uns konstruierte Parser erfüllt leider die korrekte Präfixeigenschaft nicht. Das bedeutet, er hält nicht mit einem Fehler an dem ersten Zeichen, an dem er hätte erkennen können, daß das Eingabewort nicht korrekt ist.

Führen wir uns noch einmal die Arbeitsweise des Parsers vor Augen. Bildlich gesehen wandert der Parser an dem abgeleiteten Baum gegen den Uhrzeigersinn entlang und versucht sich die bisher offenen Möglichkeiten als Zustände zu merken. Dabei merkt er sich möglicherweise wiederholtes Vorkommen von Terminalen durch wiederholtes Ablegen des gleichen Zustands (siehe Zustand 2 oder 9 in Abbildung 7) auch dem Keller. Gibt es einen Übergang zum nächsten Terminal, wird die Anzahl der bisher gelesenen Terminale nicht geprüft. Erst wenn der Parser rechts unter einem Nichtterminal ankommt, an dem eine Adjunktion stattgefunden hat, prüft er die bisher gelesenen Terminal gegen die jetzt folgenden. Betrachten wir das Eingabewort *aabeccdd*. Der Parser verarbeitet dieses Wort wie folgt:

Tabelle 3: Parsingbeispiel

Step	**Parser configuration**	**Next move**
1	(‖0,aabeccdd$)	s2
2	(‖0‖2,abeccdd$)	s2
3	(‖0‖2‖2,beccdd$)	s3
4	(‖0‖2‖2‖3,eccdd$)	s4
5	(‖0‖2‖2‖3‖4,ccdd$)	rsα@0
6	(‖0‖2‖2‖3‖4‖6,ccdd$)	s7
7	(‖0‖2‖2‖3‖4‖6‖7,cdd$)	error

Aber schon in Schritt 4 hätte er den Fehler entdecken können. Da es aber einen korrekten Übergang aus dem Zustand 3 mit einem Terminal e gibt, fährt er an dieser Stelle fort.

Abkürzungsverzeichnis

CFG	Context Free Grammar
	Kontextfreie Grammatik
CFL	Context Free Language
	Kontextfreie Sprache
TAG	Tree Adjoining Grammar
	Baumadjunktionsgrammatik
TAL	Tree Adjoining Language
	Baumadjunktionssprache
PDA	Pushdown Automaton
	Kellerautomat
EPDA	Embedded Pushdown Automaton
	Einbettender Kellerautomat
BEPDA	Bottom-Up Embedded Pushdown Automaton
	Aufwärtiger Einbettender Kellerautomat
LR(k)	Left-Right Parser with k Lookahead

Index

Abgeschlossenheit der Zustände 10
Accept 7
Aufwärtiger Einbettender Kellerautomat 4
BEPDA 4
Bottom-Up Embedded Pushdown Automaton 4
Einbettender Kellerautomat 3
Einträge der Aktionstabelle 6
Embedded Pushdown Automaton 3
EPDA 3
Error 7
Kellerautomat 1
Konfiguration eines Parsers 5
Konstruktion
  Parsingtabelle 13
  Zustandsdiagramm 11
Korrekte Präfixeingenschaft 14
LR Parser 5
LR Parsing 5
LR(0) Parsingtabelle 10
Parsing Beispiel 8
PDA 1
Pushdown Automaton 1
Reduce Root 7
Resume Right 6
Shift j 6
Sprungaktionen 6
Übergangsfunktion 6
Zustand des Automaten 10