

Der Earley-Algorithmus.

Eine Erläuterung der formalen Spezifikation mit linguistischen Beispielen

Kursskript

Karin Haenelt, 25.07.2001

1 Einleitung

In diesem Skript wird die formale Spezifikation des Earley-Algorithmus erläutert. Basis ist die Spezifikation, die Jay Earley in seinem Aufsatz von 1970 (Earley, 1970) gegeben hat. Zur Erläuterung werden linguistische Beispiele gegeben.

Dieses Skript soll zugleich eine Hilfe zum Verständnis formaler Spezifikationen bieten.

Abschnitt 2 gibt eine allgemeine Übersicht über die Funktionsweise des Earley-Algorithmus. Abschnitt 3 stellt die Notationskonventionen und Definitionen zusammen, die Jay Earley in seinem Aufsatz verwendet

In Abschnitt 4 wird die Spezifikation des Aufsatzes von Jay Earley (1970) zitiert. In Abschnitt 5 folgt dann die schrittweise detaillierte Erläuterung dieser Spezifikation.

2 Funktion des Earley-Algorithmus

Der Earley-Algorithmus ist ein Erkennungs-Algorithmus für Sprachen, die mit kontextfreien Grammatiken beschrieben werden. Er ist für formale Sprachen entwickelt worden, bildet aber auch die Basis für viele Implementierungen zur syntaktischen Analyse natürlichsprachiger Sätze.

Eine kontextfreie Grammatik ist wie folgt definiert: „Eine *kontextfreie Grammatik* ist eine endliche Menge von Variablen (auch *Nichtterminale* oder *syntaktische Kategorien* genannt), von denen jede eine Sprache repräsentiert. Die Sprachen, die durch die Variablen dargestellt werden, werden rekursiv durch die anderen Variablen und primitiven Symbole – die sogenannten *Terminale* – beschrieben. Die Regeln, die die Variablen untereinander verknüpfen, werden Produktionen genannt.“ (Hopcroft/Ullman, 1988, S. 81).

Erkennung (Recognizer): ein Algorithmus, der eine Zeichenkette als Eingabe nimmt und diese Zeichenkette akzeptiert oder verwirft, je nachdem, ob sie den Grammatikregeln entspricht oder nicht.

Parser: ein Erkennung, der zusätzlich die Ableitungsbäume der Zeichenkette ausgibt.

Der Earley-Algorithmus arbeitet eine Grammatik zum Vergleich mit einer Eingabekette top-down mit drei Funktionen ab:

Der *Predictor* expandiert alle nicht-terminalen Kategorien soweit bis die terminalen Kategorien erreicht werden.

Der *Scanner* stellt fest, ob die Eingabe den erwarteten terminalen Kategorien entspricht.

Der *Completer* sorgt dafür, dass nach der vollständigen Erkennung von Teilkonstituenten zur übergeordneten Konstituente zurückgekehrt wird.

Der Algorithmus verfolgt Schritt für Schritt die möglichen Ableitungsbäume von Sätzen. Dies sei an dem folgendem Beispiel illustriert.

Beispielgrammatik

- $\Phi \rightarrow S$
- $S \rightarrow NP VP$
- $NP \rightarrow dete\ nomn$
- $VP \rightarrow verb$

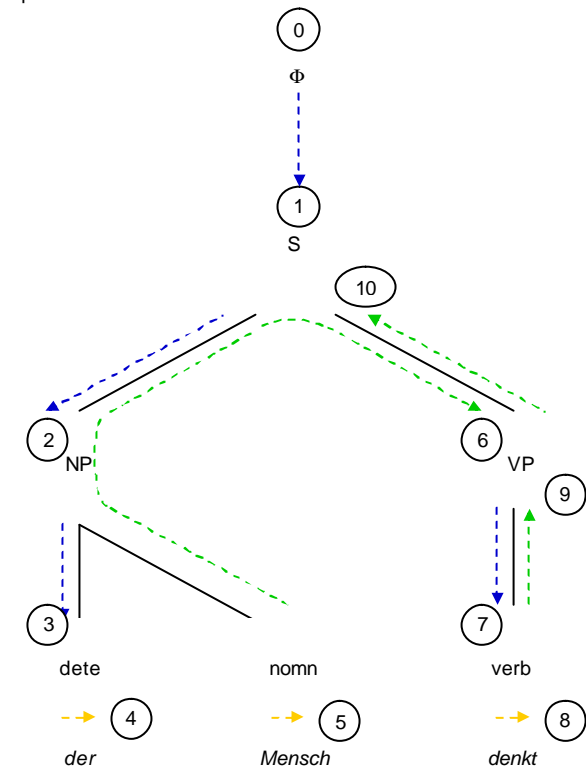


Abbildung 1: Syntaxbaum mit Ableitungsschritten 0 - 10

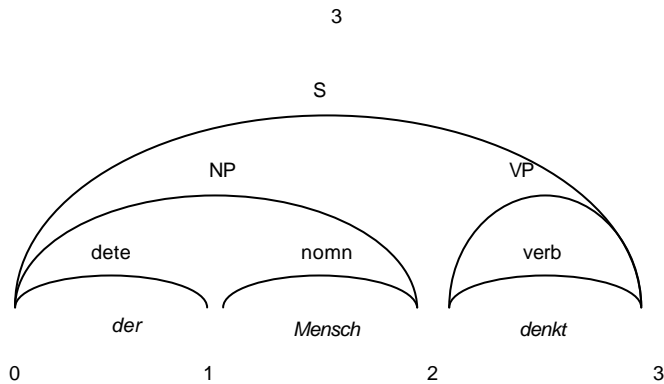


Abbildung 2: Kanten des Syntaxbaumes und ihre Abdeckung bezogen auf die Eingabe von der linken Grenze 0 (vor Wort 1) bis zur rechten Grenze 3 (hinter Wort 3)

3 Notationskonventionen und Definitionen

In dem genannten Artikel werden die folgenden Notationskonventionen und Definitionen verwendet:

3.1 Produktionsregeln

Zur Darstellung der Produktionsregeln, der Ableitungen, der Eingabekette und der Grammatik werden die folgenden Symbole verwendet:

a, b, c	Terminal-Symbole, also primitive Symbole, die für die Blätter des Satzstruktur-Baumes verwendet werden. Gewöhnlich werden hierfür die morpho-syntaktischen Kategorien wie z.B. nomn, verb, adje verwendet. Man findet aber auch Anwendungen, in denen lexikalische Zeichen direkt als Terminalsymbole verwendet werden.
A, B, C	Non-Terminal-Symbole, also syntaktische Kategorien wie z.B. S, NP, VP
α, β, γ	Ketten von Terminal - oder Non-Terminal-Symbolen
λ	leere Kette
α^n	$\alpha \dots \alpha$ (Kette mit n mal vorkommendem α)
$ \alpha $	Anzahl der Symbole in α
$A \rightarrow \alpha$	Produktionsregel
R	Wurzel, Non-Terminal-Symbol, das für "Satz" steht

$\alpha \Rightarrow \beta$	β ist von α direkt abgeleitet, wenn $\exists g, d, h, A$, wobei $\alpha = \gamma A \delta$ und $\beta = \gamma \eta \delta$ und $A \rightarrow \eta$ eine Produktion ist
$\alpha \Rightarrow^* \beta$	β ist von α (ggf. über mehrere Produktionen) abgeleitet, wenn \exists Ketten $\alpha_0, \alpha_1, \dots, \alpha_m$ ($m \geq 0$) wobei $\alpha = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow \alpha_m = \beta$. Die Folge $\alpha_0, \dots, \alpha_m$ wird Ableitung (von β aus α) genannt
$X_1 \dots X_n$	Eingabe-Kette
k	Anzahl der Look-Ahead-Symbole
G	Grammatik

3.2 Instanzen der Produktionsregeln

Für jedes Vorkommen, d.h. jede Instanz einer Regel wird zur eindeutigen Kennzeichnung eine Produktionsnummer vergeben (z.B. S_1 (NP_2 (der Architekt) VP_3 (v entwirft (NP_4 ein Haus))).

p	Nummer einer Produktion
1, ..., d - 1	Anzahl der Produktionen
Ferner werden die Konstituenten auf der rechten Seite der Regel durchnummeriert. Dabei werden die folgenden Symbole verwendet:	
\bar{p}	Anzahl der Symbole auf der rechten Seite einer Produktion
$D_p \rightarrow C_{p1} \dots C_{p\bar{p}}$	Produktion ($1 \leq p \leq d - 1$)

So ergibt sich folgende Darstellung der einzelnen Produktionen:

$D_0 \rightarrow R \#$	Produktion Nr. 0, R ist die Wurzel von G, # ist ein neues Terminal-Symbol
#	Terminal-Symbol, das in der Eingabekette nicht vorkommt; als Ketten-Ende-Markierung verwendet (im Originalartikel ist das Zeichen '-' verwendet, das in Word nicht zur Verfügung steht)

3.3 Produktionsregeln und Punkt Schreibweise (dotted rules)

Punkte in einer Produktionsregel markieren den aktuellen Bearbeitungszustand. Sie zeigen an, bis zu welcher Konstituente die Erkennung fortgeschritten ist, d.h. bis zu welcher Stelle in der Regel die Verarbeitung gediehen ist. Der Punkt steht jeweils vor der nächsten zu erkennenden Konstituente. Die Regel $S \rightarrow NP VP$ kann drei Zustände haben:

- $S \rightarrow \bullet NP VP$ noch gar keine Konstituente erkannt
- $S \rightarrow NP \bullet VP$ eine Konstituente (NP) erkannt
- $S \rightarrow NP VP \bullet$ zwei Konstituenten (NP und VP) erkannt

3.4 Zustand, Zustandsmenge, Endzustand

<i>Zustand</i> _{def.}	ein Zustand ist ein Quadrupel $\langle p, j, f, \alpha \rangle$ Dieses Quadrupel kann auch als Produktion mit einem Punkt repräsentiert werden.
$s = \langle p, j, f, \alpha \rangle$	<i>Zustand</i> _{def.}
p	Integer, $(0 \leq p \leq d - 1)$ (Nummer der Produktion)
j	Integer, $(0 \leq j \leq p)$ (Anzahl der erkannten Symbole auf der rechten Seite der Produktion) (entspricht Position des \bullet in der Punktnotation für geteilte Produktionen, z.B. $S \rightarrow \bullet NP VP$)
f	Integer, $(0 \leq f \leq n+1)$, Anfang der Produktion (entspricht der linken Grenze des durch die Produktion überspannten Teils der Eingabekette, vom Completer verwendet als „Rückwärtszeiger“ auf die Zustandsmenge, in der die Produktion erzeugt wurde)
α	Zeichenkette bestehend aus k Symbolen (Look-ahead-String), syntaktisch zulässiger Nachfolger einer Instanz einer Produktion. Die Berücksichtigung dieser Zeichenkette soll den Zeitpunkt des Abbruchs der Weiterverfolgung von Regeln, die nicht zu einer erfolgreichen Analyse beitragen, optimieren.
<i>Zustandsmenge</i>	geordnete Menge der Zustände zu einer Zustandsmenge S_i gehören jeweils die Zustände, in denen dieselbe rechte Grenze i in der Eingabekette erreicht ist. Es gibt also eine initiale Zustandsmenge S_0 und weitere Zustandsmengen S_i für jedes Eingabesymbol x_i .
<i>Endzustand</i>	Zustand mit $j = \bar{p}$

hinzufügen eines Zustandes zu einem Zustand durch Anfügen als letztem in der geordneten Menge, sofern er noch kein Element ist

3.5 Bestimmung des Look-Ahead-Symbols

$H_k(\gamma) = \{ \alpha \mid \alpha \text{ terminal ist, } |\alpha| = k, \text{ und } \exists \beta, \text{ so dass } \gamma \xrightarrow{*} \alpha\beta \}$
Menge aller k -Symbole langen terminalen Ketten die eine von γ abgeleitete Kette beginnen (verwendet zur Bildung von Look-ahead-Strings für Zustände)

Beispiele: $H_1(\text{dete adje nomn}) = \text{dete}$
 $H_1(\text{NP}) = \text{dete}$

4 Formale Spezifikation

Der Aufsatz gibt folgende formale Spezifikation des Algorithmus:

THE RECOGNIZER. This is a function of three arguments $REC(G, X_1 \dots X_n, k)$ computed as follows:

Let $X_{n+i} = \#$ $(1 \leq i \leq k+1)$.

Let S_i be empty $(0 \leq i \leq n+1)$.

Add $\langle 0, 0, 0, \# \rangle$ to S_0 .

For $i < 0$ step 1 until n do

Begin

Process the states of S_i in order, performing one of the following three operations on each state $s = \langle p, j, f, \alpha \rangle$:

(1) Predictor: If s is nonfinal and $C_{p(j+1)}$ is a nonterminal, then for each q such that $C_{p(j+1)} = D_q$, and for each $\beta \in H_k(C_{p(j+2)} \dots C_{p \bar{p}} \alpha)$ add $\langle q, 0, i, \beta \rangle$ to S_i .

(2) Completer: If s is final and $\alpha = X_{i+1} \dots X_{i+k}$,

then for each $\langle q, l, g, \beta \rangle \in S_i$
 (after all states have been added to S_i)
 such that $C_{q(l+1)} = D_p$,
 add $\langle q, l+1, g, \beta \rangle$ to S_i .

(3) Scanner: If s is nonfinal and $C_{p(j+1)}$ is terminal,
 then if $C_{p(j+1)} = X_{i+1}$,
 add $\langle p, j+1, f, \alpha \rangle$ to S_{i+1} .

If S_{i+1} is empty, return rejection.

If $i=n$ and $S_{i+1} = \{ \langle 0, 2, 0, \# \rangle \}$, return acceptance.

End

5 Erläuterung der formalen Spezifikation

Zur Erläuterung wird das in Abschnitt 2 eingeführte Beispiel verwendet. Die Sequenz der Zustände wird in einer Tabelle, der sogenannten *Agenda* (oder auch *Chart* bzw. *Parsing-Chart*) notiert. Die dem Beispiel zugehörige Agenda enthält folgende Zustände:

s (eingele- sen)	S_i	X_{i+1}	$s = \langle p, j, f, \alpha \rangle$	D_p	\rightarrow	$C_{p1} \dots C_{pP}$	$C_{p(j+1)}$				
0	0	dete	1	0	0	0	#	2	Φ	• S ##	S
1	0	dete	2	1	0	0	#	2	S	• NP VP	NP
2	0	dete	3	2	0	0	verb	2	NP	• dete nomn	dete
3	1	nomn	4	2	1	0	verb	2	NP	dete • nomn	nomn der
4	2	verb	5	2	2	0	verb	2	NP	dete nomn •	- Mensch
5	2	verb	6	1	1	0	#	2	S	NP • VP	VP
6	2	verb	7	3	0	2	#	1	VP	• verb	verb
7	3	#	8	3	1	2	#	1	VP	verb •	- denkt
8	3	#	9	1	2	0	#	2	S	NP VP •	-
9	3	#	10	0	1	0	#	2	Φ	S • ##	#
10	4	#	11	0	2	0	#	2	Φ	S# • #	#

Der Algorithmus durchläuft die Agenda, indem

- sequentiell jeweils ein Zustand eingelesen wird,
- festgestellt wird, welche Funktion (Predictor, Scanner, Completer) für die Verarbeitung des Zustandes zuständig ist,

- ein oder mehrere Zustände an das Ende der jeweiligen Zustandsmenge hinzugefügt werden

bis das Ende der Agenda bzw. ein gültiger Endzustand erreicht ist.

In der Spezifikation wird der jeweils eingelesene Zustand mit

$\langle p, j, f, a \rangle$

bezeichnet.

Die durch die drei Funktionen zu erzeugenden Zustände werden wie folgt bezeichnet :

- Predictor: $\langle q, 0, i, \beta \rangle$
- Scanner: $\langle p, j+1, f, \alpha \rangle$
- Completer: $\langle q, l+1, g, \beta \rangle$

Der Completer muss außerdem noch den Zustand aufsuchen, der den jeweiligen Mutterknoten bezeichnet:

$\langle q, l, g, \beta \rangle$

Es werden folgende Zusammenhänge zwischen den eingelesenen und den erzeugten Zuständen spezifiziert:

5.1 Predictor

Der Predictor ist für die Expansion von Non-Terminal-Symbolen zuständig. Immer wenn die Verarbeitung an einen Zustand gelangt, der die Erkennung eines nicht-terminalen Symbols (also einer syntaktischen Kategorie wie z.B. S, NP, VP) verlangt, muss der Predictor alle Regeln auf die Agenda schreiben, die für dieses nicht-terminale Symbol in der Grammatik bereitgestellt werden. Er stellt also zu einem Mutterknoten die möglichen Tochterknoten bereit.

Im Beispiel werden die folgenden Regeln durch den Predictor auf die Agenda geschrieben:

eingelesen -	erzeugt: $s = 1$ (Initialzustand)	Φ	\rightarrow • S
eingelesen: $s = 1$	erzeugt: $s = 2$	S	\rightarrow • NP VP
eingelesen: $s = 2$	erzeugt: $s = 3$	NP	\rightarrow • dete nomn
eingelesen: $s = 6$	erzeugt: $s = 7$	VP	\rightarrow • verb

Der Predictor schreibt dabei alle möglichen Alternativen einer bestimmten Regel auf die Agenda, d.h. es können pro eingelesenem Zustand immer eine Menge neuer Zustände erzeugt werden. Alle Lösungshypothesen werden quasi parallel solange weiterverfolgt, bis Scanner oder Completer keine Übereinstimmung mit der Eingabekette mehr feststellen bzw. bis das Ende der Eingabekette erfolgreich erreicht wurde.

Im Beispiel gibt es pro Konstituente nur eine Expansionsregel. In einer realistischen Grammatik für eine natürliche Sprache gibt es natürlich sehr viel mehr.

Die Spezifikation für den Predictor lautet:

Predictor: wenn s nicht final ist und $C_{p(j+1)}$ ein Non-Terminal-Symbol ist, dann soll für jedes q für das gilt $C_{p(j+1)} = D_q$, und für jedes $\beta \in H_k(C_{p(j+2)} \dots C_{p\bar{p}} \alpha)$ ein Zustand $\langle q, 0, i, \beta \rangle$ zur Zustandsmenge S_j hinzugefügt werden.

Diese Spezifikation sei am Beispiel der des eingelesenen Zustandes 1 und des erzeugten Zustandes 2 betrachtet:

s (eingelesen)	$S_i X_{i+1}$	$s = \langle p, j, f, \alpha \rangle$	\bar{p}	$D_p \rightarrow C_{p1} \dots C_{p\bar{p}} C_{p(j+1)}$
0 0 dete	1	0 0 0	#	2 Φ • S ## (S)
1 0 dete	2	1 0 0	#	2 (S) • NP VP NP

Eingelesener Zustand: $\langle p, j, f, \alpha \rangle$ (Zustand 1) $\Phi \rightarrow \bullet S$
 Erzeugter Zustand: $\langle q, 0, i, \beta \rangle$ (Zustand 2) $S \rightarrow \bullet NP VP$

Zustand 1 ist nicht final, da $j \neq \bar{p}$, d.h. es wurden noch nicht alle Symbole auf der rechten Seite der Regel erkannt. Außerdem soll $C_{p(j+1)}$ ein Non-Terminal sein. Der Index für die Elemente auf der rechten Seite der Regel läuft von 1 bis \bar{p} , wobei \bar{p} die Anzahl der Symbole auf der rechten Seite der Regel ist. Im aktuellen Fall ist $j = 0$, d.h. gesucht ist das Symbol C_{p1} . So gilt $C_{p(j+1)} = "S"$ (also ein Non-Terminal). Damit ist der Predictor zuständig.

Die Objekte der Quadrupel haben die folgende Bedeutung mit folgenden Zusammenhängen:

- Objekt: (p bzw. q) Nummer der Produktion
 Die Spezifikation bezeichnet die eingelesene Produktion ($\Phi \rightarrow \bullet S$) mit der Produktionsnummer p und die zu erzeugende Produktion mit der Produktionsnummer q, d.h. für die zu erzeugende Produktion ist eine von p verschiedene Nummer zu erzeugen

und damit eine von p verschiedene Produktion, nämlich $S \rightarrow \bullet NP VP$. Der Zähler für die Produktionsnummern wird hochgezählt und die nächste freie Nummer wird an die neue Produktion vergeben.

Der Zusammenhang zwischen der mit p und der mit q gekennzeichneten Produktion wird hergestellt durch die Bedingung $C_{p(j+1)} = D_q$:

$C_{p(j+1)}$ gehört zur eingelesenen Produktion (Index p). Mit C werden gemäß der Notationskonvention für Produktionen die Konstituenten auf der rechten Seite der Regel bezeichnet. Dabei steht $C_{p(j+1)}$ für die Konstituente, die als nächste zur Verarbeitung ansteht: es sind bisher j Konstituenten erkannt, d.h. $C_{p(j+1)}$ ist die nächste Konstituente nach den erkannten Konstituenten. Im Beispiel ist $j = 0$, da aus der Startregel $\Phi \rightarrow \bullet S$ noch gar nichts erkannt wurde (in der Punktschreibweise steht der Punkt vor dem S). Das zur Verarbeitung anstehende Symbol $C_{p(j+1)}$ ist also die erste Konstituente, nämlich das S.

D_q gehört zur neu zu erzeugenden Produktion (Index q). Mit D werden die Köpfe der Regeln bezeichnet. Also besagt diese Forderung, dass der Kopf der neu zu erzeugenden Produktion identisch sein muss mit der zur Verarbeitung anstehenden Konstituente der Mutterproduktion.

Im Beispiel ist $C_{p(j+1)} = "S"$ und $D_q = "S"$

- Objekt: (j bzw. 0) Anzahl der erkannten Symbole auf der rechten Seite
 In der zu erzeugenden Produktion ist die Anzahl der erkannten Symbole 0 (unabhängig vom Wert von j in der Mutterproduktion). Dies ist immer der Fall, wenn eine Produktion neu als Aufgabe auf die Agenda gesetzt wird, denn zu diesem Zeitpunkt ist natürlich noch keines ihrer Symbole auf der rechten Seite erkannt worden.
- Objekt: (f bzw. i) Anfang der Produktion
 In der neuen Produktion p ist der Anfang der Produktion (d.h. ihre linke Grenze bezogen auf den Eingabestring) identisch mit der rechten Grenze der Mutterproduktion (bezeichnet durch die Zustandsmenge S_j , zu der die Mutterproduktion gehört).
- Objekt: (α bzw. β) Look-Ahead-Symbol
 Eingelesen wird ein Look-Ahead-Symbol α , geschrieben wird ein Look-Ahead-Symbol β , d.h. es ist ein neues Look-Ahead-Symbol zu bestimmen. Das Look-Ahead-Symbol gibt an, welches terminale Symbol als nächstes in der Eingabe stehen soll, wenn eine Produktion abgearbeitet ist. Dieses Symbol ist nach folgender Formel zu bestimmen: $\beta \in H_k(C_{p(j+2)} \dots C_{p\bar{p}} \alpha)$, d.h. es ist die Zeichenkette der Mutterproduktion (Index p), beginnend ab der Konstituente $C_{p(j+2)}$ (d.h. ab der Konstituente hinter der gerade in Bearbeitung befindlichen Konstituente $C_{p(j+1)}$) bis einschließlich des Look-Ahead-Symbols

α der Mutterproduktion; und die Länge der hieraus als Look-Ahead-Symbole zu bestimmenden Teilkette ist k Symbole lang (meistens 1 Symbol).

Für die Verarbeitung natürlicher Sprachen ist die Verwendung eines Look-Ahead-Symbols aus folgenden Gründen aufwändig:

1. Look-Ahead-Symbole sind terminale Symbole, da der Sinn gerade darin besteht, diese Symbole mit der aktuellen Eingabekette zu vergleichen. Zur Bestimmung des nächsten terminalen Symbols kann die Verfolgung einer Ableitungskette erforderlich werden. Bei Grammatiken für natürliche Sprachen kann diese Ableitungskette lang werden. Angenommen die Mutterproduktion lautet $S \rightarrow NP VP$, dann ist das Look-Ahead-Symbol der NP das erste terminale Symbol aus der VP. Um dieses zu bestimmen, muss VP bis zum linken terminalen Knoten abgeleitet werden.

2. Es kann mehrere Look-Ahead-Symbole geben. Angenommen, die Grammatik enthält mehrere Ableitungsregeln für die VP, z.B.

$VP \rightarrow v NP$

$VP \rightarrow NP v$ und $NP \rightarrow (dete) (adje) nomn$

\cdot $NP \rightarrow pron$

so gibt es insgesamt fünf Look-Ahead-Symbole für die in Frage stehende NP, nämlich v , $dete$, $adje$, $nomn$. Bei Anwendung des Originalalgorithmus würde das zu fünf Einträgen in die Agenda führen (und zwar jeweils für alle NP-Regeln, die gerade auf die Agenda einzutragen sind). Hier ist durch geeignete Modifikation des Algorithmus dafür Sorge zu tragen, dass nicht allein wegen der Look-Ahead-Symbole eine kombinatorische Explosion von Chart-Einträgen entsteht.

3. Wie eine der unter 2. angenommenen NP-Regeln zeigt, können in Grammatiken für natürliche Sprachen manche Konstituenten fakultativ sein. Auch diese Eigenschaft wäre zu protokollieren.

5.2 Scanner

Der Scanner ist zuständig, wenn ein terminales Symbol zur Verarbeitung ansteht. Er überprüft, ob das terminale Symbol tatsächlich dem nun folgenden Element in der Eingabekette entspricht.

Im Beispiel werden die folgenden Zustände durch den Scanner auf die Agenda geschrieben:

eingelesen: $s = 3$ erzeugt: $s = 4$ $NP \rightarrow dete \bullet nomn$

eingelesen: $s = 4$ erzeugt: $s = 5$ $NP \rightarrow dete nomn \bullet$
 eingelesen: $s = 7$ erzeugt: $s = 8$ $VP \rightarrow verb \bullet$
 eingelesen: $s = 10$ erzeugt: $s = 11$ $\Phi \rightarrow S \# \bullet \#$

Die Spezifikation für den Scanner lautet:

Scanner: wenn s nicht final ist und $C_{p(j+1)}$ ein Terminal-Symbol ist, und $C_{p(j+1)} = X_{i+1}$, dann soll ein Zustand $\langle p, j+1, f, \alpha \rangle$ zur Zustandsmenge S_{i+1} hinzugefügt werden.

Diese Spezifikation sei am Beispiel des eingelesenen Zustandes 3 und des erzeugten Zustandes 4 betrachtet:

s (eingelesen)	S_i	X_{i+1}	$s = \langle p, j, f, \alpha \rangle$	\bar{p}	D_p	\rightarrow	$C_{p1} \dots C_p$	\bar{C}_p	$C_{p(j+1)}$
2	0	dete	3	2	0	verb	2 NP	$\bullet dete nomn$	dete
3	1	nomn	4	2	0	verb	2 NP	dete $\bullet nomn$	nomn der

Eingelesener Zustand: $\langle p, j, f, \alpha \rangle$ (Zustand 3) $NP \rightarrow \bullet dete nomn$

Erzeugter Zustand: $\langle p, j+1, f, \alpha \rangle$ (Zustand 4) $NP \rightarrow dete \bullet nomn$

Es wird überprüft, ob noch Symbole der Produktion zur Erkennung anstehen (s ist nicht final, d.h. $j \neq \bar{p}$), ob das anstehende Symbol $C_{p(j+1)}$ ein Terminal-Symbol ist und ob das anstehende Symbol mit dem nächsten Symbol in der Eingabekette übereinstimmt ($C_{p(j+1)} = X_{i+1}$). Dann ist der Scanner zuständig.

Bei der Erzeugung des neuen Zustandes wird dann lediglich notiert, dass „der Punkt in der Produktion eine Position nach rechts verschoben wurde“: ($j + 1$). Auch die rechte Grenze der erkannten Eingabekette wird um eine Position nach rechts verschoben: (S_{i+1}). Die anderen Objekte des Quadrupels werden vom eingelesenen Zustand übernommen, denn es handelt sich um dieselbe Produktion p , in der lediglich der neue Erkennungsstand notiert wird.

5.3 Completer

Der Completer ist zuständig, wenn eine Konstituente vollständig erkannt wurde. Er hat die Aufgabe, die Mutterkonstituente aufzusuchen und einen neuen Zustand der Mutterkonstituente zu erzeugen. In diesem Zustand wird notiert, dass die vollständige erkannte Tochter-Konstituente gefunden wurde, indem der Punkt eine Position nach rechts geschoben wird.

Im Beispiel werden die folgenden Zustände durch den Completer auf die Agenda geschrieben:

- eingelesen: s = 5 NP → dete nomn •
 Mutter: s = 2 S → • NP VP
 erzeugt: s = 6 S → NP • VP
- eingelesen: s = 8 VP → verb •
 Mutter: s = 6 S → NP • VP
 erzeugt: s = 9 S → NP VP •
- eingelesen: s = 9 S → NP VP •
 Mutter: s = 1 Φ → • S ##
 erzeugt: s = 11 Φ → S • ##

Die Spezifikation für den Completer lautet:

Completer: wenn s final ist und $\alpha = X_{i+1} \dots X_{i+k}$, und
 alle Zustände der Zustandsmenge S_i bereits erzeugt wurden,
 dann soll für jeden Zustand $\langle q, l, g, \beta \rangle \in S_i$,
 für den $C_{q(l+1)} = D_p$,
 ein neuer Zustand $\langle q, l+1, g, \beta \rangle$
 der Zustandsmenge S_i hinzugefügt werden..

Diese Spezifikation sei am Beispiel des eingelesenen Zustandes 5, des Mutterzustandes 2 und des erzeugten Zustandes 6 betrachtet:

s (eingelesen)	S_i X_{i+1}	s =	$\langle p, j, f, \alpha \rangle$	$\frac{D_p}{p}$	→	$C_{p_1} \dots C_{p_p}$	$C_{p(l+1)}$
0	dete	1	0 0 0 #	2	Φ	• S ##	S
1	dete	2	1 0 0 #	2	S	• NP VP	NP
2	dete	3	2 0 0 verb	2	NP	• dete nomn	dete
3	1 nomn	4	2 1 0 verb	2	NP	dete • nomn	nomn der
4	2 verb	5	2 2 0 verb	2	NP	dete nomn • -	Mensch
5	2 verb	6	1 1 0 #	2	S	NP • VP	VP

- Eingelesener Zustand: $\langle p, j, f, \alpha \rangle$ (Zustand 5) NP → dete nomn •
- Mutterzustand: $\langle q, l, g, \beta \rangle$ (Zustand 2) S → • NP VP
- Erzeugter Zustand: $\langle q, l+1, g, \beta \rangle$ (Zustand 6) S → NP • VP

Eingelesen wird eine mit p bezeichnete Produktion, erzeugt wird eine mit q bezeichnete Produktion, also eine von der eingelesenen Produktion verschiedene Produktion. Allerdings steht die mit q bezeichnete Produktion bereits auf der Agenda, jedoch in einem anderen Zustand. Es handelt sich nämlich um die Mutterproduktion im Zustand vor der Erkennung der jetzt vervollständigten Tochterproduktion. Diese Vervollständigung soll an die Mutterproduktion „zurückgemeldet“ werden, indem ein neuer Zustand der Mutterproduktion erzeugt wird, in der der Punkt eine Position weiter rechts notiert wird. Die Mutterproduktion (bzw. die Mutterproduktionen – es können auch mehrere sein) werden über folgende Bedingungen identifiziert:

$$\text{Zustand } \langle q, l, g, \beta \rangle \in S_i \text{ und } C_{q(l+1)} = D_p.$$

Die Zustandsmenge in der gesucht werden soll, ist durch f bezeichnet. f ist durch den eingelesenen Zustand gegeben (dessen linke Grenze). In dieser Zustandsmenge sollen die Produktionen q gefunden werden, deren zur Verarbeitung anstehende Konstituente $C_{q(l+1)}$ identisch ist mit dem Kopf der eingelesenen Produktion p, m.a.W. all die Produktionen, die zur weiteren Verarbeitung auf die gerade vervollständigte Konstituente gewartet haben. Die Beschränkung der Suche auf die Zustandsmenge f stellt sicher, dass nur an die Stellen zurückgekehrt wird, die hier nach der aktuellen NP gesucht haben und nicht ggf. auch die, die möglicherweise eine an anderer Stelle im Satz vorkommende NP erwarten.

6 Literatur

Earley, Jay (1970): An Efficient Context-Free Parsing Algorithm. In: *Communications of the ACM*, 6 (8), 451-455

Hopcroft, John E. und Ullman Jeffrey D. (1988): *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie*. Addison-Wesley.