2.6 Stochastische Parsingverfahren

In diesem Abschnitt wird die Benutzung einer trainierten stochastischen kontextfreien Grammatik vorgestellt. Natürlich können stochastische kontextfreie Grammatiken in allen Fällen eingesetzt werden, in denen nicht-stochastische kontextfreie Grammatiken eingesetzt werden. Umgekehrt ist dies leider nicht der Fall, da die stochastischen kontextfreien Grammatiken aufgrund ihrer Regelwahrscheinlichkeiten wesentlich vielfältiger eingesetzt werden können. Um diese Unterscheidung möglichst klar auszudrücken, soll im folgenden von symbolischen und stochastischen Parsingverfahren gesprochen werden:

Die Vorgehensweise, bei der für einen Satz alle seine möglichen Analysen (für kontextfreie Grammatiken seine Syntaxbäume) berechnet werden, wird *symbolisches Parsingverfahren* genannt. Im Vergleich hierzu ist ein *stochastisches Parsingverfahren* ein Vorgang, bei dem für einen Satz nicht nur seine Analysen, sondern auch deren Wahrscheinlichkeiten (oder andere relevante Zählgrössen) berechnet werden.

Für symbolisches Parsing mit kontextfreien Grammatiken sind seit vielen Jahren wohl ebenso viele Parsing-Algorithmen bekannt, von denen der sogenannte CKY-Algorithmus vielleicht der bekannteste ist. Auch für stochastisches Parsing gibt es eine Vielzahl von Parsing-Algorithmen. Während sich aber die symbolischen Parsing-Algorithmen vor allem darin unterscheiden, welche Grammatik-Voraussetzungen sie fordern und auf welche spezielle Art und Weise sie die Satzanalysen herstellen und verwalten, ist die Unterscheidung der stochastischen Parsing-Algorithmen von prinzipiellerer Art.

Hierbei geht es primär um die Frage, was ein stochastischer Parsing-Algorithmus, neben der Herstellung der Satzanalysen, zusätzlich leisten soll. Die folgende Aufstellung gibt hierzu Auskunft, in die auch der Count-Algorithmus aufgenommen wurde, weil dieser, auf rein symbolische kontextfreie Grammatiken anwendbare Algorithmus, seiner ganzen Natur nach, eher den stochastischen Parsingverfahren zuzuordnen ist:

- *Count-Algorithmus*: Berechnung der Anzahl der Satzanalysen (bzw. der Teilsatz--Analysen),

- *Inside-Algorithmus*: Berechnung der *Inside-Wahrscheinlichkeit*, d.h. der Satzwahrscheinlichkeit (bzw. der Teilsatz-Wahrscheinlichkeiten),

konsistenten Non-Standard-Modellen zu suchen, welche bessere stochastische und damit vermutlich auch bessere linguistische Eigenschaften, als die Baumbank-Grammatik, aufweisen. Chi beweist (Chi (1998), Chi (1999)), dass Baumbank-Grammatiken und die mit dem Inside-Outside-Algorithmus estimierten Grammatiken konsistent sind, wenn keine zusätzlichen Techniken, wie Smoothing von Grammatikregeln, etc. eingesetzt werden.

- *Viterbi-Algorithmus*: Berechnung der *Viterbi-Wahrscheinlichkeit*, d.h. der maximalen Wahrscheinlichkeit der Satzanalysen (bzw. der Teilsatz-Analysen),

Eine besonders vorzügliche Eigenschaft probabilistischer kontextfreier Grammatiken ist es, dass diese, dank des Viterbi-Algorithmus (bzw. des Viterbi-Parse-Algorithmus, siehe Abschnitt 2.7), effizient zur Disambiguierung herangezogen werden können, auch wenn, in einer realistischen Grammatik, ein Satz in der Regel mehrere zehntausend Analysen besitzt (Rooth et al. 1999).

Diese Eigenschaft hebt probabilistische kontextfreie Grammatiken einerseits aus der Menge der rein symbolischen Grammatiken heraus, mit denen prinzipiell keine Disambiguierung möglich ist, wenn ein Satz mehr als eine einzige Satzanalyse aufweist.

Wichtiger ist jedoch, dass der Viterbi-Algorithmus kontextfreie Grammatiken in die Lage versetzt, *effizient* zu disambiguieren, was diese auch innerhalb der nicht eben zahlreichen Menge aller bekannten stochastischen Grammatiken, insbesondere der stochastischen unifikationsbasierten Grammatiken, äusserst positiv heraushebt, da nur diese Eigenschaft eine praktische Anwendung einer stochastischen Grammatik bewirken kann.

Bis zum heutigen Zeitpunkt ist für stochastische unifikationsbasierte Grammatiken kein effizienter Algorithmus zur Bestimmung des wahrscheinlichsten Parses bekannt. Dies hat zur Folge, dass stochastische unifikationsbasierte Grammatiken leider nur in sehr eingeschränktem Umfang zur Disambiguierung beliebiger Satzanalysen zu gebrauchen sind; wobei sie allerdings für Sätze mit wenigen dutzend Analysen, für die sie anwendbar sind, gute Ergebnisse aufweisen (Riezler et al. 2000).

Für das weitere Vorgehen ist es vorteilhaft, sich den Algorithmus von Cocke, Kasami und Younger (Kasami 1965) in Erinnerung zu rufen. Der CKY-Algorithmus löst für eine gegebene kontext-freie Grammatik in Chomsky-Normalform das Parsingproblem, d.h. die Fragestellung, ob ein gegebener Satz grammatisch ist, oder in anderen Worten, ob eine gegebene Wortfolge w die Satzkategorie S tragen kann. Da der CKY-Algorithmus auf dem Prinzip der dynamischen Programmierung basiert, ist er ein sehr effizienter Algorithmus. *Dynamische Programmierung* wird dabei jeder Algorithmus genannt, der ein Problem löst, indem er das gegebene Problem in kleinere Unterprobleme zerlegt und dann die Lösung rekursiv so berechnet, *dass jedes Teilproblem nur einmal gelöst wird.*[19]

Beim CKY-Algorithmus sieht das konkret so aus, dass er die etwas allgemeinere Fragestellung, ob ein beliebiger Teilsatz w_A (eine beliebige Teil-Wortfolge des Satzes w) eine bestimmte grammatikalische Kategorie A (zum Beispiel NP, DET, N, VP, V, ...) tragen

[19]Der angegebene Zusatz ist sehr wichtig. Im Parsing macht dies den Unterschied zwischen Chartparsern und anderen Parsern mir exponentieller Laufzeit (zum Beispiel Recursive-Decent-Parsern) aus.

kann, rekursiv zu beantworten sucht. Bezeichnet $\mathcal{T}(G)$ die Menge aller Syntaxbäume einer kontextfreien Grammatik G, so ist der Schlüsselschritt, dass gezeigt wird, dass

$$A \Rightarrow^* w_A := \{x_A \in \mathcal{T}(G) \mid x_A \text{ hat die Wurzel } A \text{ und die Blattfolge } w_A\}$$

unter milden Grammatik-Voraussetzungen rekursiv aus irgendwelchen Mengen $B \Rightarrow^* w_B$ mit $|w_B| < |w_A|$ berechnet werden kann.

Als Voraussetzung des CKY-Algorithmus wird angenommen, dass die kontextfreie Grammatik in *Chomsky-Normalform* vorliegt[20], d.h. dass alle Grammatikregeln entweder von der Form

$$A \to B\ C \quad \text{oder} \quad A \to a$$

sind; hierbei ist a ein beliebiges Wort aus dem Grammatiklexikon und A, B, C sind beliebige Grammatikkategorien. Unter dieser milden Voraussetzung, die insbesondere die wenig eleganten Epsilon- und Kettenregeln ausschliessen, kann es offenbar für ein einzelnes Wort $w_A = a$ nur einen einzigen Syntaxbaum mit Wurzelknoten A und Blattknoten a geben:

$$A \Rightarrow^* w_A \ = \ \{A \to a\}\ .$$

Für eine Wortfolge w_A mit $|w_A| \geq 2$ gilt jedoch:

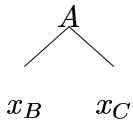
$$A \Rightarrow^* w_A \ = \ \left\{ \begin{array}{c} A \\ x_B \quad x_C \end{array} \ \middle| \ \begin{array}{c} A \to BC, \\ w_A = w_B w_C, \\ x_B \in\ B \Rightarrow^* w_B, \\ x_C \in\ C \Rightarrow^* w_C \end{array} \right\}$$

$$= \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} \left\{ \begin{array}{c} A \\ x_B \quad x_C \end{array} \ \middle| \ \begin{array}{c} x_B \in\ B \Rightarrow^* w_B, \\ x_C \in\ C \Rightarrow^* w_C \end{array} \right\}$$

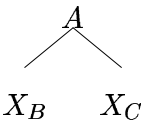
$$= \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} \begin{array}{c} A \\ B \Rightarrow^* w_B \quad C \Rightarrow^* w_C \end{array} \ .$$

[20]Für theoretische Überlegungen, nicht aber in der Praxis, ist die Annahme von Chomsky-Normalform eine milde Anforderung, da jede kontextfreie Grammatik in eine Grammatik in Chomsky-Normalform umgewandelt werden kann. Ihre Grösse kann dabei aber exponentiell anwachsen und - was schlimmer ist - es ist schwer, aus den Analysen der Grammatik in Chomsky-Normalform die Analysen der ursprünglichen Grammatik zu rekonstruieren. Wegen der grösseren praktischen Bedeutung und der Anwendbarkeit der hier vorgestellten Parsewald-Algorithmen auf allgemeine kontextfreie Grammatiken, wäre es natürlich sehr interessant, die Parseald-Algorithmen auch für den Earley-Algorithmus zu formulieren.

Hierbei weist die Notation $\sum$ daraufhin, dass es sich um eine Vereinigung disjunkter Mengen handelt. Ferner wurde für einen Syntaxbaum x_B mit Wurzel B und einen Syntaxbaum x_C mit Wurzel C und für eine Regel $A \to BC$ das Symbol



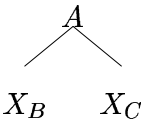
für den aus diesen drei Teilen in natürlicher Weise zusammengesetzten Syntaxbaum mit der Wurzel A benutzt. Schliesslich benutzt man für eine Menge X_A von Syntaxbäumen mit Wurzel A und eine Menge X_B von Syntaxbäumen mit Wurzel B und für eine Regel $A \to BC$ naheliegenderweise das Symbol



für die folgende Menge von Syntaxbäumen:

$$\left\{ \begin{array}{c} A \\ x_B \quad x_C \end{array} \;\middle|\; x_B \in X_B,\ x_C \in X_C \right\} .$$

Diese Definition erinnert sehr an die Definition des *Kreuzprodukts* in der Mengentheorie. Wie dort, ergibt



die leere Menge, wenn die Menge X_B oder die Menge X_C leer ist.

Die Formel:

$$A \Rightarrow^* w_A \;=\; \begin{cases} \{A \to w_A\} & \text{für } |w_A| = 1 \\ \displaystyle\sum_{\substack{A \to BC, \\ w_A = w_B w_C}} \begin{array}{c} A \\ B \Rightarrow^* w_B \quad C \Rightarrow^* w_C \end{array} & \text{für } |w_A| \geq 2 \end{cases}$$

beleuchtet in sehr anschaulicher Weise den rekursiven Charakter des Parsingproblems, d.h. der Aufgabe der Berechnung von $A \Rightarrow^* w_A$.

Während die erste Alternative einen Induktionsanfang begründen kann, ist es möglich, die zweite Alternative für einen Induktionsschluss zu benutzen, weil die Berechnung der Menge

Eingabe: Eine Grammatik in Chomsky-Normalform und ein Satz $w = w_1 \dots w_n$.

Ausgabe: Die Parsewälder $forest[s, A, s+l]$ für alle Teilsätze der Länge $1 \leq l \leq n$, Startindizes $1 \leq s \leq n+1-l$, und Wurzelkategorien A.

Prozedur:

1. $forest[.,.,.] := \emptyset$;
2. **for each** $s := 1, \dots, n$ **do**
3. **for each** $A \to w_s$ **do**
4. $forest[s, A, s+1] := \{A \to w_s\}$;
5. **for each** $l := 2, \dots, n$ **do**
6. **for each** $s := 1, \dots, n+1-l$ **do**
7. **for each** $t := 1, \dots, l-1$ **do**
8. **for each** $A \to B\ C$ **do**
9. $forest[s, A, s+l] := forest[s, A, s+l] +$
10. 

Abbildung 2.11: Parsewald-Algorithmus

$A \Rightarrow^* w_A$ auf die Berechnung der Mengen $B \Rightarrow^* w_B$ und $C \Rightarrow^* w_C$ zurückgeführt wurde, und diese sich wegen $|w_B| < |w_A|$ und $|w_C| < |w_A|$ als Teilprobleme des Ausgangsproblems auffassen lassen.

Abbildung 2.11 zeigt einen Algorithmus, der treffenderweise *Parsewald-Algorithmus* genannt werden könnte, weil er die Syntaxbäume sämtlicher Teilsätze dieser Induktionsidee folgend, verwaltet.

Die Eingabe des Parsewald-Algorithmus besteht aus einer Grammatik in Chomsky-Normalform und einem Satz $w = w_1 \dots w_n$, $(n \geq 1)$. Die Ausgabe des Parsewald-Algorithmus besteht aus den Parsewäldern

$$forest[s, A, s+l] \ := \ A \Rightarrow^* w_s \dots w_{s+l-1}$$

d.h. den Syntaxbäumen aller Teilsätze der *Länge* $1 \leq l \leq n$ und *Startindex* $1 \leq s \leq n+1-l$, mit der Grammatikkategorie A als Topknoten.

Man versteht den Algorithmus, insbesondere die 9.-10. Zeile problemlos, wenn man sich klar macht, dass in der Prozedur im Vergleich zu den vorherigen Ausführungen die folgende

Eingabe: Eine Grammatik in Chomsky-Normalform und ein Satz $w = w_1 \dots w_n$.

Ausgabe: Die Parsechart $chart[s, A, s+l] \in \{TRUE, FALSE\}$ für alle Teilsätze der Länge $1 \leq l \leq n$, Startindizes $1 \leq s \leq n+1-l$, und Grammatikkategorien A.

Prozedur:

```
1.  chart[.,.,.] := FALSE;
2.  for each s := 1,...,n do
3.      for each A → w_s do
4.          chart[s,A,s+1] := TRUE;
5.  for each l := 2,...,n do
6.      for each s := 1,...,n+1-l do
7.          for each t := 1,...,l-1 do
8.              for each A → B C do
9.                  chart[s,A,s+l] := chart[s,A,s+l] ∨
10.                     chart[s,B,s+t] ∧ chart[s+t,C,s+l]
```

Abbildung 2.12: CKY-Algorithmus

Notation benutzt wird:

$$w_A = w_s \dots w_{s+l-1}, \quad w_B = w_s \dots w_{s+t-1} \quad \text{und} \quad w_C = w_{s+t} \dots w_{s+l-1},$$

sodass, wie vorher ausgeführt:

$$w_A = w_B w_C$$

gilt. Das Füllen der Parsewälder geschieht induktiv für monoton grösser werdende Teilsätze w_A der Längen $l = 1, 2, \dots, n$. In der 1. Zeile der Prozedur werden die Parsewälder als leere Mengen initialisiert, während in der 2.-4. Zeile der Induktionsanfang und in der 5.-10. Zeile der Induktionsschluss stattfindet. Interessant ist das Zeichen $'+'$, welches in der 9. Zeile anstatt des Mengenvereinigungszeichens $'\cup'$ benutzt wird und anzeigen soll, dass eine Vereinigung disjunkter Mengen stattfindet. Da die Grammatik fest ist, können die 3. und 4. Zeile, sowie die 8.-10. Zeile in konstanter Zeit abgearbeitet werden, wobei man sich lediglich Gedanken um eine angemessene Implementierung der Parsewälder machen muss. Die Abarbeitung der 2.-4. bzw. der 5.-10. Zeile benötigt daher $O(n)$ bzw. $O(n^3)$ Zeit. Der Parsewald-Algorithmus ist somit von der Ordnung $O(n^3)$.

In Abbildung 2.12 ist der CKY-Algorithmus wiedergegeben. Die Eingabe des CKY-Algorithmus besteht aus einer Grammatik in Chomsky-Normalform und einem Satz $w = w_1 \dots w_n$, $(n \geq 1)$, dessen Grammatikalität zu überprüfen ist. Die Ausgabe des

CKY-Algorithmus besteht aus der sogenannten *Parsechart* $chart[.,.,.]$, die für einen Teilsatz der *Länge* $1 \leq l \leq n$ und dessen *Startindex* $1 \leq s \leq n+1-l$, sowie für eine Grammatikkategorie A einen Boole'schen Wert

$$chart[s, A, s+l] \in \{TRUE, FALSE\}$$

wie folgt annimmt:

$$chart[s, A, s+l] = \begin{cases} TRUE & \text{falls } A \Rightarrow^* w_s \ldots w_{s+l-1} \text{ nicht leer ist ,} \\ FALSE & \text{sonst .} \end{cases}$$

Der CKY-Algorihmus verwaltet also, anstatt der Parsewälder $forest[s, A, s+l]$, lediglich die Informationen $chart[s, A, s+l]$, unabhängig ob die Parsewälder leer sind oder nicht. Der Chart-Algorithmus ist daher eine einfache Modifikation des Parsewald-Algorithmus, die im wesentlichen auf den folgenden Booleschen Gleichungen beruht (für $|w_A| \geq 2$):

$$chart(\ A \Rightarrow^* w_A\ ) := \begin{cases} TRUE & \text{Die folgende Menge ist nicht leer:} \\ & \bigvee\limits_{\substack{A \to BC, \\ w_A = w_B w_C}} \quad \begin{array}{ccc} & A & \\ B \Rightarrow^* w_B & & C \Rightarrow^* w_C \end{array} \\ FALSE & \text{sonst} \end{cases}$$

$$= \bigvee_{\substack{A \to BC, \\ w_A = w_B w_C}} \begin{cases} TRUE & \text{Die folgende Menge ist nicht leer:} \\ & \begin{array}{ccc} & A & \\ B \Rightarrow^* w_B & & C \Rightarrow^* w_C \end{array} \\ FALSE & \text{sonst} \end{cases}$$

$$= \bigvee_{\substack{A \to BC, \\ w_A = w_B w_C}} \left(chart(B \Rightarrow^* w_B) \wedge chart(C \Rightarrow^* w_C) \right) .$$

Da der CKY-Algorithmus eine Chart benutzt und das Füllen der Chart für Wortfolgen mit der kleinster Länge 1 beginnt und für monoton grösser werdende Wortfolgen mit der Wortfolge von maximaler Länge n endet, wird der CKY-Parser auch ein *Bottom-Up-Chart-Parser* genannt. Der CKY-Algorithmus ist wie der Parsewald-Algorithmus von der Ordnung $O(n^3)$.

Der sogenannte *Count-Algorithmus* ist der erste Parsing-Algorithmus, der als ein stochastischer Parsing-Algorithmus vorgestellt werden soll. Er ist in Abbildung 2.13 wiedergegeben.