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:

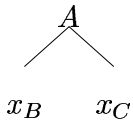
$$A \Rightarrow^* w_A \ = \ \left\{ \begin{array}{c} A \\ x_B \quad x_C \end{array} \ \middle| \ \begin{array}{c} A \to BC, \\ w_A = w_B w_C, \\ x_B \in\ B \Rightarrow^* w_B, \\ x_C \in\ C \Rightarrow^* w_C \end{array} \right\}$$

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

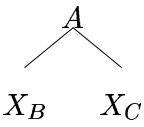
$$= \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} \begin{array}{c} A \\ B \Rightarrow^* w_B \quad C \Rightarrow^* w_C \end{array} \ .$$

[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



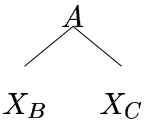
für den aus diesen drei Teilen in natürlicher Weise zusammengesetzten Syntaxbaum mit der Wurzel A benutzt. Schliesslich benutzt man für eine Menge X_A von Syntaxbäumen mit Wurzel A und eine Menge X_B von Syntaxbäumen mit Wurzel B und für eine Regel $A \to BC$ naheliegenderweise das Symbol



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.

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

Ausgabe: Die Anzahl $count[s, A, s+l]$ der grammatischen Analysen für alle Teilsätze der Länge $1 \leq l \leq n$, Startindizes $1 \leq s \leq n+1-l$, und Grammatikkategorien A.

Prozedur:

1. $count[.,.,.] := 0$;
2. **for each** $s := 1, \dots, n$ **do**
3. **for each** $A \to w_s$ **do**
4. $count[s, A, s+1] := 1$;
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. $count[s, A, s+l] := count[s, A, s+l] +$
10. $count[s, B, s+t] \cdot count[s+t, C, s+l]$

Abbildung 2.13: Count-Algorithmus

Obwohl seine Eingabe wie beim Parsewald-Algorithmus und dem CKY-Algorithmus nur aus einer rein symbolischen Grammatik besteht, ist der Count-Algorithmus seiner Natur nach eher ein stochastischer Algorithmus, da er Häufigkeiten (die Zahl der Analysen aller Teilsätze), und damit stochastische Basisgrössen, verwaltet. Ein weiterer Grund, weshalb der Count-Algorithmus eher zu den stochastischen Parsing-Algorithmen gezählt werden sollte, ist seine grosse Ähnlichkeit mit dem Inside-Algorithmus, welcher statt Häufigkeiten Wahrscheinlichkeiten von Analysen berechnet.

Die Eingabe des Count-Algorithmus besteht aus einer Grammatik in Chomsky-Normalform und einem Satz $w = w_1 \ldots w_n$, $(n \geq 1)$, dessen Anzahl der Syntaxbäume zu ermitteln ist. Die Ausgabe des Count-Algorithmus besteht aus den sogenannten *Counts*:

$$count[s, A, s+l] \;:=\; \mid A \Rightarrow^* w_s \ldots w_{s+l-1} \mid .$$

die für einen Teilsatz der *Länge* $1 \leq l \leq n$ und dessen *Startindex* $1 \leq s \leq n+1-l$ die Anzahl seiner Syntaxbäume mit Topknoten A angibt.

Auch der Count-Algorithmus ist nur eine einfache Modifikation des Parsewald-Algorithmus, weil für eine Wortfolge w_A mit $|w_A| \geq 2$ die folgende Identität gilt:

$$\begin{aligned}
count(A \Rightarrow^* w_A) \;&:=\; \mid A \Rightarrow^* w_A \mid \\
&= \left| \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} \quad \begin{array}{ccc} & A & \\ B \Rightarrow^* w_B & & C \Rightarrow^* w_C \end{array} \right| \\
&= \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} \left| \begin{array}{ccc} & A & \\ B \Rightarrow^* w_B & & C \Rightarrow^* w_C \end{array} \right| \\
&= \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} |B \Rightarrow^* w_B| \cdot |C \Rightarrow^* w_C| \\
&= \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} count(B \Rightarrow^* w_B) \cdot count(C \Rightarrow^* w_C) \; .
\end{aligned}$$

In der 1. Zeile des Count-Algorithmus werden die Analyseanzahlen mit 0 initialisiert, während in der 2.-4. Zeile der Induktionsanfang und in der 5.-10. Zeile stattfindet. Die

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

Ausgabe: Die Inside-Wahrscheinlichkeiten $inside[s, A, s+l] \in [0,1]$ für alle Teilsätze der Länge $1 \leq l \leq n$, Startindizes $1 \leq s \leq n+1-l$, und Grammatikkategorien A.

Prozedur:

1. $inside[.,.,.] := 0$;
2. **for each** $s := 1, \dots, n$ **do**
3. **for each** $A \to w_s$ **do**
4. $inside[s, A, s+1] := p(A \to w_s)$;
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. $inside[s, A, s+l] := inside[s, A, s+l] +$
10. $p(A \to B\ C) \cdot inside[s, B, s+t] \cdot inside[s+t, C, s+l]$;

Abbildung 2.14: Inside-Algorithmus

eben hergeleitete Identität wird in der 9. Zeile angewandt. Selbstverständlich ist auch der Count-Algorithmus von der Ordnung $O(n^3)$.

Der sogenannte *Inside-Algorithmus*, der in Abbildung 2.14 wiedergegeben ist, ist der erste, der hier vorgestellten Parsing-Algorithmen, der tatsächlich eine nicht-symbolische Grammatik als Eingabe fordert und somit als erster "echter stochastischer Parsing-Algorithmus" gelten kann:

Die Eingabe des Inside-Algorithmus besteht aus einer probabilistischen Grammatik in Chomsky-Normalform und einem Satz $w = w_1 \dots w_n$, $(n \geq 1)$, dessen Analyse-Wahrscheinlichkeit zu ermitteln ist. Die Ausgabe des Inside-Algorithmus besteht aus den sogenannten *Inside-Wahrscheinlichkeiten*:

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

für alle Teilsätze der *Länge* $1 \leq l \leq n$ und *Startindizes* $1 \leq s \leq n+1-l$, sowie Grammatikkategorien A. Die Inside-Wahrscheinlichkeit eines Teilsatzes ist somit die Summe der Wahrscheinlichkeiten aller Syntaxbäume (mit Wurzelkategorie A) dieses Teilsatzes und unterscheidet sich von dessen Count nur dadurch, dass ein Syntaxbaum mit seiner

Wahrscheinlichkeit, anstatt der Zahl 1, gezählt wird. Auch der Inside-Algorithmus ist eine einfache Modifikation des Parsewald-Algorithmus, weil für eine Wortfolge w_A mit $|w_A| \geq 2$ die folgenden Umformungen möglich sind, und diese ein ähnliches Ergebnis wie beim Count-Algorithmus liefern:

$$
\begin{aligned}
\mathit{inside}(\ A \Rightarrow^* w_A) \ &:= \ p(\ A \Rightarrow^* w_A) \\
&= \ p\left(\sum_{\substack{A \to BC, \\ w_A = w_B w_C}} \quad \begin{array}{ccc} & A & \\ B \Rightarrow^* w_B & & C \Rightarrow^* w_C \end{array} \right) \\
&= \ \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} p\left(\begin{array}{ccc} & A & \\ B \Rightarrow^* w_B & & C \Rightarrow^* w_C \end{array} \right) \\
&= \ \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} \ \sum_{\substack{x_B \in B \Rightarrow^* w_B \\ x_C \in C \Rightarrow^* w_C}} p\left(\begin{array}{ccc} & A & \\ x_B & & x_C \end{array} \right) \\
&= \ \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} \ \sum_{\substack{x_B \in B \Rightarrow^* w_B \\ x_C \in C \Rightarrow^* w_C}} p(A \to BC) \cdot p(x_B) \cdot p(x_C) \\
&= \ \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} p(A \to B\ C) \cdot \sum_{\substack{x_B \in B \Rightarrow^* w_B \\ x_C \in C \Rightarrow^* w_C}} p(x_B) \cdot p(x_C) \\
&= \ \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} p(A \to B\ C) \sum_{x_B \in B \Rightarrow^* w_B} p(x_B) \sum_{x_C \in C \Rightarrow^* w_C} p(x_C) \\
&= \ \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} p(A \to B\ C) \cdot p(B \Rightarrow^* w_B) \cdot p(C \Rightarrow^* w_C)
\end{aligned}
$$

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

Ausgabe: Die Viterbi-Wahrscheinlichkeiten $Viterbi[s, A, s+l] \in [0, 1]$ für alle Teilsätze der Länge $1 \leq l \leq n$, Startindizes $1 \leq s \leq n + 1 - l$, und Grammatikkategorien A.

Prozedur:

1. $Viterbi[.,.,.] := 0$;
2. **for each** $s := 1, \dots, n$ **do**
3. **for each** $A \to w_s$ **do**
4. $Viterbi[s, A, s + 1] := p(A \to w_s)$;
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. $Viterbi[s, A, s + l] := \max\{Viterbi[s, A, s + l],$
10. $p(A \to B\ C) \cdot Viterbi[s, B, s + t] \cdot Viterbi[s + t, C, s + l]\}$;

Abbildung 2.15: Viterbi-Algorithmus

$$= \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} p(A \to B\ C) \cdot inside(B \Rightarrow^* w_B) \cdot inside(C \Rightarrow^* w_C)\ .$$

In der 1. Zeile des Inside-Algorithmus werden die Inside-Wahrscheinlichkeiten mit 0 initialisiert, während in der 2.-4. Zeile der Induktionsanfang und in der 5.-10. Zeile stattfindet. Die eben hergeleitete Identität wird in der 9. Zeile angewandt. Auch der Inside-Algorithmus ist von der Ordnung $O(n^3)$.

Der sogenannte *Viterbi-Algorithmus*, der in Abbildung 2.15 wiedergegeben ist, ist vermutlich der bekannteste aller stochastischen Parsing-Algorithmen, da er das Disambiguierungsproblem angeht, indem er die grösste Wahrscheinlichkeit aller Satzanalysen ermittelt.

Die Eingabe des Viterbi-Algorithmus besteht aus einer probabilistischen Grammatik in Chomsky-Normalform und einem Satz $w = w_1 \dots w_n$, $(n \geq 1)$, dessen grösste Analyse-Wahrscheinlichkeit zu ermitteln ist. Die Ausgabe des Viterbi-Algorithmus besteht aus den sogenannten *Viterbi-Wahrscheinlichkeiten*:

$$Viterbi[s, A, s + l] \ := \ \max\{p(x_A) \mid x_A \in A \Rightarrow^* w_s \dots w_{s+l-1}\}$$

für alle Teilsätze der *Länge* $1 \leq l \leq n$ und *Startindizes* $1 \leq s \leq n+1-l$, sowie Grammatikkategorien A. Die Viterbi-Wahrscheinlichkeit eines Teilsatzes ist somit die grösste Wahrscheinlichkeit aller Syntaxbäume (mit Wurzelkategorie A) dieses Teilsatzes. Wie alle bisherigen Algorithmen ist auch der Viterbi-Algorithmus eine einfache Modifikation des Parsewald-Algorithmus, was darauf zurückzuführen ist, dass für eine Wortfolge w_A mit $|w_A| \geq 2$ die folgende Identität gilt:

$$
\begin{aligned}
\mathit{Viterbi}(A \Rightarrow^* x_A) &:= \max\{p(x_A) \mid x_A \in A \Rightarrow^* x_A\} \\
&= \max\left\{ p\left(\begin{array}{ccc} & A & \\ \diagup & & \diagdown \\ x_B & & x_C \end{array} \right) \;\middle|\; \begin{array}{c} A \to BC, \\ w_A = w_B w_C, \\ x_B \in B \Rightarrow^* w_B, \\ x_C \in C \Rightarrow^* w_C \end{array} \right\} \\
&= \max\left\{ p(A \to BC) \cdot p(x_B) \cdot p(x_C) \;\middle|\; \begin{array}{c} A \to BC, \\ w_A = w_B w_C, \\ x_B \in B \Rightarrow^* w_B, \\ x_C \in C \Rightarrow^* w_C \end{array} \right\} \\
&= \max\left\{ \begin{array}{c} p(A \to BC) \cdot \\ \max\{p(x_B) | x_B \in B \Rightarrow^* w_B\} \cdot \\ \max\{p(x_C) | x_C \in C \Rightarrow^* w_C\} \end{array} \;\middle|\; \begin{array}{c} A \to BC, \\ w_A = w_B w_C \end{array} \right\} \\
&= \max\left\{ \begin{array}{c} p(A \to BC) \cdot \\ \mathit{Viterbi}(B \Rightarrow^* x_B) \cdot \\ \mathit{Viterbi}(C \Rightarrow^* x_C) \end{array} \;\middle|\; \begin{array}{c} A \to BC, \\ w_A = w_B w_C \end{array} \right\} .
\end{aligned}
$$

In der 1. Zeile des Viterbi-Algorithmus in Abbildung 2.15 werden die Inside-Wahrscheinlichkeiten mit 0 initialisiert, während in der 2.-4. Zeile der Induktionsanfang und in der 5.-10. Zeile stattfindet. Die eben hergeleitete Identität wird in der 9. Zeile angewandt. Der Viterbi-Algorithmus ist von der Ordnung $O(n^3)$.

Damit soll dieser Abschnitt beschlossen werden. Drei weitere noch nicht besprochene stochastische Parsingverfahren, der Viterbi-Parse-, der N-Best- und der N-Best-Parse-Algorithmus, sind dem Viterbi-Algorithmus sehr ähnlich. Sie werden im Abschnitt 2.7 über State-of-the-Art Parsingverfahren als Instanzen des Semiring-Parsing-Algorithmus eingeführt.

2.7 State-of-the-Art Parsingverfahren

Die Überschrift deutet darauf hin, dass in diesem Abschnitt die allerneuesten symbolischen und stochastischen Parsingverfahren vorgestellt werden. Hierzu wird jedoch auf das Kapitel 5 verwiesen, in welchem Experimente zu lexikalisierten probabilistischen kontextfreien Grammatiken und stochastischen Unifikationsgrammatiken vorgestellt werden. Dieser Abschnitt ist vielmehr einer Arbeit gewidmet, die zu einer völlig neuen Auffassung der meisten effizienten, dynamischen Parsingverfahren für kontextfreie Grammatiken führte.

Obwohl, vielleicht auch weil, der CKY-Algorithmus (Kasami 1965) schon relativ alt ist, wurde mit seiner Hilfe eine aufregende Entdeckung für stochastische Grammatiken gemacht: Goodman (1998) wies in jüngster Zeit nach, dass eine Vielzahl der symbolischen und stochastischen, äusserst effizienten und dynamischen Parsing-Algorithmen, rund um den CKY-Algorithmus, prinzipiell als Ausprägungen eines einzigen Algorithmus, dem sogenannten *Semiring-Parsing-Algorithmus*, aufgefasst werden können. Die unterschiedlichen Leistungen der Parsing-Algorithmen (siehe Abschnitt 2.6), von denen die Fähigkeit zur Disambiguierung wohl die wichtigste ist, resultieren dabei lediglich aus unterschiedlichen Instantiierungen des Semiring-Parsing-Algorithmus. Dies hat mehrere Folgen:

Erstens ist Goodmans Entdeckung zunächst einmal von allgemeinem theoretischen Interesse, da sie zu einem besseren Verständnis der bekannten stochastischen Parsingverfahren für kontextfreie Grammatiken beitragen kann.

Zweitens kann man hoffen, dass in naher Zukunft für kontextfreie Grammatiken ein optimales (für ein gegebenes stochastisches Parsingproblem nicht weiter verbesserbares) stochastisches Parsingverfahren vorhanden sein wird. Denn aufgrund Goodman's Entdeckung darf vermutet werden, dass jedes optimale Parsingverfahren auf der mathematischen Struktur der sogenannten *Semiringe* basiert. Die Suche nach einem optimalen Parsingverfahren scheint also auf die Suche nach einem optimalen Semiring (für das gegebene Parsingproblem) zurückführbar zu sein. Dies ist eine sehr viel leichtere Aufgabe.

Drittens weist Goodman's Entdeckung den Weg zu effizienten Parsingverfahren, da der Semiring-Parsing-Algorithmus geradezu als Prototyp dynamischer Programmierung angesehen werden kann. Unterschiedliche Implementierungen eines beliebigen Parsingverfahrens müssen deshalb nicht mehr auf der Grundlage der Speicher- und Zeitanalyse der gesamten Implementierung bewertet werden, sondern können allein auf der Grundlage der Speicher- und Zeitanalyse der additiven und multiplikativen Operationen des Semirings vorgenommen werden.

Viertens ist Goodman's Entdeckung für die Entwicklung völlig neuer stochastischer Parsingverfahren interessant, da man jeden Semiring prinzipiell als ein neues Parsingverfah-

ren auffassen kann. Relevant sind in diesem Zusammenhang die verschiedenen Gewichtungsmöglichkeiten der Grammatikregeln, sowie die Frage, welche Gewichtungen sich (in annehmbarer Laufzeit) auf die Parsewälder fortsetzen lassen. Am Ende des Abschnitts wird hierzu ein Beispiel vorgestellt.

Fünftens wird die Implementierung vieler verschiedener Parsing-Algorithmen in der einen Gestalt des Semiring-Parsing-Algorithmus durch die modernen objekt-orientierte Programmiersprachen (wie C++, Java, Perl++) optimal unterstützt, weil diese gestatten, die Semiringe als sogenannte Klassen oder Objekte zu implementieren.

Aus diesen Gründen ist der Semiring-Parsing-Algorithmus ein absolutes State-of-The-Art Parsingverfahren.

Da zur Zeit für unifikationsbasierte Grammatiken kein symbolischer Standard-Parsing--Algorithmus bekannt ist, den man für analoge Untersuchungen heranziehen könnte[21], und leider auch kein effizienter stochastischer Parsing-Algorithmus existiert, kann man nicht hoffen, ähnlich vorzügliche Resultate für unifikationsbasierte Grammatiken zu erzielen.

Das State-of-The-Art Parsingverfahren wird also ein Parsingverfahren für kontextfreie (oder einfachere) Grammatiken sein. Vieles deutet daraufhin, dass die sogenannte Lexikalisierung eine wichtige Rolle spielen wird. In Kapitel 5 wird gezeigt, dass probabilistische lexikalisierte kontextfreie Grammatiken ähnlich effizient geparst und trainiert werden können, wie gewöhnliche kontextfreie Grammatiken.

Im folgenden werden Semiringe und der Semiring-Parsing-Algorithmus eingeführt und nachgewiesen, dass die im Abschnitt 2.6 vorgestellten symbolischen und stochastischen Parsing-Algorithmen spezielle Instanzen des Semiring-Parsing-Algorithmus sind. Ferner werden drei weitere stochastische Parsingverfahren direkt als Instanzen des Semiring-Parsing-Algorithmus vorgestellt werden:

- *Viterbi-Parse-Algorithmus*: Berechnung des *Viterbi-Parses*, d.h. der Satzanalyse mit der maximalen Wahrscheinlichkeit,
- *N-Best-Algorithmus*: Berechnung der *N-Best-Wahrscheinlichkeiten*, d.h. der N grössten Wahrscheinlichkeiten aller Satzanalysen,
- *N-Best-Parse-Algorithmus*: Berechnung der *N-Best-Parses*, d.h. aller Satzanalysen, welche die N-Best-Wahrscheinlichkeiten tragen.

Insbesondere der Viterbi-Parse- und der etwas allgemeinere N-Best-Parse-Algorithmus

[21] Der Early-Deduktions-Algorithmus ist fast so etwas wie ein Standardalgorithmus für das Parsen von unifikationsbasierten Grammatiken, wenn man von dem Problem der Unentscheidbarkeit des Parsens mit unifikationsbasierten Grammatiken absieht.

werden sehr oft zu Disambiguierungszwecken herangezogen. Alle drei Algorithmen ähneln dem schon in Abschnitt 2.6 vorgestellten Viterbi-Algorithmus.

In Abschnitt 2.6 wurden für kontextfreie Grammatiken in Chomsky-Normalform sehr detailliert zwei symbolische Parsingverfahren (Parsewald-, CKY-Algorithmus), ein zwischen Symbolik und Stochastik anzusiedelndes Parsingverfahren (Count-Algorithmus), sowie zwei stochastische Parsingverfahren (Inside-Algorithmus, Viterbi-Algorithmus) vorgestellt. Es fällt auf, dass die Prozeduren dieser fünf Parsingverfahren in grossen Teilen übereinstimmen. Die Unterschiede der Algorithmen sind neben unterschiedlicher Ein- und Ausgabe in den folgenden Prozedurzeilen zu lokalisieren:

- **1. Zeile**: Initialisierung der Parsechart mit den Werten $\emptyset$ (Parsewald), $FALSE$ (CKY), 0 (Count, Inside, Viterbi),
- **4. Zeile**: Induktionsanfang, d.h. Gewichtung von $A \Rightarrow^* w_s$ mit den Werten $\{A \rightarrow w_s\}$ (Parsewald), $TRUE$ (CKY), 1 (Count), $p(A \rightarrow w_s)$ (Inside, Viterbi),
- **9.+10. Zeile**: Induktionsschluss, d.h. rekursive Gewichtung von $A \Rightarrow^* w_A$ mittels der Gewichte von $B \Rightarrow^* w_B$ und $C \Rightarrow^* w_C$, sowie (eventuell) eines Gewichts, welches der Grammatikregel $A \rightarrow B\ C$ zugewiesen ist: $p(A \rightarrow B\ C)$ (Inside, Viterbi). Benutzung zweier Rechenoperationen $\vee, \wedge$ (CKY), $+, \cdot$ (Count, Inside), $max, \cdot$ (Viterbi).

Es fällt weiter auf, dass selbst diese Zeilen sehr ähnlich sind. Der in der 9. und 10. Zeile stattfindende Induktionsschluss weist hierbei die grössten Unterschiede auf. Um diese Unterschiede auf eine abstraktere Ebene anzuheben, reicht es, sich klarzumachen, dass am Induktionsschluss die folgenden Elemente beteiligt sind:

- gewisse **Werte** (die den Parsewäldern $A \Rightarrow^* w_A$ rekursiv zugewiesen werden, und den Grammatikregeln $A \rightarrow B\ C$ fest zugewiesen sind),
- eine **Addition**, die mit diesen Werten umgehen kann, sowie
- eine **Multiplikation**, die ebenfalls auf diesen Werten operiert.

Aus der Algebra sind verschiedene mathematische Strukturen $< \mathcal{A}, \oplus, \otimes >$ bekannt, die auf einer abstrakten Menge $\mathcal{A}$ von Elementen, eine abstrakte Addition $\oplus$ und eine abstrakte Multiplikation $\otimes$ erklären. Diese Strukturen heissen *Körper, Ring, Semiring, Gruppe, Ideal,...* abhängig davon, welche Eigenschaften Addition und Multiplikation haben.

Es wird sich erweisen, dass die kontextfreien Parsingverfahren die Eigenschaften sogenannter Semiringe $< \mathcal{A}, \oplus, \otimes, 0, 1 >$ haben. (0 und 1 sind neutrale Elemente der Addition

Elementmenge $\mathcal{A}$: $\{TRUE,\ FALSE\}$ (die Menge der Boolschen Elemente)

Addition $\oplus$: $\vee$ (das logische "oder")

Multiplikation $\otimes$: $\wedge$ (das logische "und")

Additives neutrales Element 0: $FALSE$ (das logische "falsch")

Multiplikatives neutrales Element 1: $TRUE$ (das logische "wahr")

Werte gemäss Grammatik:

$$\mu_{\mathcal{A}}(r) := TRUE \qquad \text{für alle Grammatikregeln } r$$

Abbildung 2.16: CKY-Semiring

Elementmenge $\mathcal{A}$: $\mathcal{N}$ (die Menge der natürlichen Zahlen)

Addition $\oplus$: $+$ (die gewöhnliche Addition)

Multiplikation $\otimes$: $\cdot$ (die gewöhnliche Multiplikation)

Additives neutrales Element 0: 0 (die gewöhnliche Null)

Multiplikatives neutrales Element 1: 1 (die gewöhnliche Eins)

Werte gemäss Grammatik:

$$\mu_{\mathcal{A}}(r) := 1 \qquad \text{für alle Grammatikregeln } r$$

Abbildung 2.17: Count-Semiring

und der Multiplikation, die erst weiter unten definiert werden sollen.) Um dies einzusehen, können mehrere Wege eingeschlagen werden. Der einfachste Weg ist, nicht direkt von einer bestimmten mathematischen Struktur auszugehen, sondern von einem der in Abschnitt 2.6 vorgestellten Parsing-Algorithmen, zum Beispiel von dem CKY-, Count-, Inside-, oder Viterbi-Algorithmus; der Parsewald-Algorithmus ist ungeeignet.

Die in dem ausgewählten Algorithmus benutzte Addition und Multiplikation kann dann gegen eine abstrakte Addition $\oplus$ und eine abstrakte Multiplikation $\otimes$ von abstrakten Chart-Werten $chart_{\mathcal{A}}[.,A,.]$ und Grammatikregel-Werten $\mu_{\mathcal{A}}(A \rightarrow BC)$ aus der abstrakten Menge $\mathcal{A}$ ausgetauscht werden. Die Abbildungen 2.16, 2.17, 2.18, und 2.19 zeigen explizit, welche mathematische Struktur $< \mathcal{A}, \oplus, \otimes, 0, 1 >$ in diesen vier Fällen benötigt wird.

Elementmenge $\mathcal{A}$: $[0,1]$ (das Intervall reeller Zahlen von 0 bis 1)

Addition $\oplus$: $+$ (die gewöhnliche Addition)

Multiplikation $\otimes$: $\cdot$ (die gewöhnliche Multiplikation)

Additives neutrales Element 0: 0 (die gewöhnliche Null)

Multiplikatives neutrales Element 1: 1 (die gewöhnliche Eins)

Werte gemäss Grammatik:

$$\mu_{\mathcal{A}}(r) := p(r) \qquad \text{für alle Grammatikregeln } r$$

Abbildung 2.18: Inside-Semiring

Elementmenge $\mathcal{A}$: $[0,1]$ (das Intervall reeller Zahlen von 0 bis 1)

Addition $\oplus$: max (das Maximum zweier reeller Zahlen)

Multiplikation $\otimes$: $\cdot$ (die gewöhnliche Multiplikation)

Additives neutrales Element 0: 0 (die gewöhnliche Null)

Multiplikatives neutrales Element 1: 1 (die gewöhnliche Eins)

Werte gemäss Grammatik:

$$\mu_{\mathcal{A}}(r) := p(r) \qquad \text{für alle Grammatikregeln } r$$

Abbildung 2.19: Viterbi-Semiring

Eingabe: Ein Semiring $< \mathcal{A}, \oplus, \otimes, 0, 1 >$, eine Grammatik in Chomsky-Normalform, eine Gewichtung $\mu_{\mathcal{A}}(r) \in \mathcal{A}$ der Grammatikregeln r mit Semiring-Elementen, sowie ein Satz $w = w_1 \ldots w_n$.

Ausgabe: Die Semiring-Chart $chart_{\mathcal{A}}[s, A, s+l]$ für alle Teilsätze der Länge $1 \leq l \leq n$, Startindizes $1 \leq s \leq n+1-l$, und Grammatikkategorien A.

Prozedur:

1. $chart_{\mathcal{A}}[.,.,.] := 0$;
2. **for each** $s := 1, \ldots, n$ **do**
3. **for each** $A \rightarrow w_s$ **do**
4. $chart_{\mathcal{A}}[s, A, s+1] := \mu_{\mathcal{A}}(A \rightarrow w_s)$;
5. **for each** $l := 2, \ldots, n$ **do**
6. **for each** $s := 1, \ldots, n+1-l$ **do**
7. **for each** $t := 1, \ldots, l-1$ **do**
8. **for each** $A \rightarrow B\ C$ **do**
9. $chart_{\mathcal{A}}[s, A, s+l] := chart_{\mathcal{A}}[s, A, s+l] \oplus$
10. $\mu_{\mathcal{A}}(A \rightarrow B\ C) \otimes chart_{\mathcal{A}}[s, B, s+t] \otimes$
11. $chart_{\mathcal{A}}[s+t, C, s+l]$;

Abbildung 2.20: Semiring-Parsing-Algorithmus

Wie man sieht, sind sich die Strukturen sehr ähnlich: Vom Count-Semiring kommt man zum Inside-Semiring, indem man die Elementmenge ($\mathcal{N}$ versus $[0,1]$) und die Gewichte der Grammatikregeln r austauscht (1 versus $p(r)$). Vom Inside-Semiring kommt man zum Viterbi-Semiring, indem man die Addition austauscht (max versus +). Viele weitere Beispiele dieser Art wären möglich. Vermutlich ähneln sich die Semiringe aller Parsingverfahren, weil sie alle für die Lösung ähnlicher Probleme konzipiert wurden.

Nach viermaliger Anwendung des Verfahrens hat man zusätzlich zu den vier vorgestellten Semiringen einen abstrakten Parsing-Algorithmus erhalten, welcher in Abbildung 2.20 zu sehen ist und recht treffend *Semiring-Parsing-Algorithmus* genannt werden könnte.

Der Semiring-Parsing-Algorithmus ist ein abstrakter Parsing-Algorithmus, der vor einer Instanziierung mit einem Semiring, weder ein symbolischer noch ein stochastischer Parsing-Algorithmus ist. Wegen der Gewichtung $\mu_{\mathcal{A}}(.)$ der Grammatikregeln ähnelt er allerdings mehr einem stochastischen Parsing-Algorithmus. Während seiner Laufzeit, ist der Semiring-Parsing-Algorithmus natürlich nicht von einem symbolischen

Parsingverfahren ($\mathcal{A}_{CKY}$) oder einem stochastischen Parsingverfahren (beispielsweise $\mathcal{A}_{count}, \mathcal{A}_{inside}, \mathcal{A}_{Viterbi}$) zu unterscheiden.

Die Eingabe des Semiring-Parsing-Algorithmus besteht aus einer Grammatik in Chomsky-Normalform und einem Satz $w = w_1 \ldots w_n$, $(n \geq 1)$, sowie einem Semiring $< \mathcal{A}, \oplus, \otimes, 0, 1 >$ und einer Gewichtung $\mu_{\mathcal{A}}(.)$ der Grammatikregeln, die in manchen Fällen aus Regel-Wahrscheinlichkeiten besteht und dann die Eingabe-Grammatik zu einer stochastischen Grammatik macht.

Die Ausgabe des Semiring-Parsing-Algorithmus besteht aus den Elementen der *Semiring-Chart*:

$$chart_{\mathcal{A}}[s, A, s+l] \;:=\; \mu_{\mathcal{A}}(\; A \Rightarrow^* w_s \ldots w_{s+l-1} \;) \;.$$

Einem Teilsatz der *Länge* $1 \leq l \leq n$ und dessen *Startindex* $1 \leq s \leq n+1-l$ wird der Wert seines Parsewaldes (der Wurzelkategorie A) zugewiesen, den dieser im Semiring $\mathcal{A}$ besitzt. Hierfür muss garantiert werden, dass die Gewichtung $\mu_{\mathcal{A}}(.)$, die ja zunächst nur für Grammatikregeln definiert ist, sich sinnvoll auf Parsewälder fortsetzen lässt. Die Frage, welche Eigenschaften die abstrakte mathematische Struktur $< \mathcal{A}, \oplus, \otimes, 0, 1 >$ erfüllen muss, damit der Semiring-Parsing-Algorithmus sinnvolle Eigenschaften besitzt, ist deshalb sehr interessant. Die wichtigste Eigenschaft des Semiring-Parsing-Algorithmus sollte sicherlich sein, dass er möglichst viele Parsingverfahren, insbesondere aber den Parsewald-Algorithmus als Instanz besitzt. Für eine Wortfolge w_A mit $|w_A| \geq 2$ sollte daher gelten, dass:

$$
\begin{aligned}
& chart_{\mathcal{A}}(A \Rightarrow^* w_A) \\
& \quad := \mu_{\mathcal{A}}(A \Rightarrow^* w_A) \\
& \quad = \mu_{\mathcal{A}} \left(\sum_{\substack{A \to BC, \\ w_A = w_B w_C}} \; \begin{array}{ccc} & A & \\ & \diagup \quad \diagdown & \\ B \Rightarrow^* w_B & & C \Rightarrow^* w_C \end{array} \right) \\
& \quad = \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} \mu_{\mathcal{A}} \left(\begin{array}{ccc} & A & \\ & \diagup \quad \diagdown & \\ B \Rightarrow^* w_B & & C \Rightarrow^* w_C \end{array} \right)
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} \mu_{\mathcal{A}}(A \to B\ C) \otimes \mu_{\mathcal{A}}(B \Rightarrow^* w_B) \otimes \mu_{\mathcal{A}}(C \Rightarrow^* w_C) \\
&= \sum_{\substack{A \to BC, \\ w_A = w_B w_C}} \mu_{\mathcal{A}}(A \to B\ C) \otimes \mathit{chart}_{\mathcal{A}}(B \Rightarrow^* w_B) \otimes \mathit{chart}_{\mathcal{A}}(C \Rightarrow^* w_B)
\end{aligned}
$$

Offensichtlich wird diese Rekursion in der 9.-11. Zeile der Prozedur des Semiring-Parsing-Algorithmus realisiert. Alle weiteren Zeilen der Prozedur sind klar: In der 1. Zeile des Semiring-Parsing-Algorithmus wird die Semiring-Chart mit dem additiv neutralen Element 0 initialisiert, während in der 2.-4. Zeile der Induktionsanfang und in der 5.-11. Zeile der Induktionsschluss stattfindet. Interessant ist noch die Benutzung der Werte $\mu_{\mathcal{A}}(A \to a)$ in der 4. Zeile.

Der Semiring-Parsing-Algorithmus ist von der Ordnung $O(n^3)$, wenn die Addition und Multiplikation zweier Elemente des Semirings in konstanter Zeit vorgenommen werden können, was bei allen vier vorgestellten Semiringen der Fall ist.

Im folgenden soll gelöst werden, welche Eigenschaften die abstrakte Addition $\oplus$ und die abstrakte Multiplikation $\otimes$ haben müssen, um im Semiring-Parsing-Algorithmus sinnvoll eingesetzt werden zu können. Wie bereits gezeigt wurde, muss für die rekursive Berechnung von $\mathit{chart}_{\mathcal{A}}(C \Rightarrow^* w_B)$ der äquivalente Ausdruck

$$
\sum_{\substack{A \to BC, \\ w_A = w_B w_C}} \mu_{\mathcal{A}}(A \to B\ C) \otimes \mathit{chart}_{\mathcal{A}}(B \Rightarrow^* w_B) \otimes \mathit{chart}_{\mathcal{A}}(C \Rightarrow^* w_B)
$$

Sinn machen. Man liest diesem Ausdruck ab, dass die Addition $\oplus$ kommutativ und assoziativ sein muss (sonst ist die Notation $\sum$ sinnlos), die Multiplikation $\otimes$ muss hingegen nur assoziativ sein (sonst macht der ungeklammerte Ausdruck $\mu_{\mathcal{A}}(A \to B\ C) \otimes \mathit{chart}_{\mathcal{A}}(B \Rightarrow^* w_B) \otimes \mathit{chart}_{\mathcal{A}}(C \Rightarrow^* w_B)$ keinen Sinn). Ferner muss die Multiplikation ein neutrales Element besitzen (weil $\mu_{\mathcal{A}}(A \to B\ C)$ in manchen Parsingverfahren konstant 1 ist). Ebenso muss die Addition ein neutrales Element besitzen (weil die Summe $\sum$ mit 0 initialisiert werden muss und auch weil ein Summand $\mu_{\mathcal{A}}(A \to B\ C) \otimes \mathit{chart}_{\mathcal{A}}(B \Rightarrow^* w_B) \otimes \mathit{chart}_{\mathcal{A}}(C \Rightarrow^* w_B)$ manchmal 0 sein kann).

Die Abbildung 2.21 fasst die erforderlichen Eigenschaften zusammen. Es sind die Eigenschaften eines Semirings, die man in jedem Buch über Algebra nachschlagen kann, zum Beispiel in (Van der Warden 1936). Erwähnenswert ist, dass die Multiplikation eines Semirings nicht kommutativ sein muss, was den Semiring zu einem Ring machen würde.

Elementmenge $\mathcal{A}$: beliebig

Addition $\oplus$:

$$\begin{array}{lll} a \oplus b = b \oplus a & \forall a, b \in \mathcal{A} & \text{(Kommutativität)} \\ (a \oplus b) \oplus c = a \oplus (b \oplus c) & \forall a, b, c \in \mathcal{A} & \text{(Assoziativität)} \end{array}$$

Multiplikation $\otimes$:

$$\begin{array}{lll} (a \otimes b) \otimes c = a \otimes (b \otimes c) & \forall a, b, c \in \mathcal{A} & \text{(Assoziativität)} \\ a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) & \forall a, b, c \in \mathcal{A} & \text{(Distributivität)} \\ a \otimes 0 = 0 = 0 \otimes a & \forall a \in \mathcal{A} & \end{array}$$

Additives neutrales Element 0:

$$a \oplus 0 = a = 0 \oplus a \quad \forall a \in \mathcal{A}$$

Multiplikatives neutrales Element 1:

$$a \otimes 1 = a = 1 \otimes a \quad \forall a \in \mathcal{A}$$

Werte gemäss Grammatik:

$$\mu_{\mathcal{A}} : \; G \to \mathcal{A}, \quad r \mapsto \mu_{\mathcal{A}}(r) \; .$$

Abbildung 2.21: Eigenschaften eines Semirings

Elementmenge $\mathcal{A}$:

$$\left\{ X \subseteq \mathcal{T}(G) \;\middle|\; \begin{array}{l} X \text{ ist leer oder von der Gestalt:} \\ \{A \to B\ C\},\ X_B, \\ \begin{array}{cc} A & \\ X_B & C \end{array} \text{ oder } \begin{array}{cc} ? & \\ X_B & X_C \end{array} \end{array} \right\}$$

Hierbei bezeichnet $\{A \to B\ C\}$ einen aus einer einzigen verzweigenden Grammatikregel bestehenden Parsewald, X_B eine Teilmenge der Syntaxbäume eines festen (aber beliebigen) Teilsatzes w_B (d.h. $X_B \subseteq B \Rightarrow^* w_B$), sowie:

$$\begin{array}{cc} A & \\ X_B & C \end{array} := \left\{ \begin{array}{cc} A & \\ x_B & C \end{array} \;\middle|\; x_B \in X_B \right\} \text{ und } \begin{array}{cc} ? & \\ X_B & X_C \end{array} := \langle X_B, X_C \rangle$$

Addition $\oplus$: wird (im Semiring-Parsing-Algorithmus) nur für disjunkte $X_1,\ X_2 \subset \mathcal{T}(G)$ der Gestalt X_A benötigt:

$$X_1 \oplus X_2 := X_1 \cup X_2\ .$$

Multiplikation $\otimes$: wird (im Semiring-Parsing-Algorithmus) nur für die folgenden vier Fälle benötigt:

$$\{A \to B\ C\} \otimes X_B := \begin{array}{cc} A & \\ X_B & C \end{array} \text{ und } \begin{array}{cc} A & \\ X_B & C \end{array} \otimes X_C := \begin{array}{cc} A & \\ X_B & X_C \end{array}$$

sowie:

$$X_B \otimes X_C := \begin{array}{cc} ? & \\ X_B & X_C \end{array} \text{ und } \{A \to B\ C\} \otimes \begin{array}{cc} ? & \\ X_B & X_C \end{array} := \begin{array}{cc} \mathbf{A} & \\ X_B & X_C \end{array}$$

Wie man leicht sieht, ist die so definierte Multiplikation wohldefiniert, abgeschlossen, assoziativ, aber nicht kommutativ.

Additives neutrales Element 0: $\emptyset$ (die leere Menge)

Multiplikatives neutrales Element 1: die den "leeren Syntaxbaum" enthaltende Menge. Wird (im Semiring-Parsing-Algorithmus) nicht benötigt.

Werte gemäss Grammatik:

$$\mu_{\mathcal{A}}(r) := \{r\} \qquad \text{für alle Grammatikregeln } r$$

Abbildung 2.22: Parsewald-Semiring

Im folgenden werden vier weitere Semiringe vorgestellt (Parsewald-, Viterbi-Parse, N-Best-, N-Best-Parse-Semiring), die sich in Goodman (1998) ebenfalls finden lassen. Bis auf den Viterbi-Parse-Semiring werden in dieser Arbeit im Vergleich zu Goodman's Arbeit abweichende, weil einfachere und genauere, Angaben gemacht. Dies betrifft vor allem den Parsewald-Semiring, der in Abbildung 2.22 angegeben ist und der bei Goodman nur sehr schwer zu verstehen ist.

Der Parsewald-Semiring instantiiert den Parsewald-Algorithmus (siehe Abbildung 2.11). Die Addition und Multiplikation des Parsewald-Semirings wird nur für die im Semiring--Parsing-Algorithmus vorkommenden Elemente eingeführt. Der Schlüsselschritt ist, dass erkannt wird, dass im Semiring-Parsing-Algorithmus zur Bildung des Produkts $\mu_{\mathcal{A}}(A \to BC) \otimes \mathit{chart}_{\mathcal{A}}(B \Rightarrow^* w_B) \otimes \mathit{chart}_{\mathcal{A}}(C \Rightarrow^* w_C)$, kürzer als

$$\{A \to BC\} \otimes X_B \otimes X_C$$

notiert, die folgenden vier Typen von Semiring-Elementen zur Verfügung stehen müssen (je nachdem in welcher Reihenfolge das Produkt berechnet wird):

$$\{A \to B\ C\}, \quad X_B, \quad \{A \to B\ C\} \otimes X_B, \quad X_B \otimes X_C\ .$$

In Abbildung 2.22 findet man diese vier Typen in etwas sinnvollerer Notation:

$$\{A \to B\ C\}, \quad X_B, \quad \begin{array}{ccc} & A & \\ \swarrow & & \searrow \\ X_B & & C \end{array}\ , \quad \begin{array}{ccc} & ? & \\ \swarrow & & \searrow \\ X_B & & X_C \end{array}\ .$$

Bemerkenswert ist, dass die Multiplikation des Parsewald-Semirings nicht kommutativ ist.

Die Abbildung 2.23 zeigt den Semiring zur Bestimmung der Viterbi-Parses. Da es natürlich mehr als einen Parse maximaler Wahrscheinlichkeit geben kann, muss zu jeder Viterbi-Wahrscheinlichkeit ein ganzer Parsewald verwaltet werden. In den meisten Implementierungen von stochastischen Parsern wird darauf allerdings verzichtet und (mittels Zufallsauswahl) lediglich ein Viterbi-Parse verwaltet. Leider zerstört ein derartiges Vorgehen die Assoziativität der Addition, sodass diese Implementatierungen mangelhaft sind.

Recht hübsch ist der Rückgriff auf Addition und Multiplikation des Viterbi- und Parsewald-Semirings, was die Verständlichkeit der gezeigten Operationen stark erhöht.

Abschliessend werden zwei Parsing-Algorithmen vorgestellt, die den Viterbi- und den Viterbi-Parse-Algorithmus verallgemeinern.

Mit dem N-Best-Algorithmus sollen die N-Best-Wahrscheinlichkeiten, d.h. die N grössten Wahrscheinlichkeiten aller Parses, bestimmt werden. Mit dem N-Best-Parse-Algorithmus werden die N-Best-Parses, d.h. die dazugehörigen Parses, ermittelt.

Elementmenge $\mathcal{A}$: $\mathcal{A}_{Viterbi} \times \mathcal{A}_{Parsewald}$

Addition $\oplus$:

$$< p, X_p > \oplus < q, X_q > := \begin{cases} < p \oplus_{Viterbi} q,\ X_p \oplus_{Parsewald} X_q > & \text{falls } p = q \\ < p \oplus_{Viterbi} q,\ X_{p \oplus_{Viterbi} q} > & \text{falls } p \neq q \end{cases}$$

Multiplikation $\otimes$:

$$< p, X_p > \otimes < q, X_q > := < p \otimes_{Viterbi} q,\ X_p \otimes_{Parsewald} X_q >$$

Additives neutrales Element 0: $< 0_{Viterbi}, 0_{Parsewald} >$

Multiplikatives neutrales Element 1: $< 1_{Viterbi}, 1_{Parsewald} >$

Werte gemäss Grammatik:

$$\mu_{\mathcal{A}}(r) := < \mu_{\mathcal{A}Viterbi}(r),\ \mu_{\mathcal{A}Parsewald}(r) > \text{ für alle Grammatikregeln } r$$

Abbildung 2.23: Viterbi-Parse-Semiring

Elementmenge $\mathcal{A}$:

$$\{M \subset \mathcal{A}_{Viterbi} \mid 1 \leq |M| \leq N\}$$

Addition $\oplus$: die gewöhnliche Mengenvereinigung, gefolgt von der Extraktion der N grössten Zahlen:

$$M_1 \oplus M_2 := N\text{–best}(M_1 \cup M_2)$$

Multiplikation $\otimes$:

$$M_1 \otimes M_2 := N\text{–best}\{m_1 \otimes_{Viterbi} m_2 \mid m_1 \in M_1,\ m_2 \in M_2\}$$

Additives neutrales Element 0: $\{0_{Viterbi}\}$

Multiplikatives neutrales Element 1: $\{1_{Viterbi}\}$

Werte gemäss Grammatik:

$$\mu_{\mathcal{A}}(r) := \{\ \mu_{\mathcal{A}Viterbi}(r)\ \} \text{ für alle Grammatikregeln } r$$

Abbildung 2.24: N-Best-Semiring ($N = 1$ (Viterbi-Semiring), $2, 3, \ldots$)

Ein einfaches Beispiel zeigt, dass die formale Definition von "N-Best" recht unangenehm ist. Sind beispielsweise $<0.4, x_1>$, $<0.4, x_2>$, $<0.05, x_3>$, $<0.05, x_4>$, $<0.05, x_5>$, $<0.03, x_6>$, $<0.02, x_7>$ die sieben Parses eines gegebenen Satzes, zusammen annotiert mit ihren Analyse-Wahrscheinlichkeiten, so ist die 1-Best-Wahrscheinlichkeit (d.h. die Viterbi-Wahrscheinlichkeit) gleich 0.4, die 2-Best-Wahrscheinlichkeiten sind $\{0.04, 0.05\}$, und die 3-Best-Wahrscheinlichkeiten sind $\{0.04, 0.05, 0.03\}$. Somit gibt es zwei 1-Best-Parses (Viterbi-Parses) $\{x_1, x_2\}$, sogar fünf 2-Best-Parses $\{x_1, \ldots, x_5\}$, sowie sechs 3-Best-Parses $\{x_1, \ldots, x_6\}$.

Der einfachste Weg zu einer formalen Definition eines $N-$best(.)-Operators führt über Ordnungen. Sind

$$<p_1, X_1> \ldots <p_1, X_{i_1}>, <p_2, X_{1+i_1}> \ldots <p_2, X_{i_2}>, <p_3, X_{1+i_2}> \ldots <p_3, X_{i_3}> \ldots$$

die Elemente einer endlichen Menge M von 2-dimensionalen Daten, die so angeordnet sind, dass in der ersten Koordinate die folgende Kette von Ungleichungen gilt:

$$p_1 > p_2 > p_3 > \ldots ,$$

so gilt $i_1 < i_2 < i_3 < \ldots \leq |M|$, sowie:

$$N\text{–best}(M) := \{\ <p_i, X_i> \in M \mid i \leq i_N\ \} .$$

Für eine Menge M von 1-dimensionalen Daten, d.h. für Zahlenmengen, erhält man natürlich analog:

$$i_1 = 1,\ i_2 = 2,\ i_3 = 3, \ldots .$$

Dieser wichtige Spezialfall besagt, dass N–best(.) aus einer Zahlenmenge deren N grössten Zahlen extrahiert:

$$N\text{–best}(M) := \{\ p_i \in M \mid i \leq N\ \} .$$

Offensichtlich kann hier gefolgert werden, dass $|N\text{–best}(M)| \leq N$. Es wurde bereits per Beispiel gezeigt, dass dies im allgemeinen nicht gelten wird.

Der $N-$best-Operator ist der Schlüssel zu beiden Semiringen, dem N-Best- und dem N-Best-Parse-Semiring. Bei einer Implementierung sollte darauf geachtet werden, dass der N–best-Operator in konstanter Zeit arbeitet, was für Zahlenmengen einfach zu realisieren ist, aber für Mengen von 2-dimensionalen Daten schwieriger erscheint.

Die Abbildung 2.24 zeigt den N-Best-Semiring. Mit diesem Semiring liegt eine Verallgemeinerung des Viterbi-Semirings vor, den man für $N = 1$ erhält. Natürlich ist es sinnvoll bei der Definition des N-Best-Semirings auf den Viterbi-Semiring zurückzugreifen: Die

Elementmenge $\mathcal{A}$:

$$\{M \subset \mathcal{A}_{Viterbi-Parse} \mid 1 \leq |M| \leq N\}$$

Addition $\oplus$: die gewöhnliche Mengenvereinigung, gefolgt von der $N-$best(.)-Extraktion:

$$M_1 \oplus M_2 := N\text{–best}(M_1 \cup M_2)$$

Multiplikation $\otimes$:

$$M_1 \otimes M_2 := N\text{–best}\{m_1 \otimes_{Viterbi-Parse} m_2 \mid m_1 \in M_1,\ m_2 \in M_2\}$$

Additives neutrales Element 0: $\{0_{Viterbi-Parse}\}$

Multiplikatives neutrales Element 1: $\{1_{Viterbi-Parse}\}$

Werte gemäss Grammatik:

$$\mu_{\mathcal{A}}(r) := \{\ \mu_{\mathcal{A}Viterbi-Parse}(r)\ \} \text{ für alle Grammatikregeln } r$$

Abbildung 2.25: N-Best-Parse-Semiring ($N = 1$ (Viterbi-Parse-Semiring), $2, 3, \ldots$)

Elemente des N-Best-Semiring sind die 1- bis N-elementigen Teilmengen des Viterbi-Semirings. Die Addition besteht aus der gewöhnlichen Mengenvereinigung, die Multiplikation aus einer kreuzweisen Produktbildung, beide Operationen gefolgt von der Anwendung des N–best-Operators, der in diesem Fall die N grössten Zahlen extrahiert.

Die Abbildung 2.25 zeigt den N-Best-Parse-Semiring. Mit diesem Semiring liegt eine Verallgemeinerung des Viterbi-Parse-Semirings vor, den man für $N = 1$ erhält. Natürlich ist es sinnvoll, bei der Definition des N-Best-Parse-Semirings auf den Viterbi-Parse-Semiring zurückzugreifen: Die Elemente des N-Best-Parse-Semirings sind die 1 bis N-elementigen Teilmengen des Viterbi-Parse-Semirings. Die Addition besteht aus der gewöhnlichen Mengenvereinigung, die Multiplikation hingegen aus einer kreuzweisen Produktbildung, für welche natürlich die Multiplikation des Viterbi-Parse-Semirings benutzt wird. Beide Operationen werden durch die Anwendung des N–best-Operators vervollständigt, der in diesem Fall die Parsewälder aller N N-Best-Wahrscheinlichkeiten extrahiert.