Diplomarbeit

Entwicklung eines Agenten zur Unterstützung der Umformatierung von Literaturhinweisen

Eckart Mansfeld

Diplomarbeit
am Fachbereich Informatik
der Universität Dortmund

3. März 2000

Betreuer:

Prof. Dr. Katharina Morik

Dipl. Inform. Ralf Klinkenberg

Inhaltsverzeichnis

Kapitel 1

Aufgabenstellung

1.1 Einleitung

Heutzutage stehen im Internet riesige Datenbestände zur Verfügung, die von den Nutzern des Internet zur Lösung ihrer Aufgaben herangezogen werden können. Mit dieser Fülle an Information kommen jedoch wieder neue Aufgaben auf einen Anwender zu. Ein Problem bei der Nutzung dieser Daten stellt die Aufgabe dar, aus der Menge aller vorhandenen Daten die jeweils relevanten Daten herauszusuchen.

Zum anderen liegen die Daten entweder völlig unstrukturiert oder aber an verschiedenen Stellen in unterschiedlichen Formaten vor. Sollen diese Daten zur Weiterverarbeitung als Eingabe eines Programmes dienen oder in eine eigene Datenbank eingetragen werden, so müssen sie zunächst in die benötigte Struktur überführt werden. Bei großen Datenmengen kann dies einen erheblichen Zeitaufwand bedeuten, was häufig dazu führt, daß nur ein kleiner Teil der verfügbaren Informationen tatsächlich verwendet wird.

Um diese Aufgaben zu vereinfachen, werden von erfahrenen Anwendern kleine Programme eingesetzt, die die Konversion bestimmter Datenformate automatisch durchführen. Dieses Vorgehen hat jedoch den Nachteil, daß für jedes neue Datenformat auch ein neues Programm entwickelt werden muß, daß die Daten in das Zielformat überführt. Diese Programme müssen also von dem Anwender, der die Daten für eigene Anwendungen weiterverwenden will, selbst erzeugt werden. Den meisten Anwendern fehlen jedoch die hierzu notwendigen Kenntnisse.

Schon seit den Anfängen der Informatik besteht der Gedanke, Computer zu entwickeln, die nicht programmiert werden müssen, sondern die man wie ein Kind erziehen kann. Der Computer wird mit leerem Speicher seiner Umwelt ausgesetzt und lernt, die ihm zugewiesenen Aufgaben zu erfüllen. Ein später formuliertes spezielleres Prinzip ist das Programming by Example. Dabei sollen Systeme entwickelt werden, die aus Beispielen für Ein- Ausgabepaare ein Programm synthetisieren, das die geforderte Aufgabe erledigt. Der Benutzer muß also keine Programmierkenntnisse besitzen, sondern kann seinem Computer durch Beispiele "beibringen", wie die Aufgabe zu erledigen ist.

1.2 Aufgabenstellung

Im Rahmen dieser Arbeit wird nun ein spezieller Typus von Daten herausgegriffen, nämlich Literaturhinweise. Im Internet sind sehr viele Dokumente verfügbar, die Literaturlisten enthalten. Darüber hinaus gibt es viele Bibliographien für bestimmte Fachgebiete.

Will ein Benutzer nun diese Angaben in seine eigene Arbeit oder in eine persönliche Bibliographie übernehmen, so muß er in der Regel den gefundenen Eintrag umformatieren, da die Formate der verwendeten Quellen meist nicht dem vom Benutzer verwendeten Format entsprechen. Hierzu soll nun ein Verfahren entwickelt werden, mit dessen Hilfe aus vom Benutzer gegebenen Beispielen gelernt werden kann, wie diese Umformatierung vorzunehmen ist, so daß die Umformatierung dann automatisiert weitergeführt werden kann.

Ein Eingabe-Ausgabepaar könnte dabei folgendermassen aussehen:

> Angluin, D. (1982). Inference of reversible languages. Journal of the Association for Computing Machinery, 29, 741-765.
>
> ==>
>
> D. Angluin. Inference of reversible languages. Journal of the Association for Computing Machinery, 29:741-765, 1982.

Beispiel 1.1

Damit ein solches System vom Benutzer akzeptiert wird und ihm tatsächlich nützlich ist, sollten nach Möglichkeit folgende Bedingungen erfüllt werden (siehe [Witten et al., 1996]):

- Der Benutzer muß schnell feststellen, daß das System ihn unterstützen wird. Daher sollte der Agent schon nach wenigen ersten Beispielen brauchbare Vorschläge für die Übersetzungen machen. Dabei können aber auch unvollständige Ausgaben, an denen nur kleinere Korrekturen vorgenommen werden müssen, eine Hilfe sein.

- Die Anzahl der Rückfragen an den Benutzer sollte so gering wie möglich gehalten werden.
- Die Rückfragen sollten so einfach wie möglich gestaltet werden. Zum einen sollte der Benutzer nicht mehr Arbeit in die Beantwortung der Fragen stecken müssen, als zur eigenhändigen Bearbeitung der Aufgabe nötig wäre. Zum anderen sollten die Fragen vom Benutzer möglichst leicht zu beantworten sein, ohne daß der Benutzer genaue Kenntniss der Struktur von Literaturhinweisen haben muß. [1]

Hauptziel ist die Entwicklung eines Verfahrens, das die Umformatierung von Literaturhinweisen aus Beispielen lernt und dabei möglichst wenig Hintergrundwissen voraussetzt. Um den Lernprozess zu beschleunigen kann zwar Hintergrundwissen vorgegeben werden, das Verfahren sollte aber im Stande sein auch ohne umfangreiches Hintergrundwissen auszukommen.

1.3 Existierende Systeme

Es existieren bereits Systeme, die zur Unterstützung eines Benutzers bei der Umformatierung von Literaturangaben dienen.

J. Schirra et al. haben z.B. in [Schirra et al., 1985] ein System beschrieben, das bei der Erfassung von Literaturhinweisen für eine Feste Literaturdatenbank eingesetzt werden soll. Dieses System benutzt umfangreiches vorgegebenes Hintergrundwissen über Literaturhinweise, um auch unvollständige und inkonsistente Literaturhinweise erfassen zu können. Bei diesem System ist noch nicht vorgesehen, daß das System das Sachbereichswissen selbstständig erweitert.

In [Witten et al., 1996] wird ein Prototyp für ein System beschrieben, das durch umfangreiche natürlichsprachliche Interaktion mit dem Benutzer lernt, seine Aufgabe zu lösen.

Parmentier und Belaid beschreiben wiederum ein Verfahren, das aus umfangreichen gelabelten Daten (bibtex Dateien) die Struktur von Literaturhinweisen in Postscriptfiles lernt ([Parmentier und Belaid, 1997]).

[1] Eventuell lernt das System eine Repräsentation des Problems, die nicht der Sicht des Benutzers entspricht. Dies kann bei der Korrektur durch den Benutzer zu erheblichen Problemen führen. Da der Benutzer hier wieder das Vorgehen des Systems durchschauen muß, um die Rückfragen sinnvoll zu beantworten.

Kapitel 2

Literaturlisten und ihre Formatierung

2.1 Literaturlisten

In diesem Abschnitt sollen zunächst die Eigenschaften von Literaturhinweisen und ihrer Formatierung genauer untersucht werden. Aus den hier gewonnenen Erkenntnissen wird dann eine Aufteilung des Problems der Umformatierung und ein grober Ablauf für eine mögliche Lösung des Problems abgeleitet.

In vielen Bereichen werden sogenannte Bibliographien oder Literaturlisten verwendet, die eine Auflistung von Publikationen darstellen, die für einen bestimmten Bereich oder eine bestimmte Publikation relevant sind. Diese Bibliographien sollen eine gute Übersicht über die Angaben der zitierten Publikationen geben, die notwendig sind, um diese in Bibliotheken, Buchhandlungen usw. ausfindig machen zu können. Eine solche Bibliographie befindet sich z.B. auch im Anhang dieser Arbeit (siehe Literaturverzeichnis), um dem Leser einen guten Zugriff auf die hier erwähnten Arbeiten zu ermöglichen. In vielen Fällen werden den Daten zu den Publikationen noch kommentierende Bemerkungen des Verfassers der Bibliographie hinzugefügt. Im Rahmen dieser Arbeit werde ich mich jedoch auf die unkommentierten Literaturlisten beschränken.

Eine solche Literaturliste ist eine Aneinanderreihung einzelner Literaturhinweise. Dabei beschreibt jeder einzelne Literaturhinweis jeweils eine Publikation. Ein einzelner Literaturhinweis ist eine Zeichenkette, die wiederum eine Aneinanderreihung von bestimmten Angaben zu der Publikation ist. Eine solche Zeichenkette könnte z. B. folgendermaßen aussehen:

Angluin, D. (1982). Inference of reversible languages. Journal of the Association for Computing Machinery, 29, 741-765.

Beispiel 2.1

Einzelne Abschnitte dieser Zeichenkette beschreiben nun bestimmte Angaben zu der bezeichneten Publikation. Z.B. bezeichnet "*Angluin*" den Nachnamen der Autorin, "*1982*" das Erscheinungsjahr, oder "*Journal [...] Computing Machinery*" die Zeitschrift, in der der Artikel erschienen ist. Diese Teilketten will ich im Folgenden als *Felder* bezeichnen. Andere Teilketten tragen selbst keine Information über die Publikation, sondern dienen nur dazu, die einzelnen Felder voneinander abzugrenzen. So dient in obigem Beispiel etwa die Zeichenfolge "). " dazu, die Jahreszahl vom Titel zu trennen. Diese Teilketten werde ich im folgenden als *Separatoren* oder *Trennzeichen* bezeichnen. Im folgenden Beispiel sind die Felder durch < und > umschlossen:

<Angluin>, <D.> (<1982>). <Inference of reversible languages>. <Journal of the Association for Computing Machinery>, <29>, <741>-<765>.

Beispiel 2.2

Man sieht also, daß auch jeder einzelne Literaturhinweis wiederum strukturiert ist und in Untereinheiten unterteilt werden kann. Ersetzt man in obigem Beispiel die Teilketten, die Felder darstellen durch Variablen ($v_1..v_8$), so ergibt sich folgende (äußere) Strukturbeschreibung (dabei wird ein Leerzeichen der Deutlichkeit halber durch $\sqcup$ ersetzt):

$$v_1, \sqcup v_2 \sqcup (v_3). \sqcup v_4. \sqcup v_5, \sqcup v_6, \sqcup v_7 - v_8.$$

Beispiel 2.3

Felder, die nicht weiter unterteilt werden können, ohne ihre Information zu verlieren, will ich im folgenden *atomar* nennen. Solche atomaren Felder sind z.B. eine Jahreszahl oder ein Titel. Diese Felder kommen immer sowohl in der Eingabe, als auch in der Ausgabe als Ganzes vor, und werden niemals durch andere Felder, bzw. Zeichenketten unterbrochen. So kann der obige Literaturhinweis in einem anderen Format niemals Zeichenketten wie "*... Inference of. 1982, reversible Languages ...*" enthalten.

Gegebenenfalls kann es jedoch vorkommen, daß diese Felder verkürzt werden oder bestimmte Ersetzungen vorgenommen werden. So könnte "*1982*" zu "*82*" oder "*Proceedings of the ninth*" zu *Proc. ninth* werden. Diese Veränderungen beziehen sich jedoch immer nur auf den Inhalt eines Feldes, ohne daß der Kontext dabei zu berücksichtigen ist. Die kleinsten Einheiten, mit denen bei der Umformatierung von Literaturhinweisen operiert werden muß, sind also diese atomaren Felder. Dabei ist es jedoch möglich, daß auch eine Zerlegung in größere Einheiten sinnvoll ist. Werden zum Beispiel in Eingabe- und Ausgabeformat Seitenzahlen in der Form <Seite 1>-<Letzte Seite> angegeben, so können diese bei der Übersetzung auch immer als Einheit betrachtet werden. Felder sind also immer atomar, oder sie bestehen aus einer Kette von atomaren Feldern und Trennzeichen, wobei immer vollständige atomare Felder enthalten sind. Unser Beispiel könnte also auch wie folgt aufgeteilt werden:

$$v_1, \sqcup v_2 \sqcup (v_3). \sqcup v_4. \sqcup v_5, \sqcup v_6, \sqcup v_7.$$

Das Beispiel 1.1 kann man damit z.B. auf folgende Arten darstellen:

$$v_1, \sqcup v_2 \sqcup (v_3). \sqcup \underline{v_4. \sqcup v_5, \sqcup v_6}, \sqcup \underline{v_7 - v_8}.$$

$$==>$$

$$v_2 \sqcup v_1. \sqcup \underline{v_4. \sqcup v_5, \sqcup v_6} : \sqcup \underline{v_7 - v_8}, \sqcup v_3.$$

oder

$$v_1, \sqcup v_2 \sqcup (v_3). \sqcup v_9, \sqcup v_{10}.$$

$$==>$$

$$v_2 \sqcup v_1. \sqcup v_9 : \sqcup v_{10}. \sqcup (v_3).$$

Beispiel 2.4

Das erste Beispiel zeigt eine Zerlegung in die kleinsten sinnvollen Einheiten. Im zweiten Beispiel werden die im ersten Beispiel durch v_4, v_5, und v_6 repräsentierten Felder für den Aufsatztitel, den Zeitschriftentitel und die Nummer der Zeitschrift zu einem einzigen Feld v_9 zusammengefaßt. Ebenso werden die Seitenzahlen in v_{10} zusammengefaßt.

Hier zeigt sich auch, daß die obige Strukturbeschreibung allein nicht ausreicht, um eine Zeichenkette in die richtigen Teile zu zerlegen: Die Trennzeichen sind nicht eindeutig, sondern können auch im Innern der Felder auftreten. Will man die Zeichenkette des Literaturhinweises aus Beispiel 2.1 nach dem zweiten Muster von oben zerlegen, so sind z.B. folgende Zerlegungen möglich:

<Angluin>, <D.> (<1982>). <Inference of reversible languages. Journal of the Association for Computing Machinery >, <29, 741-765>.

oder

<Angluin>, <D.> (<1982>). <Inference of reversible languages. Journal of the Association for Computing Machinery, 29>, <741-765>.

Für das Trennzeichen zwischen v_4 und v_5 kommen also zwei Ausschnitte der Eingabe in Frage, so daß zwei verschiedene Belegungen von v_4 und v_5 möglich sind. Dies zeigt, daß noch weitergehende Beschreibungen benötigt werden, die aus den möglichen Zerlegungen die richtige auswählt.

Desweiteren lassen sich Literaturhinweise nach der Art der Publikation, die sie beschreiben, in Klassen aufteilen. Solche Klassen können z.B. Artikel in Zeitschriften, Bücher oder Konferenzbeiträge sein. Zu jeder dieser Klassen gibt es einen bestimmten Satz von möglichen Merkmalen, bzw. Feldern. Einige dieser Merkmale treten nur bei einzelnen Publikationsklassen auf, einige bei einer Gruppe von Klassen und einige immer. Merkmale, die allen Publikationstypen gemein sind, sind z.B. Autorennamen, Titel und Erscheinungsjahr. Die Angabe von Nummer oder Jahrgang tritt nur bei Zeitschriften auf, während z.B. Seitenzahlen bei Artikeln, Konferenz- und Buchbeiträgen, nicht jedoch bei Büchern oder Technischen Berichten auftreten. Die meisten Literaturlisten enthalten Publikationen aus drei bis vier solcher Klassen. Geht man z.B. von einem bis drei Autoren und einem optionalem Feld pro Klasse aus, so ergeben sich daraus zwölf bis sechzehn mögliche Feldkombinationen. Daraus folgt eine sehr große Variationsbereite bei der Struktur von Literaturhinweisen. Es kommen in einer Liste nur wenige Hinweise mit exakt gleicher Struktur vor.

Bei den Feldern selbst gibt es sehr große Unterschiede in der Struktur und Variationsbreite. Sehr kompliziert zu behandeln sind z.B. die Titelfelder. Als Titel können prinzipiell beliebige Sätze natürlicher Sprachen auftreten. Wiederholungen desselben Titels in unterschiedlichen Literaturhinweisen kommen praktisch nicht vor. Zeitschriften- und Konferenztitel hSSaben eine ähnliche Struktur, insbesondere Zeitschriftentitel wiederholen sich jedoch häufiger. Außerdem enthalten sie häufig Schlüsselwörter wie "Journal", "Proceedings" usw., die hilfreich sind. Auf der anderen Seite der Komplexitätsskala stehen Jahres- und Seitenzahlen. Für diese Elemente kann man sehr leicht vollständige Grammatiken angeben, die sich auch leicht lernen lassen. Verlags- oder Ortsnamen sind in ihrer Struktur zwar nicht so eindeutig, wiederholen sich jedoch sehr oft.

Ein Mensch ist bei der Umformatierung von Literaturhinweisen verhältnismäßig erfolgreich. Dabei greift er auf ein sehr umfangreiches Hintergrundwissen über die einzelnen Komponenten von Literaturhinweisen und ihre Beziehungen untereinander zurück. Er weiß zum Beispiel, daß zu einem Vornamen auch ein Nachname gehört, daß Stuttgardt ein Ort ist und so weiter. Allerdings stellen Literaturhinweise selbst für einen Menschen ein Problem dar. So ist zum Beispiel der folgende Eintrag auch für einen Menschen nicht ohne weiteres eindeutig:

> Dijkstra, E. W. [1959]. A Note on Two Problems in Connexion With Graphs. Numerische Mathematik 1, 269-271.

Ist dies nun ein Hinweis auf ein Lehrbuch mit dem Titel "Numerische Mathematik 1", oder einen Zeitschriftenartikel in Nummer 1 der Zeitschrift "Numerische Mathematik"?

2.2 Idee für ein Verfahren zur automatisierten Umformatierung von Literaturhinweisen

Zusammenfassend, lassen sich Literaturlisten also duch eine hierarchisch aufgebaute Struktur beschreiben. Auf der obersten Ebene wird eine solche Liste zunächst in einzelne Literaturhinweise unterteilt, die in der Regel durch einen Zeilenumbruch oder eine Leerzeile voneinander getrennt sind. Die einzelnen Literaturhinweise lassen sich wiederum als Abfolge von Feldern und Trennzeichen auffassen.

Aus den oben beschriebenen Eigenschaften dieser Struktur läßt sich nun eine Strategie für eine automatisierte Umformatierung ableiten: Zunächst sucht man eine Aufteilung eines eingegebenen Literaturhinweises in atomare Felder und Trennzeichen. Durch Weglassen der Trennzeichen erhält man eine Liste aller in der Eingabe enthaltenen atomaren Felder. Nun erzeugt man aus diesen Feldern unter Hinzufügen von neuen Trennzeichen einen String, der dem Ausgabeformat genügt. Sinnvollerweise gibt es für jede Kombination von Feldern eine eindeutige Anordnung im Ausgabeformat.

Zur Illustration will ich hier die einzelnen Arbeitsschritte mit dem bekannten Beispielhinweis angeben:

Der Eingabestring sieht folgendermaßen aus:

> Angluin, D. (1982). Inference of reversible languages. Journal of the Association for Computing Machinery, 29, 741-765.

Die Zerlegung in Felder und Trennzeichen:

<Angluin>, <D.> (<1982>). <Inference of reversible languages>. <Journal of the Association for Computing Machinery>, <29>, <741>-<765>.

Nun kommt die Liste der enthaltenen Felder, wobei ich den einzelnen Feldern sinnvolle Namen gegeben habe. Diese Namen können aber auch willkürliche (maschinell erzeugte) Bezeichnungen wie z.B. Feld1 sein:

- Nachname=Angluin
- Vorname=D.
- Jahr=1982
- Titel=Inference of reversible languages
- Zeitschrift=Journal of the Association for Computing Machinery
- Nummer=29
- Erste_Seite=741
- Letzte_Seite=765

Daraus wird dann folgende Ausgabe erzeugt:

<D.> <Angluin>. <Inference of reversible languages>. <Journal of the Association for Computing Machinery>, <29>:<741-765>, <1982>.

An diesem Beispiel läßt sich auch zeigen, daß für dieses Verfahren nicht umbedingt eine Aufteilung in atomare Felder nötig ist. Die Umformatierung lässt sich in diesem Falle auch durchführen, wenn die Anfangs- und Endseite zu einem gemeinsamen Feld zusammengefaßt werden. Man muß also eine "geeignete" Aufteilung finden, wobei geeignet heißt, daß die gefundenen Felder sowohl Teilstrings der Eingabe als auch der Ausgabe sind.

Zusammengefaßt besteht die Umformatierung von Literaturhinweisen also aus folgenden Schritten:

1. Zerlegen der Eingabe in Felder und Feldtrenner
2. Aufsammeln der Felder und der zugehörigen Strings
3. Zusammensetzen der Ausgabe aus den gefundenen Feldern und hinzuzufügenden Trennzeichen

Kapitel 3

Lernaufgaben und Verfahren

Im vorherigen Kapitel wurde eine grobe Strategie für ein Verfahren zur Übersetzung von Literaturhinweisen beschrieben. In diesem Kapitel will ich zunächst darstellen, wie sich das gegebene Problem als Problem des Grammatiklernens auffassen läßt, und dann die oben vorgeschlagene Strategie aus der Sicht des Syntaxlernens betrachten. Insbesondere wird die Bedeutung einiger Forschungsergebnisse aus diesem Bereich für das gegebene Problem erläutert. Im Anschluss werden einige Lernverfahren für derartige Lernprobleme beschrieben.

3.1 Syntax und Grammatik

Um das gegebene Problem genauer beschreiben zu können, will ich es hier unter dem Aspekt des Grammatik- oder Syntaxlernens beschreiben. Dazu gebe ich zunächst einen kurzen Überblick über die Syntaxlehre und übertrage diese dann auf das gegebene Problem.

Definition 3.1 Syntax([Linke et al., 1991]): *Der Terminus Syntax kommt von altgriech. syntaxis und heisst ursprünglich so viel wie "Zusammenstellung" oder "Anordnung". Üblicherweise wird in der Grammatik darunter die Lehre von der Anordnung der Wörter zu Sätzen verstanden.*

Die Syntaxlehre befaßt sich also damit, wie ein gültiger Satz einer Sprache aus Wörtern, bzw. Zeichen zusammengesetzt wird. Bei den betrachteten Sprachen kann es sich dabei um natürliche Sprachen, aber auch z.B. Programmiersprachen handeln. Grundsätzlich kann jedes Problem, bei dem aus einzelnen Zeichen nach bestimmten Regeln größere Einheiten gebildet werden, als syntaktisches Problem aufgefaßt werden. So kann man auch ein bestimmtes Literaturhinweisformat als

Sprache auffassen, wobei dann ein dem Format entsprechender Literaturhinweis ein gültiger Satz dieser Sprache ist. Einen Formalismus, der diese zulässige Anordnung von Wörtern zu Sätzen beschreibt, nennt man Grammatik. Die Bildung von Wörtern aus Zeichen wird in der Linguistik dabei in der Regel gesondert beschrieben. Der hierfür verwendete Formalismus heißt Lexikon([Linke et al., 1991]).

Bei der Konstruktion einer Grammatik für eine bestimmte Sprache hat man nun viele Möglichkeiten. So kann es im Falle sehr einfacher Sprachen, die nur eine geringe (endliche) Anzahl von Sätzen enthalten, ausreichen, einfach alle gültigen Sätze der Sprache aufzuzählen. Bei komplizierteren Sprachen macht man sich jedoch häufig zunutze, daß sich bei den meisten Sprachen eine hierarchische Struktur der Sätze erkennen läßt. Dabei läßt sich ein Satz in kleinere Einheiten zerlegen, die nun selbst wieder durch eine solche hierarchische Struktur beschrieben werden können. In der Linguistik werden diese Einheiten vom Satz bis hinunter zum Wort als Konstituenten bezeichnet. Die Struktur eines Satzes wird gerne als Baum wie in Abb. 3.1 dargestellt. Die Knoten eines solchen Baumes repräsentieren die Konstituenten und die Kanten die hierarchische Beziehung der Konstituenten untereinander. Die Blätter des Baumes sind dann die Wörter. Das Berechnen eines solchen Ableitungsbaumes für einen bestimmten Satz wird auch parsen genannt.

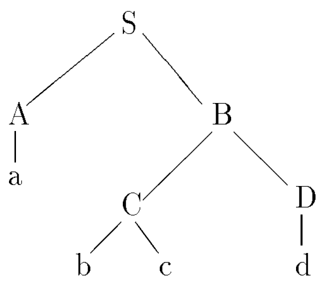

Abbildung 3.1: Ableitungsbaum

Die in der Informatik verbreitete Sicht und Terminologie (siehe z.B. ([Hopcroft und Ullman, 1979])), die aus der Theorie der formalen Sprachen stammt, unterscheidet sich etwas von der in der Linguistik verwendeten. Bei formalen Sprachen wird die Ebene des Lexikons übersprungen und direkt die Bildung von Sätzen aus Zeichen betrachtet. Verwirrenderweise heißt ein Satz hier Wort.

3.1.1 Formale Grammatiken

Eine für die formale Untersuchung von Sprachen verwendete Darstellungsform sind formale Grammatiken, die durch ein Tupel $G = (V_t, V_n, S, P)$ beschrieben werden. Dabei ist V_t eine Menge von Terminalsymbolen, V_n eine Menge von Nichtterminalsymbolen, S eine Menge von Startsymbolen und P eine Menge von Regeln. Die Regeln haben die Form $\alpha \rightarrow \beta$, wobei α und β jeweils Ketten von Terminalen und nichtterminalen Symbolen sind. Zu der von G beschriebenen Sprache gehören alle Worte, die nur aus Terminalsymbolen bestehen und in endlich vielen Schritten aus einem Startsymbol mit Hilfe der Regeln aus P abgeleitet werden können. Eine solche Ableitung kann wiederum wie in Abb. 3.1 bildlich dargestellt werden. In der Abbildung ist S das Startsymbol. Die nichtterminalen Symbole sind durch Großbuchstaben und die Terminalsymbole durch Kleinbuchstaben dargestellt.

Diese Grammatiken werden je nach Art der Regeln in vier Klassen eingeteilt. Diese Klassen unterscheiden sich in der Komplexität der von den zugehörigen Grammatiken beschreibbaren Sprachen. Beginnend mit den Chomsky-0 Grammatiken werden die Regeln immer weiter eingeschränkt bis hin zu der einfachsten Klasse der Chomsky-3 Grammatiken. Die einzelnen Klassen werden wie folgt eingeteilt([Hopcroft und Ullman, 1979]:

1. Chomsky-0 Grammatiken: Alle Grammatiken ohne weitere Einschränkungen der Regeln.

2. Chomsky-1 (kontextsensitive) Grammatiken: Die Produktionen haben die Form $c_1Ac_2 \rightarrow c_1\omega c_2$ mit $A \in V_n$, $\omega, c_1, c_2 \in V^*$ und ω ist nicht das leere Wort.

3. Chomsky-2 (kontextfreie) Grammatiken: Die Grammatiken enthalten Regeln der Form $A \rightarrow \omega$ mit $A \in V_n$, $\omega \in V^*$ und ω ist nicht das leere Wort.

4. Chomsky-3 (reguläre) Grammatiken: Die Grammatiken enthalten nur Regeln der Form $A \rightarrow aB$ und $A \rightarrow a$ mit $A, B \in V_n$ und $a \in V_t$.

Es gilt hier folgende Teilmengenbeziehung : Chomsky-3 $\subset$ Chomsky-2 $\subset$ Chomsky-1 $\subset$ Chomsky-0. Für die Chomsky-1 und Chomsky-0 Grammatiken ist das Wortproblem, also die Entscheidung, ob ein gegebenes Wort zu einer Sprache gehört NP-vollständig. Für Chomsky-2 Grammatiken ist zwar das Wortproblem effizient lösbar, jedoch sind die Frage, ob eine Sprache Teilmenge einer anderen ist und die Berechnung der Schnittmenge zweier Sprachen nicht entscheidbar.

Die Anordnung der Felder bei Literaturhinweisen sind mit regulären (Chomsky-3) Grammatiken beschreibbar. Die Inhalte der Felder können im Zweifelsfall beliebige Sätze einer natürlichen Sprache sein (zum Beispiel bei Titelfeldern). Damit sind sie in der Chomsky-Hierarchie wie natürliche Sprachen zwischen den kontextfreien und den kontextsensitiven Sprachen anzusiedeln. Allerdings muß die hier

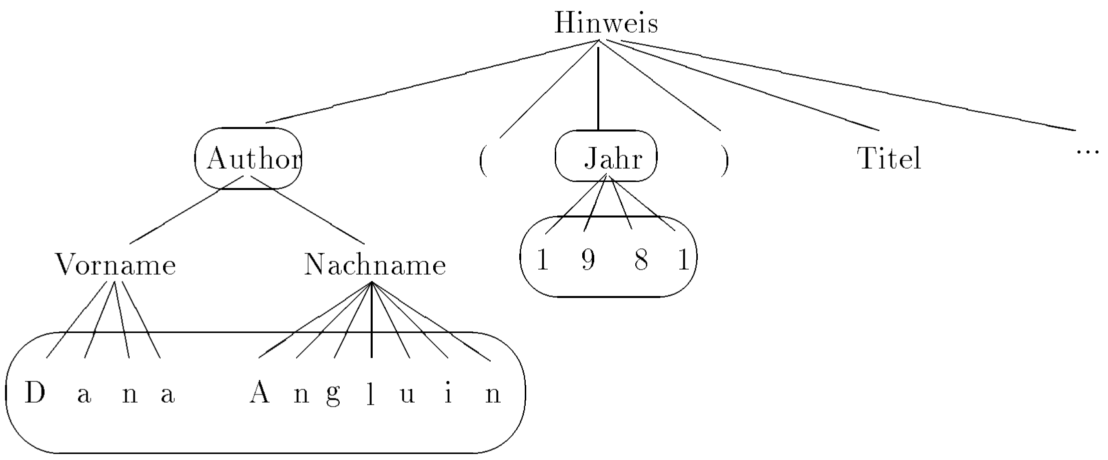

Abbildung 3.2: Ableitungsbaum

verwendete Beschreibung der Felder keine exakte Beschreibung der entsprechenden Sprache sein, so daß hier eventuell auch einfachere Beschreibungsformalismen verwendet werden können.

3.1.2 Umformatierung von Literaturhinweisen mit Hilfe von Grammatiken

3.1.2.1 Zerlegen der Eingabe

Hier will ich zunächst darstellen, daß sich Schritte 1 und 2 des in Abschnitt 2.2 beschriebenen Verfahrens als Parsen der Eingabe mit Hilfe einer geeigneten Grammatik auffassen läßt.

Man konstruiere eine Grammatik für das Eingabeformat, die für jedes Feld eine entsprechende Konstituente enthält. Aus diesen Konstituenten sollen sich alle Zeichenketten ableiten lassen, die an der Stelle des entsprechenden Feldes stehen können. Ermittelt man dann für eine Eingabe einen Ableitungsbaum mit Hilfe dieser Grammatik, so enthält dieser Ableitungsbaum für jedes Feld in der Eingabe eine entsprechende Konstituente. Ein Ausschnitt eines solchen Ableitungsbaumes könnte z.B. wie in Abb. 3.2 dargestellt aussehen.

Die gesuchten Informationen findet man nun, indem man zu jeder einem Feld entsprechenden Konstituente in dem Ableitungsbaum den unterhalb dieser Konstituente liegenden Teilstring der Eingabe aufsammelt. In Diagramm 3.2 sind z.B. die Konstituenten "Autor" und "Jahr" und die zugehörigen Zeichenketten durch Kreise gekennzeichnet. Wie man sieht, sind für die Lösung der Aufgabe also nur zwei Arten von Konstituenten wichtig: die den Feldern entsprechenden

Konstituenten und die Zeichen der Zeichenkette selbst. Eine Ableitung, die nur diese benötigten Elemente enthält, würde also wie folgt aussehen:

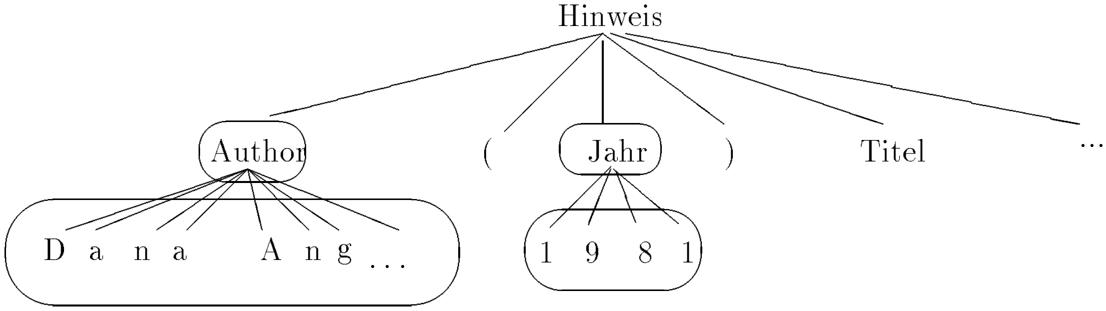

Abbildung 3.3: Flacher Ableitungsbaum

Das Lernen dieser Zerlegung aus Beispielen kann also als Lernen einer Grammatik aus Beispielen beschrieben werden. Es lassen sich nun bestimmte Randbedingungen angeben, die die gesuchte Grammatik erfüllen muß. Wie schon festgestellt, muß ein Strukturbaum eines Literaturhinweises eine Zerlegung in Felder beinhalten. Auf der anderen Seite können die Anforderungen an die gesuchte Grammatik jedoch auch gelockert werden. Die gesuchte Grammatik muß nämlich nicht zwingend exakt die Sprache der Literaturhinweise beschreiben, sondern darf auch eine Obermenge dieser Sprache beschreiben, also übergenerieren. Wichtig ist lediglich, daß ein gültiges Wort der Sprache durch eine Strukturbeschreibung mit der gesuchten Grammatik korrekt zerlegt wird. Wenn die Grammatik auch Zeichenketten akzeptiert, die keinen Literaturhinweis darstellen, so stört das nicht die Zerlegung von Literaturhinweisen.

Die Suche nach einer solchen Grammatik kann in Unterprobleme aufgeteilt werden. So kann man die Beschreibung der Literaturhinweise zunächst nur auf der Ebene der Felder und Trennzeichen beschreiben, wobei dann die Felder als Terminalsymbole betrachtet werden. Die Struktur der einzelnen Felder beschreibt man dann in einem zweiten Schritt, wobei man für jedes Feld eine eigene Grammatik angibt. Diese Aufteilung hat einige Vorteile. So sind die einzelnen Teilgrammatiken einfacher und somit auch leichter zu lernen. Außerdem hat man die Möglichkeit, für jede dieser Teilgrammatiken jeweils bestimmte Klassen von Grammatiken und Lernververfahren zu benutzen, die sich für die jeweilige Teilgrammatik am besten eignen. Zuletzt erhöht sich damit auch die Wiederverwendbarkeit des Lernergebnisses. Schließlich ist die Struktur einer Seitenzahl oder eines Vornamens bei allen oder zumindest bei vielen Formaten gleich. Hat man also einmal eine Beschreibung für ein Feld gelernt, so läßt sich diese Beschreibung in anderen Grammatiken weiterverwenden. Dies gilt natürlich insbesondere für die atomaren Felder.

Es soll allerdings nicht verschwiegen werden, daß diese Aufteilung nicht unbedingt verlustlos ist. So könnten die Inhalte benachbarter Felder Hinweise auf die Art eines bestimmten Feldes geben. Zum Beispiel kann der Name eines Fachbereiches im Verlagsfeld einen Hinweis darauf geben, daß eine Veröffentlichung ein Technischer Bericht ist. Wenn das Ausgabeformat einen Unterschied in der Formatierung zwischen Büchern und Technischen Berichten macht, so ist diese Information tatsächlich wichtig. Aus dem Inhalt des Titels selbst geht diese Information jedoch nicht hervor. In [Parmentier und Belaid, 1997] wird z.B. ein System vorgeschlagen, das solche Zusammenhänge nutzt.

Im folgenden will ich die Beschreibung der Literaturhinweise auf der Ebene der Felder als äußere Grammatik bezeichnen und die Beschreibung der einzelnen Felder als innere oder Feldgrammatik.

3.1.2.2 Erzeugen der Ausgabe

Auch Schritt 3, also das Erzeugen der Ausgabe, läßt sich mit Hilfe von Grammatiken beschreiben. Insgesamt ist dieser Schritt erheblich einfacher als der erste Schritt, die Zerlegung. Als Eingabe steht nun eine Liste der Felder des Literaturhinweises zur Verfügung und es muß eine Anordnung dieser Felder gefunden werden, die dem Ausgabeformat entspricht. Dazu benötigt man für die Ausgabesprache eine Grammatik, die die Anordnung der Felder beschreibt, also wieder eine äußere Grammatik. Da die Felder gerade so gewählt wurden, daß ihre Struktur im Ein- und Ausgabeformat identisch ist, ist hier eine innere Grammatik nicht notwendig. Um nun die Ausgabe zu erzeugen, sucht man unter allen möglichen Parsebäumen einen heraus, der genau die in Schritt 2 gefundenen Felder enthält. Der von diesem Parsebaum generierte String ist die gesuchte Ausgabe. Die hier benötigte Grammatik wird also nicht zur Analyse eines Strings, sondern zur Erzeugung eines Strings verwendet. Die gesuchte Grammatik muß also für eine gegebene Menge von Feldern genau einen String erzeugen, der dem korrekten Ausgabestring entspricht. Sie darf also nicht übergenerieren. Daher ist es vermutlich wiederum sinnvoll, für diese Grammatik eine andere Beschreibungsform und andere Lernverfahren zu verwenden, als für äußere Beschreibung der Eingabesprache.

3.1.2.3 Die zu lernenden Grammatiken (Zusammenfassung)

Um aus Beispielen zu lernen, wie Literaturhinweise umformatiert werden, muß man also drei verschiedene Grammatiken lernen.

1. Eine äußere Grammatik G_e, die die Eingabesprache auf der Ebene der Felder und Trennzeichen beschreibt. Diese Grammatik dient der Analyse der

Eingabe.

2. Für jedes Feld F_n eine innere Grammatik G_{F_n}, die die Struktur des Feldes F_n beschreibt. Dabei handelt es sich tatsächlich nicht um eine einzelne Grammatik, sondern um eine Menge von Grammatiken, nämlich für jedes Feld eine. Diese Grammatik dient ebenfalls der Analyse der Eingabe.

3. Eine äußere Grammatik G_a , die die Ausgabesprache auf der Ebene der Felder und Trennzeichen beschreibt. Diese Grammatik dient der Generierung der Ausgabe.

Diese Grammatiken werden dann wie folgt für die Umformatierung von Literaturhinweisen verwendet:

1. Erzeuge mit Hilfe der Grammatiken G_e und $F_1 \ldots F_n$ eine Strukturbeschreibung der Eingabe.

2. Erzeuge eine Liste aller in der Strukturbeschreibung gefundenen Felder.

3. Finde eine aus G_a ableitbare Strukturbeschreibung, die genau die gefundenen Felder enthält.

3.1.3 Lernen von Grammatiken

Ein wesentliches Motiv bei der Beschäftigung mit der Informatik war für viele Forscher von Anfang an die Modellierung menschlicher Intelligenz mit Hilfe von Computern. Als wesentliche Merkmale menschlicher Intelligenz werden die menschliche Sprache und die Fähigkeit zu lernen betrachtet. Daher waren diese beiden Fähigkeiten immer wichtiger Gegenstand der Forschung. Besonders interessant ist dabei natürlich die Kombination dieser beiden Fähigkeiten, nämlich das Lernen von Sprache. Da das Lernen von natürlichen Sprachen jedoch ungeheuer komplex ist, ist man immer noch weit von einer Lösung dieses Problems entfernt.

Um das Problem handhabbarer zu machen, verlegte man sich zunächst auf das Lernen von bestimmten Aspekten der Sprache. Dabei liegt ein wesentlicher Schwerpunkt der Forschung auf der Syntax. Dies liegt vor allem daran, daß sich die Syntax formalen Methoden besser erschließt als andere Aspekte der Sprache, wie die Semantik oder die Pragmatik.

Das Ziel beim Syntaxlernen ist, aus einer Menge von Wörtern eine Grammatik zu induzieren, die diese Beispiele erzeugt. Dieser Ansatz wird häufig als *grammatical inference* bezeichnet ([Langley, 1987]). Ein Problem dabei ist, daß es zu einer gegebenen Beispielmenge bis zu unendlich viele Grammatiken gibt, die diese Beispielmenge akzeptieren.

Gold stellte in [Gold, 1967] fest, daß schon reguläre Sprachen (siehe dazu 3.1.1) aus positiven Beispielen allein nicht effizient erlernbar sind. In [Gold, 1978] zeigt er darüber hinaus, daß es im allgemeinen nicht möglich ist, zu einer beliebigen Menge positiver und negativer Beispiele einen konsistenten minimalen regulären Automaten zu berechnen, der die Beispiele akzeptiert. Diese Ergebnisse waren zunächst ernüchternd, in der Folge versuchten dann aber einige Forscher, Strategien zu entwickeln, die das effiziente Lernen von Grammatiken doch noch ermöglichen. Ein Forschungszweig beschäftigte sich dabei damit, gegenüber den regulären Sprachen noch weiter eingeschränkte Sprachen zu finden, die effizient lernbar sind. In diese Richtung gehen zum Beispiel Dana Angluins Arbeiten zu k-reversiblen Sprachen ([Angluin, 1982], siehe 3.3.1) oder zu den Gap-Patterns ([Angluin, 1980], siehe 3.3.2). Ebenfalls von Dana Angluin stammen Untersuchungen dazu, wie mit Hilfe von Nachfragen an ein Orakel die negativen Ergebnisse von Gold überwunden werden können ([Angluin, 1987]). Verschiedene Orakel werden auch bei dem Verfahren EMILE von Pieter Adriaans ([Adriaans und Knobbe, 1996]) oder von Stephen Muggleton in [Muggleton, 1996a] angewendet. Eine weiterer Trick besteht darin, die verwendeten Beispiele besonders auszuwählen, oder Annahmen über ihre Verteilung auszunutzen. Das Verfahren EMILE geht von einer Anordnung der Beispiele nach ihrer Komplexität aus. In Progol ([Muggleton, 1996a]) wird die Häufigkeitsverteilung der Beispiele zur Auswahl der Hypothese genutzt.

Bei den meisten dieser Verfahren wird die erlernte Grammatik nach eher formalen Kriterien selektiert. Sehr beliebt ist als Kriterium z.B. die Komplexität einer Grammatik. Häufig identifiziert man diese mit ihrer Kodierungslänge, also der Anzahl von Bits, die benötigt werden, um die Regeln der Grammatik zu speichern. Bei vielen Verfahren wird der hierarchische Aufbau der Sprachen nicht wirklich verwendet. Bei den Verfahren zur Induktion von Automaten z.B. ist diese Art der Struktur nur implizit enthalten. Bei den n-Gram Ansätzen (siehe z.B. [Charniak, 1993]) (siehe 3.3.3.1 wird die Sprache als rein linear aufgebaut aufgefaßt.

In den späten 60ern wurde beim Sprachlernen dann ein weiterer Aspekt von Sprache berücksichtigt, nämlich die Semantik. Es setzte sich die Erkenntnis durch, daß Grammatiken nicht nur zum Parsen von Sätzen verwendet werden können, sondern auch für eine Abbildung von Sätzen auf Bedeutungen. Die Beispiele sind hier nicht mehr einfach Sätze, sondern Paare von Sätzen und mit diesen Sätzen assoziierten Bedeutungen. Diesen Ansatz bezeichnet Langley ([Langley, 1987]) als *grammatical mapping*.

Auch im Rahmen dieser Arbeit kommt dieser Aspekt der Sprache zum Tragen. Die Form der zu lernenden Grammatik ist nicht allein von formalen Präferenzkriterien wie der Komplexität abhängig, sondern ist in gewissem Umfang von der zu lösenden Aufgabe der Umformatierung bestimmt. Wie weiter oben beschrieben, soll die gesuchte Grammatik bestimmte Konstituenten enthalten, die eng mit bestimmten Eigenschaften der zugehörigen Publikationen zusammenhängen. Hier soll nun versucht werden, diese Einheiten aus strukturellen Eigenschaften der gegebenen Beispiele zu ermitteln.

3.2 Klassifikation

Die Aufgabe, Beschreibungen für die einzelnen Felder zu lernen, kann auch als Klassifikationsaufgabe angesehen werden. Dabei soll eine Repräsentation gelernt werden, mit deren Hilfe für ein gegebenes Objekt entschieden werden kann, ob es zu einer Klasse gehört oder nicht. Wie weiter oben schon festgestellt wurde, wird die Beschreibung der Felder benötigt, da die Zerlegung der Literaturhinweise allein mit einer Beschreibung auf der Ebene der Felder und Feldtrenner nicht eindeutig ist. Mit Hilfe eines Klassifikators kann nun entschieden werden, ob die Zeichenketten, in die die Eingabe zerlegt wurde, wirklich Instanzen der ihnen zugewiesenen Klassen sein können oder nicht. Somit können dann die falschen Zerlegungen zurückgewiesen werden.

3.3 Lernverfahren

Hier soll nun eine Reihe von Lernverfahren vorgestellt werden, die für die Lösung der einzelnen Teilverfahren eingesetzt werden könnten. Dabei sind, wie in Abschnitt 3.1.2 schon ausgeführt wurde, natürlich Verfahren zum Grammatiklernen interessant. Zum anderen kommen für die Erkennung der Felder auch Klassifikationsverfahren in Frage. Da vor allem die Struktur von Titeln kurzen Texten ähnelt, können hier Verfahren aus der Textklassifikation geeignet sein. Ein Bereich der sich vor allem mit dem Erkennen von Einheiten wie Literaturhiweisen, URLs und so weiter beschähftigt ist der Bereich der Informationsextraktion. In diesem Bereich werden sowohl Klassische Begriffslernverfahren wie ILP, als auch Verfahren der Textklassifikation untersucht ([Craven et al., 1998a]).

3.3.1 Automateninduktion

Deterministische endliche Automaten (DFAs) sind ein beliebter Formalismus zur Darstellung von Sprachen. Dabei ist die Aussagekraft von DFAs identisch mit der von regulären Grammatiken ([Hopcroft und Ullman, 1979]).

Ein DFA ist ein Tupel $((Q, \Sigma, q_s, F)$ und eine Funktion σ. Dabei ist Q eine endliche Menge von Zuständen, Σ ein endliches Ausgabealphabet, q_s ein Startzustand, und $F \in Q$ eine endliche Menge von akzeptierenden Zuständen. Die Transitionsfunktion $\sigma(q, w) = q'$ gibt an, daß der Automat aus dem Zustand q bei Eingabe des Zeichens $w \in \Sigma$ in den Zustand q' übergeht.

Der Automat akzeptiert ein Wort $w = v_1 \dots v_n$, wenn es eine Folge $q_1 \dots q_{n+1}$ von Zuständen gibt, so daß q_1 der Startzustand, q_{n+1} ein akzeptierender Zustand ist und für zwei aufeinanderfolgende Zustände q_i, q_{i+1} und das Eingabesymbol w_i gilt: $\sigma(q_i, w_i) = q_{i+1}$.

DFAs lassen sich durch Graphen darstellen, wobei jeder Zustand durch einen Knoten repräsentiert wird und zwischen zwei zu den Zuständen q_i und q_j gehörenden Knoten genau dann eine mit v markierte Kante existiert, wenn $\sigma(q_i, v) = q_j$.

Ein wesentlicher Forschungszweig des Syntaxlernens ist die Induktion eines DFA aus Beispielen. Eine der grundlegenden Arbeiten stammt auch hier von Dana Angluin ([Angluin, 1982]). Durch die Einschränkung des Hypothesenraumes auf k-reversible Sprachen gelingt es ihr, einen effizienten Algorithmus zur Induktion eines DFA zu entwickeln. Dieses Verfahren wird in Abschnitt 4.2.2.2 genauer beschrieben.

Das Prinzip des Verfahrens besteht darin, zunächst einen Automaten zu erzeugen, der für jedes Beispiel einen disjunkten Pfad enthält. Dann wird dieser Automat generalisiert, indem Knoten, die als äquivalent angesehen werden, miteinander verschmolzen werden. Mittlerweile existiert eine große Familie solcher *State Merging* Verfahren.

Hermens und Schlimmer [Hermens und Schlimmer, 1993] z.B. benutzen ein modifiziertes Verfahren zur Vorhersage von Benutzereingaben für Benutzeroberflächen.

Automaten geben keinen echten Einblick in die Struktur der beschriebenen Sprache. Die von ihnen verwendete Sicht auf die Sprache ist eine eher sequenzielle. Sie dienen häufig zur Entscheidung des Wort-Problems für eine gegebene Sprache, oder zur Vorhersage von Eingabezeichen. Mit ihrer Hilfe lassen sich die Eingaben nicht in eine Struktur überführen. Für diese Arbeit kommen DFAs also im wesentlichen als Klassifikatoren in Frage.

3.3.2 Gap-Pattern Verfahren

Ein Gap-Pattern ist eine nichtleere Zeichenkette, die aus Konstanten (Terminalsymbole) und Variablensymbolen (nichtterminale Symbole) besteht. Die von einem Gap-Pattern beschriebene Sprache ist die Menge aller Zeichenketten, die aus dem Pattern durch Ersetzen aller Variablensymbole durch beliebige Strings aus Konstanten erzeugt werden können. Ein Gap-Pattern könnte z.B. folgendermaßen aussehen:

$$\text{“}v_1,\ v_2(v_3)\text{”}$$

Die durch dieses Gap-Pattern erzeugten Wörter erhält man, indem man die Variablensymbole v_1, v_2 und v_3 durch beliebige Zeichenketten ersetzt.

Einer der ersten Ansätze zum Lernen von Gap-Patterns stammt von Dana Angluin ([Angluin, 1980]), die ein effizientes Verfahren zur Berechnung eines deskriptiven Patterns für eine gegebene Menge S von Zeichenketten angibt. Für ein bezüglich einer Menge S von Zeichenketten deskriptives Pattern p gilt, daß die von p beschriebene Sprache die Menge S enthält und keine Sprache eines anderen Patterns als Teilmenge enthält, dessen Sprache seinerseits S enthält. Der Hypothesenraum wird bei Angluin auf Gap-Patterns mit einer Variable beschränkt.

Von den Arbeiten von Angluin beeinflußt beschreibt Robert Nix in [Nix, 1985] ein Verfahren, um aus Beispielen zu lernen, wie Tabellen umformatiert werden. Dabei vergleicht das Verfahren jeweils zwei Eingabebeispiele und berechnet den längsten gemeinsamen Teilstring (lcs). Für die Strings "(53-71)" und "(62-86)" wäre der lcs z.B. die Zeichenkette "(-)". Nun wird der lcs zu einem Gap-Pattern erweitert. Dazu geht er die beiden Strings und den lcs parallel Zeichen für Zeichen durch, solange alle drei Strings an der aktuellen Position dasselbe Zeichen enthalten. Sobald wenigstens einer der beiden Eingabestrings ein anderes Zeichen enthält als der lcs, wird an der aktuellen Position ein Variablensymbol eingefügt. Dann werden die Ausgangsstrings so lange durchlaufen, bis das aktuelle Zeichen aus dem lcs gefunden wurde. An dieser Stelle wird der Vorgang dann widerholt, bis das Ende des lcs erreicht wird. Wurde dann noch nicht das Ende der anderen Strings erreicht, wird noch ein Variablensymbol angehängt. Das Ergebnis für die Beispielstrings ist "(v_1-v_2)". Mit diesem Gap-Pattern werden die Eingaben geparsed und die sich ergebenden Belegungen für die Variablensymbole ermittelt. Bei unserem Beispiel sind dies (v_1 =53,v_2=71) und (v_1 =62,v_2=86). Nun wird ein Gap-Pattern konstruiert, das dieselben Variablensymbole enthält und mit diesen Belegungen die korrespondierenden Ausgabestrings erzeugt. Das so ermittelte Pattern-Paar wird dann dem Benutzer zur Kontrolle vorgelegt bevor es zur Umformatierung der nächsten Beispiele verwendet wird.

Robert Nix ermittelt also allein durch den Vergleich der Eingaben eine für die Umformatierung geeignete Struktur. Die Ausgaben werden dann in einem zweiten Schritt verwendet, um eine Regel für die Umformatierung mit Hilfe der gefundenen Struktur zu erzeugen.

Dieses Verfahren hat mehrere Nachteile. Das Verfahren funktioniert nur, wenn die Beispiele alle die gleiche Struktur haben, da für alle Beispiele nur ein einziges Gap-Pattern gesucht wird. Zum anderen kann es Mehrdeutigkeiten bei der Belegung der Gap-Patterns geben. Für das obige Beispiel-Pattern und den String

"8-5-6" gibt es zwei Belegungen (v_1=8,v_2=5-6) und (v_1=8-5,v_2=6), von denen das Verfahren einfach die erste nutzt, die es findet. Das Verfahren funktioniert also nur in Domänen mit eindeutigen Trennzeichen.

3.3.3 ILP

Der Bereich der Induktiven Logischen Programmierung (ILP) beschäftigt sich mit Lernverfahren, die Prädikatenlogik als Repräsentationssprache verwenden. Eine der Standard-Lernaufgaben in diesem Bereich ist das Begriffslernen. Dabei sind jeweils eine Menge positiver Beispiele E^+, eine Menge negativer Beispiele E^- in der Beispielsprache L_E und Hintergrundwissen B in einer Sprache L_B gegeben. Gesucht ist eine Hypothese in der Hypothesensprache L_H, die folgende Bedingungen erfüllt ([Morik, 1995]):

1. *Konsistenz:* Aus B, H und E folgt kein Widerspruch.
2. *Vollständigkeit:* Aus B und H lassen sich alle positiven Beispiele ableiten.
3. *Korrektheit:* Aus B und H läßt sich kein negatives Beispiel ableiten.

Bei ILP ist die Hypothesensprache in der Regel eine eingeschränkte Prädikatenlogik. Das System Foil ([Quinlan, 1990]) lernt z.B. funktionenfreie Hornklauseln aus variablen- und funktionenfreien Fakten. Bei Progol ([Muggleton, 1995a]) werden L_H, L_B und L_E durch Hornklauseln repräsentiert. Progol verwendet darüber hinausgehend Stochastische Logische Programme ([Muggleton, 1996b]). Es bietet damit auch ein Verfahren an, die Zielhypothese aufgrund der Verteilung der Beispiele auszuwählen. In diesem Fall kommt Progol auch ohne negative Beispiele aus.

Da sich die Prädikatenlogik sehr gut für die Darstellung von Grammatiken eignet, liegt es nahe, ILP-Verfahren auch zum Grammatiklernen zu verwenden. Der zu lernende Begriff ist dann ein Prädikat, das dem Startsymbol einer Grammatik entspricht. Im Hintergrundwissen werden die möglichen Konstituenten der Grammatik modelliert. Für die Beispiele werden Beispielsätze in L_E modelliert.

Ein Verfahren, auf diese Weise eine Grammatik für ein Lesebuch für die Grundschule zu lernen, beschreibt Stephen Muggleton in ([Muggleton, 1996a]). Er geht dabei von einer handgemachten einfachen Basisgrammatik aus, die er im Hintergrundwissen vorgibt. Für den Fall, daß die im Hintergrundwissen vorgegebenen Konstituenten nicht ausreichen, gibt Muggleton ein Verfahren an, bedarfsgestützt neue Prädikate zu generieren ([Muggleton, 1995b]). Da die Lesebücher in der Grundschule mit einfachen Texten beginnen und dann immer komplexer werden, ist diese Strategie sehr erfolgreich. Es wird hier also auch eine bewußte

Auswahl der Beispiele verwendet. Das Lernergebnis, das heißt die gefundenen Prädikate und Regeln, wird einem Orakel (dem Benutzer) vorgelegt. Weist das Orakel die Regeln zurück, so werden neue gesucht. Somit wird hier das Lernen also vom Benutzer gesteuert.

Prädikatenlogische Verfahren können auch verwendet werden,um einfach ein Prädikat zu lernen, mit dessen Hilfe entschieden werden kann, ob eine Zeichenkette eine Instanz des Feldes ist oder nicht. Dayne Freitag ([Freitag, 1998]) und Mark Craven et al. ([Craven et al., 1998b]) haben mit Hilfe von Foil Regeln gelernt, um bestimmte Informationen in Webseiten zu finden, oder um Webseiten zu klassifizieren.

3.3.3.1 Statistisches Sprachlernen

Ein großer Bereich beim Lernen von Sprache ist der Bereich der statistischen Lernverfahren ([Charniak, 1993]). Ein großer Vorteil der statistischen Verfahren ist, daß die hier entwickelten Repräsentationen keine scharfe Entscheidung, ob ein Wort zu einer Sprache gehört oder nicht, erfordern, sondern angeben, mit welcher Wahrscheinlichkeit ein Wort zu einer Sprache gehört.

Ein klassischer Ansatz aus diesem Bereich sind die n-Gram Verfahren. Dabei wird das Problem des Sprachaufbaus als lineares Vorhersageproblem betrachtet. Ein n-Gram Model beschreibt die Wahrscheinlichkeit des nächsten Zeichens in der Eingabe in Abhängigkeit von den letzten n-1 Zeichen der Eingabe. Für jedes n-tupel $(x_1, \ldots, x_n)$ wird die Wahrscheinlichkeit $P(x_n|x_1 \ldots x_{n-1})$ angegeben, mit der das Zeichen x_n als nächste Eingabe kommt, wenn $x_1, \ldots, x_{n-1}$ die letzten $n-1$ Zeichen der Eingabe waren. Die Wahrscheinlichkeit eines Wortes $M = x_1 \ldots x_l$ ist dann das Produkt der n-Gram Wahrscheinlichkeiten der x_i. Als sehr erfolgreich haben sich dabei Trigramme, also n-Gramme mit n=3 erwiesen. Ein Spezialfall der n-Gramme sind die Unigramme mit n=1. Hier wird die Wahrscheinlichkeit eines Zeichens angegeben, ohne einen Kontext zu berücksichtigen. Um die Vorhersagekraft zu erhöhen, verwendet man häufig nicht n-Gramme mit einer festen Länge, sondern man kombiniert die n-Gram-Wahrscheinlichkeiten der einzelnen Zeichen für mehrere n. Beliebt ist hier die Kombination von n=1,2,3. Ian Witten beschreibt in [Witten et al., 1999] ein Verfahren zur Erkennung von URLs und ähnlichen Angaben in Texten, dem ein Textkompressionsverfahren zugrunde liegt, das n-Gram-Modelle verwendet. Diese Verfahren sind vor allem sehr robust gegen leichte Abweichungen der Eingaben von den Beispielen. In der Praxis haben sich vor allem n-Gramme mit $n = 3$ bewährt.

Ein weiterer großer Bereich der statistischen Lernverfahren sind die auf Hidden Markov Models (HMM) basierenden Verfahren. Ein HMM wird durch ein Tupel (Q, Σ, q_i, q_F) und zwei Funktionen p_t und p_e beschrieben. Dabei ist Q eine

Menge von Zuständen, Σ ein Ausgabealphabet, p_i ein Startzustand und q_F ein Endzustand. Die Funktion $p_t(q \rightarrow q')$ gibt die Wahrscheinlichkeit an, mit der das HMM von Zustand q in Zustand q' übergeht. $p_e(q \uparrow \sigma)$ gibt an, mit welcher Wahrscheinlichkeit ein Zustand q das Zeichen $\sigma \in \Sigma$ ausgibt.

Die Forschung zum Lernen von HMMs bezieht sich dabei auf zwei Faktoren, die die Ausgabewahrscheinlichkeiten von HMMs beeinflussen. Vor allem bei älteren Ansätzen konzentrierte man sich auf die Abschätzung der Übergangs- und Ausgabewahrscheinlichkeiten, während die Struktur von HMMs von Hand modelliert wurde. In neueren Ansätzen wird auch das Lernen der HMM-Struktur aus Beispielen betrachtet. Dabei werden auch einige Ansätze zur Induktion von DFAs verwendet.

HMMs und Trigramme dienen dazu, festzustellen, mit welcher Wahrscheinlichkeit ein gegebenes Wort zu einer Sprache gehört.

3.3.3.2 Sequenzen

Aufbauend auf den Arbeiten von Agrawal und Srikant [Agrawal und Srikant, 1995] hat Helena Ahonen-Myka ([Ahonen-Myka, 1999]) ein Verfahren entwickelt, um alle maximal häufigen Sequenzen von Wörtern in Texten zu finden. Diese Sequenzen können dann zur Klassifikation von Texten verwendet werden, indem man feststellt, welche der in einer gesuchten Klasse von Texten häufige Sequenz in einem gegebenen Text vorkommt.

Kapitel 4

Lernen von Übersetzungsregeln für Literaturhinweise

In Kapitel 2 wurde eine Strategie für die Umformatierung von Literaturhinweisen entwickelt, die dann in Kapitel 3 in das Syntaxlernen eingeordnet wurde. Ebenfalls in Kapitel 3 wurden diverse Lernverfahren näher beschrieben. In diesem Kapitel soll nun ein anwendbares Verfahren vorgeschlagen werden, mit dessen Hilfe ein Computer aus Beispielen lernen kann, Literaturhinweise umzuformatieren.

Grundsätzlich läßt sich das Verfahren in zwei Teilverfahren aufspalten. Der erste Teil, die Lernkomponente, wird angewendet, um eine Repräsentation der Umformatierung von Literaturhinweisen aus Beispielen zu erlernen. Der zweite Teil, die Anwendungskomponente, wendet dann die gelernte Repräsentation an, um einen eingegebenen Literaturhinweis in das Ausgabeformat zu überführen. Dabei interagieren die beiden Komponenten miteinander. Insbesondere verwendet die Lernkomponente die Anwendungskomponente, um die aus alten Beispielen gelernte Repräsentation zu nutzen.

4.1 Allgemeiner Ablauf

In Kapitel 1 wurden einige Rahmenbedingungen dargestellt, die das zu entwickelnde Verfahren erfüllen soll. Diese will ich hier noch einmal kurz zusammenfassen.

Das Verfahren soll so entworfen werden, daß kein spezielles Hintergrundwissen erforderlich ist. Dies schließt natürlich nicht aus, daß die Vorgabe von Informationen das Lernergebnis positiv beeinflussen kann. Das Verfahren soll interaktiv mit dem Benutzer zusammenarbeiten und diesen so schnell wie möglich bei seiner

Tätigkeit unterstützen. Dies bedeutet, daß der Benutzer nicht erst große Mengen von Beispielen geben muß, um von dem System erste Vorhersagen zu erhalten. Vielmehr sollte er schon nach wenigen Beispielen sichtlich unterstützt werden. Aus diesen Randbedingungen ergibt sich nun der im folgenden dargestellte Ablauf.

Zu Anfang stellt der Benutzer dem System den ersten Literaturhinweis im Eingabe- und im Ausgabeformat zur Verfügung. Es wird schließlich mindestens ein Beispiel benötigt, um überhaupt irgendwelche sinnvollen Vorhersagen machen zu können.

Bei den übrigen Literaturhinweisen wird dann immer wie folgt vorgegangen. Der Benutzer gibt zunächst einen Literaturhinweis im Eingabeformat ein. Darauf macht das System eine Vorhersage für die Ausgabe. Der Benutzer kontrolliert die Ausgabe und korrigiert sie gegebenenfalls. Die Eingabe und die korrigierte Ausgabe werden dann der Lernkomponente des Verfahrens als Beispiel zur Verfügung gestellt. Dieser Ablauf wird in Abb. 4.1 als Flußdiagramm dargestellt.

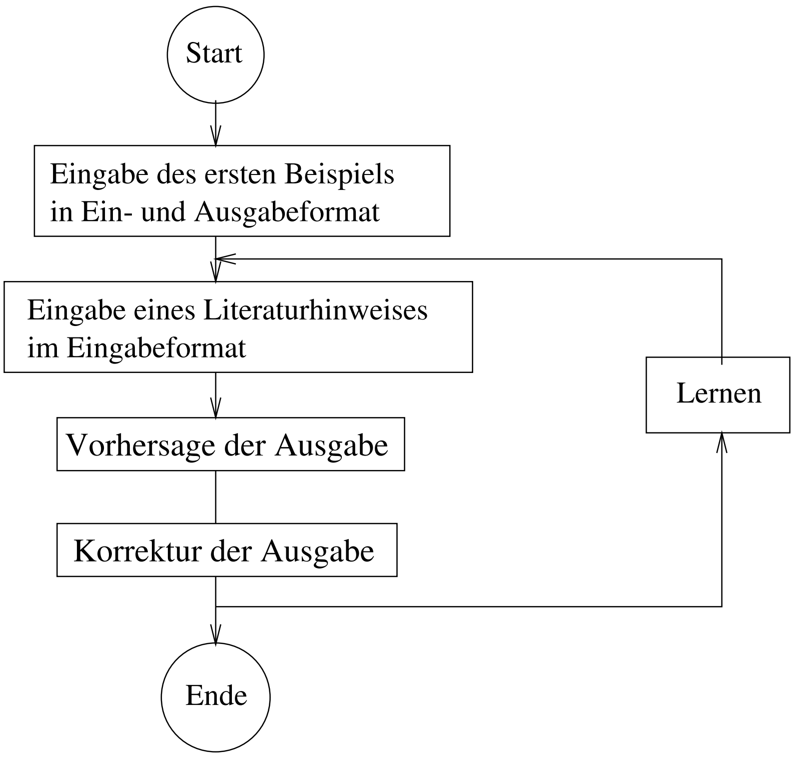

Abbildung 4.1: Ablauf

4.1.1 Ablauf bei der Umformatierung

Hier wird nun das in Abschnitt 2.2 angegebene Vorgehen genauer beschrieben. Vor allem wird hier die Form der Zwischenergebnisse festgelegt.

Die Eingabe des Benutzers liegt zunächst als Zeichenkette vor. Diese wird als erstes in Felder und Trennzeichen zerlegt. Diese Zerlegung wird dann als geordnete Liste $(e\_1, ..., e\_n)$ dargestellt. Die Elemente e_i sind die Felder und Feldtrenner. Dabei ist jedes Element ein Paar aus einem Bezeichner, der das Element als Feld einer bestimmten Klasse oder als Feldtrenner kennzeichnet, und dem Teilstring der Eingabe, der diesem Element entspricht.

Die folgende Eingabe

> D. Angluin (1988). Queries and Concept Learning. Machine Learning, 2:319–342.

würde beispielsweise in folgende Liste überführt:

> (1, "D."), (T," "),(2, "Angluin"), (T, " ("), (3, "1988"), (T, "). "), (4, "Queries and Concept Learning"), (T, ". "), (5, "Machine Learning"), (T, ", "), (6, "2"), (T, ":"), (7, "319–342"), (T, ".")}

Hier wurden die Felder zur Bezeichnung durchnummeriert, T bezeichnet einen Feldtrenner. Diese Liste will ich weiterhin als Strukturliste bezeichnen. Aus dieser Liste werden dann alle Feldtrenner eliminiert und man erhält eine geordnete Liste der gefundenen Felder, die ich weiterhin als Feldliste bezeichnen will.

> {(1, "D."),(2, "Angluin"), (3, "1988"), (4, "Queries and Concept Learning"), (5, "Machine Learning"), (6, "2"), (7, "319–342")}

Aus dieser Liste wird die Ausgabe generiert. Diese liegt zunächst wieder als Strukturliste vor, die dann in eine Zeichenkette umgewandelt wird, indem alle Zeichenketten der Listenelemente aneinandergehängt werden.

In Abbildung 4.2 sind die die einzelnen Schritte in noch einmal zusammenfassend dargestellt. Auf diese Darstellung wird dann im nächsten Abschnitt Bezug genommen. Dieses Schema entspricht dem Linken Teil des Diagrammes für den Gesamtablauf (Abb. 5.1).

1. Eingabe einer Zeichenkette
2. Zerlegen der Eingabe in Felder und Trennzeichen => Geordnete Liste von Feldern und Trennzeichen (Strukturliste).
3. Eliminieren der Trennzeichen => Geordnete Liste der Felder (Feldliste).
4. Erzeugen der Struktur der Ausgabe => Geordnete Liste der Felder und Trennzeichen (Strukturliste).
5. Generieren Ausgabe aus der Strukturliste für die Ausgabe. => Zeichenkette

Abbildung 4.2: Ablauf der Umformatierung

4.2 Die einzelnen Aufgaben

Die Gesamtaufgabe läßt sich wie schon gesagt in Einzelaufgaben aufteilen. Die erste Aufgabe besteht darin, eine Strukturierung der Beispiele zu finden, die für eine Umformatierung nutzbar ist. Diese Strukturierung soll in der Zerlegung der Beispiele in Felder und Feldtrenner bestehen. Als nächstes muß eine Repräsentation dieser Zerlegung aufgebaut werden, die es ermöglicht, eine neue Eingabe entsprechend der gefundenen Struktur zu zerlegen. Dabei wird inkrementell vorgegangen, d.h. nach der Eingabe eines Beispiels wird aus diesem Beispiel und dem alten Lernergebnis ein neues Lernergebnis ermittelt.

4.2.1 Finden einer für die Umformatierung geeigneten Struktur

Der erste Arbeitsschritt besteht also darin, eine Zeichenkette, die als Beispiel vorgegeben wird, zu strukturieren. Dazu wird eine Aufteilung eines gegebenen Beispiels gesucht, die eine Umformatierung dieses Beispiels ermöglicht. Diese Aufteilung wird zunächst allein aus einem Beispielpaar ohne Berücksichtigung der restlichen Beispiele ermittelt. Dargestellt wird diese Zerlegung als Strukturliste (s.o.), wobei die Felder hier nur mit Platzhaltern für die später noch zuzuordnenden Feldklassen dienen. Dann wird in einem zweiten Schritt versucht, die einzelnen Komponenten der Zerlegung mit den in den vorherigen Beispielen gefundenen Komponenten zu identifizieren und damit die Felder einzelnen Klassen zuzuordnen.

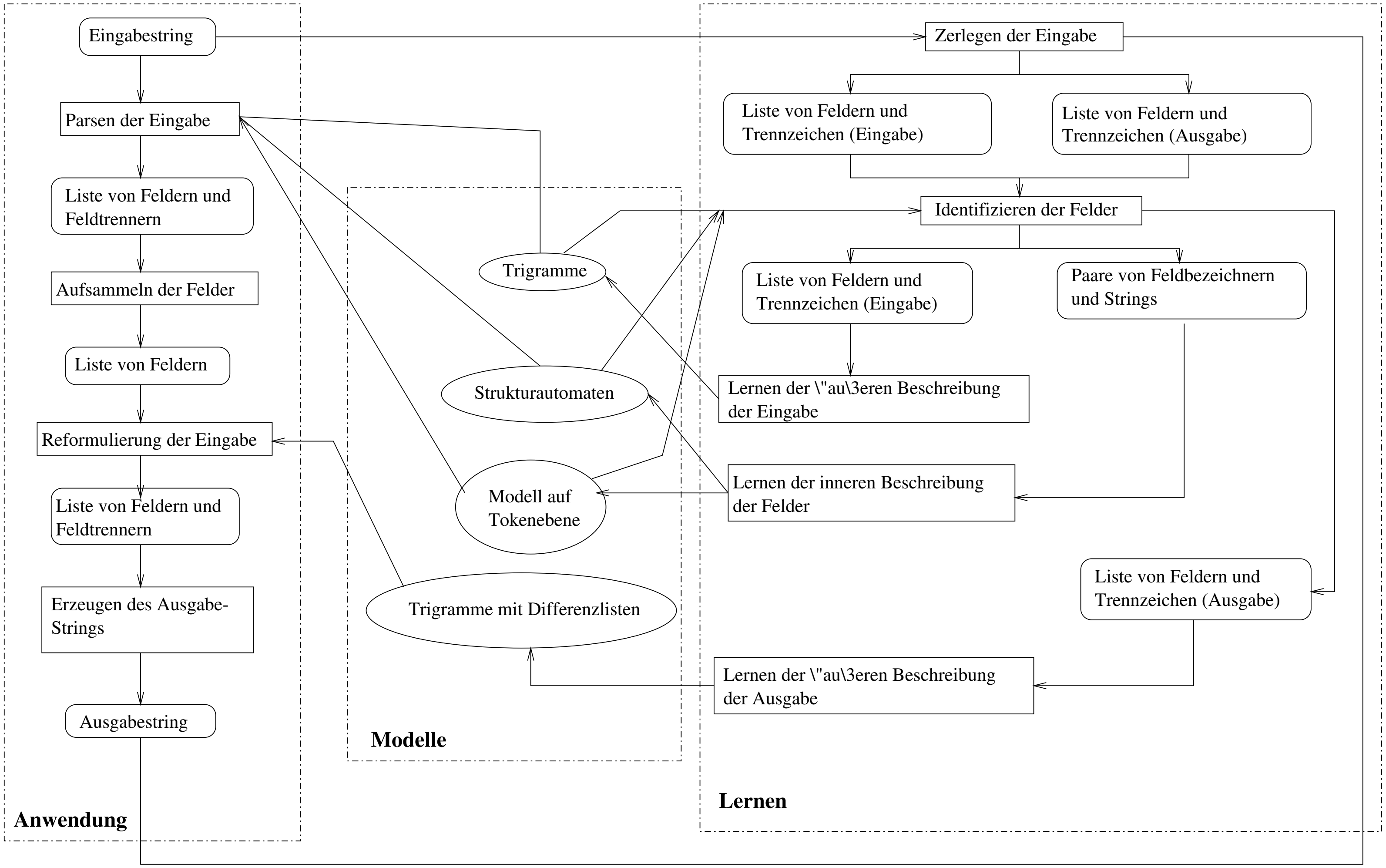

Abbildung 4.3: Gesamtüberblick

4.2.1.1 Finden von sinnvollen Zerlegungen

Für die erste Lernaufgabe hat man als Beispielsprache also die Menge aller Paare von Zeichenketten. Die Hypothesensprache besteht aus allen Mengen möglicher Zerlegungen der Eingabepaare. Das Lernergebnis soll ein Element der Hypothesensprache sein, daß eine Umformatierung von Literaturhinweisen nach dem oben entwickelten Vorgehen ermöglicht.

Zunächst wird die Eingabe in eine Kette von Token überführt, um die Zerlegung zu erleichtern. Im weiteren werden dann diese Token als Zeichen der Eingabe aufgefaßt. Dazu werden alle ununterbrochenen Ketten von Buchstaben jeweils zu einem Token zusammengefaßt. Alle anderen Zeichen stellen jeweils selbst ein Token dar. Der Grund für diese Maßnahme wird erst bei der Beschreibung der Analyse der Eingabe deutlich, da dort die Länge der Eingabe wie sich zeigen wird eine sehr große Rolle spielt Damit ist die Beispielsprache nicht mehr die Menge aller Zeichenkettenpaare, sondern die Menge aller Tokenkettenpaare.

Die Schwierigkeit liegt jetzt darin, allein aus Tokenketten eine Struktur dieser Ketten zu finden, die mit der Bedeutung dieser Zeichenketten zusammenhängt. In Abschnitt 3.3 wurden schon einige Verfahren vorgestellt, die sich mit diesem Problem beschäftigen. Obwohl die Aufgabe, die Robert Nix in [Nix, 1985] gelöst hat, der Aufgabe hier sehr ähnlich ist, ist das Verfahren, wie weiter oben schon erkärt wurde, unter anderem wegen der inhomogenen Struktur von Literaturhinweisen für diese nicht geeignet. Nix' Verfahren lernt die Struktur der Eingabe nur aus den Eingabebeispielen. Die Ausgabebeispiele verwendet er dann nur, um die Regeln zur Umformatierung zu lernen. Die Idee, die ich hier nutzen möchte, ist, die Ausgabebeispiele auch schon bei der Strukturierung der Eingabe zu nutzen.

Aus dem Vergleich zwischen dem Ausgangsbeispiel und dem Zielbeispiel lassen sich wichtige Informationen über die Einheiten der Beispiele gewinnen, vor allem natürlich im Hinblick auf die Umformatierung, die ja gerade auf den Unterschieden zwischen den beiden Sprachen basiert. Das Prinzip dabei ist, gemeinsame Teilketten in den Eingangs- und Ausgangsbeispielen zu finden.

Gegeben seien zwei Wörter $B_a = w_{a_1}..w_{a_n}$ und $B_z = w_{b_1} \dots w_{b_m}$, wobei w_{a_j} und w_{b_k} einzelne Token sind. Gesucht ist eine Sequenz $Z = f_1 t_1 f_2 t_2 ... f_l t_l$, wobei die f_i und die t_i wiederum Sequenzen von Token sind. Die f_k sind dann die Felder der Zerlegung, die f_k sind die Feldtrenner. Diese Sequenz soll folgende Bedingungen erfüllen:

1. Die Tokenketten B_z und Z sollen identisch sein. $Z = B_z$

2. Die einzelnen Teilsequenzen f_i sollen in B_a enthalten sein. (Da $Z = B_z$ sind die f_i natürlich auch in B_z enthalten.) $f_i = w_{a_j} w_{a_{j+1}} w_{a_{j+2}} ... w_{a_{j+k}}$

3. Die einzelnen f_i sind maximal, d.h. : Sei f_i=$w_{a_j} \dots w_{a_{j+k}}$ und $Z = \dots w_x f_i w_y \dots$ und w_x und w_y sind Token, so darf f_i nicht um w_x oder w_y verlängert werden können, ohne daß Bedingung zwei verletzt wird.

4. Kein w_{a_i} darf mehr als einmal in Z vorkommen. D.h. es muß eine Darstellung Z=$w_{a_i} \dots w_{a_j}$ geben wobei für je zwei w_{a_k}, w_{a_l} gilt $k \neq l$.

5. Kein t_i darf eine Sequenz enthalten, die die Bedingungen für die f_i erfüllt.

6. Die Länge l von Z, d.h. die Zahl der enthaltenen f_i, k_i ist minimal.

B_a läßt sich nun entsprechend zerlegen, indem man in B_a alle Teilketten zusammenfaßt, die jeweils einem f_i entsprechen. Die übrigen Teilketten sind dann die t.

Als Beispiel gebe ich hier einen Ausschnitt des bereits zuvor benutzten Beispielhinweises. Bei der Darstellung der Zerlegung werden die f_i durch eckige und die t_i durch geschweifte Klammern gekennzeichnet.

> B_a=Angluin, D. (1982). Inference of reversible languages.
> B_z=D. Angluin. Inference of reversible languages. 1982.
> Z=<D. ><Angluin><. Inference of reversible languages.>{ }<1982>{.}
> Z_a=<Angluin>{, }<D. >{(}<1982>{)}<. Inference of reversible languages.>

Diese Zerlegung sichert nun bestimmte Eigenschaften der f_i bezüglich der Bestandteile eines Literaturhinweises zu. Genauer gesagt besteht ein Zusammenhang zwischen den f-Segmenten und den in Kapitel 2 eingeführten Feldern und zwischen den t-Segmenten und den dort eingeführten Feldtrennern.

Jedes atomare Feld eines Literaturhinweises ist vollständig in einem f_i enthalten. Sei der Ausschnitt $W = w_{ai} \dots w_{aj}$ aus B_a ein atomares Feld. Dann ist W nach Definition auch eine Teilkette von B_z. Wenn W nicht in einem f_i vollständig enthalten ist, dann gibt es zwei Möglichkeiten: Es gibt ein f_i, das sich mit W überschneidet, oder W ist disjunkt zu allen f_i. Der erste Fall wäre eine Verletzung von Eigenschaft 3 der Zerlegung. Im zweiten Falle wäre W in einem t_i enthalten, was Eigenschaft 5 der Zerlegung widerspricht. Daraus und aus der Maximalität der f_i, ergibt sich, daß sie genau die maximalen Felder eines Literaturhinweises bezüglich seiner Umformatierung darstellen. Wie bei der Beschreibung des Verfahrensansatzes schon erläutert wurde, wäre es zwar wünschenswert, die atomaren Felder eines Literaturhinweises zu finden. Jedoch sind diese nicht ohne zusätzliche Informationen zu finden.

Mit Hilfe einiger Heuristiken, die sich aus grundlegenden Eigenschaften von Literaturhinweisen ergeben, können die gefundenen Zerlegungen jedoch noch etwas verfeinert werden. Bei genauerer Betrachtung fällt zum Beispiel auf, daß zwei atomare Felder eines Literaturhinweises immer mindestens durch ein Zeichen, das nicht ein Buchstabe oder eine Zahl ist voneinander getrennt werden. In allen von mir bisher gefundenen Literaturlisten beginnen und enden alle atomaren Felder mit einem alphanumerischen Zeichen. Die einzige Ausnahme stellen dabei Punkte dar, die am Ende von abgekürzten Feldern stehen können. In diesem Falle wird der Punkt jedoch immer von einem Leerzeichen gefolgt.

Aus diesem Grunde werden nicht alphanumerische Zeichen am Anfang oder Ende der Felder von diesen abgetrennt und den Trennzeichen zugeschlagen. Einzig ein einem Buchstaben folgender Punkt darf am Ende eines Feldes stehenbleiben. Punkte, die am Ende des gesamten Literaturhinweises stehen, werden immer abgetrennt. Nach diesem Schritt ist sichergestellt, daß zwei gefundene Felder immer durch einen Feldtrenner voneinander getrennt sind.

Damit sehen die Zerlegungen für das obige Beispiel dann so aus:

> Z=<D.> { } <Angluin>{. }<Inference of reversible languages>{. }<1982>{.}
> Z_a=<Angluin>{, }<D.>{ }>{(}<1982>{). }<Inference of reversible languages>{.}

4.2.1.2 Ein Algorithmus für die Zerlegung

Als nächstes werde ich einen Algorithmus angeben, der eine solche Zerlegung findet.

Sei die Eingabe die Tokenfolge $(e_1, \ldots, e_m)$ und die Ausgabe die Tokenfolge $(a_1, ..., a_n)$. Zunächst werden die Elemente der Eingabe von 1 bis m durchnumeriert. Dann baut man einen Graphen für die Ausgabe folgendermaßen auf. Der Graph hat $n + 1$ Zustandsknoten $k_1, ..., k_{n+1}$. k_1 sei der Startzustand und k_{n+1} der Endzustand. Im folgenden repräsentieren alle Kanten von k_i nach k_{i+1} die Ausgabe an der Position i.

Nun wird jeder Knoten k_i mit dem Knoten k_{i+1} durch eine Kante verbunden, die mit dem Token a_i markiert ist. Der so entstandene Graph hat genau einen Pfad von k_1 nach k_{i+1}, der mit der Ausgabe markiert ist. Für jedes e_i sucht man nun alle a_j mit $e_i = a_j$, das heißt alle Vorkommen des Eingabetokens in der Ausgabe, und verbindet jeweils k_j und k_{j+1} mit einer Kante, die mit i eschriftet ist, also der Position des korrespondierenden Tokens in der Eingabe markiert ist.

Jetzt enthält der Graph für jede Sequenz $e_i, \ldots, e_j$, die sowohl in der Eingabe, als auch in der Ausgabe enthalten ist, einen Pfad, der diese Sequenz in der

Ausgabe darstellt und nur durchgehend numerierte Kanten enthält.

Nun durchläuft man die Zustände des Graphen von k_1 bis k_n und verbindet alle Paare von Zuständen, die durch einen durchgehend numerierten Pfad miteinander verbunden sind, jeweils mit einer Kante. Diese Kante markiert man mit einer Variablen, wobei für jede Kante eine eigene Variable verwendet wird. Damit ist jede gemeinsame Teilsequenz von Ein- und Ausgabe durch eine mit einer Variablen markierten Kante repräsentiert. Also gibt es auch für jedes Feld eine solche Kante. Also ist die gesuchte Zerlegung als akzeptierender Pfad in dem Graphen enthalten. Dieser Pfad enthält für jedes Feld der Zerlegung eine mit einer Variablen markierte Kante und für jeden Feldtrenner eine Folge von mit Token markierten Kanten. Der gesuchte Pfad entspricht genau dem kürzesten akzeptierenden Pfad, wenn man vorher alle mit Zahlen markierten Kanten entfernt.

Diesen kürzesten Pfad kann man z.B. mit dem Algorithmus von Dijkstra [Dijkstra, 1959] effizient berechnen. Der oben beschriebene Graph enthält für eine Eingabe der Länge m aufgrund seiner Konstruktion $m+1$ Knoten. Die Laufzeit des Algorithmus bei einer Implementierung mit Adjazenzmatrizen liegt also in $O(m^2)$.

4.2.1.3 Probleme bei der Zerlegung

Zu guter Letzt müssen hier noch einige Probleme behandelt werden, die sich aus der rein strukturellen Sicht des Verfahrens ergeben. Eine Möglichkeit, diesen Problemen zu entgehen, besteht in der Verwendung eines Orakels. Grundsätzlich sind alle gefundenen Zerlegungen geeignet, die Umformatierung durchzuführen. Bestimmte Zerlegungen können jedoch die Leistung des Verfahrens beeinträchtigen. Ein Ausweg ist hier, einem Orakel, in diesem Falle dem Benutzer, die gefundenen Zerlegungen zur Korrektur vorzulegen. Der Benutzer kann dann gegebenenfalls durch Einfügen zusätzlicher Trennzeichen die vorgeschlagene Zerlegung verfeinern.

Eine besonders ungünstige Konstellation, die zu schlechten Zerlegungen führen kann, ist aus folgendem Ausschnitt aus einem Literaturhinweis ersichtlich.

... H. Ahonen, H. Mannila ...
... Ahonen, H., Mannila, H. ...

In diesem Falle würde die gefundene Zerlegung wie folgt aussehen:

... <H.> <Ahonen, H.> <Mannila> ...
... <Ahonen, H.>, <Mannila>, <H. > ...

Hier wird also der Vorname des zweiten Autors mit dem Nachnamen des ersten Autors zusammengefaßt. Diese Zerlegung hat alle aufgeführten Eigenschaften, und ist damit auch für die Umformatierung dieses Literaturhinweises geeignet. Da diese Zerlegung aber nur für den Sonderfall von zwei Autoren mit gleichen Initialen anwendbar ist, wäre folgende Zerlegung wünschenswert:

... <H.> <Ahonen>, <H.> <Mannila> ...
... <Ahonen>, <H.>, <Mannila>, <H. > ...

Der Vorteil dieser Zerlegung ist jedoch nur aus semantischer Sicht erkennbar, da man dazu die Beziehungen zwischen den Vor- und Nachnamen der Autoren kennen muß.

Es läßt sich jedoch relativ leicht erkennen, ob ein solcher Fall vorliegen könnte. Er kann nämlich nur eintreten, wenn im Ein- oder Ausgabebeispiel ein Feld von zwei Feldern mit identischem Inhalt umgeben ist. Ob dies eingetreten ist, erkennt man, indem man in beiden Zerlegungen feststellt, ob ein Feld Präfix des vorhergehnden oder Suffix des folgenden Feldes ist. In diesem Falle kann dann dem Benutzer vorgeschlagen werden, das Feld an der entsprechenden Stelle aufzuteilen. Glücklicherweise kommt dieser Fall jedoch nur sehr selten vor.

Das zweite Problem besteht wie schon erwähnt darin, daß nicht nur atomare Felder gefunden werden, sondern, wenn zwei atomare Felder in beiden Formaten hintereinander auftreten und durch dieselben Zeichen getrennt werden, das Verfahren diese Felder zusammenfaßt. Dies hat verschiedene Nachteile. Zum einen sind diese Felder komplexer als die atomaren Felder. Dadurch wird es schwieriger, eine gute Charakterisierung dieser Felder zu lernen. Zum anderen hat dies zur Folge, daß sich unter Umständen die Anzahl der zu lernenden Felder drastisch erhöht. Werden z.B. in beiden Formaten Nummer und Seitenzahlen eines Zeitschriftenbeitrags wie folgt formatiert, so werden sie als ein Feld betrachtet:

“2, 51-68”

Angenommen, die Bandzahl würde unterschiedlich angegeben, kommt aber zwischen Nummer und Seite:

“2, 5:51-68”
“2, (5)51-68”

und nach einer Reihe von Beispielen ohne Bandangabe kommt als nächstes Beispiel ein Zeitschriftenbeitrag mit Bandangabe. Dann würden zusätzlich jeweils eigene Felder für Nummer, Bandzahl und Seitenzahl als Felder eingeführt.

Auch hier wäre eine Möglichkeit, die Zerlegung dem Benutzer vorzulegen und von ihm gegebenenfalls diese Zerlegung verfeinern zu lassen.

4.2.1.4 Identifizieren von Konstituenten

Nun kommen wir zur nächsten Aufgabe, der Klassifizierung der Felder. Das aktuelle Beispiel liegt als Liste von Feldern und Feldtrennern vor. Wären Literaturhinweise homogener strukturiert, so würde sich auch hier das Verfahren von Nix (siehe Abschn. 3.3.2) anbieten. Man berechnet ein Gap-Pattern aus dem aktuellen und einem alten Beispiel, parsed beide Beispiele damit und identifiziert dann diejenigen Felder miteinander, die auf dieselben Variablen abgebildet werden. Dies setzt jedoch voraus, daß die zugrundeliegenden Beispiele diegleiche Struktur haben. Dazu müßte man z.B. wissen, daß die verwendeten Beispiele ähnliche Publikationen beschreiben.

Im Rahmen dieser Arbeit soll nun das Lernergebnis der vorhergehenden Beispiele angewendet werden, um diese Aufgabe zu lösen. Dazu wird das aktuelle Beispiel wie weiter unten in Abschn. 4.2.2.3 beschrieben analysiert. Dabei werden jedoch die möglichen Ergebnisse auf solche beschränkt, die die Eingabe genau in die Komponenten der gefundenen Zerlegung aufteilt. Das Ergebnis dieser Analyse ordnet dann jedes Element der Zerlegung einer der bisher gefundenen Feldklassen zu.

Das Orakel
Diese Zuordnung muß nun von einem Orakel (dem Benutzer) kontrolliert werden. Falsche Zuordnungen sind hier unvermeidbar. Schließlich ist diese Zuordnung erst dann einigermaßen sicher, wenn schon eine gewisse Anzahl von Beispielen gesehen wurde. Außerdem wird hier jeder Komponente der Zerlegung eine der bekannten Klassen zugeordnet, was auf jeden Fall falsch ist, wenn die betreffende Komponente einer neuen Klasse angehört. Dem Benutzer wird also die Strukturliste für ein Beispiel vorgelegt. Die Felder sind dabei mit Bezeichnern versehen, die es dem Benutzer ermöglichen, festzustellen mit welchen Bestandteilen alter Beispiele die Felder identifiziert wurden. Der Benutzer kann dann die vorgelegte Strukturliste korrigieren, indem er den Bezeichner eines falsch identifizierten Feldes gegen einen anderen austauscht, oder das Feld als neu kennzeichnet. Einem neuen Feld wird dann ein noch nicht vergebener Bezeichner zugeordnet.

Nach diesem Schritt liegt das Beispiel also als Liste von klassifizierten Feldern und Feldtrennern vor.

4.2.2 Lernen eines Modells für die Eingabe

Durch die vorhergehenden Schritte wurden die als Beispiele vorgegebenen Zeichenketten strukturiert. Sie liegen nun als Liste von klassifizierten Feldern und Feldtrennern, die ich weiterhin als Strukturlisten bezeichnen möchte, vor. Die nächste Aufgabe besteht darin, eine Repräsentation aufzubauen, die es ermöglicht, eine vorgegebene Zeichenkette korrekt in eine solche Liste von Feldern und Feldtrennern zu transformieren.

Gesucht wird eine Repräsentation, die nach wenigen Beispielen schon gute Vorhersagen machen kann und robust gegen kleine Veränderungen der Eingaben ist. Eine Literaturliste enthält nur wenige Literaturhinweise mit absolut identischer Struktur. Zum einen gibt es, wie in Kapitel 2 schon bemerkt wurde, mehrere Klassen von Literaturhinweisen, zum anderen ist auch die Variationsbreite innerhalb dieser Klassen sehr hoch. Mal hat eine Publikation einen Autor, mal sind es fünf. Zu einem Konferenzband ist der Herausgeber angegeben, zu dem nächsten stattdessen der Erscheinungsort und so weiter. Auf der Tokenebene sind die möglichen Variationen noch grösser. Es ist unmöglich, eine vollständige, hundertprozentig korrekte Grammatik zu lernen.

Zunächst wird diese Repräsentation in zwei Ebenen aufgeteilt. Die äßere Grammatik beschreibt die Struktur der Literaturhinweise auf der Ebene der Felder, die hier als feste Einheiten (Zeichen) angesehen werden. Die innere Grammatik beschreibt die Struktur der Felder, ohne den Kontext in dem die Felder stehen, zu berücksichtigen.

Aufgrund der Unsicherheit der zu treffenden Entscheidungen wären hier Probabilistische Repräsentationen gut geeignet, da bei diesen Verfahren diese Unsicherheit im gelernten Modell ausgedrückt wird. Auf der anderen Seite sollen aber gerade nach wenigen Beispielen schon Vorhersagen gemacht werden. Probabilistische Verfahren benötigen jedoch recht viele Beispiele, um gute Abschätzungen für die jeweils verwendeten Wahrscheinlichkeiten zu erhalten. Daher werden hier an probabilistische Modelle angelehnte Modelle verwendet, die die Sicherheit einer Zerlegung durch Zuordnung eines Gewichtes beschreiben.

4.2.2.1 Die äußere Grammatik

Ein gegen leichte Änderungen der Eingaben sehr robustes Modell sind n-Gramme (siehe 3.3.3.1), die in der Informationsextraktion sehr erfolgreich eingesetzt werden. Dieses Modell wird vor allem dazu eingesetzt, für eine gegebene Zeichenkette zu ermitteln, mit welcher Wahrscheinlichkeit sie als Wort der Modellierten Sprache auftritt. Die Bewertung eines Wortes ergibt sich hier aus dem Produkt der Trigramwahrscheinlichkeiten der einzelnen Zeichen. Dabei ist jeder dieser Faktoren ein Wert kleiner 1. Dadurch werden bei einer Anwendung dieses Modells zur Suche der besten Strukturliste Zerlegungen mit wenigen Elementen gegenüber Zerlegungen mit vielen Elementen bevorzugt, was dann zu falschen Vorhersagen führt.

In diesem Fall wird eine Zerlegung der Eingabe in eine Strukturliste gesucht, deren genaue Zusammensetzung, insbesondere deren Länge nicht bekannt ist. Es geht hier also nicht einfach darum, den Elementen einer vorggegebenen Liste eine Klasse zuzuweisen. Daher ist ein Trigrammodell hier so nicht anwendbar.

Hier wird nun ein relationales Modell gewählt. Die Beispiele sind durch Strukturlisten der Form $(e_1, \ldots, e_n)$ dargestellt. Diese werden durch ein besonderes Element ergänzt, das den Anfang und das Ende des Beispiels kennzeichnet: $(\$, \$, e_1, \ldots, e_n, \$)$. Diese Listen werden nun durch eine dreistellige Relation R modelliert. Dabei gibt $R(e_i, e_{i+1}, e_{i+2})$ an, daß innerhalb einer Strukturliste das Element e_{i+3} auf die Elemente e_i, e_{i+1} folgt. Den Tripeln dieser Relation wird nun ein Gewicht zugeordnet, das nach einem ähnlichen Schema wie die Wahrscheinlichkeiten eines geglätteten Trigrammodells ermittelt werden. Dazu wird zunächst die relative Häufigkeit $H_R(e_i, e_{i+1}, e_{i+2})$ berechnet, mit der e_{i+2} auf die Elementfolge e_i, e_{i+1} folgt. Diese ergibt sich aus den absoluten Häufigkeiten, mit denen die Folgen e_i, e_{i+1} in den Beispielen vorkommen:

$$H_R(e_i, e_{i+1}, e_{i+2}) = \frac{H(e_i, e_{i+1}, e_{i+2})}{H(e_i, e_{i+1})}$$

Auf ähnliche Weise wird die relative Häufigkeit, mit der e_{i+2} auf e_{i+1} folgt, ermittelt:

$$H_R(e_{i+1}, e_{i+2}) = \frac{H(e_{i+1}, e_{i+2})}{H(e_{i+1})}$$

Dabei ist $H(e_{i+1})$ die absolute Häufigkeit, mit der das Element e_{i+1} in den Beispielen vorkommt. Sei N die Summe der Längen aller Beispiellisten, also die Gesamtzahl aller Elemente, so ist $H_R(e_{i+2}) = \frac{H(e_{i+2})}{N}$ die relative Häufigkeit von e_{i+2} ohne Berücksichtigung des Kontextes.

Das Gewicht $W(e_i, e_{i+1}, e_{i+2})$ ergibt sich nun als gewichtete Summe

$$W(e_i, e_{i+1}, e_{i+2}) = \alpha * H_R(e_i, e_{i+1}, e_{i+2}) * \beta * H_R(e_{i+1}, e_{i+2}) * \gamma * H_R(e_{i+2})$$

Die Summe der Faktoren α, β und γ muß 1 betragen. Die Werte der Faktoren wurden bei dieser Anwendung heuristisch festgelegt mit $\alpha = 0.7$, $\beta = 0.2$ und $\gamma = 0.1$. Das Gewicht einer Strukturliste $(\$, \$, e_1, \ldots, e_n, \$)$ ergibt sich nun aus dem durchschnittlichen Gewicht aller Elementfolgen der Länge drei, die diese Liste enthält:

$$\begin{aligned} W(\$, \$, e_1, \ldots, e_n, \$) = \frac{1}{n} * \quad (\quad & W(\$, \$, e_1) + W(\$, e_1, e_2) + W(e_1, e_2, e_3) + \ldots + \\ & W(e_i, e_{i+1}, e_{i+2}) + \ldots + W(e_{n-2}, e_{n-1}, e_n) + \\ & W(e_{n-1}, e_n, \$) \quad) \end{aligned}$$

4.2.2.2 Die Charakterisierungen der Felder

Für die Charakterisierung der einzelnen Felder müssen nun Beschreibungen gefunden werden, mit deren Hilfe entschieden werden kann, ob eine bestimmte Zeichenkette eine Instanz eines gegebenen Feldes ist, oder nicht.

Betrachtet man die einzelnen Felder genauer, so stellt man fest, daß diese Felder sehr unterschiedliche Eigenschaften haben. Manche haben eine sehr einfache Struktur, wie zum Beispiel Jahreszahlen. Andere Felder sind zwar ebenfalls deutlich strukturiert, diese Struktur ist aber etwas komplizierter. Einige Felder, wie z.B. die Titelfelder, können aus praktisch beliebigen Zeichenketten bestehen. Eine große Schwierigkeit besteht auch darin, negative Beispiele zu erzeugen. Ein naheliegender Gedanke wäre, Instanzen der jeweils anderen Felder als negative Beispiele zu verwenden. Das Problem dabei ist jedoch, daß die durch die Felder gegebenen Sprachen nicht unbedingt disjunkt sind. So kann z.B. "1982" sowohl eine Jahres-, als auch eine Seitenzahl sein. Manche Felder können sogar exakt dieselben Inhalte haben. Dies betrifft zum Beispiel Autoren und Herausgeber. Beides können beliebige Namen sein. Die einzige Möglichkeit an zuverlässige negative Beispiele zu kommen wäre, diese vom Benutzer vorgeben zu lassen oder einige Zeichenketten als negative Beispiele auszuwählen und diese dem Benutzer zur Kontrolle vorzulegen.

Man benötigt hier also Lernverfahren, die ohne negative Beispiele auskommen. Außerdem sollte die Laufzeit der Verfahren nicht zu groß sein, da das Lernen nach jedem verarbeitetem Literaturhinweis wieder neu angestoßen wird.

Aufgrund der oben beschriebenen Eigenschaften der Felder ist es günstig, auch hier wieder Verfahren zu verwenden, die keine exakte Entscheidung liefern, ob eine Zeichenkette Instanz eines Feldes ist oder nicht, sondern ein Gewicht liefern, das die Sicherheit dieser Zugehörigkeit beschreibt.

Hier wird nun nicht ein einziges Verfahren für die Klassifizierung der Felder verwendet, sondern es werden mehrere Gewichte mit verschiedenen Verfahren ermittelt, die dann zu einem Gesamtgewicht zusammengefaßt werden.

Dazu wird zusätzlich zu der Tokenkette, die aus Wörtern, Zahlen und nichtalphanumerischen Zeichen besteht, noch eine zweite Darstellung der Beispiele erzeugt. Diese zweite Darstellung ist wiederum eine Tokenkette, wobei die Token aber eine Abstraktion des ursprünglichen Beispiels darstellen. Diese Tokenkette wird erzeugt, indem die einzelnen Token wie folgt durch neue Token ersetzt werden:

1. Wörter werden durch ein Token $< Wort >$ ersetzt.

2. Einzelne Buchstaben, die also nicht Bestandteil eines Wortes sind werden

durch ein Token <Buchstabe> ersetzt.

3. Ziffern werden durch ein Token $< Ziffer >$ ersetzt.
4. Alle anderen Zeichen bleiben, wie sie sind.

Ein Beispiel für die beiden Darstellungsarten könnte wie folgt aussehen:

(“Machine”,” “,”Learning”,”, “,”2”)

(<Wort>,” ”,<Wort>,”, “,<Ziffer>)

Dann werden im wesentlichen zwei Verfahren angewandt. Aus den abstrahierten Tokenketten werden Automaten gelernt, die die Struktur dieser Ketten beschreiben. Um Automaten zu lernen, die die Felder auf der Ebene der tatsächlichen Token beschreiben werden sehr viele Beispiele benötigt. Durch die Beschränkung auf die abstrakten Token werden der Hypothesenraum und die Beispielsprache stark vereinfacht, wodurch auch die Induktion eines Automaten erheblich vereinfacht ist.

Als zweites wird ein Verfahren verwendet, das die Häufigkeiten, mit denen bestimmte Zeichen oder Token in den ursprünglichen Beispielen vorkommen, nutzt. Dasselbe geschieht noch einmal mit den abstrahierten Tokenketten.

Aus dem aufgrund der Struktur einer Tokenkette ermittelten Gewicht W_S und dem aus den Tokenhäufigkeiten ermitteltn Gewicht W_H wird ein Gesamtgewich $W_{Gesamt} = 0.5 * W_S + 0.5 * W_H$ berechnet.

Strukturautomaten

Aus den abstrahierten Tokenketten werden DFAs gelernt. Dafür wird der Algorithmus von Dana Angluin ([Angluin, 1982]) verwendet. Dieser Algorithmus ist geeignet, k-reversible Sprachen effizient zu lernen. Eine Sprache ist genau dann k-reversibel, wenn jeweils zwei Präfixe, deren letzten k Zeichen übereinstimmen und die ein gemeinsames Suffix haben, alle Suffixe gemeinsam haben.

Nun sind die Feldsprachen im allgemeinen nicht k-reversibel. Folgendes Beispiel zeigt, daß die Sprache der Seitenzahlangaben nicht 3-reversibel ist ($< Z >$ steht hier für <Ziffer>):

$$\begin{array}{rl} \text{(P1)}\ < Z >< Z >< Z > - & \\ & < Z >< Z >< Z > \\ \text{(P2)}\ < Z >< Z > - & \end{array}$$

Hier sind also zwei Präfixe, deren letzten drei Zeichen identisch sind. An beide Präfixe kann das Suffix $< Z >< Z >< Z >$ angehängt werden. Jedoch ist die Endseitenzahl immer größer, als die Anfangsseitenzahl. Daher ist $< Z >< Z >$ kein gemeinsames Suffix für (1) und (2), da eine Seitenangabe der Form $< Z >< Z >< Z > - < Z >< Z >$ nicht erlaubt ist. Im Allgemeinen ist eine genaue Kenntnis der einzelnen zu findenen Felder nötig, um feststellen zu können, ob und für welches k die Sprachen k-reversibel sind.

Für die Anwendung auf die Literaturhinweise wird hier einfach davon ausgagangen, daß die gesuchte Sprache 0-reversibel ist. Dies sichert vor allem eine schnelle Generalisierung. Die Folge davon ist, daß die gelernten Automaten im allgemeinen eine Sprache beschreiben, die eine Obermenge der gesuchten Sprache ist, also übergeneralisieren.

Wie aber schon gesagt ist hier eine exakte Identifizierung der Sprache nicht notwendig. Es reicht, wenn die Felder so weit klassifiziert werden, daß eine korrekte Erkennung unter Zuhilfename der strukturellen Information aus der äußeren Grammatik möglich ist. Im Falle der Seitenangaben kommt der oben konstruierte Fall sogar überhaupt nicht vor, so daß es gar keine Rolle spielt, ob diese Worte von dem gelernten Automaten erkannt werden oder nicht.

Der Algorithmus von Angluin beginnt zunächst mit dem Aufbau eines Präfixautomaten für alle Beispiele. Zunächst wird das erste Beispiel verarbeitet. Dabei wird das Beispiel Token für Token durchlaufen. Ausgehend vom Startknoten wird dabei für jedes Token eine Transition vom aktuellen Knoten zu einem neuen Knoten eingefügt. Die Transition wird mit dem Token markiert und der neue Knoten wird der aktuelle Knoten. Der letzte Knoten dieser Sequenz ist ein akzeptierender Knoten. Für die Verarbeitung der nächsten Beispiele beginnt man jeweils wieder mit dem Startknoten und durchläuft das Beispiel. Solange von dem aktuellen Knoten eine Transition ausgeht, die mit dem Token markiert ist, wird der Zielknoten dieser Transition zum aktuellen Knoten und man geht ein Token weiter. Sobald das erste Token gefunden wird, für das vom aktuellen Knoten aus keine Transition existiert, geht man ab hier so vor wie beim ersten Beispiel. Der letzte aktuelle Knoten wird ebenfalls ein akzeptierender Knoten.

Dieser Automat akzeptiert nun genau die Beispiele. Um diesen Automaten zu generalisieren werden in dem Automaten Knoten nach den folgenden Regeln zusammengefaßt:

- **Regel 1:** Zwei Zustände q_1 und q_2 werden zusammengefaßt, wenn gilt:q_1 und q_2 haben einen gemeinsamen Vorgängerknoten q_V und von diesem Vorgängerknoten aus gibt es zu den beiden Knoten jeweils eine Transition mit derselben Markierung v. Das heist es gilt $\delta(q_V, v) = q_1$ und $\delta(q_V, v) = q_2$.

Diese Regel garantiert, daß der erzeugte Automat deterministisch ist, also von jedem Knoten aus für die Ausgabe eines bestimmten Tokens nur eine Transition benutzt werden kann.

- **Regel2:** Zwei Zustände q_1 und q_2 werden zusammengefaßt, wenn es ein Wort $w = v_1 \dots v_k$ gibt, so daß beide Zustände über einen Pfad der Länge k erreicht werden können, der mit w markiert ist, und eine der beiden folgenden Bedingungen erfüllt ist:

  1. q_1 und q_2 sind akzeptierende Zustände.
  2. Es gibt einen Knoten q_N, der von q_1 und q_2 aus mit einer Transition erreicht werden kann, die mit demselben Token v markiert ist. Es gilt dann also $\delta(q_1, v) = q_N$ und $\delta(q_2, v) = q_N$.

Bei der Generalisierung werden zunächst alle Knotenpaare des Automaten untersucht, ob sie diese Regeln erfüllen, oder nicht. Knoten, die die Regeln erfüllen werden zusammengefaßt. Solange während eines solchen Durchlaufes Knoten zusammengefaßt wird der Vorgang wiederholt.

Um zwei Zustände q_1 und q_2 zusammenzufassen, werden zunächst für alle Transitionen, die in q_2 enden, neue Transitionen eingeführt, die in q_1 enden. Das heißt für alle Transitionen v und Zustände q mit $\delta(q, v) = q_2$ wird $\delta(q, v) = q_1$ gesetzt. Ebenso werden für alle von q_2 ausgehenden Transitionen neue von q_1 ausgehenden Transitionen eingesetzt. Also für alle v und q mit $\delta(q_2, v) = q$ wird $\delta(q_1, v) = q$ gesetzt. Dann werden q_2 und alle ein- und ausgehenden Transitionen aus dem Automaten entfernt.

Durch diesen Algorithmus wird ein k-reversibler DFA erzeugt, der die Beispiele akzeptiert. Für die Anwendung im Rahmen dieser Arbeit wird der Automat noch durch ein Verfahren ergänzt, das es erlaubt, die von dem erzeugten DFA akzeptierten Wörter mit Gewichten zu versehen. Dazu werden alle Zustände und Transitionen jeweils mit einem Zähler versehen. Für jedes Beispiel wird nun der das Beispiel akzeptierende Pfad durch den Automaten gesucht. Da der Automat deterministisch ist, gibt es jeweils nur einen solchen Pfad. Dann wird gezählt wie oft jeder Zustand und jede Kante von diesen akzeptierenden Pfaden durchlaufen wird. Daraus wird die relative Häufigkeit $P(q, v)$, mit der die mit v markierte Transition durchlaufen wird, wenn sich der Automat beim Parsen eines Besipiels im Zustand q befindet. Das Gewicht eines Wortes ist dann das Produkt der relativen Häufigkeiten aller Transitionen des dieses Wort akzeptierenden Pfades.

Tokenbasierte Gewichte

Da durch die Strukturautomaten nur die abstrakte Struktur der Felder beschrieben wird, werden weitere Modelle benötigt, die die Felder aufgrund der tatsächlichen Tokenketten beschreiben. Dazu werden hier einige naive Häufigkeitsmaße verwendet, um die Erkennung der Felder zu verbessern. Insbesondere sollen diese Maße schon in einem frühen Stadium hilfreich sein, wenn noch nicht genügend Beispiele gesehen wurden, um die Automaten genügend zu generalisieren. Das erste hier verwendete Maß beshreibt für jedes Token x, in welchem Maße es einen Hinweis auf die Feldklasse einer Tokenkette ist. Dazu weden die Häufigkeit $H_F(x)$, mit der ein Token in einem Feld F vorkommt und $H_G(x)$, mit der das Token insgesamt vorkommt. Das daraus ermittelte Gewicht $W_1(x) = H_F(x)/H_G(x)$ beschreibt, wie sicher die Zuordnung einer Tokenkette, die x enthält zu Feld F ist. Token, die nur in F vorkommen erhalten das Gewicht 1 und solche die nie vorkommen, das Gewicht 0. Das Gesamtgewicht einer Tokenkette $w = (x_1, ..., x_n)$ ist der Quotient $W_1(w) = \frac{1}{n}\Sigma_{i=1}^{n} W_1(x_i)$, also das durchschnittliche Gewicht der einzelnen Token. Analog wird aus der Häufigkeit H_{NF}, mit der x in anderen Feldern als F vorkommt das Gewicht $W_2(x) = H_{NF}(x)/H_G(x)$ bzw. $W_2(w) = \frac{1}{n}\Sigma_{i=1}^{n} W_2(x_i)$ berechnet. Dieses Gewicht beschreibt, in welchen Maß ein Token x in einer Tokenkette ein Anhaltspunkt dafür ist, daß sie keine Instanz von F ist.

Dieselben Maße werden noch einmal auf die abstrakten Token angewandt. Das heißt es werden die entsprechenden Häufigkeiten für die den x_i zugeordneten abstrakten Token x'_i ermittelt und die Gewichte $W_3(w') = \frac{1}{n}\Sigma_{i=1}^{n} H_F(x_i)/H_G(x_i)$ und $W_4(w') = \frac{1}{n}\Sigma_{i=1}^{n} H_{NF}(x_i)/H_G(x_i)$ Dabei ist $w' = (x'_1, \ldots, x'_n)$. Im Gegensatz zu den Strukturautomaten werden hier nicht die Abfolgen der Token betrachtet, sondern das Auftreten eines Tokens von einem bestimmten Typs wird als unabhängig von den umgebenden Token angesehen. Dadurch lassen sich hier schneller Aussagen machen, als mit den Automaten. Da sich die abstrakten Token häufig wiederholen, sind die so gewonnenen Gewichtungen sehr schnell brauchbar. Vor allem Felder, die nur aus bestimmten Tokentypen bestehen, wie Zahlen oder Namen können mit Hilfe dieser Gewichte schon nach wenigen Beispielen gut von anderen Feldern getrennt werden.

Das Gesamtgewicht der Tokenkette w ist die gewichtete Summe der Einzelgewichte $W_T(w) = \alpha W_1(w) - \beta W_2(w) + \gamma W_3(w) - \delta W4(w)$. In der Anwendung haben sich die Gewichtungen $\alpha = \beta = \gamma = \delta = 0.5$ als die brauchbarsten Gewichtungen herausgestellt.

Schließlich wird noch eine heuristische Bewertung vorgenommen. Eine gegebene Tokenkette wird nach Klammern durchsucht. Enthält eine Tokenkette nicht geschlossene Klammern, so wird das Gewicht der Zeichenkette auf 0 gesetzt. Klammerpaare gehen schließlich nie über Feldgrenzen hinweg. Entweder fungieren Klammern als Feldtrenner, dann gehört die zugehörige schließende Klammer zu einem anderen Feldtrenner, oder Klammerpaare befinden sich vollständig innerhalb eines Feldes. Daher kann man ausschließen, das Tokenketten mit offenen

Klammern gültige Instanzen eines Feldes sind.

4.2.2.3 Die Analyse der Eingabe

Gesucht wird nun eine Strukturliste $L_S = (e_1, ..., e_n)$, die die Zerlegung der Eingabe in Felder und Feldtrenner beschreibt. Die e_i sind dabei Tokenketten, die zusätzlich als Feld vom Typ F oder als Trennzeichen gekennzeichnet sind. Die Konkatenation der Tokenketten in der gegebenen Reihenfolge soll der Eingabe entsprechen und das Gewicht der Liste entsprechend der oben beschriebenen Modelle maximal sein. Das Gesamtgewicht der Liste ist dabei die Summe aus dem Gewicht der Liste nach dem Modell für die äußere Grammatik und dem Gewicht der Elemente nach den Modellen für die Felder:

$$\begin{aligned} W(L_S) &= W_A(L) + \sum_{i=1}^{n} W_F(e_i) \\ &= [\ W_A(\$, \$, e_1) + W_A(\$, e_1, e_2) + \\ &\quad \sum_{i=1}^{n-2} W_A(e_i, e_{i+1}, e_{i+2}) + W_A(e_{n-1}, e_n, \$)\ ] + \\ &\quad \frac{1}{n} \sum_{i=1}^{n} W_F(e_i) \end{aligned}$$

Das durchschnittliche Gewicht eines Elementes dieser Strukturliste ist der Quotient aus dem Gewicht der Strukturliste und der Anzahl n der Elemente:

$$W_D(L_S) = \frac{W(L_S)}{n}$$

Das innere Gewicht $W_F(e_i)$ für ein e_i, das ein Trennzeichen ist, ist 1. Der Beitrag eines e_i zum Gesamtgewicht der Liste ergibt sich also aus seiner Position innerhalb der äußeren Struktur und durch das Gewicht, das sich aus den Feldmodellen ergibt.

Um nun die beste Zerlegung zu finden wird eine Best-First-Suche ausgeführt, die durch einige Heuristiken beschränkt ist. Die Beschränkung ist nötig, da der Suchraum hier sehr groß ist.

Bei der Suche werden, angefangen mit einer leeren Strukturliste, bereits gefundene Strukturlisten, die ein Anfangsstück der Eingabe abdecken, so lange um jeweils ein Element erweitert, bis sie die gesamte Eingabe abdecken. Dazu wird eine Menge M verwendet, die alle bisher erzeugten Strukturlisten enthält. Zu Anfang befindet sich in ihr nur eine Strukturliste, die zwei Startsymbole $\{\$, \$\}$ enthält. Diese Liste enthält also noch keine Elemente einer möglichen Zerlegung. Jeder Liste in M wird ihr Gewicht gemäß der Modelle für die äußere und die innere Struktur zugeordnet. Aus diesem Gewicht und der Anzahl der Elemente in der Liste (ohne Start- und Endsymbole) wird das durchschnittliche Gewicht der Liste berechnet. Das durchschnittliche Gewicht der Liste $\{\$, \$\}$ wird mit 0 festgelegt. Aus der Menge M wird dann in jedem Schritt der Suche die Strukturliste A mit dem höchsten durchschnittlichen Gewicht herausgenommen. Für jedes Element, das als nächstes Element der gesuchten Strukturliste in Frage kommt, wird eine neue Liste erzeugt, indem die alte Liste um dieses Element erweitert wird. Die erweiterten Strukturlisten werden dann in M eingefügt. Wird dabei eine Strukturliste gefunden, die die gesamte Eingabe abdeckt, also nicht erweitert werden kann, wird diese als vorläufige Lösung behalten, wenn sie die erste vollständige Liste ist oder ihr durchschnittliches Gewicht höher als das der bisherigen vorläufigen Lösung ist.

Um die möglichen Erweiterungen einer Struktuliste zu ermitteln werden zunächst alle Abschnitte der Eingabe gesucht, die als nächstes Element dieser Teilausgabe in Frage kommen. Dies sind alle Anfangsstücke des noch nicht abgedeckten Teils der Eingabe, die von einem möglichen Feldtrenner gefolgt werden. Für die Eingabe

“D. Angluin (1982). Inference of reversible Languages ...”

sehen die ersten Abschnitte so aus:

“D”
“D.”
“D. Angluin”
“D. Angluin (1982”
...

Die erlaubten Teilketten müssen dabei von einem gültigen Feldtrenner gefolgt werden. Daher käme “D. “ in dem Beispiel nicht in Frage. Der nächste Feldtrenner müßte dann mit “Angluin” beginnen, und so ein Feldtrenner ist in den bisherigen Beispielen nicht vorgekommen.

Da sich in der Ausgabe immer Felder und Feldtrenner abwechseln, kann der folgende Feldtrenner auch gleich ermittelt werden. Für die ersten Beispiele der obigen Liste erhält man dann:

“D”, “. “
“D.”, “ “
“D. Angluin”, “ (“
“D. Angluin (1982”, “). “
...

"." als Feldtrenner nach "D" kann ausgeschlossen werden, da dann das folgende Feld mit einem Leerzeichen beginnt. Dies ist aber nie der Fall. Felder beginnen immer mit einem alphanumerischen Zeichen oder eventuell einer Klammer.

Die Teilketten für das nächste Feld werden mit allen möglichen Feldbezeichnern zu Feldelementen erweitert. Die Teilketten für die Feldtrenner entsprechend mit dem Bezeichner für Feldtrenner. Diese Paare werden dann als mögliche Erweiterung an die Strukturliste A angehängt. Für die erste Strukturliste $\{\$, \$\}$ werden z.B. folgende Erweiterungen betrachtet (n sei die Anzahl der Felder):

{$, $, (1, "D"), (T, ". ")}
{$, $, (2, "D"), (T, ". ")}
...
{$, $, (n, "D"), (T, ". ")}
...
{$, $, (1, "D."), (T, " ")}
...
{$, $, (n, "D."), (T, " ")}
...

Die so erweiterten Strukturlisten werden nun in die Menge M aufgenommen und die Strukturliste A, die erweitert wurde wird entfernt. Nun wird wieder die Teilausgabe aus M mit dem höchsten Gewicht gesucht und wie beschrieben erweitert. Umfaßt die Teilausgabe die gesamte Eingabe, so wird an diese ein Endsymbol gehängt und das endgültige Durchschnittsgewicht ermittelt. Ist dies die erste vollständige Strukturliste oder ist ihr Gewicht höher als das der besten bisher gefundenen vollständigen Liste, so wird sie als vorläufige Lösung behalten.

Sobald die erste vollständige Lösung gefunden wurde, kann der Suchraum weiter beschränkt werden. Wenn das maximal erreichbare Durchschnittsgewicht einer Teilausgabe abgeschätzt werden kann, so brauchen alle Teilausgaben, die das Durchschnittsgewicht der besten bisher gefundenen Strukturliste nicht mehr erreichen können, nicht weiter berücksichtigt zu werden. Dazu muß bekannt sein, um wieviele Felder eine Strukturliste maximal noch erweitert werden kann. Angenommen, eine Strukturliste $(e_1, ...e_n)$ mit n Elementen hat bisher das Gewicht W und kann um bis zu k Elemente erweitert werden. Für jedes zusätzliche Element e_i erhöht sich das Gewicht einer Strukturliste um das äußere Gewicht des Elementes $(W_A(e_{i-2}, e_{i-1}, e_i))$ und das innere Gewicht des Elementes $W_F(e_i)$. Beide Gewichte können maximal 1 betragen. Dann kann das maximal mögliche Gewicht W' folgendermaßen abgeschätzt werden: $W' = W + 2k$. Das maximale Durchschnittsgewicht ist dann $W'' = \frac{W'}{k+n}$.

Um die Anzahl der folgenden Felder abzuschätzen, wird zum einen der noch nicht abgedeckte Teil der Eingabe betrachtet. Dabei wird ermittelt, wie lang die längste mögliche Kette von Feldern und Feldtrennern in der Resteingabe ist. Die Länge dieser Kette gibt eine Obergrenze für die Zahl der folgenden Felder an. Dies kann jedoch zu sehr großen Anzahlen führen, wenn der Rest der Eingabe sehr lang ist, und viele potentielle Trennzeichen enthält. Daher wird zum anderen die Gesamtzahl der Felder in einer Strukturliste beschränkt. Es wird angenommen, daß eine Eingabe maximal vier Felder mehr enthält, als das größte bisher gesehene Beispiel. Diese Annahme ist in sofern vernünftig, da Eingaben, deren Feldanzahl stärker von den Beispielen abweichen in der Regel strukturell stark von den Beispielen abweichen, so daß sie auch ohne diese Beschränkung nicht zuverlässig erkannt werden können.

4.2.3 Die Reformulierung der Eingabe

Die Reformulierung der Eingabe ist verglichen mit der Analyse der Eingabe eine erheblich leichtere Aufgabe. Hier geht es nur noch um ein reines Anordnungsproblem. Alle Fragen, die mit der Bedeutung einzelner Komponenten von Literaturhinweisen zu tun haben, spielen hier keine Rolle. Außerdem ist die Komplexität des Problems erheblich reduziert worden. Die Reformulierungskomponente kann schliesslich die Felder als Black Boxes betrachten, da sie ja genauso, wie sie sind, in der Ausgabe vorkommen.

Die Ausgabe kann schrittweise aus den vorher gefundenen Feldern aufgebaut werden. Dabei wird zunächst mit einem speziellen Startsymbol $ begonnen. Dann wird die Ausgabe abwechselnd um Trennzeichen und Felder erweitert. Die bereits verwendeten Felder werden aus der Liste gestrichen. Zu jedem Zeitpunkt hat man also die bisher konstruierte Ausgabe und die Liste der noch verbleibenden Felder und muß nun entscheiden, welches Element der Restliste das nächste Element der Ausgabe ist. Diese Entscheidung ist aus den gegebenen Informationen eindeutig zu treffen. Gesucht ist also eine Menge von Regeln, die diese Entscheidung ermöglichen. Diese Aufgabe läßt sich sehr gut als Begriffslernaufgabe (siehe 3.3.3) repräsentieren.

Die so gelernten Regeln funktionieren jedoch nicht sehr gut in Fällen, für die die bisher gesehenen Beispiele keine eindeutige Entscheidung zulassen. Dies trifft insbesondere zu, wenn in der Eingabe gegenüber den Beispielen bestimmte Felder fehlen oder neue Kombinationen bekannter Felder auftreten. Angenommen, es wurden zum Beispiel bisher nur Literaturhinweise für Konferenzbeiträge gesehen, die Konferenztitel, Seitenzahl, Verlag und Verlagsort enthielten, und die wie folgt formatiert wurden:

… <*Titel*>, pages <*Seiten*>, <*Ort*>:<*Verlag*> …

Enthält die aktuelle Eingabe diesmal keine Seitenangabe, so kann nicht entschieden werden, welches Element nach dem Titel auftreten soll. Mit den gelernten Entscheidungsregeln kommt man jetzt nicht weiter. Eine sichere Entscheidung ist hier zwar nicht möglich, jedoch könnte man versuchen, die Ausgabe zu raten. Daher werden hier wieder genau wie für die Eingabesprache dreistellige Relationen verwendet, die durch die wie beschrieben abgeschätzten Gewichte auch Vorhersagen bei noch nicht gesehenen Kombinationen erlauben. Gesucht wird dann wieder eine Abfolge von Feldern, deren Gewicht maximal ist. Im Falle der Ausgabe müssen jedoch nicht die Feldgewichte berücksichtigt werden.

Bei unvollständigen Daten kann man so die beste Anordnung "raten". Der Vorteil dieses Modells liegt darin, daß hier auch bei ungenügenden Beispielen noch Vorhersagen gemacht werden können, ohne wirklich blind zu raten. Der Nachteil ist, daß hier die Informationen der Restliste nicht berücksichtig werden. Bei Literaturhinweisen ist dies beispielsweise bei den Autorenlisten problematisch:

...<*VName1*> <*Name1*>, <*VName2*> <*Name2*> and
<*VName3*> <*Name3*> ...

Hier würde sowohl zwischen *Name1* und *VName2*, als auch zwischen *Name2* und *VName3* das wahrscheinlichste Zeichen gesetzt, je nach ihrer Häufigkeit ein Komma oder ein "and".

Um nun zu guten Vorhersagen zu kommen, wird eine Kombination der oben beschriebenen Methoden angewandt. Mit Hilfe des Begriffslernens werden für die Fälle, in denen das relationale Modell mehrdeutig ist, Regeln gelernt, die eine eindeutige Vorhersage ermöglichen. Dann wird die Ausgabe mit dem nach dem relationalen Modell höchsten Gewicht gesucht, die diese Regeln erfüllt. Dadurch wird sichergestellt, daß in allen Fällen, für die die bisherigen Beispiele ausreichend sind, die richtige Lösung gefunden wird und in den unsicheren Fällen eine Lösung gefunden wird, die immer noch die gesehenen Beispiele nutzt.

4.2.3.1 Lernen der Reformulierung als Begriffslernen

Für jedes Element (Feld oder Feldtrenner), auf das in den Beispielen mindestens zwei unterschiedliche Elemente folgen können, sollen Regeln gelernt werden, mit deren Hilfe entschieden werden kann, welches Element in einem gegebenen Fall als nächstes ausgegeben werden soll. Dabei wird jedes Paar aufeinanderfolgender Elemente als zu lernender Begriff aufgefaßt. Als Beispiel will ich das obige Problem bei Autorenlisten nehmen. Dabei seien Vornamenfelder als Feld1 und

Nachnamenfelder als Feld2 bezeichnet. Im Beispielformat können auf Feld2 die Trennzeichen “, “, “ and “ und “ (“ folgen, je nachdem, wieviele Autorennamen noch in der Restliste sind. Nun soll gelernt werden, wann Feld2 von “, “ gefolgt wird. Als bisherige Beispielhinweise seien ein Hinweis mit drei Autoren und einer mit zwei Autoren gegeben:

<*Feld1*> <*Feld2*>, <*Feld1*> <*Feld2*> and <*Feld1*> <*Feld2*> (...
<*Feld1*> <*Feld2*> and <*Feld1*> <*Feld2*> (...

Um positive Beispiele zu erzeugen, werden nun alle Stellen in den Beispielen gesucht, an denen Feld2 von “, “ gefolgt wird. Ein Beispiel besteht aus der Ausgabe bis zur kritischen Stelle (als Strukturliste AL) und der Restliste (als Feldliste RL) für jede dieser Stellen. Da hier die zu den Feldern gehörenden Tokenketten weggelassen werden können, werden Felder nur mit ihrem Bezeichner und Feldtrenner mit der zugehörigen Tokenkette angegeben. Da Feld 2 in den Beispielen nur einmal von “, “ gefolgt wird, gibt es auch nur ein positives Beispiel ($P1$):

- $P1$: AL=(Feld1,” “,Feld2)
  RL=(Feld1,Feld2,Feld1,Feld2,Feld3,...)

Die negativen Beispiele werden entsprechend für alle Stellen erzeugt, an denen Feld2 von einem anderen Feldtrenner gefolgt wird. In unserem Beispiel gibt es drei negative Beispiele ($N1$-$N3$):

1. $N1$: AL=(Feld1,” “,Feld2,”, “,Feld1,” “,Feld2)
   RL=(Feld1,Feld2,Feld3,...)
2. $N2$: AL=(Feld1,” “,Feld2,”, “,Feld1,” “,Feld2,” and “,Feld1,” “Feld2)
   RL=(Feld3,...)
3. $N3$: AL=(Feld1,” “,Feld2,” and “,Feld1,” “,Feld2)
   RL=(Feld3,...)

Für das Lernen der Regeln werden die Beispiele und das Hintergrundwissen als variablen- und funktionenfreie Fakten dargestellt. Der Hypothesenraum sind die Hornklauseln mit einem Kopfliteral. Der Zielbegriff ist ein Prädikat *translation()*, das jeweils der bisherigen Ausgabe und der Liste der übriggebliebenen Felder das nächste Element der Ausgabe zuordnet.

Nun zur Darstellung der Beispiele. Ein Beispiel besteht also aus der bisherigen Ausgabe, die durch eine Strukturliste dargestellt wird, einer Liste von noch einzubauenden Feldern und dem nächsten Element der Ausgabe. Um diese recht komplizierten Beispiele als Grundfakten darstellen zu können, wird ein Trick angewendet. Dazu werden allen Beispielen jeweils eindeutige Atome als Bezeichner zugeordnet. Diese Atome heißen *id_1,id_2* usw.. Die tatsächliche Struktur der Beispiele wird dann im Hintergrundwissen modelliert. Ebenso werden den Feldern und Feldtrennern eindeutige Atome (*field_1,field_2...* und *sep_1,sep_2...*) zugeordnet. Im Beispiel von oben werden die Felder mit *field_1-field3* bezeichnet. Die Feldtrenner " ", ", ", " and " und " (" mit *sep_1-sep_4*. Das Zielprädikat ist dann das dreistellige Prädikat *translation(E,L,N)*. Dabei bedeutet *translation(id_5,sep_7,field_1)*, daß bei Beispiel Nr. 5 das letzte Element der aktuellen Eingabe der Feldtrenner Nr. 7 und das nächste Element der Eingabe Feld Nr. 1 ist.

Bezeichnet man die Beispiele $P1$ und $N1$-$N3$ mit den Atomen *id_1-id_4*, erhält man folgende Beispielprädikate:

- $P1$:*translation(id1,field2,sep_2)*
- $N1$:*translation(id2,field2,sep_2)*
- $N2$:*translation(id3,field2,sep_2)*
- $N3$:*translation(id4,field2,sep_2)*

Die Beispiele werden dann im Hintergrundwissen genauer beschrieben. Dazu werden folgende Prädikate verwendet:

1. *contains(ID,F)* bedeutet, daß bei Beispiel ID das Feld F mindestens ein mal in der Restliste enthalten ist.
2. *contains_sing(ID,F)* bedeutet, daß bei Beispiel ID das Feld F genau einmal in der Restliste enthalten ist.
3. *contains_mult(ID,F)* bedeutet, daß bei Beispiel ID das Feld F mehrfach in der Restliste enthalten ist.
4. *previous(ID,E)* bedeutet, daß E das vorletzte Element in der Ausgabekette von Beispiel ID ist (das letzte Element ist ja schon im Beispielfakt angegeben).
5. *empty_rest(ID)* bedeutet, daß die Restliste bei Beispiel ID leer ist.

Dazu kommen noch die Sortenprädikate field(F) für Felder, sep(F) für Feldtrenner und id(ID) für Beispiele. Bei den Versuchen mit dieser Repräsentation hat sich gezeigt, daß die aktuelle Ausgabe nicht weiter als bis zu den letzten beiden Elementen untersucht werden muß. Daher reicht das *previous*-Prädikat in dieser Form aus.

Ein Ausschnitt aus dem Hintergrundwissen zu dem einzigen positiven Beispiel $P1$ sieht folgendermaßen aus:

contains_mult(id_1,field_1)
contains(id_1,field_1)
contains_mult(id_1,field_2)
contains(id_1,field_2))
previous(id_1,sep_1)

Mit dieser Repräsentation werden für die Anwendung in dieser Arbeit mit Hilfe des Systems Progol Regeln gelernt. Eine so gelernte Regel könnte z.B. sein:

translation(A,field_2,sep_2):-contains_mult(id_1,field_1).

Wenn f

Kapitel 5

Experimente

Ziel dieser Arbeit war es, ein Vefahren zu entwickeln, das einen Benutzer bei der Umformatierung von Literaturhinweisen unterstützt. Dabei soll ohne umfangreiches Hintergrundwissen aus Beispielen gelernt werden, wie diese Umformatierung auszuführen ist. Dieses Verfahren soll weiterhin möglichst folgende Randbedingungen erfüllen, wobei der Schwerpunkt auf der Grundsätzlichen Lösung des Problems lag.

- Der Benutzer soll schon nach wenigen Beispielen merklich bei der Umformatierung unterstützt werden.

- Die Anzahl der Rückfragen soll so gering wie möglich gehalten werden.

- Die Rückfragen sollten möglichst einfach gehalten werden.

Im letzten Kapitel wurde ein Verfahren beschrieben, das die gestellte Aufgabe löst. Um das Verfahren zu testen wurden zunächst 80 Literaturhinweise aus den Literaturhinweisen verschiedener Publikationen zum maschinnellen Lernen in drei verschiedenen Formaten formatiert. Verwendet wurden dabei die beiden BibTeX Stile `apalike` und `plain`. Ein drittes Format wurde für die Experimente konstruiert, das sich möglichst stark von den anderen beiden Formaten unterscheiden sollte. Auf diese Literaturlisten wurde das Verfahren angewendet, um die Performanz des Verfahrens zu testen.

Im folgenden werden zunächst einige qualitative Ergebnisse dieser Tests dargestellt. Dort wird vorallem untersucht, wieviele Beispiele nötig sind, um eine gute Genauigkeit bei der Umformatierung zu erreichen. Danach werden einige qualitative Merkmale der Testergebnisse genauer beschrieben.

5.1 Quantitative Ergebnisse

Die Literaturhinweise wurden mit Hilfe des hier entwickelten Verfahrens von jeweils einem der Formate in eines der anderen überführt. Dieses wurde jeweils fünfmal mit einer zufälligen Anordnung der Beispiele durchgeführt. Dabei wurde der Anteil korrekt übersetzter Literaturhinweisen an allen Eingaben berechnet. Dieser Anteil ist die Genauigkeit (Accuracy), mit der die Literaturhinweise umformatiert werden. Die Genauigkeit wurde über alle Durchläufe gemittelt.

Da das in dieser Arbeit vorgestellte Verfahren erst aus den verarbeiteten Literaturhinweisen lernt, ist das Ergebnis für die ersten paar Literaturhinweise sehr schlecht. Wenn man den Fehler über alle verarbeiteten Beispiele berechnet, beeinflussen die ersten Beispiele das Ergebnis sehr stark. Daher wird unter Umständen erst nach sehr vielen Beispielen der Anteil richtig umgeformter Hinweise akzeptabel, obwohl die letzten Beispiele alle richtig umgeformt waren. Bessere Aussagen erhält man hier, wenn man den Fehler nicht über alle Eingaben ermittelt, sondern über ein Fenster der letzten n verarbeiteten Eingaben. Für die hier durchgeführten Experimente wurde ein Fenster der größe 10 gewählt. Abbildung 5.1 zeigt die mittlere Genauigkeit bei den Experimenten in Abhängigkeit von der Anzahl der verarbeiteten Eingaben. Dabei ist auf der Y-Achse der Anteil richtiger Ausgaben unter den letzten zehn Ausgaben aufgetragen.

Wie man in der Abbildung erkennen kann, steigt der Anteil der richtig erkannten Beispiele recht schnell auf 30%. Nach etwa 20 Eingaben ist etwa die Hälfte der Ausgaben korrekt und nach etwa 30 Beispielen werden ca. 60% bis 65% erreicht. Die höchste Genauigkeit innerhalb eines Fensters lag bei etwa 70%. Auch mit zunehmender Zahl verarbeiteter Beispiele wächst die Genauigkeit nicht weiter.

5.2 Qualitative Ergebnisse

Um festzustellen, inwieweit die vorgegebenen Randbedingungen erfüllt wurden, werden nun die Art der Ausgaben im einzelnen betrachtet.

5.2.1 Reformulierung

Wie zu erwarten war stellt die Reformulierung bei dem gestellten Problem die kleinste Hürde dar. Jeder Literaturhinweis, der richtig zerlegt wurde, wurde auch korrekt umformatiert, sofern die bisher gesehenen Beispiele dies zuließen. In den Fällen, in denen die bisher gesehenen Beispiele keine eindeutige Umformatierung

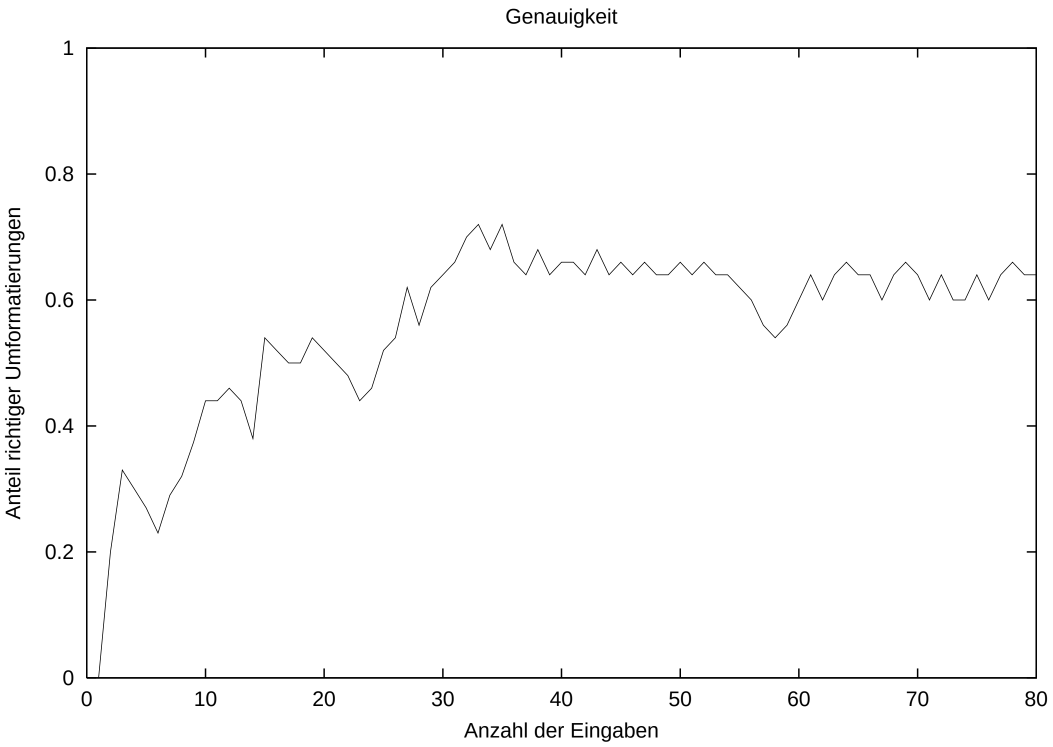

Abbildung 5.1: Mittlere Genauigkeit bei der Umformatierung

nahelegten, wurde zumindest die Reihenfolge der Felder richtig geraten. Die Fehler die hier auftraten lagen in der Regel in falsch eingefügten Trennzeichen. Die Reihenfolge der Felder wurde fast immer korrekt vorhergesagt. Dabei können für Menschen zunächst irritierende Fehler auftreten, die jedoch erklärbar sind. So wurde z.B. bei einem Konferenzbeitrag Titel und Ort durch "pages" getrennt. Einem Menschen fällt dies sofort auf, da er die Bedeutung des Wortes pages und den Zusammenhang mit Seitenzahlen kennt. In diesem Falle waren jedoch auf den Konferenztitel immer die Seitenzahlen und dann erst der Ort gefolgt. Da kein Sachbereichswissen verwendet wird, muß in diesen Fällen also einfach ein Feldtrenner gewählt werden, der durch seine räumliche Beziehung zu den Feldern geeignet scheint. Da "pages" bisher immer auf das Titelfeld folgte, wurde es als Feldtrenner ausgewählt. Weitere Verbesserungen des Verfahrens sollten sich daher auf die Verbesserung der Erkennung der Eingaben konzentrieren.

Man kann weiterhin feststellen, daß auch die nicht korrekt umformatierten Literaturhinweise in vielen Fällen immer noch eine Arbeitserleichterung für den Benutzer darstellen. Viele dieser Ausgaben sind nämlich zumindest teilweise bereits richtig umformatiert. Zum Beispiel werden praktisch immer die Autorenlisten richtig umformatiert. Damit entfällt für den Benutzer zumindest die lästige Umstellung der Autoren.

Was sich bei den Experimenten ebenfalls zeigt, ist die unterschiedliche Schwierigkeit bei der Erkennung der einzelnen Felder. Autorenlisten, Jahreszahlen und Seitenzahlen werden fast immer richtig erkannt. Diese Felder haben eine sehr festgelegte Struktur und vorallem bei den Zahlen einen sehr eingeschränktes Alphabet. Etwas schwieriger sind Orte und Verlage, die in ihrer Struktur etwas stärker variieren können. Andererseits wiederholen sich hier die Einträge oft. Die größten Probleme bereiten die Titelfelder, Zeitschriftentitel und ähnliche Felder.

Das Lernergebnis ist sehr abhängig von der Reihenfolge der Beispiele. Zum Beispiel hängen die gefundenen Felder von der Reihenfolge der Beispiele ab. In zwei untersuchten Formaten werden Titel, Ort und Verlag bei Büchern folgendermaßen formatiert:

Format1:<Titel>. <Ort>: <Verlag>
Format2:<Titel>, <Ort>. <Verlag>

Format1:<Titel>. <Verlag>
Format2:<Titel>. <Verlag>

Sowohl *Titel* und *Ort*, als auch *Ort* und *Verlag* werden also in den beiden Formaten unterschiedlich voneinander getrennt und müssen daher eigene Felder sein. *Titel* und *Verlag* werden jedoch durch dasselbe Zeichen voneinander getrennt und können somit gemeinsam verschoben werden. Welche Felder eingeführt werden hängt nun sowohl von der Eingabereihenfolge, als auch von der Zuverlässigkeit der Analyse ab. Wird zuerst ein Beispiel ohne *Ort* und dann eines mit gegeben, so werden in jedem Falle vier Felder eingeführt. Zusätzlich zu den atomaren Feldern wird noch das Kombinationsfeld *Titel+Verlag* eingeführt. Bei umgekehrter Reihenfolge wird das Beispiel ohne *Ort* häufig richtig zerlegt. In diesem Falle würde kein Kombifeld eingeführt.

Durch bewußte Auswahl der Reihenfolge, in der der Benutzer die Beispiele vorgibt läßt sich die Genauigkeit der Vorhersagen leicht verbessern. Das obige Beispiel zeigt einen Fall, in dem durch die Reihenfolge der Beispiele die Zahl der verwendeten Feldklassen klein gehalten werden kann. Grundsätzlich ist es ebenfalls besser, mit möglichst vollständigen Literaturhinweisen zu beginnen, da Auslassungen in vielen Fällen bei der Analyse toleriert werden.

Bei der Identifizierung der Felder zeigt sich, daß diese Aufgabe auch für den Menschen nicht ganz einfach ist. Somit sind dann auch die Antworten auf die Orakel-Fragen nicht leicht zu beantworten. In vielen Fällen gibt es nicht unbedingt eine richtige Lösung, sondern verschiedene Möglichkeiten. Die Art der Beantwortung dieser Rückfragen kann das Lernergebnis im Zweifelsfall entschieden beeinflussen. So könnte der Benutzer zum Beispiel versucht sein, Autorennamen und Herausgebernamen als eine Klasse anzugeben, da beides Namen sind. In diesem Falle gäbe es aber bei der Reformulierung keine Möglichkeit, zwischen Autoren und Herausgebern zu unterscheiden und diese Felder richtig anzuordnen. Die richtige Beantwortung der Nachfragen erfordert also doch einen gewissen Einblick in das Vorgehen des Verfahrens.

Ein Problem sind die Rückfragen an den Benutzer. Letztendlich müssen auch richtig übersetzte Beispiele vom Benutzer kontrolliert werden. In einigen wenigen Fällen kam es bei den Experimenten vor, daß bei richtig umformatierten Literaturhinweisen einzelne Felder falsch zugeordnet wurden. Diese Beispiele müssen korrigiert werden, da sonst aus falschen Beispielen gelernt wird. Allerdings muß der Benutzer nur bei falsch erkannten Eingaben eingreifen und eine Korrektur vornehmen.

Kapitel 6

Fazit und Ausblick

Die Umformatierung von Literaturhinweisen ist, wie sich gezeigt hat, trotz ihrer scheinbaren Einfachheit tatsächlich eine sehr komplexe Aufgabe, die selbst ein Mensch nicht immer hundertprozentig lösen kann. Um diese Aufgabe lösen zu können wurde sie hier in Teilaufgaben zerlegt, durch die das Problem Handhabbarer wurde. Die erste getroffene Aufteilung war die Trennung zwischen der Zerlegung der Eingabe und der Generierung der Ausgabe aus dieser Zerlegung. Für die Analyse der Eingabe müssen widerum zwei Probleme gelöst werden: Es muß eine Struktur gefunden werden, die eine Umformatierung der Eingabe ermöglicht. Dann muß ein Modell dieser Struktur gefunden werden, mit dessen Hilfe eine Strukturbeschreibung einer Eingabe ermittelt werden kann. Für die Ausgabe muß ein Modell der Ausgabesprache gefunden werden, mit dessen Hilfe aus einer Strukturbeschreibung der Eingabe die korrekte Ausgabe erzeugt werden kann. Das Modell für die Eingabesprache wurde in zwei Ebenen aufgeteilt. Die äußere Ebene, die die Anordnung der für die Umformatierung wichtigen Elemente beschreibt und die innere Ebene, die diese Elemente selbst beschreibt. Für die einzelnen Teilprobleme wurden dann Lösungen vorgeschlagen.

Mithilfe des so entwickelten Verfahrens ist es Möglich, aus Beispielen die Umformatierung von Literaturhinweisen zu lernen, ohne Domänenwissen vorauszusetzen. In den Experimenten hat sich gezeigt, daß das Verfahren tatsächlich schon nach wenigen Beispielen eine Unterstützung des Benutzers ermöglicht. Die Zuverlässigkeit der Umformatierungen läßt allerdings noch zu Wünschen übrig. Wie weiter oben schon festgestellt wurde sind einige Gründe für die Fehler, die das Verfahren macht im Bereich der Literaturhinweise kaum zu vermeiden. Eine Ursache für falsche Umformatierungen ist, daß selbst nach einer größeren Anzahl von Beispielen noch unbekannte Strukturen auftreten können. Außerdem sind einige Literaturhinweise auch für Menschlichen Benutzern nicht immer eindeutig Strukturierbar. Zum Beispiel kann die Entscheidung, ob es sich bei einer gegebenen Zeichenkette um einen Buchtitel mit Untertitel, oder einen Konferenzbeitrag mit Konferenztitel handelt ist manchmal schwierig. Absolute Genauigkeit kann also von einem maschinellen Verfahren auch nicht erwartet werden.

Für einige der Teilkomponenten lassen sich jedoch vermutlich noch bessere Lösungen finden. Vor allem die Modellierung der Felder kann vermutlich noch verbessert werden. Durch die vorgenommene Aufteilung des Problems hat man hier den Vorteil, daß man die Methoden für die einzelnen Teilkomponenten ändern oder austauschen kann, ohne sich unbedingt um die anderen Komponenten kümmern zu müssen.

6.1 Mögliche Erweiterungen des Verfahrens

Wie in Kapitel 2 bereits angemerkt, habe ich mich bei dem Entwurf dieses Systems auf Umformatierungen beschränkt, die allein durch Verschieben der Sinntragenden Elemente erreicht werden können. Ein- und Ausgabeformat müssen also dieselben Informationen zu den einzelnen Publikationen vorsehen. Die Welt der Literaturhinweise ist leider jedoch erheblich komplizierter.

Ein Problem, das nicht unbedingt mit der Umformatierung zusammenhängt, ist die Ergänzung unvollständiger Informationen in der Eingabe. Dies ist auch von einem Menschen nur zu bewältigen, wenn er eine andere Quelle für diese Publikation hat, oder aber durch sein umfangreiches Sachbereichswissen. So kann ein Mensch z.B. den Verlagsort eines Buches in vielen Fällen ergänzen, wenn er den Verlag kennt.

Ein anderer Fall sind Transformationen einzelner Felder. Ein typisches Beispiel hierfür sind Abkürzungen. Soll zum Beispiel der Vorname eines Autors im Zielformat grundsätzlich durch seine Initialen angegeben werden, so muß man lernen, wie aus den vollständigen Namen Initialen gemacht werden. Noch schwieriger ist der umgekehrte Weg, aus Initialen wieder Vornamen zu machen. Dies ist wiederum nur mit umfangreichem Sachbereichswissen möglich. So weiß ich z.B., daß bei “D. Angluin” das “D.” vermutlich “Dana” bedeutet. Bei “D. Aha” würde ich jedoch auf “David” tippen. Bei sehr häufigen Nachnamen, müssen unter Umständen noch Hinweise aus dem Titel verwendet werden. Wie weiter oben schon angedeutet, kann man dies als eigene neue Lernaufgabe auffassen, die nach der Zerlegung und der Identifizierung der Felder ansetzt. Daher habe ich diese Fälle in meiner Arbeit ausgespart. Hier will ich kurz vorschlagen, wie eine entsprechende Erweiterung aussehen könnte. Die transformierten Teile der Felder würden in meinem Verfahren nach der Zerlegung als Feldtrenner erkannt werden. Um diese Transformationen zu entdecken könnte man die Feldtrenner nach verdächtigen Zeichen (Ziffern und Buchstaben) durchsuchen.. Solche Feldtrenner sind dann potentielle Kandidaten für Transformationen, für die dann Transformationsregeln gelernt werden müssen. Hier kann dann wieder der Benutzer als Orakel zu Rate gezogen werden.

Ein Ansatz, wissensgestützte Ergänzungen und Transformationen zu behandeln, wird in [Parmentier und Belaid, 1997] vorgestellt. Hier wird aus großen Mengen gelabelter Beispiele eine Art semantisches Netz für die behandelten Objekte aufgebaut.

Einfacher in das vorgestellte Verfahren zu integrieren sind Auslassungen. Dabei könnte das Ausgabebeispiel durch ein eindeutiges Zeichen, das nicht aus dem Alphabet der Literaturhinweise besteht, in zwei Teile aufgeteilt werden. Der erste Teil ist die korrekte Ausgabe und der zweite Teil ist eine Art "Mülleimer", der die nicht verwendeten Komponenten der Eingabe enthält und bei der Ausgabe nicht gedruckt wird.

Das Ziel dieser Arbeit war, ein Verfahren zu entwickeln, daß während der Benutzung durch den Benutzer lernt und möglichst kein Vorwissen benötigt. Nach jedem Literaturhinweis, den ein Benutzer mit Hilfe des Systems umformatiert, wird dieser Literaturhinweis wieder als Beispiel verwendet. Ein Benutzer will jedoch möglichst zügig mit dem System arbeiten können. Zu lange Wartezeiten zwischen zwei Eingaben werden von einem Benutzer nicht toleriert. Daher habe ich mich hier auf Lernverfahren konzentriert, die die Beispiele möglichst schnell verarbeiten können.

Um die Performanz beim Lernen zu verbessern wäre es sinnvoll, diese Online-Komponente um eine Offline-Komponente zu erweitern. Dabei könnten die während der Benutzung eingegebenen Beispiele später noch einmal verwendet werden, um eine bessere Repräsentation zu lernen. Dies betrifft vor allem die Beschreibung der Felder, für die hier eher einfache Verfahren angewandt wurden. Denkbar wäre, hier z.B. mit einem ILP Lernverfahren eine exaktere Beschreibung der Felder zu erhalten. Bei einigen anfänglichen Experimenten mit diesen Verfahren ergaben sich jedoch Laufzeiten von mehreren Minuten bis Stunden. Das wesentliche Problem dabei ist das Fehlen negativer Beispiele. Daher kommen hier nur Verfahren in Frage, die wie Foil oder Progol in der Lage sind, auch ohne negative Beispiele zu lernen. Eine andere Klasse von Verfahren, die zu diesem Zweck geeignet sein könnten, sind Verfahren, die charakteristische Sequenzen in den Feldern finden.

Literaturverzeichnis

[Adriaans und Knobbe, 1996] Adriaans, P. und Knobbe, A. J. (1996). EMILE: Learning Context-free Grammars from Examples. unpublished.

[Agrawal und Srikant, 1995] Agrawal, R. und Srikant, R. (1995). Mining sequential patterns. In *International Conference on Data engineering*.

[Ahonen-Myka, 1999] Ahonen-Myka, H. (1999). Finding all maximal frequent sequences in text. In *Proceedings of the ICML-99 Workshop on Machine Learning in Text Data Analysis*, Seiten 11–17.

[Angluin, 1980] Angluin, D. (1980). Finding patterns common to a set of strings. *Journal of Computer and System Science*, 21:46–62.

[Angluin, 1982] Angluin, D. (1982). Inference of reversible languages. *Journal of the Association for Computing Machinery*, 29:741–765.

[Angluin, 1987] Angluin, D. (1987). Learning regular sets from queries and counter-examples. *Information and Computation*, (75):87–106.

[Charniak, 1993] Charniak, E. (1993). *Statistical Language Learning*. MIT Press, Cambridge, MA.

[Craven et al., 1998a] Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K. und Slattery, S. (1998). Learning to Extract Symbolic Knowledge from the World Wide Web. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*.

[Craven et al., 1998b] Craven, M., Slattery, S. und Nigam, K. (1998). First-Order Learning for Web Mining. In *Proceedings of the 10th European Conference on Machine Learning*, Seiten 250–255.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A Note on Two Problems in Connexion With Graphs. *Numerische Mathematik*, (1):269–271.

[Freitag, 1998] Freitag, D. (1998). Information Extraction From HTML: Application of a General Learning Approach. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*.

[Gold, 1967] Gold, E. M. (1967). Language Identification in the Limit. *Information and Control*, 14:447–474.

[Gold, 1978] Gold, E. M. (1978). Complexity of Automaton Identification from Given Data. *Information and Control*, 37:302–320.

[Hermens und Schlimmer, 1993] Hermens, L. A. und Schlimmer, J. C. (1993). A machine-learning apprentice for the completion of repetitive forms. In *Proceedings of the Ninth IEEE Conference on Artificial Intelligence*, Seiten 96–103, Orlando.

[Hopcroft und Ullman, 1979] Hopcroft, J. E. und Ullman, J. D. (1979). *Introduction to automata theory, languages and computation*. Addison Wesley, New York [u. a.].

[Langley, 1987] Langley, P. (1987). Machine Learning and Grammar Induction. *Machine Learning*, 2:5–8.

[Linke et al., 1991] Linke, A., Nussbaumer, M. und Portmann, P. R. (1991). *Studienbuch Linguistik*. Niemeyer, Tübingen.

[Morik, 1995] Morik, K. (1995). Maschinelles Lernen. Vorlesungsskript.

[Muggleton, 1995a] Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, (13):245–286.

[Muggleton, 1995b] Muggleton, S. (1995). Inverting entailment and Progol. In Furukawa, K., Michie, D. und Muggleton, S., Herausgeber, *Machine Intelligence*, Ausgabe 14. Oxford University Press.

[Muggleton, 1996a] Muggleton, S. (1996). Experimental acquisition of grammar from early reader books. Technical Report PRG-TR-18-96, Oxford University Computing Laboratory.

[Muggleton, 1996b] Muggleton, S. (1996). Stochastic logic programs. In de Raedt, L., Herausgeber, *Advances in Inductive Logic Programming*, Seiten 254–264. IOS Press.

[Nix, 1985] Nix, R. P. (1985). Editing by Example. *ACM Transactions on Programming Languages and Systems*, 7(4):600–621.

[Parmentier und Belaid, 1997] Parmentier, F. und Belaid, A. (1997). Logical Structure Recognition of Scientific Bibliographic References. In *Proceedings of the ICDAR '97*.

[Quinlan, 1990] Quinlan, J. (1990). Learning Logical Definitions from Relations. *Machine Learning*, (5):239–266.

[Schirra et al., 1985] Schirra, J., Brach, U., Wahlster, W. und Woll, W. (1985). WILIE – Ein wissensbasiertes Literaturerfassungssystem. In Endres-Niggemeyer, B. und Krause, J., Herausgeber, *Sprachverarbeitung in Information und Dokumentation*, Seiten 101–112. Springer, Berlin.

[Witten et al., 1999] Witten, I. H., Bray, Z., Mahoui, M. und Teahan, W. J. (1999). Using language models for generic entity extraction. In *Proceedings of the ICML-99 Workshop on Machine Learning in Text Data Analysis*, Seiten 25–35.

[Witten et al., 1996] Witten, I. H., Nevill-Manning, C. G. und Maulsby, D. (1996). Interacting with learning agents: Implications for ML from HCI. In *ICML '96 Workshop on Machine Learning meets Human Computer Interaction*, Bari, Italy.