Schema und Schema Integration

Thosten Dollmann Dennis Schade Carsten Karl

30. Juni 2003

Inhaltsverzeichnis

1 Überblick **4**

2 Einführung in XTRACT **4**

2.1 Motivation . . . 4
2.2 Problembeschreibung . . . 4
2.3 Ein einführendes Beispiel . . . 4
2.4 Was ist eine "gute" DTD? . . . 5
2.5 Das MDL Prinzip . . . 5

3 XTRACT Learning DTDs from XML-document collections **6**

3.1 Schematischer Überblick . . . 6
3.2 Das Generalisierungsmodul . . . 7
3.2.1 DiscoverSeqPattern . . . 7
3.2.2 DiscoverOrPattern . . . 8
3.3 Das Faktorisierungsmodul . . . 8
3.3.1 Auswahl geeigneter Teilmengen . . . 9
3.3.2 Der Faktorisierungsalgorithmus . . . 10
3.4 Das MDL Modul . . . 10
3.4.1 Bitkodierung . . . 10
3.4.2 Berechnung der DTD mit minimalen MDL-Kosten . . . 11
3.5 Experimentelle Validierung . . . 12
3.6 Schlussfolgerung . . . 13

4 Inkrementelle Validierung von XML-Dokumenten **14**

4.1 Motivation . . . 14
4.2 Grundlagen . . . 15
4.3 Inkrementelle Validierung von Strings . . . 16
4.4 Inkrementelle DTD-Validierung . . . 18
4.5 Inkrementelle XML Schema-Validierung . . . 19
4.6 Schlussfolgerung . . . 20

1 Überblick

Diese Ausarbeitung behandelt das Thema *Schema und Schema Integration*. Es basiert auf den Papern XTRACT (LITERATUR) und Incremental Valiation (LITERATUR). Wir stellen mit XTRACT ein Werkzeug vor, mit dem es möglich ist aus einer Menge von XML-Beispieldokumenten, zu denen es noch keine DTD gibt, eine möglichst intuitive und präzise DTD abzuleiten. Zusätzlich stellen wir Methoden vor, um ein XML-Dokument, an dem Veränderungen durchgeführt wurden, effizient auf seine Gültigkeit bezüglich seiner DTD bzw. seines Schemas zu überprüfen.

2 Einführung in XTRACT

2.1 Motivation

Durch die weite Verbreitung von XML werden schon vorhandene Datenbestände mit einfachen Mitteln automatisch zu XML Dokumenten konvertiert. Zu vielen dieser Dokumente existiert keine DTD. Da man DTDs zum Beispiel zur Optimierung von XML-Queries benötigt, ist es sinnvoll aus einer gegeben Anzahl verschiedener XML-Dokumente, automatisch eine DTD zu generieren.

2.2 Problembeschreibung

Gegeben sei eine Menge von XML-Dokumenten. Ziel ist es, eine möglichst akurate, kompakte und aussagekräftige DTD für jedes Tag e aus diesen Dokumenten zu erzeugen. Sei I die Menge aller Sequenzen von Tags, die innerhalb des Tags e vorkommen. Für jedes Vorkommen von e wird eine solche Sequenz zu I hinzugefügt. Gegeben sei die Menge I aller im Tag e geschachelten Beispielsequenzen. Berechne eine DTD D für das Tag e, so dass jede Beispielsequenz S aus I konform zur DTD D ist.

2.3 Ein einführendes Beispiel

Betrachten wir zunächst folgendes XML-Dokument:

```
<book>
   <title> </title>
   <author>
      <name> </name>
      <organisation> </organisation>
   </author>
   <author>
      <name> </name>
   </author>
   <editor>
      <name> </name>
   </editor>
</book>
```

Zu jedem in diesem Dokument vorkommenden Tag e wird eine Menge I aus Beispielsequenzen erstellt. Eine solche Beispielsequenz enthält alle Kinder des Tags e. Aus dem obigen Beispiel ergeben sich somit folgende Mengen I von Beispielsequenzen,

```
<book>   =  { <title><author><author><editor> }
<author> =  { <name><organisation>, <name> }
<editor> =  { <name> }
```

wobei auf die Angabe von leeren Mengen für Tags wie `<title>` verzichtet wurde.

XTRACT versucht für jede dieser Mengen I, d.h. separat für jedes einzelne Tag, eine "gute" DTD zu berechnen.

2.4 Was ist eine "gute" DTD?

Sei $I = \{ab, abab, ababab\}$ eine Menge von Sequenzen zu einem Tag. DTDs, die diese Sequenzen korrekt beschreiben sind beispielsweise:

1. (a|b)*
2. (ab|abab|ababab)
3. (ab)*
4. ab|ab(ab|abab)

Die DTDs (1) und (3) sind jeweils intuitiv und leicht verständlich, jedoch sind sie zu generell, d.h. sie beschreiben eine wesentlich grössere als die gegebene Sprache I. Die DTDs (2) und (4) definieren zwar genau die Sprache I, jedoch sind sie im Gegensatz zu (1) und (3) recht unintuitiv und wirken durch ihre Grösse sehr komplex.
Zusammenfassend können wir sagen, dass eine *gute* DTD D die beiden folgenden Anforderungen so gut wie möglich erfüllen sollte:

R1: D sollte übersichtlich sein (d.h. sie sollte nicht zu lang sein)

R2: D sollte präzise sein (d.h. sie sollte nicht zu viele nicht in I enthaltene Sequenzen abdecken)

Offenbar sind die beiden Restriktionen R1 und R2 komplementär. Um nun einen möglichst guten Trade-Off zwischen R1 und R2 zu erhalten, bedient man sich des Prinzips der *minimum description length (MDL)*.

2.5 Das MDL Prinzip

Das MDL-Prinzip besagt, dass die bestmögliche Theorie, welche eine Menge von Daten beschreibt, diejenige ist, die die Summe aus

1. der Länge der Theorie kodiert in bits und
2. der Länge der mit der Theorie kodierten Daten in bits

minimiert. Näheres dazu ist in [Ris78] zu finden.

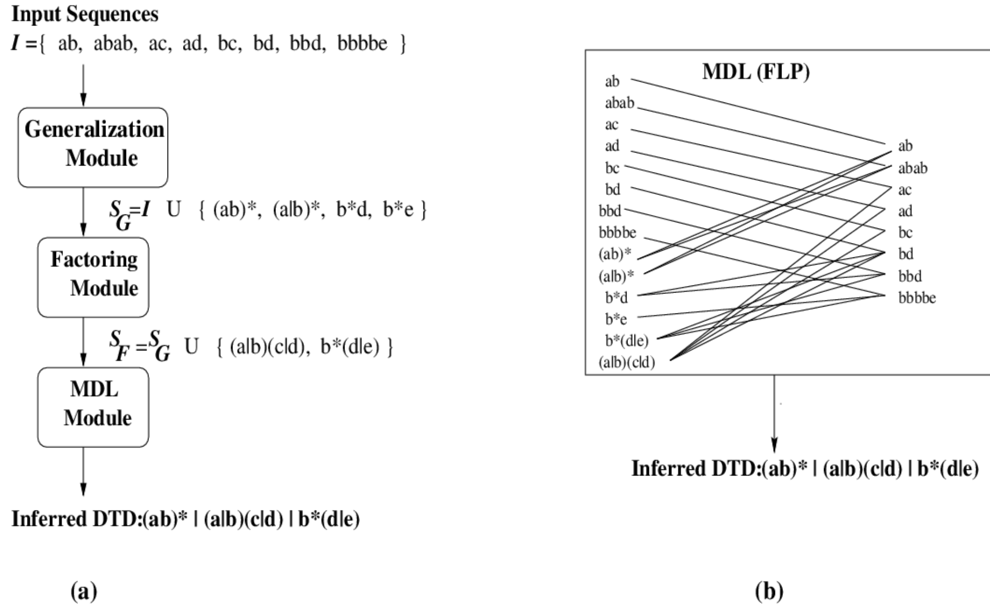

Abbildung 1: XTRACT: Schematischer Aufbau

3 XTRACT Learning DTDs from XML-document collections

3.1 Schematischer Überblick

Das XTRACT-System ist in drei Module aufgeteilt (Abbildung 1 (a))

- Das Generalisierungsmodul erzeugt aus der Menge der Beispielsequenzen I eine Menge von Kandidaten DTDs, indem es für jede Beispielsequenz aus I verallgemeinernde reguläre Ausdrücke einführt. S_G ist die Menge I vereint mit der Menge dieser regulären Ausdrücke.

- Das Faktorisierungsmodul erzeugt aus der Menge der Kandidaten DTDs S_G kompaktere DTDs, indem es verschiedene Kandidaten DTDs mit gemeinsamen Präfixen oder Suffixen miteinander verbindet. So kann zum Beispiel aus den DTDs ab, ac die neue DTD $a(b|c)$ erzeugt werden. S_F ist die Menge der so erzeugten DTDs vereinigt mit der Menge S_G.

- Das MDL-Modul berechnet aus der Menge S_F der Kandidaten DTDs eine DTD D, die alle Beispielsequenzen in I mit minimalen MDL-Kosten abdeckt. Diese DTD D ist eine VerODERung aller Elemente einer der Teilmengen von S_F, die alle Beispielsequenzen in I mit minimalen MDL-Kosten abgedeckten. (siehe Abbildung 1 (b))

3.2 Das Generalisierungsmodul

Die Qualität der finalen DTD, die das MDL-Modul errechnet, hängt sehr stark von der durch das Faktorisierungsmodul generierten Kandidatenmenge S_F ab. Würde S_F nur die Sequenzen aus I enthalten, dann wäre die finale DTD, die das MDL Modul errechnen würde, ein einfaches ODER über alle Beispielsequenzen aus I. Eine solche DTD wäre jedoch nicht erstrebenswert, da sie, wie wir oben schon gesehen haben, weder präzise noch übersichtlich ist. Aus diesem Grund ist es entscheidend, dass die Kandidaten DTDs in S_F möglichst allgemein sind. Das Ziel des Generalisierungsmoduls ist es, eine Menge S_G von allgemeinen DTDs zu erzeugen. S_G dient dann als Eingabe des Faktorisierungsmoduls. Das Generalisierungsmodul in XTRACT leitet reguläre Ausdrücke aus jeder Sequenz aus I her. [HU79] Insbesondere werden zwei Sprachmerkmale erkannt:

- Kleenesterne, z.B. kann ababab durch (ab)* ersetzt werden
- VerODERungen, z.B. kann abac durch (a(b|c))* ersetzt werden

Das Generalisierungsmodul findet diese Merkmale mit Hilfe der Funktionen $DiscoverSeqPattern$ und $DiscoverOrPattern$. Diese Algorithmen können aber nicht alle möglichen Kombinationen aller regulären Sprachmerkmale finden, z.B. kann ein ?, das in einem Kleenestern geschachtelt ist (ab?c)*, durch den benutzten Algorithmus nicht gefunden werden. Der Verallgemeinerungsalgorithmus leitet für jede Beispielsequenz in I mehrere Kandidaten DTDs her und fügt sie zu S_G hinzu. Da jeweils nur eine Sequenz als Grundlage für die Herleitung genutzt wird, werden auch zu allgemeine Kandidaten DTDs generiert. Diese Kandidaten DTDs werden aber nicht durch das MDL-Modul als Teil der endgültigen DTD ausgewählt.

3.2.1 DiscoverSeqPattern

Die Prozedur $DiscoverSeqPattern$ nimmt als Eingabe eine Beispielsequenz s aus I und gibt eine Kandidaten DTD D zurück. Diese Kandidaten DTD D entsteht durch das Ersetzen von sich wiederholenden Mustern der Form $xxxx\cdots x$ durch den regulären Ausdruck $x*$. Neben der Sequenz s wird noch ein weiterer Parameter r an $DiscoverSeqPattern$ übergeben. Dieses r bestimmt die minimale Anzahl kontinuierlicher Wiederholungen der Teilsequenz x, damit sie durch $x*$ ersetzt wird. Falls mehrere Teilsequenzen diesen Schwellwert überschreiten, dann wird die längste Teilsequenz ausgewählt. Dabei wird aber, anstatt den regulären Ausdruck in die Beispielsequenz einzufügen, ein Hilfssymbol $A_i = x*$ eingefügt. Zwischen einem Hilfssymbol und einem regulären Ausdruck besteht eine Eins-zu-Eins Beziehung. Zum Beispiel ergibt $DiscoverSeqPattern$ mit den Parametern s=*abababcababc* und r=2 nach der ersten Iteration die Sequenz A_1cA_1c, wobei A_1 ein Hilfssymbol für den regulären Ausdruck $(ab)*$ ist. Nach der zweiten Iteration liefert die Prozedur nur noch das Hilfssymbol A_2, mit $A_2 = ((ab)*c)*$. Aus diesem Grund können Kleenesterne innerhalb anderer Kleenesterne geschachelt werden. Die Laufzeit der Prozedur wird durch das Auffinden der Teilsequenz x mit der maximalen Anzahl von kontinuierlichen

Wiederholungen dominiert. Da die Sequenz s maximal $O(|s|^2)$ Teilsequenzen enthält und das Errechnen der Anzahl der Wiederholungen für jede Teilsequenz $O(|s|)$ Schritte kostet, ist die Laufzeit pro Iteration im schlimmsten Fall $O(|s|^3)$. Weitere Details zu den Grundlagen des Algorithmus sind in [Bra93] beschrieben.

3.2.2 DiscoverOrPattern

Die Prozedur $DiscoverOrPattern$ erzeugt Muster der Form $(a_1|a_2|a_3 \cdots |a_m)*$ basierend auf dem gehäuften Vorkommen dieser Zeichen innerhalb eines Teilbereichs einer Sequenz s. Solche Teilbereiche werden durch das Partionieren der Eingabesequenz s in kleinst mögliche Teilsequenzen $s_1, s_2, s_3, ..., s_n$ erreicht. Für jede dieser Teilsequenzen gilt, dass für jedes Vorkommen eines Symbols a in einer Sequenz s_i, a nicht innerhalb eines vorgebenen Abstands d in einer anderen Teilsequenz s_j vorkommt. Das Erstellen solcher Teilsequenzen übernimmt die Prozedur $Partition$. Jede dieser Teilsequenzen s_i aus s wird anschliessend durch den regulären Ausdruck $(a_1|a_2|...|a_m)*$ ersetzt, wobei $a_1, ..., a_m$ die Menge der verschiedenen in s_i vorkommenden Zeichen ist. Diese können wiederum Hilfssymbole sein, die reguläre Ausdrücke repräsentieren. Betrachten wir zum Beispiel s=$abcbac$, dann liefert $DiscoverOrPattern$ mit verschiedenen Distanzparametern d folgende Muster:

- aA_1ac mit $d = 2$ und $A_1 = (b|c)*$,
- aA_2 mit $d = 3$ und $A_2 = (a|b|c)*$ und
- A_2 mit $d = 4$

Die Prozedur $Partition$ ist ein wichtiger Teil von $DiscoverOrPattern$, weshalb wir sie nun nähern erläutern werden. Angenommen s_1 bis s_j wären schon erzeugt worden, dann würde s_{j+1} direkt nach dem Ende von s_j beginnen, und würde so weit nach rechts erweitert, bis gewährleistet ist, dass kein Symbol, das in s_j vorkommt, innerhalb der Distanz d rechts vom Ende von s_{j+1} auftaucht. Diese kann in $O(|s|)$ getestet werden, wenn man initial für jedes Symbol aus s eine sortierte Liste seines Vorkommens in s anlegt. Deswegen ist die Laufzeit von $DiscoverOrPattern$ $O(|s|^2)$.

3.3 Das Faktorisierungsmodul

Das Faktorisierungsmodul erzeugt zu geeigneten Teilmengen S aus S_G zusätzliche Kandidaten DTDs, indem es Teilausdrücke faktorisiert. So erzeugt es z.B. aus der Kandidatenmenge $S = \{ac, ad, bc, bd\}$ eine DTD der Form $(a|b)(c|d)$, welche wiederum zur Kandidatenmenge S_F hinzugefügt wird. Das Faktorisieren erweitert also die Menge der Kandidaten DTDs um neue zusätzliche DTDs, deren MDL-Kodierungskosten geringer sind als die Summe der MDL-Kosten der DTDs aus denen sie hervorgegangen sind. Bei geeigneter Wahl besitzt die faktorisierte DTD neben geringerer MDL-Kodierungskosten auch noch eine höhere Abdeckung der Beispielsequenzen aus I. Der Vollständigkeit halber müssten alle

Teilmengen von S_G faktorisiert werden, was aber offentsichtlich nicht praktikabel ist, da die Anzahl $|\mathcal{P}|$ der Teilmengen exponentiell zur Anzahl der Elemente in S_G wächst ($|\mathcal{P}| = 2^{|S_G|}$). Aus diesen Gründen verwendet man im Faktorisierungsmodul Heuristiken zur Auswahl geeigneter Teilmengen $S \subset S_G$.

3.3.1 Auswahl geeigneter Teilmengen

Ziel der Heuristik ist es, die Auswahl der zu faktorisierenden Kandidaten DTDs so zu gestalten, dass die resultierenden Kandidaten DTDs geringe Kosten im Sinn des MDL-Prinzips besitzen und die auszuschliessen, die durch ihre geringere Abdeckung von Beispielsequenzen im MDL-Modul mit hoher Wahrscheinlichkeit verworfen werden. Die Heuristik des Faktorisierungsmoduls sollte also diejenigen Teilmengen auswählen, die sich besonders gut faktorisieren lassen und mit die zusätzlich eine hohe Abdeckung der Beispielsequenzen in I besitzen. Deswegen sollte eine Teilmenge $S \subset S_G$, aus der man gute DTDs generieren kann, die beiden folgenden Eigenschaften erfüllen:

1. Jede DTD $D \in \mathcal{S}$ besitzt ein gemeinsames Präfix oder Suffix mit einer möglichst grossen Anzahl anderer DTDs $D' \in \mathcal{S}$. Je mehr DTDs in $\mathcal{S}$ ein gemeinsames Präfix oder Suffix besitzen und je grösser die Länge der gemeinsamen Präfixe oder Suffixe ist, desto höher wird die Wahrscheinlichkeit, dass die Qualität im Sinne des MDL Prinzips der resultierenden faktorisierten DTDs ansteigt.

2. Die Überlappung zwischen jedem Paar DTDs D, D' aus $\mathcal{S}$ ist minimal. Unter Überlappung versteht man die Anzahl der Beispielsequenzen, die sowohl von D als auch D' abgedeckt werden. Geringe Überlappung ist wichtig, da eine faktorisierte DTD, mit der man nicht signifikant mehr Beispielsequenzen abdecken kann als durch die DTDs D und D', aus denen sie enstanden ist, aus MDL Sicht fast wertlos ist.

Diese Eigenschaften lassen sich folgendermassen formalisieren:

Abdeckung $cover(D)$ ist die Menge der Inputsequenzen aus I, die Wörter der durch $L(G)$ definierten regulären Sprache sind.

Überlappung $overlap(D, D')$ ist die Anzahl der Inputsequenzen, die sowohl von D und als auch von D' abgedeckt werden.

$$overlap(D, D') = |\frac{cover(D) \cap cover(D')}{cover(D) \cup cover(D')}|$$

Um die erste Eigenschaft formal zu beschreiben, werden wir $score(D, \mathcal{S})$, ein Mass zur Beschreibung der Güte der Präfixe und Suffixe, einführen. Hierzu ist jedoch einige Vorarbeit nötig.
Für eine DTD D sei $pref(D)$ und $suf(D)$ jeweils die Menge der Präfixe bzw. Suffixe von D. Sei $psup(p, \mathcal{S})$ die Anzahl der DTDs aus $\mathcal{S}$ mit den Präfix p und sei $ssup(s, \mathcal{S})$ die Anzahl der DTDs aus $\mathcal{S}$ mit dem Suffix s. Damit kann nun das Mass $score(D, \mathcal{S})$ folgendermassen definiert werden:

$$score(D, \mathcal{S}) = max(\{|p| \cdot psup(p, S) : p \in pref(D)\} \cup \{|s| \cdot ssup(s, S) : s \in suf(D)\})$$

Ein langes Präfix oder Suffix einer DTD D, das häufig in der Menge der Inputsequenzen $\mathcal{S}$ vorkommt, liefert eine gute Faktorisierung, die viele Inputsequenzen abdeckt.
Die Laufzeit der Heuristik beträgt $O(N^2 \cdot (N + L))$, wobei $N = |I|$ und L die maximale Länge einer Sequenz aus I ist.

3.3.2 Der Faktorisierungsalgorithmus

Der von XTRACT verwendete Alorithmus zur Faktorisierung von geeigneten Präfixen oder Suffixen basiert auf einem Algorithmus zur Optimierung von Boolschen Ausdrücken von Wang [Wan89] Da die Eigenschaften von regulären Ausdrücken aber nicht Eins-zu-Eins mit den Eigenschaften von Boolschen Ausdrücken übereinstimmen, mussten einige hier nicht näher erläuterte Modifikationen vorgenommen werden. Da dieser Algorithmus ein NP-hartes Problem lösen muss, ist dies ein zusätzlicher Grund für die Anwendung der oben schon genannten Heuristiken zur Auswahl geeigneter Teilmengen von S_G. [BM82]

3.4 Das MDL Modul

Das MDL-Modul bildet das Herzstück von XTRACT. Es wählt aus der durch das Generalisierungs- und Faktorisierungsmodul erzeugten Kandidatenmenge S_F eine Teilmenge von DTDs aus, die

1. alle Sequenzen aus der Menge der Beispielsequenzen I umfasst und
2. minimale MDL-Kosten hat.

3.4.1 Bitkodierung

Um das MDL-Prinzip anwenden zu können, muss sowohl die Kodierung einer Theorie (DTD), die Länge einer kodierten Theorie, als auch die Kodierung einer Datensequenz und die Länge der Kodierung der Datensequenz definiert werden.

Kodierung einer DTD Sei Σ die Menge der in einer DTD vorkommenden Tags e, $\mathcal{M}$ die Menge der Metazeichen |,*,+,?,(,). Die Länge n einer DTD ist die Anzahl der Zeichen aus $\Sigma \cup \mathcal{M}$, die benötig werden um die DTD zu beschreiben. Die Länge von D in bits ist dann:

$$n \log(|\Sigma| + |\mathcal{M}|)$$

Als Beispiel betrachten wir die DTD a*b*. Die Länge dieser DTD ist

$$4 \log(2 + 6) = 12$$

Neben der Kodierung der DTD müssen die Daten mit dieser Theorie kodiert werden.

DTD	Datenkodierung	Gesamtlänge
ab|abb|abbb|abbbb	0,1,2,3	17+7=24
(a|b)*	201, 3011, 44111, 501111	6+21=27
ab*	1, 2, 3, 4	3+7=10

Tabelle 1: Kodierung und Kosten der Sequenzen aus I={ ab,abb,abbb,abbbb }

Kodierung einer Sequenz Im folgenden Beispiel soll die prinzipielle Arbeitsweise illustriert werden:

1. Die Kodierung der Sequenz a mit Hilfe der DTD a ist ϵ und hat die Kodierungskosten 0.

2. Die Kodierung der Sequenz c mit Hilfe der DTD $a|b|c$ ist 2 und hat die Kodierungskosten 1. Die 2 stellt den zu wählenden Index in der VerODERung $a|b|c$ dar.

3. Die Kodierung der Sequenz ccc mit DTD c^* ist 3 und hat die Kodierungskosten 1. Die 3 gibt die Anzahl der Wiederholungen des Kleenesterns an.

Auf diesen einfachen Vorgehensweisen basiert die Kodierung komplexerer DTDs. Zum Beispiel können $ababcabc$ beschrieben durch die DTD $((ab)*c)*$ als 2,2,1 kodiert werden.

Betrachten wir mit Hilfe dieser Kodierung wieder die Beispielsequenzen I=$\{ab, abb, abbb, abbbb\}$. Mögliche Kodierungen und die zugehörigen Kosten sind in der Tabelle 1 aufgeführt.

3.4.2 Berechnung der DTD mit minimalen MDL-Kosten

Im Folgenden wenden wir uns dem Problem der Berechnung der finalen DTD D zu. Sie ist eine VerODERung einer Teilmenge S aus S_F. D muss folgenden Anforderungen genügen:

- D deckt alle Beispielsequenzen in I ab
- D hat minimale MDL-Kosten

Dieses Minimierungsproblem ist eine Instanz des *Facility Location Problems(FLP)*. Das FLP ist wie folgt definiert: Sei C die Menge von Kunden und J eine Menge von Anlagen, die die Kunden i bedienen kann. Es gibt Kosten $c(j)$ für das Auswählen einer Anlage $j \in J$ und Kosten $d(i,j)$, wenn Kunde $i \in C$ von Anlage $j \in J$ bedient wird. Nun soll eine Teilmenge $F \subset J$ gewählt werden, die die Summe aus beiden Kosten bei Bedienung aller Kunden minimiert:

$$\min_{F \subset J}\{\sum_{j \in F} c(j) + \sum_{i \in C} \min_{j \in F} d(j,i)\}$$

Das Problem eine DTD mit minimalen MDL Kosten herzuleiten kann folgendermassen auf das FLP reduziert werden: Sei C die Menge der Beispielsequenzen I und sei J die Menge der Kandidaten DTDs S_F. Die Kosten der Auswahl

einer Anlage seien die Kodierungskosten der zugehörigen DTD. Die Kosten, wenn ein Kunde i von einer Anlage j, $d(j,i)$ bedient wird, ist die Länge der Kodierung einer Sequenz, die mit Hilfe der der zugehörigen DTD kodiert wurde. Das FLP Problem ist NP-hart, kann aber auf das *set cover problem* reduziert werden, für das es logarithmische Approximationen gibt. Der in XTRACT angewandte Algorithmus hat eine Laufzeit von $O(N^2 \cdot \log N)$ mit $N = |I|$ und wurde im Paper nicht näher erläuert. Näheres über die angewandten Verfahren ist in [CG99] und [Hoc82] nachlesbar.

3.5 Experimentelle Validierung

Um die Effektivität der durch XTRACT angewandten Methoden zum Herleiten von DTDs aus Beispieldokumenten zu bestimmen, wurde eine Studie sowohl mit synthetischen als auch mit Real-life DTDs durchgeführt. Dabei wurden die Ergebnisse von XTRACT mit den Ergebnissen des IBM alphaworks DTD Extraktionswerkzeugs DDbE (Data Description by Example) verglichen. Die Ergebnisse zeigen, dass XTRACT bei den meisten vorgebenen DTDs bei weitem bessere DTDs erzeugt als DDbE.

Synthetische Datensätze In Tabelle 2 sind die DTDs aufgeführt, die in den Experimenten betrachtet wurden. Der Übersichtlichkeit wegen werden die DTDs durch reguläre Ausdrücke repräsentiert.

Nr.	Vereinfachte DTD
1	$abcde\|efgh\|ij\|klm$
2	$(a\|b\|c\|d\|e)*gh$
3	$(a\|b\|c\|d)*\|e$
4	$(abcde)*f$
5	$(ab)*\|cdef\|(ghi)*$
6	$abcdef(g\|h\|i\|j)(k\|l\|m\|n\|o)$
7	$(a\|b\|c)d*e*(fgh)*$
8	$(a\|b)(cdefg)*hijklmnopq(r\|s)*$
9	$(abcd)*\|(e\|f\|g)*\|h\|(ijklm)*$
10	$a*\|(b\|c\|d\|e\|f)*\|gh\|(i\|j\|k)*\|(lmn)*$

Tabelle 2: Eine Auswahl synthetischer DTDs

Real-life Datensätze Als Datensätze wurden die Classified Advertising Standards XML DTD der Newspaper Association of America (NAA) benutzt. In Tabelle 3 werden sechs hierfür repräsentative DTDs gezeigt. Interessant sind insbesondere die drei letzten DTDs. DTD 4 enthält ein ? in Verbindung mit einem Kleenestern. DTD 4 und DTD 5 enthalten zwei reguläre Ausdrücke mit Kleenesternen. Dei DTD 6 ist ein Kleenestern in einem anderen geschachtelt.

Qualität der erzeugten DTDs Die DTDs, die XTRACT und DDbE aus den synthetischen Datensätzen erzeugt haben, sind in Tabelle 4 zu sehen. Dort

Nr.	Original DTD	Vereinfachte DTD
1	!ENTITY % included-elements “audio-clip|blind-box-reply|graphic|linkpi-char|video-clip”	$a\|b\|c\|d\|e$
2	!ELEMENT communicatons-contacts (phone|fax|email|pager|web-page)	$(a\|b\|c\|d\|e)*$
3	!ELEMENT employment-services (employment-service.type, employment-service.loaction*(e.zz-generic-tag)*)	$ab*c*$
4	!ENTITY % location “addr*,geographic-area?,city?, state-province?,postal-code?,country?	$a*b?c?d?$
5	!ELEMENT transfer-info(transfer-number, (from-to,company-id)+,contact-info)*	$(a(bc)+d)*$
6	!ELEMENT real-estate-services(real-estate-service.type, real-estate-service.location?,r-e.response-modes*,r-e.comment?)*	$(ab?c*d?)*$

Tabelle 3: Real vorkommende DTDs der Newspaper Assosication of America

kann man erkennen, dass XTRACT jede einzelne dieser DTDs korrekt herleitet. Dagegen berechnet DDbE nur die erste DTD, die zugleich die einfachste ist, korrekt. Bei den einfachen DTDs 2-5 ist DDbE nicht in der Lage eine korrekte DTD herzuleiten oder die hergeleitete DTD deckt nicht alle Beispieldokumente ab. DDbE neigt scheinbar zur Überverallgemeinerung, wenn die Orignal DTD Kleenesterne enthält. Bei den repräsentativ ausgewählten real vorkommenden DTDs erkennt XTRACT die ersten fünf korrekt (Tabelle 5). Dagegen kann DDbE nur die ersten beiden DTDs herleiten und eine Näherung bei DTD 3. Mit weiteren Faktorisierungschritten könnte DDbE auch dort noch die korrekte DTD finden. Bei DTD 4 ist die Faktorisierungschwäche von DDbE noch deutlicher zu erkennen. Bei den DTDs 5 und 6 zeigt sich die Tendenz von DDbE zur Übergeneralisierung. Hier wurde ein einfaches ODER über alle Tags erstellt. Die von XTRACT für DTD 6 hergeleitete DTD 6 ist sehr komplex und lässt die fehlende Möglichkeit ? innerhalb von Kleenesternen zu verarbeiten erkennen.

3.6 Schlussfolgerung

XTRACT nutzt das Prinzips der *minimum description length* um aus einer Menge von Kandidaten DTDs eine finale DTD zu errechnen. Das $MDL-Prinzip$ sorgt hierbei für den Ausgleich zwischen der Einfachheit und dem Verallgemeinersgrad einer erzeugten DTD. Neben dem Erstellen von DTDs aus einer Menge von Beispiel XML-Dokumenten kann XTRACT auch im Allgemeinen zum Herleiten von regulären Ausdrücken aus Beispieldaten genutzt werden. Die Experimente mit künstlichen und real vorkommenden Dokumenten zeigen, dass der Ansatz richtig gewählt wurde. Sie zeigen aber auch, dass es in manchen Bereichen noch Möglichkeiten zur Verbesserung gibt. Einserseits ist dies eine Verbesserung im Generalisierungsmodul, die es möglich machen sollte, reguläre Ausdrücke der Form $(ab?)*$ zu finden. Weiterhin sollte es das Generalisierungsmodul ermöglichen komplexere reguläre Sprachen zu erkennen.

Nr.	Vereinfachte DTD	DTD XTRACT	DTD DDbE
1	$abcde \mid efgh \mid ij \mid klm$	$abcde \mid efgh \mid ij \mid klm$	$abcde \mid efgh \mid ij \mid klm$
2	$(a \mid b \mid c \mid d \mid e) * gh$	$(a \mid b \mid c \mid d \mid e) * gh$	$gh(a \mid b \mid c \mid d \mid e) + gh$
3	$(a \mid b \mid c \mid d) * \mid e$	$(a \mid b \mid c \mid d) * \mid e$	$(e(a \mid c \mid d \mid b) + e)$
4	$(abcde) * f$	$(abcde) * f$	$(f(a \mid e \mid d \mid c \mid b) + f)$
5	$(ab) * \mid cdef \mid (ghi)*$	$(ab) * \mid cdef \mid (ghi)*$	$cdef(a \mid b \mid g \mid h \mid i \mid h) + cdef$
6	$abcdef(g \mid h \mid i \mid j)(k \mid l \mid m \mid n \mid o)$	$abcdef(g \mid h \mid i \mid j)$ $(k \mid l \mid m \mid n \mid o)$	$abcdef(j(o \mid l \mid m \mid n \mid k) \mid$ $g(o \mid l \mid n \mid m \mid k) \mid h(m \mid l \mid n \mid k \mid o) \mid$ $i(o \mid l \mid n \mid m \mid k))$
7	$(a \mid b \mid c)d * e * (fgh)*$	$(a \mid b \mid c)d * e * (fgh)*$	$((a \mid b \mid a)d + e + \mid ad + \mid$ $bd + \mid c(e + \mid d+)? \mid ad * \mid be*))$ $(a \mid b \mid c)d * e * (fgh)*$ $(f \mid h \mid g) + ((a \mid b \mid c)d + e + \mid$ $c(e + \mid d+)? \mid a(e + \mid d+)? \mid$ $a(e + \mid d+)? \mid b(e + \mid d+)?)$
8	$(a \mid b)(cdefg) * hijkl$ $mnopq(r \mid s)*$	$(a \mid b)(cdefg) * hijkl$ $mnopq(r \mid s)*$	$((((a \mid b)hijabc$ $defg) \mid b \mid a)(c \mid g \mid f \mid e \mid d \mid s \mid r)+$ $((b \mid a)?hijkamnopq))$
9	$(abcd) * \mid (e \mid f \mid g) * \mid h \mid$ $(ijklm)*$	$(abcd) * \mid (ijklm) * \mid$ $h \mid (e \mid f \mid g)*$	$h(a \mid d \mid c \mid b \mid e \mid g \mid$ $f \mid i \mid m \mid l \mid k \mid j) + h$
10	$a * \mid (b \mid c \mid d \mid e \mid f) * \mid gh \mid$ $(i \mid j \mid k) * \mid (lmn)*$	$a * \mid (b \mid c \mid d \mid e \mid f) * \mid gh \mid$ $\mid (i \mid j \mid k) * (lmn)*$	$(a + \mid gh)(e \mid f \mid d \mid i \mid j \mid l \mid n \mid m \mid k \mid c \mid b)+$ $(a + \mid gh)$

Tabelle 4: Ergebnisse von XTRACT und DDbE bei synthetischen DTDs

Nr.	Vereinfachte DTD	DTD XTRACT	DTD DDbE
1	$a \mid b \mid c \mid d \mid e$	$a \mid b \mid c \mid d \mid e$	$a \mid b \mid c \mid d \mid e$
2	$(a \mid b \mid c \mid d \mid e)*$	$(a \mid b \mid c \mid d \mid e)*$	$(a \mid b \mid c \mid d \mid e)*$
3	$(ab * c*)$	$ab * c*$	$(ab + c*) \mid (ac*)$
4	$a * b?c?d?$	$a * b?c?d?$	$(a + b(c \mid (c?d))?) \mid ((b \mid a+)?cd) \mid$ $((a + \mid b)?d) \mid ((a + \mid b)?c) \mid a + \mid b)$
5	$(a(bc) + d)$	$(a(bc) * d)*$	$(a \mid b \mid c \mid d)*$
6	$(sb?c * d?)*$	-	$(a \mid b \mid c \mid d)+$

Tabelle 5: Vergleich der Ergebnisse von XTRACT und DDbE bei real vorkommenden DTDs

4 Inkrementelle Validierung von XML-Dokumenten

4.1 Motivation

Mit der Verbreitung von XML als Standard-Format zum Repräsentieren von Daten im Word-Wide-Web wächst auch die Zahl der Datenbanken, die XML-Daten speichern. Um einen effizienten Datenaustausch zu gewährleisten, werden durch Document Type Definitions (DTD's) und XML Schema Bedingungen an die Struktur der Datenbank gestellt.
Wird ein XML-Dokument in der Datenbank verändert, muss es auf seine Gültig-

```
<dealer>
 <UsedCars>
  <ad>
   <model>Honda</model>
   <year>92</year>
  </ad>
 </UsedCars>
 <NewCars>
  <ad>
   <model>BMW</model>
  </ad>
 </NewCars>
</dealer>
```

dealer
UsedCars NewCars
ad ad
model year model

```
<!DOCTYPE dealer>
<!ELEMENT dealer
 (UsedCars, NewCars)>
<!ELEMENT UsedCars (ad*)>
<!ELEMENT NewCars (ad*)>
<!ELEMENT ad (model, year?)>
<!ELEMENT model PCDATA>
<!ELEMENT year PCDATA>
```

$$
\begin{array}{lcl}
root & : & dealer \\
dealer & \rightarrow & UC\ NC \\
UC & \rightarrow & ad^* \\
NC & \rightarrow & ad^* \\
ad & \rightarrow & model\ (year|\epsilon) \\
model & \rightarrow & \epsilon \\
year & \rightarrow & \epsilon
\end{array}
$$

$$
\begin{array}{ll}
root : d^t & \\
d^t \rightarrow UC^t\ NC^t & \mu(d^t) = dealer \\
UC^t \rightarrow (ad^u)^* & \mu(UC^t) = UC \\
NC^t \rightarrow (ad^n)^* & \mu(NC^t) = NC \\
ad^u \rightarrow m^t\ y^t & \mu(ad^u) = ad \\
ad^n \rightarrow m^t & \mu(ad^n) = ad \\
m^t \rightarrow \epsilon & \mu(m^t) = model \\
y^t \rightarrow \epsilon & \mu(y^t) = year
\end{array}
$$

Abbildung 2: XML, DTD und spezialisierte DTD (UC und NC stehen für usedCars und NewCar

keit bezüglich der Constraints der DTD oder des XML Schema hin überprüft werden. Da eine Validierung der gesamten Datenbank nach jedem Update nicht praktisch ist, wird im folgenden ein effizienter inkrementeller Validierungsalgorithmus vorgestellt.

4.2 Grundlagen

Ein XML-Dokument kann als Baum von geschachtelten Elementen betrachtet werden. Hierbei wird der Inhalt von Elementen und Attributwerte ignoriert, da sie fr die Validierung keine Rolle spielen. Ein *geordneter beschrifteter Baum* über einem Alphabet Σ ist ein Paar $T = \langle t, \lambda \rangle$, wobei t ein geordneter Baum ist und λ jedem Knoten n aus t ein Label $\lambda(n) \in \Sigma$ zuordnet. Ebenso kann eine DTD als erweiterte kontextfreie Grammatik (CFG) über einem Alphabet Σ abstrahiert werden. Erweitert bedeutet, dass die Produktionen auf der rechten Seite reguläre Ausdrücke über Σ enthalten. Ein geordneter beschrifteter Baum $\langle t, \lambda \rangle$ über Σ erfüllt also eine DTD d, falls $\langle t, \lambda \rangle$ ein Ableitungsbaum der zur DTD gehörenden erweiterten kontextfreien Grammatik ist.

Betrachten als Beispiel das XML-Dokument in Abbildung 2 mit seiner Abstraktion als geordneter beschrifteten Baum. Dieses Dokument erfüllt die angegebene DTD, da der Baum, wie leicht zu erkennen ist, ein Ableitungsbaum der Grammatik ist. In der Menge $sat(d)$ sind genau die beschrifteten Bäume enthalten, welche die DTD d erfüllen.

Sei $root(t)$ das Startsymbol einer DTD d. Wir nehmen o.B.d.A. an, dass die DTD für alle $a \in \Sigma$ eine einzelne Regel $a \rightarrow r_a$ enthält. Mit N_a bezeichnen wir den nichtdeterministischen endlichen Automaten (NEA), der die Sprache r_a erkennt.

Nichtdeterministische endliche Automaten Ein NEA ist ein 5-Tupel $N = \langle \Sigma, Q, q_0, F, \delta \rangle$ mit Σ als endlichem Alphabet, Q als endlicher Menge der Zustände, $q_0 \in Q$ als *Startzustand* und δ als Zustandsübergangsfunktion von $\Sigma \times Q \rightarrow \mathcal{P}(Q)$. Ein String $a_1 ... a_n$ wird von N akzeptiert, falls es eine Zuordnung $\sigma : \{1, ..., n\} \rightarrow Q$ gibt mit $\sigma(a_1) \in \delta(a_1, q_0)$, $\sigma(a_n) \in F$ und $\forall i, 1 \leq i < n : \sigma(a_{i+1}) \in \delta(a_{i+1}, \sigma(a_i))$. Die Menge aller von N akzeptierten Strings wird mit

$L(N)$ bezeichnet.

Das Inkrementelle Validierungs-Problem Gegeben eine DTD τ, einen Baum $T \in sat(\tau)$ und eine Sequenz s von Updates in T sowie den Baum T' nach den Veränderungen. Ziel ist es, geringere Kosten zu erhalten als durch eine komplette Validierung von T'. Folgende Updates sind zugelassen:

(a) Ersetzen des aktuellen Labels eines Knotens v

(b) Einfügen eines neuen Blattknotens an einen Knoten v

(c) Einfügen eines neuen Blattknotens als erstes Kind eines Knotens v

(d) Löschen eines Blattknotens v

Wir gehen hier davon aus, dass der NEA, der die der DTD zugrundeliegende reguläre Sprache erkennt, im Zuge der Initialisierung vorberechnet wird und so kostenlos zur Verfügung steht. Folgende Kosten fliessen dann in die Laufzeit ein:

(a) Die Zeit, die benötigt wird den Baum T' mit Hilfe von T und der Hilfsstruktur $A(T)$ zu validieren

(b) Die Zeit zum Berechnen von $A(T')$ aus $A(T)$, T und s

(c) Die Grösse der Hilfsstruktur $A(T)$

4.3 Inkrementelle Validierung von Strings

Zunächst betrachten wir die inkrementelle Validierung eines String $a_1...a_n$ in Bezug auf die durch den NEA $N = \langle \Sigma, Q, q_0, F, \delta \rangle$ spezifizierte reguläre Sprache. Wir betrachten den String als eine Folge von Knoten mit dazugehörigen Labels.

Renamings Sei $u(a_{i_1}, b_i), ...u(a_{i_m}, b_i)$, $i_1 < i_2 < i_m$ eine Folge von Element-Renamings. Ziel ist es, effizient zu überprüfen, ob der aktualisierte String
$a_1...a_{i_1-1}b_1a_{i_1+1}...a_{i_m-1}b_ma_{i_m+1}...a_m \in L(N)$ ist.
Validierung des neuen Strings durch komplettes Durchlaufen des Automaten N würde Zeit $O(n|Q|^2 \log |Q|)$ benötigen. Durch die Verwendung einer Hilfsstruktur können wir die Laufzeit aber verbessern: Sei $u(i, b)$ ein einzelnes Renaming und die Mengen $Pre(i) = \delta(q_0, a_1...a_{i-1})$ und $Post(i) = \{s|\delta(q_0, a_{i+1}...a_n) \in F\}$ für alle i bereits vorberechnet. Der veränderte String ist in $L(N)$, falls es einen Zustand $s_1 \in Pre(i)$ und einen Zustand $s_2 \in Post(i)$ gibt mit $s_2 \in \delta(b, s_1)$. Problematisch an dieser Technik ist, dass bei einer Folge von Updates das Aufrechterhalten der Mengen sehr teuer ist, da nach jedem Update $u(i, b)$ alle $Pre(j)$ für $j > i$ und alle $Post(k)$ für $k < i$ berechnet werden müssen und so die Laufzeit auf $O(n|Q|^2 \log |Q|)$ steigt.
Wir benutzen nun eine andere Hilfsstruktur:
Sei T_{ij} die *Transitionsrelation* $\{\langle p, q \rangle \,|p, q \in Q, q \in \delta(p, a_i...a_j)\}$.
Dabei gilt $T_{ij} = T_{ik} \circ T_{kj}$, $i < k < j$.
Mit δ_a definieren wir die Relation $\{\langle p, q \rangle \,|q \in \delta(p, a)\}$ für ein $a \in \Sigma$.

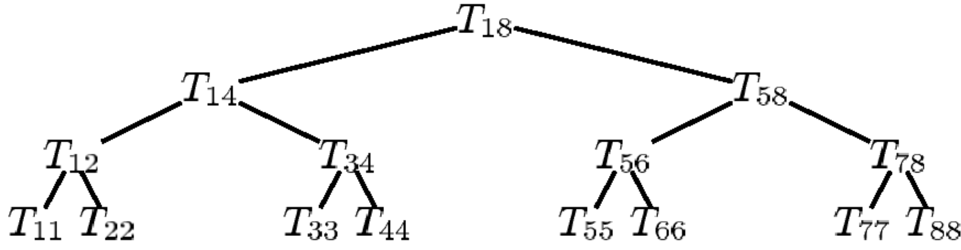

Abbildung 3: Transistionsbaum T18

Divide and Conquer Ansatz Sei $n = 2^k$. Wir benutzen als Hilfsinformation nur die T_{ij} für Intervalle $[i, j]$, die man durch rekursives Aufsplitten von $[1, n]$ erhält. Der *Transitionsrelationsbaum* $\mathcal{T}_n$ wird induktiv wie folgt definiert:

- die Wurzel ist die Relation $T_{1,2^k}$
- jeder Knoten T_{ij} mit $j - i > 0$ hat genau die Kinder T_{ik} und $T_{(k+1),j}$, mit $k = \frac{j-i+1}{2}$.
- die Blätter sind die Relationen T_{ii}, $1 \leq i \leq n$.

Somit ist $\mathcal{T}_n$ ein balancierter Binärbaum der Tiefe $\log n$ mit genau $n + (n/2) + ... + 2 + 1 = 2n - 1$ Knoten. Insgesamt hat die Hilfsstruktur somit eine Grösse von $O(n|Q^2|)$. Als Beispiel ist in Abbildung 3 der Transistionsrelationsbaum $\mathcal{T}_8$ dargestellt.

Sei nun $u(a_{i_1}, b_i), ...u(a_{i_m}, b_i)$, $i_1 < i_2 < i_m$ wieder die Folge von Updates auf dem String $a_1...a_n$. Zu beachten ist, dass nur die Relationen T_{ij} durch die Updates betroffen sind, die auf dem Weg von einem Blatt T_{i_v,i_v} $(1 \leq v \leq m)$ zur Wurzel von $\mathcal{T}_n$ liegen. Die Menge $\mathcal{I}$ dieser Relationen hat eine Grösse von höchstens $m \log n$.

Der Baum $\mathcal{T}_n$ kann nun aktualisiert werden, indem man die T_{ij} in $\mathcal{I}$ bottom-up neu berechnet: zuerst werden die Blätter $T_{i_v,i_v} \in \mathcal{I}$ auf δ_{b_v}, $(1 \leq v \leq m)$ gesetzt. Dann wird jedes $T_{i,i} \in \mathcal{I}$ mit Kindern T_{iv} und T_{vj}, von denen mindestens eines neu berechnet wurde, durch $T_{iv} \circ T_{vj}$ ersetzt. Insgesamt wurden also höchstens $m \log n$ der T_{ij} in einer Zeit von je $O(|Q|^2 \log |Q|)$ neu berechnet.

Zur Validierung des Strings $a_1...a_{i_1-1}b_1a_{i_1+1}...a_{i_m-1}b_m a_{i_m+1}...a_m$ genügt es nun, in der aktualisierten Hilfsstruktur zu testen, ob $\langle q_0, f\rangle \in T_{1n}$ für ein $f \in F$. Somit hat die Validierung insgesamt eine Laufzeit von $O(m|Q|^2 \log |Q| \log n)$.

Einfügen und Löschen Der Ansatz für Renamings kann hier nicht weiter verfolgt werden, da sowohl die Positionen der Knoten im String verändert werden als auch die Länge n des Strings variiert und damit der Transitionsrelationsbaum $\mathcal{T}_n$ jedesmal neu berechnet werden müsste.Wir bauen nun unseren Algorithmus auf dem Konstrukt der *B-Tree's* neu auf. Damit kann unsere neue

Baumstruktur $\mathcal{T}$ auch bei Einfüge- und Löschoperationen effizient aufrecht erhalten werden. Insbesondere haben wir auch weiterhin einen balancierten Baum mit Tiefe $O(\log n)$. Unser B-Tree für $\mathcal{T}$ enthält 3 Zellen pro Knoten, von denen höchstens 2 Zellen leer sein dürfen. Jede Zelle ist entweder leer oder enthält eine zu einem Substring s gehörende Menge T_s. Jede nichtleere Zelle ist entweder in einem Blattknoten oder besitzt einen Kindknoten.
In $\mathcal{T}$ muss der zu T_s gehörende Teilstring s nicht explizit berechnet werden. Folgende Invariante bleibt dabei immer erfüllt:

- die Folge von nichtleeren Zellen in den Blättern ist $T_{s_1}...T_{s_n}$
  (n= Länge des aktuellen Strings, $T_{s_i} = T_{ii}$, $1 \leq i \leq n$)

- wenn eine Zelle eines inneren Knotens auf einen Kindknoten mit Relationen T_{s_1}, T_{s_2} (und T_{s_3}) in den Zellen zeigt, dann gilt für die Relation T_s in dieser Zelle: $T_s = T_{s_1} \circ T_{s_2} (\circ T_{s_3})$.

Daneben benutzen wir Pointer, die in $O(1)$ zu einem Element v im String das entsprechende Blatt T_s finden. Sobald $\mathcal{T}$ für den aktuellen String berechnet wurde, ist die Validierung einfach: es genügt zu testen, dass für ein $f \in F$ $\langle q_0, f \rangle$ zur Komposition der Mengen T_s in den Zellen der Wurzel von $\mathcal{T}$ gehört. Dies benötigt Zeit $O(|Q|^2 \log |Q|)$.
Die Hilfsstruktur $\mathcal{T}$ wird zu Beginn aus dem leeren String durch eine Folge von Inserts aufgebaut. Bei einem Renaming von einem Element v in s wird der entprechende Knoten im Baum gesucht und alle Mengen T_s von T_v bis zur Wurzel aktualisiert. Dies benötigt Zeit $O(\log n)$.
Insertions und Deletions werden analog zu den Algorithmen von B-Bäumen durchgeführt. Insbesondere kann ein *Node Split* in einem Blatt ebenfalls bis zur Wurzel propagieren und wir müssen neue Relationen im Blatt, der neuen Wurzel und in den inneren Knoten berechnen.
Dafür benötigen wir eine Zeit von $O(|Q|^2 \log |Q| \log n)$. Da $\mathcal{T}$ aufgrund der Struktur von B-Bäumen immer eine Tiefe von $O(\log n)$ für einen String der Länge n hat, benötigt die Aufrechterhaltung von $\mathcal{T}$ unter m Updates Zeit $O(m|Q|^2 \log |Q| \log n)$.

4.4 Inkrementelle DTD-Validierung

Sei d eine DTD und $T = \langle t, \lambda \rangle$ ein beschrifteter Baum der d erfüllt.

Renamings Sei m eine Folge von Label-Updates, die zu einem neuen Baum $T' = \langle t', \lambda' \rangle$ führen. Wir müssen überprüfen, ob für jeden Knoten v in t' mit (modifizierten) Kindern $v_1...v_n$ die Folge der Labels $\lambda'(v_1), ...\lambda'(v_n)$ zu $r_{\lambda'(v)}$ gehört. Falls $\lambda'(v) = \lambda(v)$ ist, läuft die Validierung analog der String-Validierung im letzten Abschnitt. Wurde das Label von v geändert, muss die Folge $\lambda'(v_1), ...\lambda'(v_n)$ in Bezug auf die neue reguläre Sprache $r_{\lambda'(v)}$ validiert werden. Um eine Validierung von neuem zu verhindern, verwalten wir vorsorglich Informationen über die Gültigkeit von jedem String von Geschwistern in Bezug auf *alle* regulären Sprachen r_a für $a \in \Sigma$: Für jede Folge $s = s_1, \cdots, s_k$ von Geschwistern im Baum berechnen wir die Transitionsrelationen T_s^a für

jeden NEA N_a zu r_a, $a \in \Sigma$. Da jeder NEA für Strings der Länge n eine Grösse von $O(|Q|^2 n)$ hat, hat die neue Hilfsstruktur eine Grösse von höchstens $O(|\Sigma||d|^2|T|)$, wobei $|d| = max\{|r_a||a \to r_a \in d\}$. Die Aufrechterhaltung der Hilfsstruktur wird analog der Stringstruktur durchgeführt und führt so zu Kosten von $O(m|\Sigma||d|^2 \log |d| \log |T|)$ für eine Folge von m Veränderungen.
Der modifizierte Baum T' ist in Bezug auf d gültig, wenn für jeden Knoten v mit Label a in T' (wobei entweder v oder eines seiner Kinder verändert wurde) $\langle q_0, f\rangle$ in der Realtion T_s^a ist, wobei s die Liste der Kinder von v, q_0 der Startzustand von N_a und f einer seiner Endzustände ist. Jeder solche Test braucht $O(|d|^2 \log |d|)$ Zeit, und im worst case sind m Tests durchzuführen.

Somit benötigt die Validierung insgesamt eine Zeit von

$$O(m|\Sigma||d|^2 \log |d| \log |T|)$$

Insertions und Deletions Einfüge- und Löschoperationen werden analog dem Ansatz der Validierung von Strings mithilfe von B-Bäumen durchgeführt.

4.5 Inkrementelle XML Schema-Validierung

XML Schema stellt eine Erweiterung der DTD's dar. Am wichtigsten dabei ist die Fähigkeit, den Typ eines Elements von seinem Namen trennen zu können. Für einen Autohändler wie in Abbildung 1 ist es zum Beispiel der Fall, dass gebrauchte Autos eine Modellbezeichnung und ein Baujahr haben, während Neuwagen nur eine Modellbezeichnung haben. Diesen Aspekt kann man mit DTD's nicht darstellen, da die Regeln nur vom Namen des Elements abhängen, nicht aber vom Kontext. Durch die Verwendung von XML Schema werden kontextabhängige Definitionen der Struktur von Elementen erlaubt.
Ein XML Schema kann als *spezialisierte* DTD abstrahiert werden. Leider sind die Auswirkungen von Updates auf einen einzelnen Knoten global, da sich das Typing des gesamten Baumes ändern kann. Der Algorithmus zur Validierung von Dokumenten bezüglich einer einfachen DTD kann nicht weiter verwendet werden.
Wie sich herausgestellt, definieren die spezialisierten DTD's genau die Sprache der regulären Ausdrücke von *Unranked Trees* und sind äquivalent zu *nichtdeterministischen Tree-Automaten*. Der Unranked Tree T, der das XML-Dokument repräsentiert, wird als binärer Baum enkodiert. Dann wird die spezialisierte DTD in einen bottom-up nichtdeterministischen Tree-Automaten überführt, der genau die Enkodierungen von gültigen Dokumenten akzeptiert.
Der Validierungsalgorithmus benutzt ebenfalls einen Divide-and-Conquer Ansatz, um Berechnungen entlang bestimmter Pfade aufzuteilen. Zu jedem dieser Pfade wird wieder eine Hilfsstruktur aufgebaut. Auf diese Weise erreichen wir eine Laufzeit von $O(m|\Sigma^t|^2|d|^2 \log(|\Sigma^t||d|) \log^2 |T|)$, wobei Σ^t das endliche Alphabet der Typen darstellt. Die Grösse der verwendeten Hilfsstruktur liegt bei $O(|\Sigma^t|^2|d|^2|T|)$.

4.6 Schlussfolgerung

Die vorgestellten Algorithmen zur inkrementellen Validierung stellen eine deutliche Verbesserung gegenüber einer kompletten Neuvalidierung des geänderten XML-Dokuments dar. Gegeben eine Folge von m Updates auf ein Dokument der Grösse n, hat unserer Algorithmus bezüglich einer DTD eine Laufzeit von $O(m \log(n))$ und benutzt eine Hilfsstruktur der Grösse $O(n)$. Der Algorithmus für spezialisierte DTD's hat eine Laufzeit von $O(m \log^2(n))$ und eine Hilfsstruktur der Grösse von ebenfalls $O(n)$.

Ein Optimierungsansatz wäre die Handhabung von mehreren Updates. Anstatt bei mehreren Updates die Hilfsstruktur nach jedem Update einzeln zu aktualisieren und die Gültigkeit des veränderten Baumes am Ende zu testen, wäre es sicher manchmal effizienter, Gruppen von Updates zu betrachten. Ebenso wäre eine komplette Revalidierung besser, falls die Anzahl von Updates die Grösse des finalen Baumes übersteigt.

Darüber hinaus haben wir nur Updates betrachtet, die jeweils nur einen einzelnen Knoten betreffen. In vielen Situationen finden aber mehrere Updates auf einmal statt, zum Beispiel in XML-Editoren (Cut-and-Paste, usw.). Eine Möglichkeit wäre es, die Updates auf Singletons zu reduzieren und die vorgestellten Algorithmen weiter zu verwenden. Effizienter wäre es in diesem Fall aber sicherlich, einen Algorithmus zu entwerfen, der komplexere Updates als Ganzes handhaben kann.

Literatur

[BM82] R. K. Brayton and C. McMullen. The decomposition and factorization of boolean expressions. In *International Symposium on Circuits and Systems*, pages 49–54, 1982.

[Bra93] Alvis Brazma. Efficient identification of regular expressions from representative examples. In *COLT*, pages 236–242, 1993.

[CG99] M. Charikar and S. Guha. Improved combinatorial algorithms for facility location problem an k median problems. In *40th annual Symposium on Foundations of Computer Sciene*, 1999.

[Hoc82] D. S. Hochbaum. Heuristics for fixed cost median problem. *Mathematical Programming*, pages 148–162, 1982.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automaton Theory, Languages and Computation*. Addision-Wesley, 1979.

[Ris78] J. Rissanen. Modeling shortest data description. *Automatica*, 14:465–471, 1978.

[Ris89] J. Rissanen. Stochastic complexity in statical inquiry. 1989.

[Wan89] A. R. R. Wang. *Algorithms for multi-level Logic optimization*. PhD thesis, The University of California, Berkley, 1989.

Abbildungsverzeichnis

1 XTRACT: Schematischer Aufbau 6
2 Incremental Validation: XML, DTD, XSchema 15
3 Incremental Validation: Transistionsbaum 17

Tabellenverzeichnis

1 XTRACT: MDL-Kodierung und MDL-Kosten 11
2 XTRACT: Auswahl synthetischer DTDs 12
3 XTRACT: Real vorkommende DTDs 13
4 XTRACT: Ergebnisse bei synthetischen DTDs 14
5 XTRACT: Ergebnisse bei real vorkommenden DTDs 14