Komplexität des typechecking-Problems für Top-down XML-Transformationen

Johannes Neubauer

3. April 2013

XML hat sich zu dem weitest verbreiteten Format für semistrukturierte Daten entwickelt. Häufig besteht die Notwendigkeit, Dokumente zu transformieren, um sie für unterschiedliche Zwecke, z.B. Medien, aufzubereiten. Besonders Anwendungen, die die Transformation von Daten in einem größeren Prozess verwenden, erfordern eine syntaktisch und semantisch korrekte Transformation. Nur dann können die Ausgaben im weiteren Verlauf verwendet werden. Die statische Analyse der Typkorrektheit einer Transformation ist als das Typechecking Problem bekannt und wird im Folgenden hinsichtlich seiner Komplexität, für eingeschränkte Transformationen, untersucht.

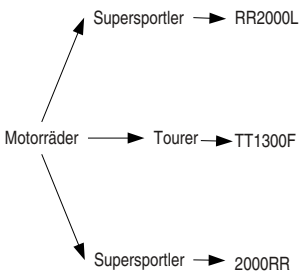
1 Überblick

Das Typechecking-Problem wird sehr schnell unentscheidbar, wenn bei einer Transformation Vergleiche von Datenwerten berücksichtigt werden (siehe [2]). Für die Praxis ist interessant, ob es eine Untermenge von Transformationen gibt, die einen möglichst großen Anwendungsbereich abdeckt und dabei effizient lösbar ist. Im weiteren Verlauf dieser Ausarbeitung werden daher nur Transformationen behandelt, die ausschließlich die Struktur eines Dokumentes berücksichtigen. Die Struktur eines XML-Dokumentes kann leicht durch einen unbeschränkten Baum dargestellt werden. Der Vorteil dieser Sichtweise ist, dass es in der theoretischen Informatik bereits viele Untersuchungen und Ergebnisse über Baumautomaten und Baumsprachen gibt, die weiterverwendet werden können. In [3] haben Milo, Suciu und Vianu gezeigt, dass sich die meisten Möglichkeiten der gängigen Transformations-Sprachen mit k-pebble (engl.: Marke) tree Transducern auf binären Bäumen beschreiben lassen. Binäre Bäume sind stärker erforscht und jeder unbeschränkte Baum lässt sich in einen binären Baum übersetzen. Ein unranked tree Transducer kann nicht durch einen deterministischen Top-down ranked tree Transducer simuliert werden (siehe [5]). Der Artikel, der dieser Arbeit zugrunde liegt (siehe

Dokument:

```
01 <Motorräder>
02  <Supersport>
03   <Hersteller>Nukunaki</Hersteller>
04   <Bezeichnung>RR2000L</Bezeichnung>
05   <Leistung>178 PS</Leistung>
06  </Supersport>
07
08  <Tourer>
09   <Hersteller>Kokuniki</Hersteller>
10   <Bezeichnung>TT1300F</Bezeichnung>
11   <Leistung>109 PS</Leistung>
12  </Tourer>
13
14  <Supersport>
15    <Hersteller>Sanikani</Hersteller>
16    <Bezeichnung>2000RR</Bezeichnung>
17    <Leistung>170 PS</Leistung>
18  </Supersport>
19 </Motorräder>
```

Baumstruktur:



DTD:

```
01 Motorräder    -> (Supersport+Tourer)*
02 Supersport    -> Hersteller, Bezeichnung, Leistung
03 Tourer        -> Hersteller, Bezeichnung, Leistung
```

Abbildung 1: Eingabedokument

[5]), konzentriert sich auf Top-down deterministische XML-Transformationen auf unbeschränkten Bäumen. Die Struktur eines wohlgeformten XML-Dokumentes entspricht einem unbeschränkten Baum.

Zusätzlich werden weitere Einschränkungen gemacht. Als Schemasprachen finden NTAs, DTAs und DTDs mit NFAs, DFAs bzw. SL-Formeln als rechte Seite der Regeln Verwendung. Ferner werden nicht löschende und kopierbeschränkte uniform tree Transducer eingesetzt. Die Komplexität bewegt sich dabei von PTIME bis EXPTIME. Die statische Analyse einer Transformation macht keine Aussagen über die Semantik der Eingabe- und Ausgabedokumente, sondern über die Syntax und Semantik der Transformation.

2 Motivation

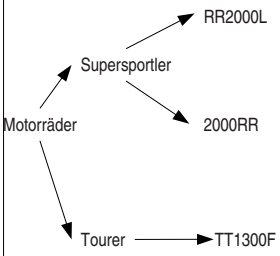
Die Relevanz des Typechecking-Problems soll an einem Beispiel motiviert werden. Ein Motorradhändler möchte die Zweiräder, die er aktuell im Programm hat, speichern. Dazu verwendet er XML. Ein Beispieldokument, seine Baumstruktur und die zugehörige DTD sind in Abbildung 1 dargestellt. Unter dem Wurzel-Tag "Motorräder"befinden sich die einzelnen Motorräder, unterschieden nach Supersportlern und Tourern. Jedes Motorrad hat drei Tags, die den Hersteller, die Bezeichnung und die Leistung beschreiben. Die Struktur des Dokumentes stellt sich als unvorteilhaft heraus. Die Sortierung nach Motorradtypen ist relativ aufwändig, da sie sich ungeordnet auf einer Ebene befinden. Es ist wünschenswert, die Struktur des Dokumentes dahingehend zu ändern, dass z.B. ein XPATH-Ausdruck direkten Zugriff auf alle Supersportler ermöglicht (’`/Motorräder/Supersport/`’). In Abbildung 2 sind die Motorräder daher nach Motorradtyp sortiert.

Eine weit verbreitete Transformations-Sprache ist XSLT. In Abbildung 3 werden

Dokument:

```
01 <Motorräder>
02  <Supersport>
03    <Motorrad>
04     <Hersteller>Nukunaki</Hersteller>
05     <Bezeichnung>RR2000L</Bezeichnung>
06     <Leistung>178 PS</Leistung>
07    </Motorrad>
08    <Motorrad>
09     <Hersteller>Sanikani</Hersteller>
10     <Bezeichnung>2000RR</Bezeichnung>
11     <Leistung>170 PS</Leistung>
12    </Motorrad>
13  </Supersport>
14  <Tourer>
15    <Motorrad>
16     <Hersteller>Kokuniki</Hersteller>
17     <Bezeichnung>TT1300F</Bezeichnung>
18     <Leistung>109 PS</Leistung>
19    </Motorrad>
20  </Tourer>
21 </Motorräder>
```

Baumstruktur:



DTD:

```
01 Motorräder    -> Supersport, Tourer
02 Supersport    -> Motorrad*
03 Tourer        -> Motorrad*
04 Motorrad      -> Hersteller, Bezeichnung,
Leistung
```

Abbildung 2: Ausgabedokument

das XSL-Template für die Transformation des Motorraddokumentes und der entsprechende uniform tree Transducer gegenübergestellt. Der uniform tree Transducer ist nah an diese Transformations-Sprache angelehnt, umfasst jedoch nur eine Untermenge der Funktionalität (siehe Kapitel 4). Würde in dem Transducer die Regel $(q_0, Motorräder) \rightarrow Supersport(s)Tourer(t)$ direkt auf die Zustände s und t verweisen, würde die Information über den Typ eines Motorrades gelöscht und das Ausgabedokument nicht der DTD in Abbildung 2 entsprechen. Eine statische Analyse einer Transformation könnte derartige Fehler verhindern, bevor eine größere Datenmenge konvertiert wurde.

3 Top-down deterministische Baumtransformationen

Top-down deterministische Baumtransformationen sind eine Einschränkung von allgemeinen Baumtransformationen. In dieser Ausarbeitung werden derartige Transformationen von uniform tree Transducern repräsentiert, deren Funktionsweise in Kapitel 4 näher beleuchtet wird. Sie haben die folgenden Eigenschaften. Top-down tree Transducer können durch 1-pebble tree Transducer simuliert werden. Zustände des Transducers kommen nur in Blättern vor. Der Ausgabebaum wird nur nach unten erweitert. Es gibt nur einen Zustand/Marker, damit keine Rückverweise möglich sind. Es erfolgt nur ein Durchlauf durch den Eingabebaum. Sie arbeiten auf unbeschränkten Bäumen. Sie können die Kinder eines Knotens in mehreren Zuständen parallel bearbeiten.

Diese Transformationen sind nicht so mächtig, wie k-pebble tree Transducer, bilden aber eine sehr relevante Untermenge der Funktionen von XSLT ab und haben einen intuitiven, an XSLT orientierten Aufbau.

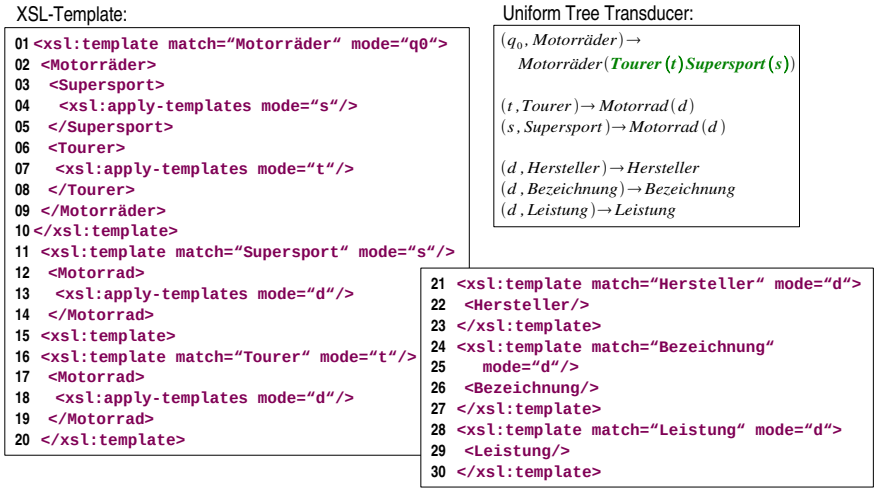


Abbildung 3: Transformation

4 uniform tree Transducer

Ein Top-down deterministic uniform tree Transducer ist ein Automat, der aus einer endlichen Zustandsmenge Q, einem Startzustand q_0, einem endlichen Ein-/Ausgabe-Alphabet Σ und einem endlichen Satz von Regeln R besteht. Im Folgenden wird dieser Begriff mit Transducer bzw. tree Transducer abgekürzt. Die Einschränkung auf ein Alphabet für Ein- und Ausgaben, erleichtert die Erklärung der Beweise. Die Einführung von getrennten Alphabeten würde die Ergebnisse aus [5] von Martens und Neven nicht beeinflussen.

Regeln

Die Regeln eines tree Transducers sind von der Form $(q, a) \rightarrow h$, wobei $q \in Q$, $a \in \Sigma$ und h ein **hedge** ist. Ein **hedge** ist eine Sequenz von Bäumen. Ein Baum ohne Wurzel ist ein **hedge**. Ein Wald ist im Gegensatz dazu eine Menge von Bäumen, bei der die Reihenfolge keine Rolle spielt. Eine gute Einführung in Baum- und **hedge**-Sprachen über unbeschränkten Alphabeten ist in [1] zu finden. Damit die Ausgabe einer Transformation XML-konform ist, sind die rhs[1] der vom Startzustand ausgehenden Regeln lediglich Bäume und beinhalten als Wurzel ein Symbol aus dem Alphabet. Eine rhs enthält Zustände lediglich in Blättern. Das gewährleistet die Top-Down Eigenschaft eines tree Transducers.

Arbeitsweise

Ein tree Transducer liest die Wurzel des Eingabebaums, der die Struktur des XML-Dokumentes widerspiegelt, im Startzustand q_0 ein. Die Erklärungen in diesem Abschnitt beziehen sich auf das Beispiel aus Kapitel 2. Die rhs der Regel $(q_0, Motorräder)$ muss, wie bereits erwähnt, ein Baum sein, um XML-Konformität im Ausgabedokument zu gewährleisten. Da es sich um eine deterministische Transformation handelt, ist jede Regel für ein Paar aus $\Sigma \times Q$ eindeutig. In diesem Falle ist die Startregel $(q_0, Motorräder) \rightarrow Motorräder(Supersport(s)Tourer(t))$. Der Transducer gibt Mo-

[1] rhs = right hand side bzw. rechte Seite einer Regel

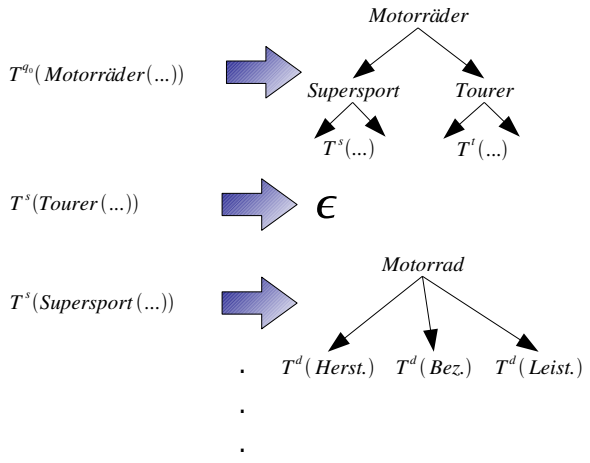


Abbildung 4: Transducer

torräder aus und kopiert den Eingabebaum zweimal jeweils in die Tags Supersport und Tourer im Zustand s beziehungsweise t. Dabei bearbeitet der Transducer die Kinder von `Motorräder` parallel in den Zuständen s und t. Das entspricht den Konstrukten `template` und `apply-templates` in XSLT:

```
<xsl:template match="Motorräder" mode="q_0">
  <Motorräder>
    <Supersport><xsl:apply-templates mode="s" /></Supersport>
    <Tourer><xsl:apply-templates mode="t" /></Tourer>
  </Motorräder>
</xsl:template>
```

Abbildung 4 veranschaulicht die Arbeitsweise des Transducers. Die Notation $T^q(t)$ bedeutet, dass sich Transducer T in Zustand $q \in Q$ befindet und den Baum t liest. In dieser Situation gibt es drei Fälle, die entscheiden, wie der Transducer agiert. Ist $t = \varepsilon$, gibt der Transducer ε aus. Handelt es sich bei t um einen Baum $a(t_1 \dots t_n)$, es gibt aber keine Regel mit (q, a) als lhs[2], wird ebenfalls ε ausgegeben. In Abbildung 4 gibt es z.B. im Zustand s keine Regel für `Tourer`-Tags. Das verhindert, dass Tourer in die Kategorie Supersport eingeordnet werden. Die dritte Möglichkeit ist, dass es sich bei t um einen Baum $a(t_1 \dots t_n)$ handelt und es eine Regel für das Paar (q, a) gibt. Hier ist a das Symbol der Wurzel von t. In diesem Fall bearbeitet der Transducer jeden Teilbaum $t_1, \dots, t_n$ von t parallel in den Zuständen der rhs der gefundenen Regel.

5 Einschränkungen

Das Typechecking Problem ist für allgemeine tree Transducer EXPTIME-Vollständig, daher werden weitere Einschränkungen vorgenommen, die in den nächsten Unterka-

[2] lhs = left hand side bzw. linke Seite einer Regel

piteln beschrieben werden. Neben der Einschränkung der Schemasprachen auf NTAs, DTAs und DTDs mit NFAs, DFAs und SL-Formeln als rhs werden in [5] auch der Transducer auf nicht löschende und kopierbeschränkte Transducer reduziert.

5.1 Nicht löschende Transducer

Ein nicht löschender Transducer besitzt nur Regeln $(q, a) \to h$, in denen es in h einen korrespondierenden Knoten zu a gibt. Das ist der Fall, wenn sich keine Zustände in den Wurzeln von h befinden. In löschenden Regeln werden die Kinder des Baumes mit Wurzel a auf der gleichen Ebene verarbeitet wie a. Abbildung 5 illustriert den Unterschied zwischen löschenden und nicht löschenden Regeln.

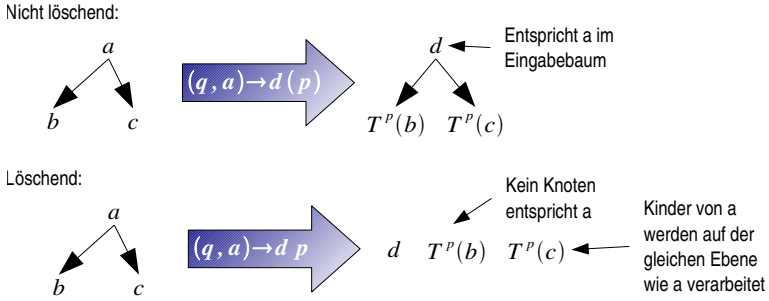


Abbildung 5: Unterschied zwischen löschenden und nicht löschenden Transducern

5.2 Transducer mit Kopierbreite k

Die Anzahl der Kopien, die ein Transducer von den Kindern eines Teilbaumes der Eingabe macht, entspricht der Anzahl der Zustände in den rhs der Regeln, in denen ein Knoten verarbeitet wird. Wird ein Knoten mit Symbol a im Zustand q gelesen, macht der Transducer für jeden Zustand in der rhs der Regel eine Kopie der Kinder von a. Ein Transducer mit Kopierbreite k erfüllt die Formel $k = max(\{\#(r)|r \in R\})$. Hierbei ist $\#$ die Anzahl der Zustände in der rhs von r.

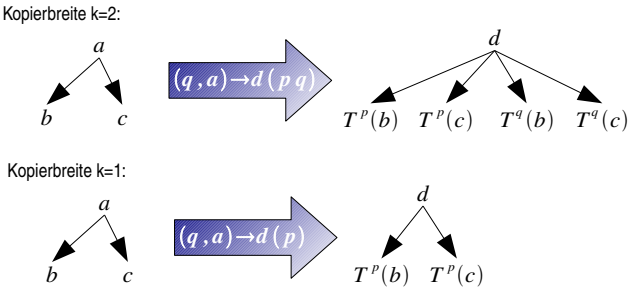


Abbildung 6: Kopierbreite eines Transducers

6 Komplexität des typechecking-Problems

Die folgenden Abschnitte werden die Ergebnisse von Martens und Neven in [5] näher beleuchten und einen Einblick in die Beweisführung geben. Für eine tiefergehende Auseinandersetzung mit diesem Thema sei auf dieses Dokument verwiesen.

6.1 Allgemeine uniform tree Transducer

Für allgemeine Transducer ist das typechecking-Problem unabhängig von der Wahl der Schemasprache EXPTIME-Vollständig. Die obere Schranke, dass $TC[T_A, NTA]$[3] in EXPTIME ist, folgt aus der Reduktion auf $TC[T_{nl}, NTA]$. Der Beweis, dass letzteres in EXPTIME ist, wird im nächsten Abschnitt behandelt. Gegeben ist der Transducer T_A und die NTAs A_{in} und A_{out}. Ziel ist es, einen Transducer T_{nl} zu konstruieren, der bezüglich A_{in} und NL_{out} genau dann Typ-korrekt ist, wenn T_A bezüglich A_{in} und A_{out} Typ-korrekt ist. Obere Schranke

Intuitiv werden alle löschenden Regeln in T_A durch nicht löschende Regeln ersetzt, in dem an die Wurzel der rhs ein Knoten mit Symbol $\# \notin \Sigma$ eingefügt wird. Die Regel $(q, a) \to cp$ mit $p, q \in Q$ und $a, c \in \Sigma$ wird ersetzt durch $(q, a) \to c\#(p)$. Zusätzlich wird eine Funktion γ definiert, die die eingefügten Symbole aus dem Ausgabebaum wieder löscht:

$$\gamma(\#(h)) = \gamma(h)$$

$$\gamma(a(h)) = a(\gamma(h)), \text{ mit } a \neq \#$$

Der NTA NL_{out} muss so konstruiert werden, dass $\gamma(t) \in L(A_{out}) \Leftrightarrow t \in L(NL_{out})$. NL_{out} sei o.B.d.A. bottom-up deterministic. Der Baumautomat verhält sich auf einem Baum ohne # Symbole wie A_{out}. Liest NL_{out} den Baum $a(t_1 \dots t_n)$ mit $a \neq \#$ und seien die Kinder mit Zuständen $q_1 \dots q_n$ markiert, dann schaut NL_{out} in der Funktionstabelle der Transitionsfunktion δ_A von A_{out} nach möglichen Paaren aus $Q \times \Sigma$ zu den Zuständen $q_1 \dots q_n$ und entscheidet sich nichtdeterministisch für eine Transition $\delta(q, a)$. Hierbei kann jede Transition als ein NFA aufgefasst werden, und $top(t_1 \dots t_n)$[4] als ein String aus Q^*, den der NFA einliest und entweder akzeptiert, wenn die Zustandsfolge in $\delta(q, a)$ enthalten ist oder ansonsten verwirft.

Stößt der NTA auf einen Knoten der mit einem #-Symbol gekennzeichnet ist, muss er das Verhalten von A_{out} auf dem Baum ohne das #-Symbol nachahmen. NL_{out} notiert einen Zustand (p_1, p_2) an diesem Knoten, der repräsentiert, dass der NFA von $\delta_A(q, a)$ im Zustand p_2 hält, wenn er den String $top(\delta(\#(t_1 \dots t_n))$ startend im Zustand p_1 liest. Die Transitionsfunktion von NL_{out} - δ_{NL} - springt nach dem Lesen der linken Geschwisterknoten von dem aktuellen #-Knoten nach p_2, falls er sich vor dem Lesen des Zustands (p_1, p_2) in Zustand p_1 befunden hat. Die Konstruktion ist in LOGSPACE durchführbar.

Die untere Schranke für $TC[T_A, DTD(SL)]$ wird durch eine Reduktion des Schnitt- Untere Schranke

[3] TC steht für typechecking-Problem. Die Parameter T_A und NTA besagen, dass allgemeine Transducer mit NTAs als Ein- und Ausgabe-Schemasprache verwendet werden

[4] $top(t_1 \dots t_n)$ beschreibt den String $q_1 \dots q_n \in Q^*$, mit dem die Wurzeln der Bäume $t_1, \dots, t_n$ gelabelt sind

problems für n Top-down deterministische Baumautomaten, das als EXPTIME-hard bekannt ist, auf $TC[T_A, DTD(SL)]$ bewiesen. Die Eingabe DTD lässt beliebige[5] Eingabebäume zu, während die Ausgabe-DTD lediglich ein *error*-Symbol verlangt:

$$d_{in}(\$) = true$$

$$d_{out}(\$) = error^{\geq 1}$$

Die Idee ist, dass der Transducer den Eingabebaum n mal kopiert, die Baumautomaten darauf simuliert und ein *error*-Symbol ausgibt, wenn einer der Baumautomaten den Baum nicht akzeptiert. Der Transducer macht dabei starken Gebrauch von löschenden Regeln, damit der Ausgabebaum die Tiefe eins hat. Nur dann kann die einfache Ausgabe-DTD, den Ausgabebaum akzeptieren. T_A ist typ-korrekt bezüglich d_{in} und d_{out}, genau dann wenn $\bigcap_{i=1}^{n} L(A_i) = \emptyset$.

6.2 Nicht löschende Transducer

$TC[T_A, NTA]$

Der Beweis aus Kapitel 6.1 für die obere Schranke von $TC[T_A, NTA]$ stützte sich darauf, dass $TC[T_{nl}, NTA]$ EXPTIME-Vollständig ist. Die untere Schranke folgt direkt aus der EXPTIME-Hardness des Containment Problems für NTAs. Die obere Schranke kann durch die Angabe eines Algorithmus durchgeführt werden, der in EXPTIME ist.

Algorithmus

Der Algorithmus berechnet eine Menge P von Paaren (S, f). S ist die Menge der Zustände, die A_{in} (Der NTA der als Baum-Schemasprache für die Eingabe dient) erreichen kann, ausgehend von der Wurzel eines Baumes t. $f(q)$ ist die Folge von Mengen von Zuständen, die A_{out}, startend an der Wurzel von $T^q(t)$, erreichen kann. Bei letzterem kann es sich um eine Folge von Mengen von Zuständen handeln, da in den Regeln von T_{nl} auch **hedge**s in rhs vorkommen können. Der Transducer ist nicht typ-korrekt, falls ein $(S, f) \in P$ existiert, mit $S \cap F_{in} \neq \emptyset$ (F_{in}, F_{out} sind die Mengen der akzeptierenden Zustände von A_{in} und A_{out}) und $f(q_T^0) \cap F_{out} = \emptyset$. In diesem Fall würde A_{out} einen Ausgabebaum verwerfen, obwohl A_{in} den zugehörigen Eingabebaum akzeptiert.

$TC[T_{nl}, DTA]$

Das typechecking-Problem für nicht löschende Transducer ist für DTAs ebenfalls EXPTIME-Vollständig:

- Obere Schranke: Das TC Problem für NTAs ist bereits in EXPTIME.
- Untere Schranke: Beweis durch Reduktion des Schnittproblems bei Top-down DTAs (Beweisführung, wie bei allgemeinen Transducern; EXPTIME-hart).

$DTD(NFA)$

Für DTDs mit NFAs als rhs ist die Komplexität der statischen Validierung der Typ-korrektheit PSPACE-Vollständig:

- Obere Schranke: Reduktion von TC auf das emptiness-Problem von NTAs (in PSPACE).

[5] Das $ wird als neue Wurzel an den Eingabebaum angehängt. Das vereinfacht die DTDs und lässt sich leicht konstruieren.

- Untere Schranke: containment-Problem von regulären Ausdrücken (bekannt als PSPACE-hard).

Das typechecking-Problem für DTDs mit DFAs bleibt PSPACE-Vollständig. Das Problem ist in PSPACE, da bereits DTDs mit NFAs in PSPACE sind. Die untere Schranke folgt aus einer Reduktion auf das Schnittproblem für n DFAs, das als PSPACE-hard bekannt ist. Die Idee ähnelt der in dem Beweis für allgemeine Transducer mit NTAs als Schemasprachen. Die Eingabe DTD erlaubt beliebige Bäume mit Tiefe eins. Die Kinder der Wurzel bilden ein beliebiges Wort aus Σ^*. Der Transducer macht n Kopien des Strings, getrennt durch $\#_*$ Zeichen, die nicht in Σ sind. Die Ausgabe-DTD akzeptiert, falls einer der n DFAs das Eingabewort zurückweist. $DTD(DFA)$

Mit SL-Formeln als rhs der DTDs sinkt die Komplexität auf CONP-Vollständig. Die Idee für den Beweis der oberen Schranke ist, einen Baum t zu raten, der von DTD d_{in} akzeptiert wird. Weiterhin wird ein Knoten v in t mit Label a geraten, ferner ein Wort w, das aus den Labeln der Kindern von v konstruiert wird, sowie ein Zustand $q \in Q_t$. Ziel ist es, ein Gegenbeispiel für die Typ-korrektheit von T_{nl} zu finden, d.h. $T^q(a(w))$ darf nicht von d_{out} akzeptiert werden. Der Baum t kann jedoch exponentiell in d_{in} sein. Daher wird lediglich ein Pfad zu einem Knoten v geraten, der zu einem Baum erweitert werden kann, welcher von d_{in} akzeptiert wird. Der Pfad besteht aus Paaren (a_i, q_i), wobei a_0 das Startlabel des Pfades ist und q_0 der Startzustand von T_{nl}. Für alle $i = 1 \dots n$ muss gelten, das T_{nl} a_i im Zustand q_i liest. $a_0 \dots a_n$ ist der Pfad von der Wurzel zu dem Knoten v. $DTD(SL)$

Die untere Schranke für $TC[T_{nl}, DTD(SL)]$ folgt aus einer Reduktion des Tautologieproblems für logische Formeln erster Ordnung. Die Idee ist einfach. Gegeben ist eine logische Formel mit den Variablen $v_1, \dots, v_n$. Das Alphabet des Transducers entspricht $\Sigma = \{a_1, \dots, a_n\}$. Die Eingabe-DTD erlaubt beliebige Bäume der Tiefe eins auf Σ. Die logische Formel ϕ wird in eine SL-Formel ϕ' übersetzt, in dem jedes Vorkommen einer Variable v_i durch die atomare SL-Formel $a_i^{\geq 1}$ ersetzt wird. Aus $(v_1 \wedge v_2) \vee \neg v_3$ würde demnach $(a_1^{\geq 1} \wedge a_2^{\geq 1}) \vee \neg a_3^{\geq 1}$. Die logische Formel ist genau dann eine Tautologie, wenn der Transducer typ-korrekt bezüglich d_{in} und d_{out} ist.

6.3 Kopierbeschränkte Transducer

Die Konstruktionen für nicht löschende Transducer lassen sich auf das typechecking-Problem für nicht löschende Transducer mit Kopierbreite k übertragen.

7 Ergebnisse und Ausblick

Das typechecking-Problem kann durch geschicktes Einschränken der Schemasprachen und Transformationen effizient gelöst werden. Dabei wird eine relevante Untermenge der Funktionalitäten gängiger Transformationssprachen unterstützt. Leider sind in [5] die kopierbeschränkten Transducer nur kurz erwähnt worden, obwohl erst diese einen Sprung in polynomielle Laufzeiten ermöglichen. Daher ist dieses Thema auch in diesem Dokument nicht weiter ausgeführt. Eine logische Weiterführung dieser Arbeit

findet sich in [4]. In Abbildung 7 sind die Ergebnisse von Martens und Neven in [5] zusammenhängend aufgeführt.

Einschränkung	*NTA*	*DTA*	*DTD(NFA)*	*DTD(DFA)*	*DTD(SL)*
Allgemeiner Fall	EXPTIME	EXPTIME	EXPTIME	EXPTIME	EXPTIME
Nicht-löschend	EXPTIME	EXPTIME	PSPACE	PSPACE	CONP
Nicht-löschend + begrenztes Kopieren	EXPTIME	In EXPTIME/PSPACE-hard	PSPACE	PTIME	CONP

Abbildung 7: Ergebnisse

Literatur

[1] D. Wood A. Brüggemann-Klein, M. Murata. Regular tree and regular hedge languages over unranked alphabets. Technical report, The Hongkong University of Science and Technology, 3 April 2001.

[2] F. Neven N. Alon, T. Milo. XML with data values: typechecking revisited. *J. Comput. System Sci.*, 66(4):688–727, 2003.

[3] V. Vianu T.Milo, D.Suciu. Typechecking for XML transformers. *J. Comput. System Sci.*, 66(1):66–97, 2003.

[4] F. Neven W. Martens. Frontiers of tractability for typechecking simple XML transformations. *Proc. 23rd Sympos. Principles of Database Systems (PODS 2004)*, pages 23–34, 2004.

[5] F. Neven W. Martens. On the complexity of typechecking top.-down XML transformations. *Theoretical Computer Science*, 336:153–180, 2005.