XML Research for Formal Language Theorists

Wim Martens

TU Dortmund
Goal of this talk

XML Research vs Formal Languages

XML benefits from Formal Language Theory

XML schemas ≈ tree automata

XPath patterns ≈ regular expressions

Formal Language Theory has a nice algorithmic toolbox

Formal Language Theory benefits from XML

XML motivates interesting Formal Language problems

Warning

Rather informal strongly biased survey
Goal of this talk

XML Research vs Formal Languages

- XML benefits from Formal Language Theory
  - XML schemas ≈ tree automata
  - XPath patterns ≈ regular expressions
  - Formal Language Theory has a nice algorithmic toolbox

Warning
Rather informal strongly biased survey

Wim Martens (TU Dortmund)
Goal of this talk

XML Research vs Formal Languages

- XML benefits from Formal Language Theory
  - XML schemas $\approx$ tree automata
  - XPath patterns $\approx$ regular expressions
  - Formal Language Theory has a nice algorithmic toolbox
- Formal Language Theory benefits from XML
  - XML motivates interesting Formal Language problems

Warning

- Rather informal strongly biased survey
Outline

1. Introduction to XML

2. An FLT Approach to XML Research
   - Document Type Definitions
   - XML Queries
   - Extended Document Type Definitions and XML Schema
   - Characterizations of single-type EDTDs

3. From XML to Formal Language Theory
   - Complexity of Regular Expressions
   - Constructions on Regular Expressions
   - Automata Minimization
Outline

1. Introduction to XML

2. An FLT Approach to XML Research
   - Document Type Definitions
   - XML Queries
   - Extended Document Type Definitions and XML Schema
   - Characterizations of single-type EDTDs

3. From XML to Formal Language Theory
   - Complexity of Regular Expressions
   - Constructions on Regular Expressions
   - Automata Minimization
Searching the Internet

Enough with these sissy keyword searches!
A real search

Where can I buy a flatscreen-TV, in a store at most 20km from Dresden, that is open tomorrow until 18:00?
An Example
An Example
An Example

XML Schema

Internet

Wim Martens (TU Dortmund)
XML for Formal Language Theorists
May 14, 2008 6 / 65
A self-describing data format

<store>
  <normal>
    <guitar type="electric">
      <maker> Tandler </maker>
      <price> 3500 </price>
    </guitar>
    <guitar type="electric">
      <maker> Fender </maker>
      <price> 1000 </price>
    </guitar>
  </normal>
  <discount>
    <guitar type="electric">
      <maker> Gibson </maker>
      <price> 2500 </price>
      <discount> 10% </discount>
    </guitar>
  </discount>
</store>

element: <title>...</title>

start tag: <title>
end tag: </title>
XML as a hierarchical structure

Example

Abstraction: ordered, unranked, labeled tree (with data-values)
XML schema languages

Schema

A schema defines the set of allowable labels and the way they can be structured.

Advantages

- automatic validation
- automatic integration of data
- automatic translation
- query optimization
- provides a user with a concrete semantics of the document
- aids in the specification of meaningful queries over XML data
XML schema languages

In formal language theoretic terms
A schema defines a tree language.

Example
- DTDs (W3C)
- XML Schema (W3C)
- Relax NG (Clark, Murata)
- several dozen others (DSD, Schematron, . . . )

CFGs with REs
\(\not\approx\) tree automata
\(\approx\) tree automata
What to remember?

- XML is an international standard for data exchange
- XML documents or XML data are simply ordered unranked labeled trees with data values
- A schema defines a tree language (no data values — in this talk)
Outline

1 Introduction to XML

2 An FLT Approach to XML Research
   - Document Type Definitions
   - XML Queries
   - Extended Document Type Definitions and XML Schema
   - Characterizations of single-type EDTDs

3 From XML to Formal Language Theory
   - Complexity of Regular Expressions
   - Constructions on Regular Expressions
   - Automata Minimization
Outline

1 Introduction to XML

2 An FLT Approach to XML Research
   • Document Type Definitions
   • XML Queries
   • Extended Document Type Definitions and XML Schema
   • Characterizations of single-type EDTDs

3 From XML to Formal Language Theory
   • Complexity of Regular Expressions
   • Constructions on Regular Expressions
   • Automata Minimization
Example

```xml
<!DOCTYPE store [ 
  <!ELEMENT store (normal,discount)> 
  <!ELEMENT normal (guitar*)> 
  <!ELEMENT discount (guitar+)> 
  <!ELEMENT guitar (maker,price,discount?)> 
  <!ELEMENT maker (#PCDATA)> 
  <!ELEMENT price (#PCDATA)> 
  <!ELEMENT discount (#PCDATA)> 
]
```

Corresponding grammar (start symbol store)

```
store  →  normal discount
normal →  guitar*
discount →  guitar+
guitar  →  maker price discount?
maker  →  DATA
price  →  DATA
discount  →  DATA
```
Document Type Definitions (DTDs)

XML Document

```
store
   normal
      guitar
         maker “Tandler”
         price “3500”
   discount
      guitar
         maker “Gibson”
         price “2500”
         discount “10%”
      guitar
         maker “Fender”
         price “1000”
```

Corresponding grammar (start symbol store)

```
store → normal discount
normal → guitar
discount → guitar
  guitar → maker price discount?
maker → DATA
price → DATA
discount → DATA
```
Extended Context-free grammars as a formal abstraction

Definition

A DTD is a triple \((\Sigma, d, s_d)\) where

- \(\Sigma\) is a finite alphabet
- \(s_d \in \Sigma\) is the start symbol
- \(d : \Sigma \rightarrow \text{RE}(\Sigma)\) maps every \(\Sigma\)-symbol to a regular expression over \(\Sigma\)

Definition

A tree \(t\) satisfies \(d\) (is valid) iff

- the root of \(t\) is labeled \(s_d\)
- for every node \(v\) labeled \(a\) the string formed by the children of \(v\) belongs to \(d(a)\).
Schema containment ($\subseteq$)

Given: Schemas $d_1, d_2$

Question: Is $L(d_1) \subseteq L(d_2)$?
Schema containment ($\subseteq$)

Given: Schemas $d_1, d_2$
Question: Is $L(d_1) \subseteq L(d_2)$?

DTD containment reduces to containment of regular expressions

$$d_1 \subseteq d_2 \iff d_1(a) \subseteq d_2(a), \forall a \in \Sigma$$

(when $d_1$ and $d_2$ are reduced).
Schema containment ($\subseteq$)

Given: Schemas $d_1$, $d_2$
Question: Is $L(d_1) \subseteq L(d_2)$?

DTD containment reduces to containment of regular expressions

$$d_1 \subseteq d_2 \quad \text{iff} \quad d_1(a) \subseteq d_2(a), \forall a \in \Sigma$$

(when $d_1$ and $d_2$ are reduced).

Theorem (Meyer, Stockmeyer, 1973)

*Containment of regular expressions is $\mathsf{PSPACE}$-complete.*
Optimization questions: from FLT to XML

Schema containment ($\subseteq$)

Given: Schemas $d_1$, $d_2$
Question: Is $L(d_1) \subseteq L(d_2)$?

DTD containment reduces to containment of regular expressions

$$d_1 \subseteq d_2 \text{ iff } d_1(a) \subseteq d_2(a), \forall a \in \Sigma$$

(when $d_1$ and $d_2$ are reduced).

Theorem (Meyer, Stockmeyer, 1973)

Containment of regular expressions is \textbf{PSPACE}-complete.

Corollary

DTD containment is \textbf{PSPACE}-complete.
Outline

1 Introduction to XML

2 An FLT Approach to XML Research
   • Document Type Definitions
   • XML Queries
   • Extended Document Type Definitions and XML Schema
   • Characterizations of single-type EDTDs

3 From XML to Formal Language Theory
   • Complexity of Regular Expressions
   • Constructions on Regular Expressions
   • Automata Minimization
Queries for XML
Conjunctive Queries over Trees

**XPath**

Tree:
```
  a
 / \
 b   c
 |   |
 e   d
 |   |
 d   
```

Pattern:
```
  a
 / \
 b   c
 |   |
 d   d
```

**Pattern Matching**

- Tree matches Pattern if there is a homomorphism $h : \text{Pattern} \rightarrow \text{Tree}$
- Homomorphism *doesn’t have to be injective*
Queries for XML
Conjunctive Queries over Trees

XPath

Tree:  Pattern:

a  a
|   |
|   |
b  b  c
|   |
e  d  d
c  |   |
d  |

Pattern Matching

- Tree matches Pattern if there is a homomorphism \( h : \text{Pattern} \to \text{Tree} \)
- Homomorphism doesn’t have to be injective
Queries for XML
Conjunctive Queries over Trees

Conjunctive Queries over Trees

Tree: 

Pattern:

Tree matches Pattern if there is a homomorphism $h : \text{Pattern} \rightarrow \text{Tree}$

Homomorphism doesn’t have to be injective
Con conjunctive queries over Trees

Tree matches Pattern if there is a homomorphism \( h : \text{Pattern} \rightarrow \text{Tree} \)

Homomorphism doesn’t have to be injective
Conjunctive Queries over Trees

Tree matches Pattern if there is a homomorphism $h : \text{Pattern} \rightarrow \text{Tree}$

Homomorphism doesn’t have to be injective
Conjunctive Queries over Trees

Tree: 
```
  a
  | |
  b
  | |
  e
  | |
  c
  | |
  d
```

Pattern: 
```
  a
  ↩️ ↩️
  ↩️ ↩️
  b
  ↓  ↓
  ↩️ ↩️
  c
  ↓  ↓
  ↩️ ↩️
  d
```

Pattern Matching
- Tree matches Pattern if there is a homomorphism $h : \text{Pattern} \rightarrow \text{Tree}$
- Homomorphism doesn’t have to be injective
Query Optimization

$L(Q)$: the set of trees that match query $Q$

**Query Containment**

Given two queries $Q_1$ and $Q_2$, is $L(Q_1) \subseteq L(Q_2)$?

**Query Containment w.r.t. a DTD**

Given $Q_1$, $Q_2$, and a DTD $d$, is $L(Q_1) \cap L(d) \subseteq L(Q_2)$?
Lemma

For each XPath query $Q$ there is an Alternating Tree Automaton $A$ s.t.

$$L(Q) = L(A)$$
For each XPath query $Q$ there is an Alternating Tree Automaton $A$ s.t.

$$L(Q) = L(A)$$

Moreover, $|A|$ is polynomial in $|Q|$
For each XPath query $Q$ there is an Alternating Tree Automaton $A$ s.t.

$$L(Q) = L(A)$$

Moreover, $|A|$ is polynomial in $|Q|$, even if $Q$ uses disjunction and negation.
Lemma

For each XPath query $Q$ there is an Alternating Tree Automaton $A$ s.t.

$$L(Q) = L(A)$$

Moreover, $|A|$ is polynomial in $|Q|$, even if $Q$ uses disjunction and negation.

Theorem

- XPath Containment is in EXPTIME
- XPath Containment w.r.t. DTDs is in EXPTIME
**Lemma**

For each XPath query $Q$ there is an Alternating Tree Automaton $A$ s.t.

$$L(Q) = L(A)$$

Moreover, $|A|$ is polynomial in $|Q|$, even if $Q$ uses disjunction and negation.

**Theorem**

- XPath Containment (tree pattern fragment) is NP-complete [Miklau, Suciu 2002]
- XPath Containment (with $\neg$ and $\lor$) is EXPTIME-complete [Marx 2004]
- XPath Containment w.r.t. DTDs is EXPTIME-complete [Neven, Schwentick 2003]
Conjunctive Query Optimization
Formal Language Theory to the Rescue!

Lemma (Björklund, Mar., Schwentick 2008)

For each Conjunctive Query $Q$ there is an Alternating Tree Automaton $A$

such that

$L(Q) = L(A)$
Lemma (Björklund, Mar., Schwentick 2008)

For each Conjunctive Query $Q$ there is an Alternating Tree Automaton $A$ s.t.

$$L(Q) = L(A)$$

But, $|A|$ is exponential in $|Q|$
Lemma (Björklund, Mar., Schwentick 2008)

For each Conjunctive Query $Q$ there is an Alternating Tree Automaton $A$ s.t.

$$L(Q) = L(A)$$

But, $|A|$ is exponential in $|Q|$ and this is optimal.
Lemma (Björklund, Mar., Schwentick 2008)

For each Conjunctive Query $Q$ there is an Alternating Tree Automaton $A$ s.t.

$$L(Q) = L(A)$$

But, $|A|$ is exponential in $|Q|$ and this is optimal

Theorem

- CQ Containment w.r.t. DTDs is $\text{2EXPTIME}$-complete
  
  [Björklund, Mar., Schwentick 2008]
Lemma (Björklund, Mar., Schwentick 2008)

For each Conjunctive Query $Q$ there is an Alternating Tree Automaton $A$ s.t.

$$L(Q) = L(A)$$

But, $|A|$ is exponential in $|Q|$ and this is optimal

Theorem

- CQ Containment is $\Pi^P_2$-complete [Björklund, Mar., Schwentick 2007]
- CQ Containment w.r.t. DTDs is 2EXPTIME-complete [Björklund, Mar., Schwentick 2008]
Outline

1. Introduction to XML

2. An FLT Approach to XML Research
   - Document Type Definitions
   - XML Queries
   - Extended Document Type Definitions and XML Schema
   - Characterizations of single-type EDTDs

3. From XML to Formal Language Theory
   - Complexity of Regular Expressions
   - Constructions on Regular Expressions
   - Automata Minimization
Extended DTDs
Grammar based approach to unranked regular tree languages

**Example**

```
store    →   (guitar¹)* (guitar²)⁺
guitar¹  →   maker price
guitar²  →   maker price discount
```
Extended DTDs
Grammar based approach to unranked regular tree languages

Typed tree $t'$

Example

store $\rightarrow$ (guitar$^1$)* (guitar$^2$)$^+$
guitar$^1$ $\rightarrow$ maker price
guitar$^2$ $\rightarrow$ maker price discount
Definition (Papakonstantinou, Vianu, 2000)

Let $\Sigma^\mathbb{N} := \{\sigma^n \mid \sigma \in \Sigma, n \in \mathbb{N}\}$ be the alphabet of types.

An extended DTD (EDTD) is a tuple $D = (\Sigma, d, s_d)$, where $(d, s_d)$ is a (finite) DTD over $\Sigma \cup \Sigma^\mathbb{N}$.

A tree $t$ is valid w.r.t. $D$ if there is an assignment of types such that the typed tree is a derivation tree of $d$.

Example

store $\rightarrow$ (guitar$^1$)* (guitar$^2$)$^+$

\begin{align*}
guitar^1 & \rightarrow \text{maker price} \\
guitar^2 & \rightarrow \text{maker price discount}
\end{align*}
EDTDs versus Tree Automata

Theorem (Papakonstantinou, Vianu, 2000, BMW)

Non-deterministic (unranked) tree automata and EDTDs define precisely the class of (homogeneous) regular unranked tree languages.
EDTDs versus Tree Automata

Theorem (Papakonstantinou, Vianu, 2000, BMW)

Non-deterministic (unranked) tree automata and EDTDs define precisely the class of (homogeneous) regular unranked tree languages.

Example

EDTD

store → (guitar\(^1\))\(^*\) (guitar\(^2\))\(^+\)
guitar\(^1\) → maker price
guitar\(^2\) → maker price discount

NTA

\[ \delta(\text{store, store}) = (\text{guitar}^1)^* (\text{guitar}^2)^+ \]
\[ \delta(\text{guitar}^1, \text{guitar}) = \text{maker price} \]
\[ \delta(\text{guitar}^2, \text{guitar}) = \text{maker price discount} \]
<xs:element name="store">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="guitar" type="1"
                  minOccurs="0"
                  maxOccurs="unbounded"/>
      <xs:element name="guitar" type="2"
                  minOccurs="1"
                  maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
Does XML Schema correspond to EDTDs?

rejected by XML Schema validator

Violates the Element Declarations Consistent Constraint.

minOccurs="1"
maxOccurs="unbounded"/>
XML Schema 1: Element Declarations Consistent constraint (Section 3.8.6)

It is illegal to have two elements of the same name […] but different types in a content model […].
A formalization of XML Schema: single-type EDTDs

**XML Schema 1: Element Declarations Consistent constraint (Section 3.8.6)**

It is illegal to have two elements of the same name [...] but different types in a content model [...].

**Definition (Murata, Lee, Mani, 2001)**

A single-type EDTD is an EDTD for which in no regular expression two types $b^i$ and $b^j$ with $i \neq j$ occur.
**A formalization of XML Schema: single-type EDTDs**

**XML Schema 1: Element Declarations Consistent constraint (Section 3.8.6)**

It is illegal to have two elements of the same name [...] but different types in a content model [...].

**Definition (Murata, Lee, Mani, 2001)**

A single-type EDTD is an EDTD for which in no regular expression two types $b^i$ and $b^j$ with $i \neq j$ occur.

**Not single-type**

- `store` $\rightarrow$ `(guitar^1)^* (guitar^2)^+`
- `guitar^1` $\rightarrow$ `maker price`
- `guitar^2` $\rightarrow$ `maker price discount`
A single-type EDTD is an EDTD in which in no regular expression two types $b^i$ and $b^j$ with $i \neq j$ occur.

Example

<table>
<thead>
<tr>
<th>Store</th>
<th>normal discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>(guitar$^1$)$^*$</td>
</tr>
<tr>
<td>Discount</td>
<td>(guitar$^2$)$^+$</td>
</tr>
<tr>
<td>Guitar$^1$</td>
<td>maker price</td>
</tr>
<tr>
<td>Guitar$^2$</td>
<td>maker price discount</td>
</tr>
</tbody>
</table>
A formalization of XML Schema: single-type EDTDs

Formal abstraction

XML Schema \(\approx\) single-type EDTDs
A formalization of XML Schema: single-type EDTDs

Formal abstraction

XML Schema \approx \text{single-type EDTDs}

Immediate Questions

- What kind of languages can be defined by single-type EDTDs?
- Is it decidable whether an EDTD rewritten to an equivalent single-type EDTD?
1. Introduction to XML

2. An FLT Approach to XML Research
   - Document Type Definitions
   - XML Queries
   - Extended Document Type Definitions and XML Schema
   - Characterizations of single-type EDTDs

3. From XML to Formal Language Theory
   - Complexity of Regular Expressions
   - Constructions on Regular Expressions
   - Automata Minimization
Properties of single-type EDTDs

Three properties

1. Single-type EDTDs admit unique top-down typing
2. Closure under a certain form of subtree exchange
3. Characterization as a pattern-based language
(1) Single-type EDTDs: simple top-down typing

Example

store → normal discount
normal → (guitar¹)*
discount → (guitar²)+
guitar¹ → maker price
guitar² → maker price discount
(1) Single-type EDTDs: simple top-down typing

Example

store → normal discount
normal → (guitar\(^1\))^* 
discount → (guitar\(^2\))^+
guitar\(^1\) → maker price
guitar\(^2\) → maker price discount
(1) Single-type EDTDs: simple top-down typing

Example

store → normal discount
normal → (guitar\(^1\))^* 
discount → (guitar\(^2\))^+ 
guitar\(^1\) → maker price 
guitar\(^2\) → maker price discount
(1) Single-type EDTDs: simple top-down typing

Algorithm to validate and type a tree (Murata et al., 2001)

Given: tree $t$ and single-type EDTD $D = (\Sigma, d, a^0)$

1. Check if root of $t$ is labeled with $a$, assign type $a^0$
2. For every interior node $u$ with type $b^i$, test whether the children of $u$ match $\mu(d(b^i))$. If so, assign unique type to every child. Else fail.

$$\mu(a^1 + b^1c^2) = a + bc$$
Single-type EDTDs: simple top-down typing

Algorithm to validate and type a tree (Murata et al., 2001)

Given: tree \( t \) and single-type EDTD \( D = (\Sigma, d, a^0) \)

1. Check if root of \( t \) is labeled with \( a \), assign type \( a^0 \)
2. for every interior node \( u \) with type \( b^i \), test whether the children of \( u \) match \( \mu(d(b^i)) \). If so, assign unique type to every child. Else fail.

\[
\mu(a^1 + b^1c^2) = a + bc
\]

Corollary

Single-typedness implies unique top-down typing.
(2) An exchange property of single-type EDTDs

The Ancestor-String
(2) An exchange property for single-type EDTDs

Ancestor-Guarded Subtree Exchange

$T$ is a regular tree language

\[ \in T \quad \in T \quad \Rightarrow \quad \in T \]

Theorem (Mar., Neven, Schwentick 2005)

A regular tree language is definable by a single-type EDTD iff it is closed under ancestor-guarded subtree exchange.
(2) Tool for proving inexpressibility

“At least one discount guitar” is not single-type

```
store
   | guitar
   |   | maker
   |   | price
   |   | “Tandler” “3500”
   |   | maker
   |   | price
   |   | “Gibson” “2500” “10%”
```

```
store
   | guitar
   |   | maker
   |   | price
   |   | discount
   |   | “Fender” “1000” “10%”
   |   | maker
   |   | price
   |   | “Gibson” “2500”
```
“At least one discount guitar” is not single-type
(2) **Tool for proving inexpressibility**

“**At least one discount guitar**” is not single-type

Single-type EDTDs are **not** closed under union or complement.
(3) Pattern-based Language
Making dependencies explicit

Definition
An ancestor-based DTD $A$ is a set of rules $r \rightarrow s$ where $r$ and $s$ are regular expressions over $\Sigma$.

Definition
A tree $t$ is valid w.r.t. $A$ iff for every vertex $v$ there is some $r \rightarrow s$ such that $v$'s ancestor string matches $r$ and the children of $v$ match $s$. 
(3) Pattern-based Language
Making dependencies explicit

**single-type EDTD**

<table>
<thead>
<tr>
<th>Store</th>
<th>→</th>
<th>Normal discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>→</td>
<td>(guitar(^1))^*</td>
</tr>
<tr>
<td>Discount</td>
<td>→</td>
<td>(guitar(^2))^+</td>
</tr>
<tr>
<td>Guitar(^1)</td>
<td>→</td>
<td>Maker price</td>
</tr>
<tr>
<td>Guitar(^2)</td>
<td>→</td>
<td>Maker price discount</td>
</tr>
</tbody>
</table>

**Ancestor-guarded DTD**

<table>
<thead>
<tr>
<th>Store</th>
<th>→</th>
<th>Normal discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>→</td>
<td>Guitar^*</td>
</tr>
<tr>
<td>Discount</td>
<td>→</td>
<td>Guitar^+</td>
</tr>
<tr>
<td>*· Normal · Guitar</td>
<td>→</td>
<td>Maker price</td>
</tr>
<tr>
<td>*· Discount · Guitar</td>
<td>→</td>
<td>Maker price discount</td>
</tr>
</tbody>
</table>
Theorem (Mar., Neven, Schwentick, 2005)

Deciding whether an EDTD is equivalent to a single-type EDTD or a DTD is \textbf{EXPTIME}-complete.

Upper bound

Compute single-type closure $D'$ of given EDTD $D$:
E.g, $a^1 \rightarrow b^1 b^2$, $b^1 \rightarrow c^1$, $b^2 \rightarrow c^2$ becomes

$a^{\{1\}} \rightarrow b^{\{1,2\}} b^{\{1,2\}}$

$b^{\{1,2\}} \rightarrow c^{\{1\}} + c^{\{2\}}$

$L(D') = L(D)$ iff $L(D)$ is single-type.
We know that $L(D) \subseteq L(D')$.
So, only need to test $L(D') \subseteq L(D)$: $D' \cap \neg D = \emptyset$. 
Theorem (Mar., Neven, Schwentick, 2005)

Deciding whether an EDTD is equivalent to a single-type EDTD or a DTD is **EXPTIME**-complete.

**Upper bound**

Compute single-type closure $D'$ of given EDTD $D$:
E.g, $a^1 \rightarrow b^1 b^2$, $b^1 \rightarrow c^1$, $b^2 \rightarrow c^2$ becomes

\[
\begin{align*}
    a^{\{1\}} & \rightarrow \ b^{\{1,2\}} \ b^{\{1,2\}} \\
    b^{\{1,2\}} & \rightarrow \ c^{\{1,2\}} + c^{\{1,2\}}
\end{align*}
\]

$L(D') = L(D)$ iff $L(D)$ is single-type.
We know that $L(D) \subseteq L(D')$.
So, only need to test $L(D') \subseteq L(D)$: $D' \cap \neg D = \emptyset$. 

Wim Martens  (TU Dortmund)  XML for Formal Language Theorists  May 14, 2008  42 / 65
What to remember?

- XML Schema $\approx$ single-type EDTDs $\subseteq$ regular tree languages
- single-type EDTDs admit top-down unique typing
- XML Schema can be simply characterized without using types
- Relax NG corresponds to unranked regular tree languages (EDTDs)
Outline

1. Introduction to XML

2. An FLT Approach to XML Research
   - Document Type Definitions
   - XML Queries
   - Extended Document Type Definitions and XML Schema
   - Characterizations of single-type EDTDs

3. From XML to Formal Language Theory
   - Complexity of Regular Expressions
   - Constructions on Regular Expressions
   - Automata Minimization
Outline

1. Introduction to XML

2. An FLT Approach to XML Research
   - Document Type Definitions
   - XML Queries
   - Extended Document Type Definitions and XML Schema
   - Characterizations of single-type EDTDs

3. From XML to Formal Language Theory
   - Complexity of Regular Expressions
   - Constructions on Regular Expressions
   - Automata Minimization
Theorem (Mar., Neven, Schwentick 2004)

Let $R$ be a class of regular expressions and $\mathcal{C}$ a complexity class. Then the following are equivalent:

- **CONTAINMENT** for $R$ is in $\mathcal{C}$;
- **CONTAINMENT** for $\text{DTD}(R)$ is in $\mathcal{C}$;
- **CONTAINMENT** for single-type $\text{EDTD}(R)$ is in $\mathcal{C}$;

Theorem (Seidl 1990, 1994)

**CONTAINMENT** and **EQUIVALENCE** are EXPTIME-complete for $\text{EDTDs}$ (even with deterministic REs).
Complexity of basic decision problems

**INTERSECTION**: Given a number of schemas $S_1, \ldots, S_n$, decide if $\bigcap_{i=1}^n L(S_i) \neq \emptyset$.

Theorem (Mar., Neven, Schwentick 2004)

Let $R$ be a class of regular expressions and $\mathcal{C}$ a complexity class. Then the following are equivalent:

- **INTERSECTION** for $R$ is in $\mathcal{C}$;
- **INTERSECTION** for $DTD(R)$ is in $\mathcal{C}$.

Remark: **INTERSECTION** for deterministic REs is $PSPACE$-complete.
**Complexity of basic decision problems**

**INTERSECTION**: Given a number of schemas $S_1, \ldots, S_n$, decide if $\bigcap_{i=1}^{n} L(S_i) \neq \emptyset$.

---

**Theorem (Mar., Neven, Schwentick 2004)**

Let $R$ be a class of regular expressions and $C$ a complexity class. Then the following are equivalent:

- **INTERSECTION** for $R$ is in $C$;
- **INTERSECTION** for $DTD(R)$ is in $C$.

---

**Theorem (Mar., Neven, Schwentick 2004)**

There is a class of regular expressions $\mathcal{X}$ such that

- **INTERSECTION** for $\mathcal{X}$ is NP-complete;
- **INTERSECTION** for single-type $EDTD(\mathcal{X})$ is EXPTIME-complete.

---

**Remark**: **INTERSECTION** for deterministic REs is PSPACE-complete.
Focus on Regular Expressions

What to remember?

- Decision problems for XML Schema translate to decision problems for regular expressions.
Focus on Regular Expressions

What to remember?

- Decision problems for XML Schema translate to decision problems for regular expressions.

What regular expression classes are interesting?

Regular expressions that occur in schemas!

- A **base symbol** is a regular expression $w$, $w\?$, or $w^*$ where $w$ is a non-empty string;
- A **factor** is of the form $e$, $e\?$, $e^+$, or $e^*$ where $e$ is a disjunction of base symbols.
- A **CHAin Regular Expression (CHARE)** is $\varepsilon$, $\emptyset$, or a sequence $f_1 \cdots f_k$ of factors.

[Bex, Neven, Van den Bussche 2004]: $>90\%$ of expressions in practical DTDs or XSDs are **CHAREs**
<table>
<thead>
<tr>
<th>Fragment</th>
<th>CONTAINMENT</th>
<th>EQUIVALENCE</th>
<th>INTERSECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a, a^+$</td>
<td>in PTIME (DFA!)</td>
<td>in PTIME</td>
<td>in PTIME</td>
</tr>
<tr>
<td>$a, a^*$</td>
<td>coNP</td>
<td>in PTIME</td>
<td>NP</td>
</tr>
<tr>
<td>$a, a?$</td>
<td>coNP</td>
<td>in PTIME</td>
<td>NP</td>
</tr>
<tr>
<td>$a, (a)^*$</td>
<td>PSPACE</td>
<td>in PSPACE</td>
<td>NP</td>
</tr>
<tr>
<td>$all - {(+w)^*, (+w)^+}$</td>
<td>PSPACE</td>
<td>in PSPACE</td>
<td>PSPACE [Bala 2002]</td>
</tr>
<tr>
<td>$a, (+w)^*$</td>
<td>PSPACE</td>
<td>in PSPACE</td>
<td>PSPACE</td>
</tr>
<tr>
<td>RE</td>
<td>PSPACE</td>
<td>PSPACE</td>
<td>PSPACE</td>
</tr>
</tbody>
</table>
### Regular Expression Analysis Revisited

<table>
<thead>
<tr>
<th>Fragment</th>
<th>CONTAINMENT</th>
<th>EQUIVALENCE</th>
<th>INTERSECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a, a^+$</td>
<td>in PTIME (DFA!)</td>
<td>in PTIME</td>
<td>in PTIME</td>
</tr>
<tr>
<td>$a, a^*$</td>
<td>coNP</td>
<td>in PTIME</td>
<td>NP</td>
</tr>
<tr>
<td>$a, a?$</td>
<td>coNP</td>
<td>in PTIME</td>
<td>NP</td>
</tr>
<tr>
<td>$a, (a)^*$</td>
<td>PSPACE</td>
<td>in PSPACE</td>
<td>NP</td>
</tr>
<tr>
<td>$all - {(+w)^*, (+w)^+}$</td>
<td>PSPACE</td>
<td>in PSPACE</td>
<td>PSPACE [Bala 2002]</td>
</tr>
<tr>
<td>$a, (w)^*$</td>
<td>PSPACE</td>
<td>in PSPACE</td>
<td>PSPACE</td>
</tr>
<tr>
<td>RE</td>
<td>PSPACE</td>
<td>PSPACE</td>
<td>PSPACE</td>
</tr>
</tbody>
</table>

### Observation

Not many **PTIME** results...
What Regular Expressions are Allowed in Schemas?

Counting and shuffle

- Numerical occurrence operator (#): \((a^{[4,5]}(b + c^*)^7)\)
- shuffle operator \((a \& b = \{ab, ba\})\)

Theorem (Mayer, Stockmeyer 1994)

**CONTAINMENT and EQUIVALENCE** for \(RE(\&)\) is **EXPSPACE-complete**
What Regular Expressions are Allowed in Schemas?

Counting and shuffle

- Numerical occurrence operator (\#): \(a^{[4,5]}(b + c^*)^7\)
- Shuffle operator \((a\&b = \{ab, ba\})\)

Theorem (Mayer, Stockmeyer 1994)

**CONTAINMENT** and **EQUIVALENCE** for RE(\&) is **EXPSPACE-complete**

Theorem (Gelade, Mar., Neven 2007)

**CONTAINMENT** and **EQUIVALENCE** is **EXPSPACE-complete** for

- RE(\#) and
- RE(\#,\&)

Wim Martens (TU Dortmund)  XML for Formal Language Theorists  May 14, 2008  50 / 65
On the Search for more **PTIME** fragments

**Theorem (Ghelli, Colazzo, Sartiani 2007)**

**CONTAINMENT** *is in PTIME* for conflict-free regular expressions

**Conflict-free**

- counting and interleaving allowed!
Theorem (Ghelli, Colazzo, Sartiani 2007)

CONTAIENMENT is in PTIME for conflict-free regular expressions

Conflict-free

- counting and interleaving allowed!
- single occurrence
- Kleene star only applied to disjunctions single symbols
Outline

1. Introduction to XML

2. An FLT Approach to XML Research
   - Document Type Definitions
   - XML Queries
   - Extended Document Type Definitions and XML Schema
   - Characterizations of single-type EDTDs

3. From XML to Formal Language Theory
   - Complexity of Regular Expressions
   - Constructions on Regular Expressions
   - Automata Minimization
Complementing schemas

Schema Complementation

- I have a schema $S$ which I update to $S'$
- What are the documents I admitted in $S$, but not in $S'$ anymore?

This should be $L(S) - L(S') = L(S) \cap \overline{L(S')}$
Complementing regular expressions

Given a regular expression \( r \), define a regexp for \( \overline{L(r)} \).

Naive approach: transform to an NFA, determinize, complement, and transform again to a regular expression (2EXPTIME)
Complementing regular expressions

Given a regular expression $r$, define a regexp for $\overline{L(r)}$.

Naive approach: transform to an NFA, determinize, complement, and transform again to a regular expression (2EXPTIME)

Lemma [Gelade and Neven 2008]

For every $n$, there is a regular expression $r$ of size $\mathcal{O}(n)$, such that any regular expression defining $\overline{L(r)}$ must be of size $\Omega(2^{2^n})$
Complementing regular expressions

Given a regular expression \( r \), define a regexp for \( \overline{L(r)} \).

Naive approach: transform to an NFA, determinize, complement, and transform again to a regular expression (2EXPTIME)

Lemma [Gelade and Neven 2008]
For every \( n \), there is a regular expression \( r \) of size \( O(n) \), such that any regular expression defining \( \overline{L(r)} \) must be of size \( \Omega(2^{2n}) \)

Idea
- Ehrenfeucht, Zeiger (1974): There is a class of DFAs \( K_n \) whose smallest equivalent regular expression is at least \( 2^n \). (States = \{1, \ldots, n\}, edges between \( i \) and \( j \) labeled with \( a_{i,j} \))
- Generalize this theorem to four-letter alphabets
- Construct \( r \) of size \( O(n) \) for \( \overline{K_{2^n}} \)
Outline

1. Introduction to XML

2. An FLT Approach to XML Research
   - Document Type Definitions
   - XML Queries
   - Extended Document Type Definitions and XML Schema
   - Characterizations of single-type EDTDs

3. From XML to Formal Language Theory
   - Complexity of Regular Expressions
   - Constructions on Regular Expressions
   - Automata Minimization
Schema Minimization

Given a schema $D$, compute the smallest equivalent schema $D'$

Why relevant?

- Recall: Query Optimization
- Input: Queries $Q_1$, $Q_2$, and a schema $D$

Smaller schema improves the run-time of the query optimization problems!
Minimization is typically studied on automata models.
Minimization is typically studied on automata models
and the results look prettier on deterministic automata
Minimization is typically studied on automata models and the results look prettier on deterministic automata.

Question

What’s the deterministic automata model for XML?

- single-type EDTDs with DFAs?
- deterministic unranked tree automata?
Minimization is typically studied on automata models and the results look prettier on deterministic automata.

**Question**

What’s the deterministic automata model for XML?

- single-type EDTDs with DFAs? \(\approx\) top-down det.
- deterministic unranked tree automata? \(\approx\) bottom-up det.
Theorem (Mar., Niehren 2005)

- **Single-type EDTD with DFA Minimization is in PTIME**
- **Minimal models are unique**

Minimization Algorithm

Reduce the input single-type EDTD
For every pair of states $q_1, q_2$, decide equivalence
If equivalent, merge $q_1$ and $q_2$
In the resulting EDTD, minimize each DFA
A bottom-up unranked tree automaton is *deterministic* if for every pair of rules \( a(L_1) \rightarrow q_1 \) and \( a(L_2) \rightarrow q_2 \),

\[
L_1 \cap L_2 = \emptyset
\]

Additional requirement: \( L_1, L_2 \) represented by DFAs

**Theorem (Mar., Niehren 2005)**

MINIMIZATION is *NP*-complete for deterministic unranked tree automata
Unranked Tree Automaton Minimization

For the right definition of bottom-up determinism:

Theorem (Mar., Niehren 2005)

- **MINIMIZATION** is in P for bottom-up deterministic tree automata
- the Myhill-Nerode theorem for unranked tree languages holds
For tree language $L$, define relation $\equiv_L$ on trees

**Definition**

$$t_1 \equiv_L t_2 \text{ if } \forall E : E \cdot t_1 \in L \iff E \cdot t_2 \in L$$

$\equiv_L$ is an equivalence relation on unranked trees.
Theorem (Myhill-Nerode for Unranked Trees (Mar., Niehren 2005))

Let $L$ be an unranked tree language. The following are equivalent:

- $L$ is regular
- $\equiv_L$ has finitely many equivalence classes

Moreover, the equivalence classes of $\equiv_L$ correspond to states of minimal (new) bottom-up deterministic unranked TA for $L$
Back to the Basics

NFA Minimization

Question
How much non-determinism can be admitted for PTIME minimization?

Theorem (Jiang, Ravikumar 1993)

\[ \text{DFA} \rightarrow \text{unambiguous FA} \]

MINIMIZATION is NP-complete

Theorem (Malcher 2003)

MINIMIZATION is NP-complete for NFAs with fixed branching (\( \geq 3 \))

NFAs with at least two start states

Question Revisited
Can there be any non-determinism at all?
Question

How much non-determinism can be admitted for \textbf{PTIME} minimization?
Question
How much non-determinism can be admitted for PTIME minimization?

Theorem (Jiang, Ravikumar 1993)
DFA → unambiguous FA MINIMIZATION is NP-complete

Theorem (Malcher 2003)
MINIMIZATION is NP-complete for
- NFAs with fixed branching (≥ 3)
- NFAs with at least two start states
Question
How much non-determinism can be admitted for PTIME minimization?

Theorem (Jiang, Ravikumar 1993)
\(DFA \rightarrow \text{unambiguous FA} \) MINIMIZATION is NP-complete

Theorem (Malcher 2003)
MINIMIZATION is NP-complete for
- NFAs with fixed branching \((\geq 3)\)
- NFAs with at least two start states

Question Revisited
Can there be any non-determinism at all?
Definition ($\delta$NFA)

The class of NFAs that

- have at most one pair $(q, a)$ such that $(q, a) \rightarrow q_1$ and $(q, a) \rightarrow q_2$
- are unambiguous
- do not loop
Definition ($\delta$NFA)

The class of NFAs that
- have at most one pair $(q, a)$ such that $(q, a) \rightarrow q_1$ and $(q, a) \rightarrow q_2$
- are unambiguous
- do not loop

Theorem (Björklund, Mar., ICALP 2008)

For every class $\mathcal{C}$ of NFAs such that $\delta$NFA $\subseteq \mathcal{C}$:

$$DFA \rightarrow \mathcal{C} \text{ MINIMIZATION is NP-hard}$$
XML and Formal Languages are great for cross-fertilization

- Many problems in XML research are solved through FLT techniques
- XML research poses interesting questions for FLT
XML and Formal Languages are great for cross-fertilization

- Many problems in XML research are solved through FLT techniques
- XML research poses interesting questions for FLT

So, …

- if you like formal language theory, but also want a PODS/ICDT paper have a look at XML
- if you like formal language theory, and you want more formal language theory have a look at XML