

Tree automata and XML

Frank Neven¹

¹Theoretical Computer Science Group
Hasselt University
Agoralaan, 3590 Diepenbeek, Belgium

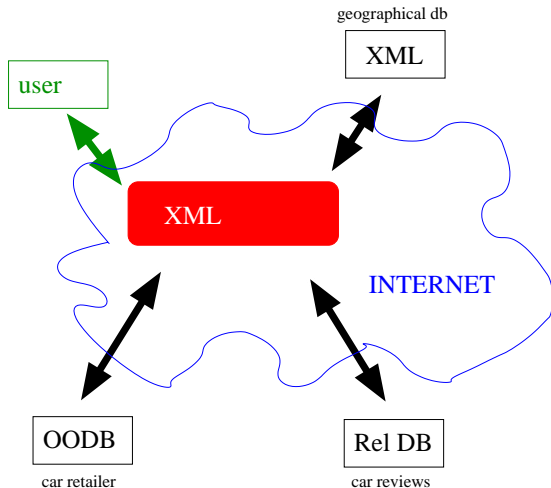
June 8, 2006

Outline

- 1 Introduction to XML
- 2 Document Type Definitions
- 3 Extended Document Type Definitions and XML Schema
- 4 Characterizations of single-type EDTDs
- 5 1-pass preorder typing
- 6 Single-type EDTDs in practice: ancestor-guarded DTDs
- 7 Applications of ancestor-guarded DTDs
 - Optimization problems
 - Learning of schema's
- 8 Conclusion

XML is a data exchange format

W3C standard



A self-describing data format

Example

```
<store>
  <dvd>
    <title> Fabuleux destin d'Amelie </title>
    <price> 17 </price>
  </dvd>
  <dvd>
    <title> Goodbye Lenin </title>
    <price> 20 </price>
    <discount> 4 </discount>
  </dvd>
</store>
```

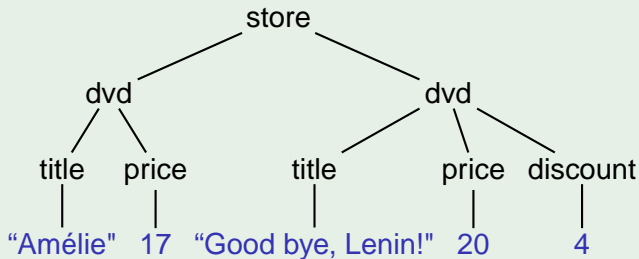
start tag: <title>

end tag: </title>

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

XML as a hierarchical structure

Example



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

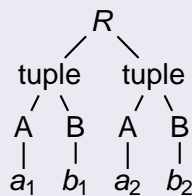
Flexibility of XML

Representation of the relational model

Relation

R	A	B
	a_1	b_1
	a_2	b_2

XML Tree



XML encoding

```

<R>
  <tuple>
    <A> a1 </A>
    <B> b1 </B>
  </tuple>
  <tuple>
    <A> a2 </A>
    <B> b2 </B>
  </tuple>
</R>
  
```

XML schema languages

Schema

A **schema** defines the set of allowable tags 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) | CFGs with REs |
| ● XML Schema (W3C) | \neq tree automata |
| ● Relax NG (Clark, Murata) | \approx tree automata |
| ● several dozen others (DSD, Schematron, ...) | |

Summary slide

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)

Summary slide

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)

Focus of this talk

- Grammars and tree automata as a formal model for XML schema languages
- ⊖ Highly biased non-technical overview
- ⊕ Cross fertilization between automata theory and XML in practice

Outline

- 1 Introduction to XML
- 2 Document Type Definitions
- 3 Extended Document Type Definitions and XML Schema
- 4 Characterizations of single-type EDTDs
- 5 1-pass preorder typing
- 6 Single-type EDTDs in practice: ancestor-guarded DTDs
- 7 Applications of ancestor-guarded DTDs
 - Optimization problems
 - Learning of schema's
- 8 Conclusion

Outline

- 1 Introduction to XML
- 2 Document Type Definitions**
- 3 Extended Document Type Definitions and XML Schema
- 4 Characterizations of single-type EDTDs
- 5 1-pass preorder typing
- 6 Single-type EDTDs in practice: ancestor-guarded DTDs
- 7 Applications of ancestor-guarded DTDs
 - Optimization problems
 - Learning of schema's
- 8 Conclusion

Document Type Definitions (DTDs)

Example

```

<!DOCTYPE store [
  <!ELEMENT store      (dvd,dvd*)>
  <!ELEMENT dvd        (title,price,discount?)>
  <!ELEMENT title      (#PCDATA)>
  <!ELEMENT price      (#PCDATA)>
  <!ELEMENT discount   (#PCDATA)>
]>

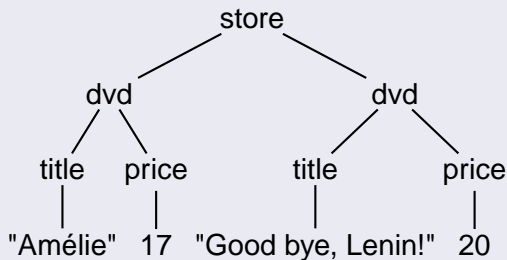
```

Corresponding grammar (start symbol store)

store	→	dvd dvd*
dvd	→	title price(discount + ϵ)
title	→	DATA
price	→	DATA

Document Type Definitions (DTDs)

XML Document



Corresponding grammar (start symbol store)

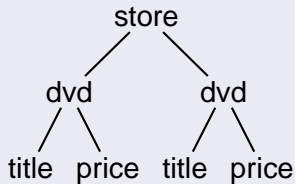
```

store    → dvd dvd*
dvd     → title price(discount + ε)
title   → DATA
price   → DATA
discount → DATA
  
```

Document Type Definitions (DTDs)

No data values

XML Document



Corresponding grammar (start symbol store)

store \rightarrow dvd dvd^{*}
dvd \rightarrow title price(discount + ϵ)

Extended Context-free grammars as a formal abstraction

Definition

A **DTD** is a pair (d, s_d) where

- $s_d \in \Sigma$ is the start symbol
- d maps every Σ -symbol to a regular expression over Σ

Definition

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

- the root of t is labeled s_d
- for every vertex v labeled a the string formed by the children of v belongs to $d(a)$.

DTD validator

Optimization questions: from FLT to XML

Schema containment (\subseteq)

Given: schema's d_1, d_2

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

Optimization questions: from FLT to XML

Schema containment (\subseteq)

Given: schema's 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 trimmed).

Optimization questions: from FLT to XML

Schema containment (\subseteq)

Given: schema's 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 trimmed).

Theorem (Meyer, Stockmeyer, 1973)

*Containment of regular expressions is **PSPACE**-complete.*

Optimization questions: from FLT to XML

Schema containment (\subseteq)

Given: schema's 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 trimmed).

Theorem (Meyer, Stockmeyer, 1973)

Containment of regular expressions is **PSPACE**-complete.

Corollary

DTD containment is **PSPACE**-complete.

Regular Expressions in DTDs Should Be Deterministic

How accurate is our abstraction?

Backward compatibility with SGML

The XML specification requires regular expressions to be deterministic: *for every input symbol in the input string we can uniquely determine by which symbol in the regular expression it should match without looking ahead in the input string.*

Example

- The expression $(a + b)^* a$ is **not** deterministic.
Counterexample: *baa*.
- The expression $b^* a(b^* a)^*$ is deterministic.

Why this restriction?

Regular Expressions in DTDs Should Be Deterministic

Relevant questions

- 1 How do we recognize deterministic regular expressions?
DTD validator
- 2 Can every regular language be denoted by a deterministic regular expression?
- 3 Are deterministic regular languages a robust class?
- 4 If a regular expression is not deterministic, can you find an equivalent one that is?
smart DTD validator

Formalization by Brüggemann-Klein and Wood [1998]

From XML to FLT

Definition

- A **marking** r' of a regular expression r is an assignment of numbers to every symbol in r .

Example

- $(a_1 + b_2)^* a_3$ is a marking of $(a + b)^* a$

Formalization by Brüggemann-Klein and Wood [1998]

From XML to FLT

Definition

- A **marking** r' of a regular expression r is an assignment of numbers to every symbol in r .
- For $w \in L(r')$, we denote by $w^\#$ the corresponding unmarked string in $L(r)$.

Example

- $(a_1 + b_2)^* a_3$ is a marking of $(a + b)^* a$
- For $w = b_2 a_1 a_3$, $w^\# = baa$

Formalization by Brüggemann-Klein and Wood [1998]

From XML to FLT

Definition

A regular expression r is **deterministic** (one-unambiguous) iff there are no strings $uxv, uyw \in L(r')$ with

- $|x| = |y| = 1$,
- $x \neq y$, (x and y are different marked symbols)
- $x^\# = y^\#$ (their unmarking is the same).

Example

$(a + b)^* a$ is not deterministic: $((a_1 + b_2)^* a_3)$

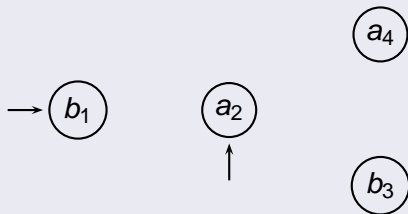
take $u \ x \ v$ and $u \ y \ w$
 $b_2 \ a_1 \ a_3$ $b_2 \ a_3 \ \varepsilon$

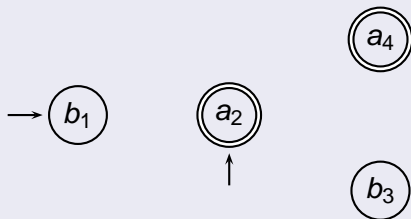
Tool

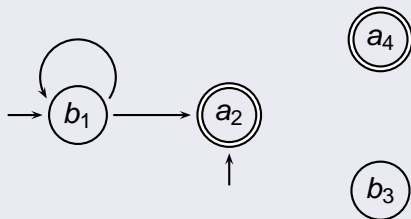
Glushkov construction preserves one-step unambiguity

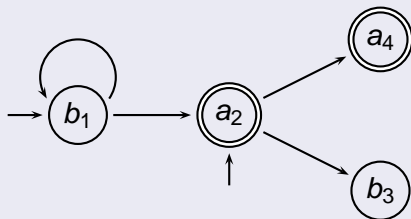
Glushkov automaton for $b^*a(b^*a)^*$

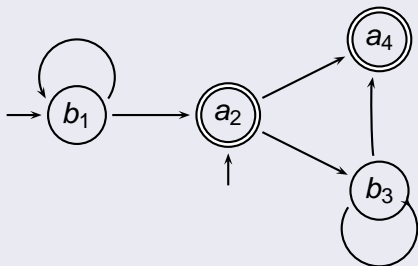
Glushkov automaton for $b^* a (b^* a)^*$ $b_1^* a_2 (b_3^* a_4)^*$ b_1 a_2 a_4 b_3

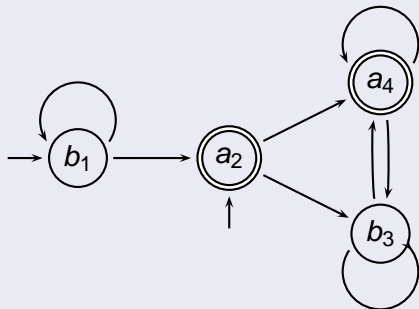
Glushkov automaton for $b^* a (b^* a)^*$ $b_1^* a_2 (b_3^* a_4)^*$ 

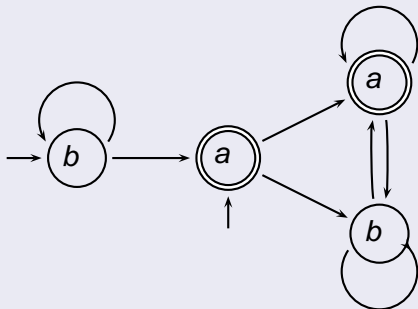
Glushkov automaton for $b^* a (b^* a)^*$ $b_1^* a_2 (b_3^* a_4)^*$ 

Glushkov automaton for $b^*a(b^*a)^*$ $b_1^*a_2(b_3^*a_4)^*$ 

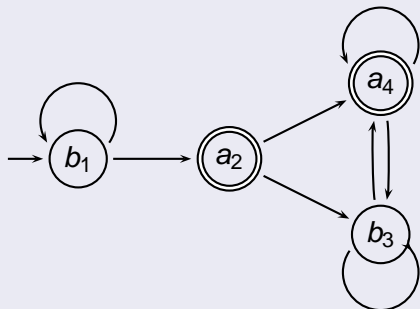
Glushkov automaton for $b^* a (b^* a)^*$ $b_1^* a_2 (b_3^* a_4)^*$ 

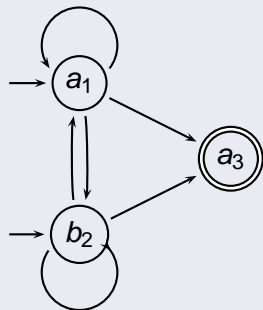
Glushkov automaton for $b^*a(b^*a)^*$ $b_1^*a_2(b_3^*a_4)^*$ 

Glushkov automaton for $b^*a(b^*a)^*$ $b_1^*a_2(b_3^*a_4)^*$ 

Glushkov automaton for $b^*a(b^*a)^*$ $b_1^*a_2(b_3^*a_4)^*$ 

Glushkov automaton construction

$$b_1^* a_2 (b_3^* a_4)^*$$


$$(a_1 + b_2)^* a_3$$


Recognition of deterministic regular expressions

Theorem (Book et al 1971, Brüggemann-Klein, Wood, 1998)

- A regular expression is *deterministic* (one-unambiguous) iff its Glushkov automaton is *deterministic*.
- It is decidable in quadratic time whether a regular expression is *deterministic*.

Properties of deterministic regular languages

Theorem (Brüggemann-Klein, Wood, 1998)

- *Not every regular language can be denoted by a deterministic regular expression.*

E.g., $(a + b)^ a(a + b)$.*

Properties of deterministic regular languages

Theorem (Brüggemann-Klein, Wood, 1998)

- *Not every regular language can be denoted by a deterministic regular expression.*

E.g., $(a + b)^ a(a + b)$.*

- *Deterministic regular languages are not closed under union, concatenation, or Kleene-star.*

No syntax for deterministic regular languages

Properties of deterministic regular languages

Theorem (Brüggemann-Klein, Wood, 1998)

- *Not every regular language can be denoted by a deterministic regular expression.*

E.g., $(a + b)^ a(a + b)$.*

- *Deterministic regular languages are not closed under union, concatenation, or Kleene-star.*

No syntax for deterministic regular languages

- *It can be decided in PTIME whether a DFA denotes a deterministic regular language.*

Orbit property.

Properties of deterministic regular languages

Theorem (Brüggemann-Klein, Wood, 1998)

- *Not every regular language can be denoted by a deterministic regular expression.*

E.g., $(a + b)^ a(a + b)$.*

- *Deterministic regular languages are not closed under union, concatenation, or Kleene-star.*

No syntax for deterministic regular languages

- *It can be decided in PTIME whether a DFA denotes a deterministic regular language.*

Orbit property.

- *If it exists, an equivalent deterministic regular expression can be constructed in exponential time.*

Properties of deterministic regular languages

Theorem (Brüggemann-Klein, Wood, 1998)

- *Not every regular language can be denoted by a deterministic regular expression.*

E.g., $(a + b)^ a(a + b)$.*

- *Deterministic regular languages are not closed under union, concatenation, or Kleene-star.*

No syntax for deterministic regular languages

- *It can be decided in PTIME whether a DFA denotes a deterministic regular language.*

Orbit property.

- *If it exists, an equivalent deterministic regular expression can be constructed in exponential time.*

Results provide formal machinery for dealing with DTDs.

Complexity of basic decision problems revisited

Schema containment (\subseteq)

Given: Schema's 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 trimmed).

Theorem

Containment of DTDs with deterministic regular expressions is in PTIME.

Summary slide

What to remember?

- XML DTDs are context-free grammars with **deterministic** regular expressions

Summary slide

What to remember?

- XML DTDs are context-free grammars with **deterministic** regular expressions
- deterministic regular expressions are a semantical notion: no easy syntax – non-transparent to users

Summary slide

What to remember?

- XML DTDs are context-free grammars with **deterministic** regular expressions
- deterministic regular expressions are a semantical notion: no easy syntax – non-transparent to users
- advantage: optimization problems are tractable

Summary slide

What to remember?

- XML DTDs are context-free grammars with **deterministic** regular expressions
- deterministic regular expressions are a semantical notion: no easy syntax – non-transparent to users
- advantage: optimization problems are tractable

Question

What is the largest robust/natural class of regular expressions for which the inclusion problem is in **PTIME**?

Outline

- 1 Introduction to XML
- 2 Document Type Definitions
- 3 Extended Document Type Definitions and XML Schema**
- 4 Characterizations of single-type EDTDs
- 5 1-pass preorder typing
- 6 Single-type EDTDs in practice: ancestor-guarded DTDs
- 7 Applications of ancestor-guarded DTDs
 - Optimization problems
 - Learning of schema's
- 8 Conclusion

Extended DTDs

Grammar based approach to unranked regular tree languages

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 $\rightarrow (dvd^1 + dvd^2)^* dvd^2 (dvd^1 + dvd^2)^*$

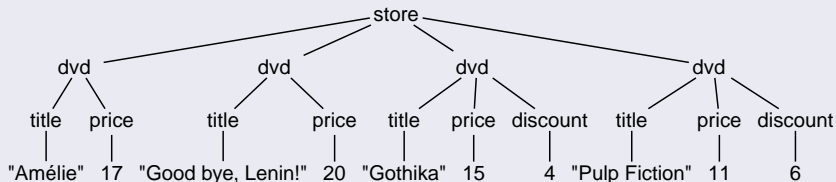
dvd¹ \rightarrow title price

dvd² \rightarrow title price discount

Extended DTDs

Grammar based approach to unranked regular tree languages

tree t



Example

store \rightarrow $(\text{dvd}^1 + \text{dvd}^2)^* \text{dvd}^2 (\text{dvd}^1 + \text{dvd}^2)^*$

dvd^1 \rightarrow title price

dvd^2 \rightarrow title price discount

Extended DTDs

Grammar based approach to unranked regular tree languages

Typed tree t'



Example

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

dvd^1 \rightarrow title price

dvd^2 \rightarrow title price discount

EDTDs versus Tree Automata

Theorem (Papakonstantinou, Vianu, 2000)

NTAs and EDTDs define precisely the class of (homogeneous) regular unranked tree languages.

EDTDs versus Tree Automata

Theorem (Papakonstantinou, Vianu, 2000)

NTAs and EDTDs define precisely the class of (homogeneous) regular unranked tree languages.

Example

EDTD

$$\begin{aligned} 0^0 &\rightarrow \varepsilon, 1^1 \rightarrow \varepsilon \\ \wedge^0 &\rightarrow .* (0^0 + \vee^0 + \wedge^0) .* \\ \wedge^1 &\rightarrow (1^1 + \vee^1 + \wedge^1)^* \\ \vee^1 &\rightarrow .* (1^1 + \vee^1 + \wedge^1) .* \\ \vee^0 &\rightarrow (0^0 + \vee^0 + \wedge^0)^* \end{aligned}$$

NTA

$$\begin{aligned} \delta(f, 0) &= \{\varepsilon\}; \delta(t, 1) = \{\varepsilon\}; \\ \delta(f, \wedge) &= .* f .* \\ \delta(t, \wedge) &= t^* \\ \delta(t, \vee) &= .* t .* \\ \delta(f, \vee) &= f^* \end{aligned}$$

Does XML Schema correspond to EDTDs?

```
<xs:element name="store">
  <xs:complexType>
    <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="dvd" type="1"/>
        <xs:element name="dvd" type="2"/>
      </xs:choice>
      <xs:element name="dvd" type="2"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="dvd" type="1"/>
        <xs:element name="dvd" type="2"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Does XML Schema correspond to EDTDs?

```

<xs:element name="store">
  <xs:complexType>
    <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="dvd" type="1"/>
        <xs:element name="dvd" type="2"/>
      </xs:choice>
      <xs:element name="dvd" type="2"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Rejected by XML Schema validator

Violates the Element Declarations Consistent Constraint.

```

      <xs:element name="dvd" type="1"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
</xs:element>

```

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 [...].

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	→	$(\text{dvd}^1 + \text{dvd}^2)^* \text{dvd}^2 (\text{dvd}^1 + \text{dvd}^2)^*$
dvd ¹	→	title price
dvd ²	→	title price discount

A formalization of XML Schema: single-type EDTDs

Definition (Murata, Lee, Mani, 2001)

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

store	→	regulars discounts
regulars	→	$(\text{dvd}^1)^*$
discounts	→	$\text{dvd}^2 (\text{dvd}^2)^*$
dvd^1	→	title price
dvd^2	→	title price discount

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 single-type EDTDs

Immediate Questions

- Can you recognize single-type EDTDs?

Trivial

XML Schema validator

- What kind of languages can be defined by single-type EDTDs?
- Is it decidable whether an EDTD is equivalent to a single-type EDTD?

smart XML Schema validator

Outline

- 1 Introduction to XML
- 2 Document Type Definitions
- 3 Extended Document Type Definitions and XML Schema
- 4 Characterizations of single-type EDTDs**
- 5 1-pass preorder typing
- 6 Single-type EDTDs in practice: ancestor-guarded DTDs
- 7 Applications of ancestor-guarded DTDs
 - Optimization problems
 - Learning of schema's
- 8 Conclusion

Properties of single-type EDTDs

Two properties

- 1 Single-type EDTDs admit unique top-down typing
- 2 Closure under a certain form of subtree exchange

(1) Validation and typing

Validation and typing:

Given a tree t and an EDTD $D = (\Sigma, d, a^0)$

- **validation**: does $t \in L(D)$, i.e., does there exist a typed tree $t' \in L(d)$?
- **typing**: compute for every b -labeled node its type b^i in t' (not necessarily unique)

(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.

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

(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.

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

Corollary

Single-typedness implies unique top-down typing.

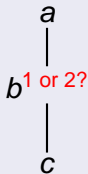
Motivation

(1) Two-pass and ambiguous typing

Example

$$a \rightarrow b^1 + b^2, \quad b^1 \rightarrow c, \quad b^2 \rightarrow d$$

Tree



(1) Two-pass and ambiguous typing

Example

$$a \rightarrow b^1 + b^2, \quad b^1 \rightarrow c, \quad b^2 \rightarrow d$$

Example

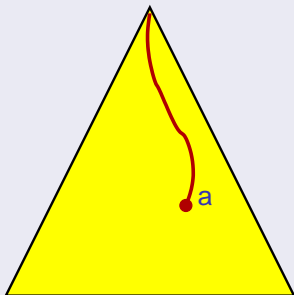
$$a \rightarrow b^1 + b^2, \quad b^1 \rightarrow c^*, \quad b^2 \rightarrow d^*$$

Tree

$$\begin{array}{c}
 a \\
 | \\
 b^1 \text{ or } 2?
 \end{array}$$

(2) An exchange property of single-type EDTDs

The Ancestor-String



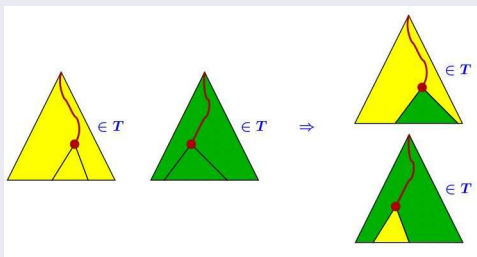
Notation

$\text{anc-str}^t(u)$ = the ancestor-string of a tree t at node u

(2) An exchange property for single-type EDTDs

Ancestor-Guarded Subtree Exchange

T is a regular tree language

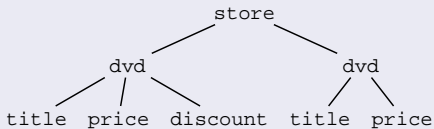
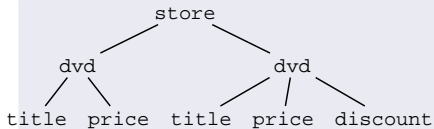


Theorem (Martens, Nev., Schw., 2005)

A regular tree language is definable by a single-type EDTD iff it is closed under ancestor-guarded subtree exchange.

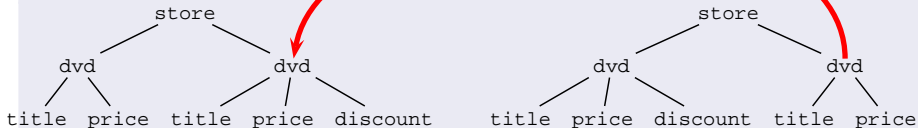
(2) Tool for proving inexpressibility

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



(2) Tool for proving inexpressibility

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



(2) Tool for proving inexpressibility

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



(2) Tool for proving inexpressibility

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



Single-type EDTDs are **not** closed under union.

Smart XML Schema validator

Theorem (Martens, Nev., Schw., 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. $a^1 \rightarrow b^1 b^2$ and $a^2 \rightarrow b^3$ becomes

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

$$a^{\{2\}} \rightarrow b^{\{3\}}$$

$$a^{\{1,2\}} \rightarrow b^{\{1,2,3\}} b^{\{1,2,3\}} + b^{\{1,2,3\}}$$

$L(D') = L(D)$ iff $L(D)$ is single-type.

We know that $L(D) \subseteq L(D')$.

So, only need to test $L(D') \subseteq L(D)$: $D' \cap \neg D = \emptyset$.

Summary slide

What to remember?

- XML Schema \approx single-type EDTDs \subsetneq regular tree languages
- single-type EDTDs admit top-down unique typing

Question

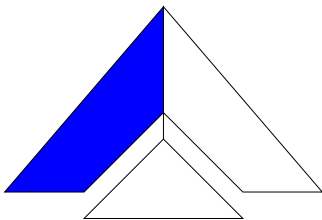
What is the largest subclass of EDTDs that admits top-down unique typing?

Outline

- 1 Introduction to XML
- 2 Document Type Definitions
- 3 Extended Document Type Definitions and XML Schema
- 4 Characterizations of single-type EDTDs
- 5 1-pass preorder typing**
- 6 Single-type EDTDs in practice: ancestor-guarded DTDs
- 7 Applications of ancestor-guarded DTDs
 - Optimization problems
 - Learning of schema's
- 8 Conclusion

1-Pass preorder typing

Streaming as a generalization of top-down processing



```
<store><regulars><dvd>  
<title>Amelie</title>  
<price>17</price>  
</dvd></regulars>  
<discounts>...
```

1-Pass Preorder Typing versus single-type EDTDs

Observations

- Streaming (preorder) typing is not possible for every EDTD:

$$a \rightarrow b^1 + b^2$$

$$b^1 \rightarrow c$$

$$b^2 \rightarrow d$$

$$\begin{array}{c} a \\ | \\ b \\ | \\ c \end{array}$$

- Every **single-type** EDTD is preorder typable: type of child depends only on type of parent
- Single-type EDTDs are not the largest class which is preorder typeable:

$$a \rightarrow b^1 b^2$$

$$b^1 \rightarrow c$$

$$b^2 \rightarrow d$$

$$\begin{array}{c} a \\ / \quad \backslash \\ b \quad b \\ | \quad | \\ c \quad d \end{array}$$

Restrained Competition EDTDs

left-to-right unique typing

Definition (Murata, Lee, Mani, 2001)

A regular expression r **restrains competition** if there are no strings $wa^i v$ and $wa^j v'$ in $L(r)$ with $i \neq j$.

An EDTD is **restrained competition** iff all regular expressions occurring in rules restrain competition.

Not restrained-competition

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

$dvd^1 \rightarrow$ title price

$dvd^2 \rightarrow$ title price discount

$dvd^1 dvd^2 dvd^2$

$dvd^2 dvd^2 dvd^2$

Restrained Competition EDTDs

left-to-right unique typing

Definition (Murata, Lee, Mani, 2001)

A regular expression r **restrains competition** if there are no strings $wa^i v$ and $wa^j v'$ in $L(r)$ with $i \neq j$.

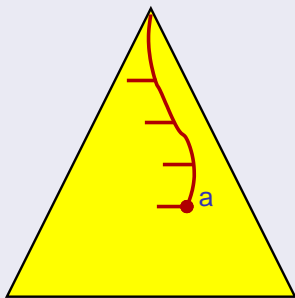
An EDTD is **restrained competition** iff all regular expressions occurring in rules restrain competition.

Restrained-competition

store	→	$(\text{dvd}^1)^* \text{discounts} \text{dvd}^2 (\text{dvd}^2)^*$
discounts	→	ε
dvd^1	→	title price
dvd^2	→	title price discount

Towards characterizations of 1-pass preorder typing

The ancestor-sibling string



Towards characterizations of 1-pass preorder typing

Theorem (Martens, Nev., Schw., 2005)

For a regular tree language T , the following are equivalent

- *T is 1-pass preorder typable*
- *T is definable by a restrained-competition EDTD*
- *T is closed under ancestor-sibling-guarded subtree exchange*

Summary slide

What to remember?

- DTD \approx extended context-free grammars
- XML Schema \approx single-type EDTDs
- single-typedness is not the most liberal restriction to get unique top-down (1-pass) typing: restrained-competition EDTDs.
- Neither single-type nor restrained-competition EDTDs are closed under union and negation: problematic for schema integration.

Question

Find a subclass of the EDTDs closed under the Boolean operators and admitting unique top-down typing.

Outline

- 1 Introduction to XML
- 2 Document Type Definitions
- 3 Extended Document Type Definitions and XML Schema
- 4 Characterizations of single-type EDTDs
- 5 1-pass preorder typing
- 6 Single-type EDTDs in practice: ancestor-guarded DTDs**
- 7 Applications of ancestor-guarded DTDs
 - Optimization problems
 - Learning of schema's
- 8 Conclusion

A practical study of XSDs

Corpus

- 819 XSDs from the Cover pages.
- 726 XSDs through Google's web services.

A practical study of XSDs

Corpus

- 819 XSDs from the Cover pages.
- 726 XSDs through Google's web services.

Practical XSDs are local

- 85% of the XSDs are structurally equivalent to a DTD: at most one type is associated to every element name.
- One example used types: $a^1 \rightarrow b$ and $a^2 \rightarrow b$.

How do the 15% non-local XSDs look like?

- 90% of the cases, types only depend on parent context:

store	→	regulars discounts
regulars	→	(dvd ¹)*
discounts	→	dvd ² dvd ² (dvd ²)*
dvd ¹	→	title price
dvd ²	→	title price discount

How do the 15% non-local XSDs look like?

- 90% of the cases, types only depend on parent context:

store	→	regulars discounts
regulars	→	$(\text{dvd}^1)^*$
discounts	→	$\text{dvd}^2 \text{dvd}^2 (\text{dvd}^2)^*$
dvd^1	→	title price
dvd^2	→	title price discount

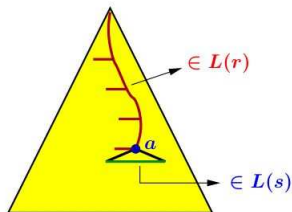
- Remaining 10% are of the following form:

a	→	b + c	h^1	→	j^1
b	→	e d^1 f	h^2	→	j^2
c	→	e d^2 f	j^1	→	k l
d^1	→	g h^1 i	j^2	→	m n
d^2	→	g h^2 i			

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 Σ .



Definition

A tree t is **valid** w.r.t. A iff for every vertex v there is some $r \rightarrow s$ such that $\text{anc-str}^t(v) \in L(r)$ and the children of v match s .

Making dependencies explicit

Theorem

Ancestor-based DTDs and single-type EDTDs define the same class of tree languages.

Ancestor-guarded DTDs can be used as a light-weight front-end for XML Schema

Similar formalism for restrained competition EDTDs.

Making dependencies explicit

single-type EDTD

store	→	regulars discounts
regulars	→	$(\text{dvd}^1)^*$
discounts	→	$\text{dvd}^2 \text{dvd}^2 (\text{dvd}^2)^*$
dvd^1	→	title price
dvd^2	→	title price discount

Ancestor-guarded DTD

store	→	regulars discounts
regulars	→	dvd^*
discounts	→	$\text{dvd} \text{dvd} \text{dvd}^*$
$* \cdot \text{regulars} \cdot \text{dvd}$	⇒	title price
$* \cdot \text{discounts} \cdot \text{dvd}$	⇒	title price discount

Outline

- 1 Introduction to XML
- 2 Document Type Definitions
- 3 Extended Document Type Definitions and XML Schema
- 4 Characterizations of single-type EDTDs
- 5 1-pass preorder typing
- 6 Single-type EDTDs in practice: ancestor-guarded DTDs
- 7 Applications of ancestor-guarded DTDs**
 - Optimization problems
 - Learning of schema's
- 8 Conclusion

Outline

- 1 Introduction to XML
- 2 Document Type Definitions
- 3 Extended Document Type Definitions and XML Schema
- 4 Characterizations of single-type EDTDs
- 5 1-pass preorder typing
- 6 Single-type EDTDs in practice: ancestor-guarded DTDs
- 7 Applications of ancestor-guarded DTDs**
 - Optimization problems
 - Learning of schema's
- 8 Conclusion

Complexity of basic decision problems

Proposition

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 $DTD(R)$ is in \mathcal{C} ;
- CONTAINMENT for single-type $EDTD(R)$ is in \mathcal{C} ;
- CONTAINMENT for restrained-competition $EDTD(R)$ is in \mathcal{C} .

Theorem (Seidl 1990, 1994)

CONTAINMENT, EQUIVALENCE, and INTERSECTION are EXPTIME-complete for EDTDs (even with 1-unambiguous REs).

Complexity of regular expressions

- Upper bounds for the complexity of containment for REs carry over to schema languages

Complexity of regular expressions

- Upper bounds for the complexity of containment for REs carry over to schema languages
- Containment problem for REs has been studied in depth (Hunt III et al., Kozen, Meyer and Stockmeyer, ...)

Complexity of regular expressions

- Upper bounds for the complexity of containment for REs carry over to schema languages
- Containment problem for REs has been studied in depth (Hunt III et al., Kozen, Meyer and Stockmeyer, ...)
- more than ninety percent of the regular expressions occurring in practical DTDs and XSDs are **Chain Regular Expressions (CHAREs)**. (Bex et al. 2004).

Complexity of regular expressions

Definition

- A **chain regular expression (CHARE)** is a sequence of factors where a factor is $(a_1 + \dots + a_n)^*$, $(a_1 + \dots + a_n)^+$, or $(a_1 + \dots + a_n)^?$.
- $(c^* + b^*)(a + b)^?(ab)^+(ac + b)^*$ is a CHARE
- $(a + b) + (a^*b^*)$ is not a CHARE.

Theorem (Martens, Nev, Schwentick (2004))

CONTAINMENT for CHAREs with factors of the form a or a^* is **coNP**-complete.

Complexity of regular expressions

Definition

- A **chain regular expression (CHARE)** is a sequence of factors where a factor is $(a_1 + \dots + a_n)^*$, $(a_1 + \dots + a_n)^+$, or $(a_1 + \dots + a_n)^?$.
- $(c^* + b^*)(a + b)^?(ab)^+(ac + b)^*$ is a CHARE
- $(a + b) + (a^*b^*)$ is not a CHARE.

Theorem (Martens, Nev, Schwentick (2004))

CONTAINMENT for CHAREs with factors of the form a or a^* is **coNP**-complete.

Cheating: none of these are 1-unambiguous

Complexity of basic decision problems

Proposition

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} .

Complexity of basic decision problems

Proposition

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} .

Theorem (Martens, Nev., Schw. 2005)

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 1-unambiguous REs is **PSPACE**-complete.

Summary slide

What to remember?

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

Summary slide

What to remember?

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

Further research

- Not much is known about EQUIVALENCE of fragments of CHAREs: in **PTIME** for $\text{CHARE}(a, a^*)$.
What is the largest class of CHARES/REs for which equivalence is in **PTIME**?
- Numerical occurrence operator $(a^4(b + c^*)^7)$ and shuffle operator $(a\&b = \{ab, ba\})$.
Questions regarding 1-unambiguity and complexity. (Preliminary results by Kilpeläinen (2005,2006), Brüggemann-Klein (1994), Gelade, Martens, Nev (2006))

Outline

- 1 Introduction to XML
- 2 Document Type Definitions
- 3 Extended Document Type Definitions and XML Schema
- 4 Characterizations of single-type EDTDs
- 5 1-pass preorder typing
- 6 Single-type EDTDs in practice: ancestor-guarded DTDs
- 7 Applications of ancestor-guarded DTDs**
 - Optimization problems
 - **Learning of schema's**
- 8 Conclusion

DTD and XML Schema learning

DTD learning

- Given a set of XML documents, find a DTD that represents them

DTD and XML Schema learning

DTD learning

- Given a set of XML documents, find a DTD that represents them
- DTD learning reduces to learning of regular expressions from positive example strings (more or less the same for XML Schema)

DTD and XML Schema learning

DTD learning

- Given a set of XML documents, find a DTD that represents them
- DTD learning reduces to learning of regular expressions from positive example strings (more or less the same for XML Schema)
- the complete class of regular expressions can not be learned from positive examples only (Gold (1967))

DTD and XML Schema learning

DTD learning

- Given a set of XML documents, find a DTD that represents them
- DTD learning reduces to learning of regular expressions from positive example strings (more or less the same for XML Schema)
- the complete class of regular expressions can not be learned from positive examples only (Gold (1967))
- some learnable subclasses are identified (Brazma (1993), Fernau (2005))

DTD and XML Schema learning

DTD learning

- Given a set of XML documents, find a DTD that represents them
- DTD learning reduces to learning of regular expressions from positive example strings (more or less the same for XML Schema)
- the complete class of regular expressions can not be learned from positive examples only (Gold (1967))
- some learnable subclasses are identified (Brazma (1993), Fernau (2005))
- **Observation:** in practice, every alphabet symbol in a CHARE occurs at most once (Bex et al. (2005))

Single-occurrence regular expressions (SOREs)

Definition

SOREs: single-occurrence regular expressions, every alphabet symbol can occur at most once

$((b?(a + c))^+d)^+e$ is a SORE, $(a + b)^*b$ is not a SORE

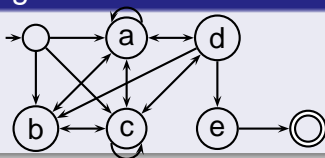
Single-occurrence regular expressions (SOREs)

Definition

SOREs: single-occurrence regular expressions, every alphabet symbol can occur at most once

$((b?(a + c))^+d)^+e$ is a SORE, $(a + b)^*b$ is not a SORE

Corresponds to a single-occurrence automaton



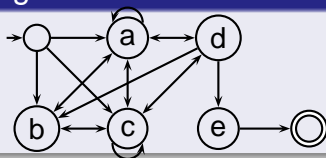
Single-occurrence regular expressions (SOREs)

Definition

SOREs: single-occurrence regular expressions, every alphabet symbol can occur at most once

$((b?(a + c))^+d)^+e$ is a SORE, $(a + b)^*b$ is not a SORE

Corresponds to a single-occurrence automaton



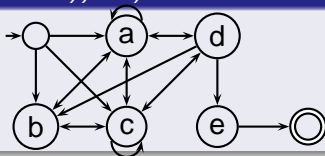
Remark

Every SORE defines a local language.

Local languages are efficiently learnable in the limit from positive data (Garcia and E. Vidal, 1990).

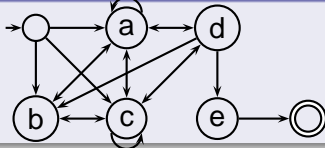
Learning REs \neq learning automata

Automaton for $((b?(a + c))^+d)^+e$



Learning REs \neq learning automata

Automaton for $((b?(a+c))^+d)^+e$

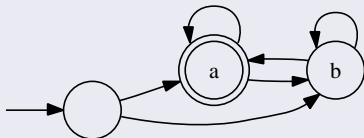


RE obtained through state elimination

$$\begin{aligned}
 & (aa^*d + (c + aa^*c)(c + aa^*c)^*(d + aa^*d) + (b + aa^*b + (c + \\
 & aa^*c)(c + aa^*c)^*(b + aa^*b)))(aa^*b + (c + aa^*c)(c + aa^*c)^* \\
 & (b + aa^*b))^*(aa^*d + (c + aa^*c)(c + aa^*c)^*(d + aa^*d)))(aa^*d + \\
 & (c + aa^*c)(c + aa^*c)^*(d + aa^*d) + (b + aa^*b + (c + aa^*c)(c + \\
 & aa^*c)^*(b + aa^*b))(aa^*b + (c + aa^*c)(c + aa^*c)^*(b + aa^*b))^* \\
 & (aa^*d + (c + aa^*c)(c + aa^*c)^*(d + aa^*d)))^*e,
 \end{aligned}$$

Some observations

Not every SOA corresponds to a SORE $((a + b)^* a)$



Ehrenfeucht, Zeiger (1974)

There are SOAs whose smallest equivalent regular expression is **exponential** in the size of the automaton (even in the size of the alphabet).

But ...

The size of a SORE is always **linear** in the size of the alphabet.

DTD learning (Bex, Nev, Schwentick, Tuyls (2006))

Theorem

There is a **PTIME** algorithm that rewrites a SOA to a corresponding SORE when one exists.

Learning of DTDs from sufficient data

- 1 Learn SOA.
- 2 Translate to equivalent SORE.

Learning of DTDs from insufficient data

- 1 Learn SOA.
- 2 Translate to equivalent SORE, while repairing SOA.

Outperforms existing systems in speed and accuracy on real world and synthetic data.

Outline

- 1 Introduction to XML
- 2 Document Type Definitions
- 3 Extended Document Type Definitions and XML Schema
- 4 Characterizations of single-type EDTDs
- 5 1-pass preorder typing
- 6 Single-type EDTDs in practice: ancestor-guarded DTDs
- 7 Applications of ancestor-guarded DTDs
 - Optimization problems
 - Learning of schema's
- 8 Conclusion**

Conclusion

For the XML-theorist

- DTDs are almost extended context-free grammars
- XML Schema is much closer to DTDs than to tree automata, is reflected in:
 - modeling power
 - optimization
 - learning

Conclusion

For the XML-theorist

- DTDs are almost extended context-free grammars
- XML Schema is much closer to DTDs than to tree automata, is reflected in:
 - modeling power
 - optimization
 - learning

For the formal language theorist

- expressiveness of tree and string automata, complexity of tree automata, string automata and REs, translations from REs to automata
- XML (schema) research is a good excuse to do theory