XML Research for Formal Language Theorists

Wim Martens

TU Dortmund

Goal of this talk

XML Research vs Formal Languages

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

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

Outline

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

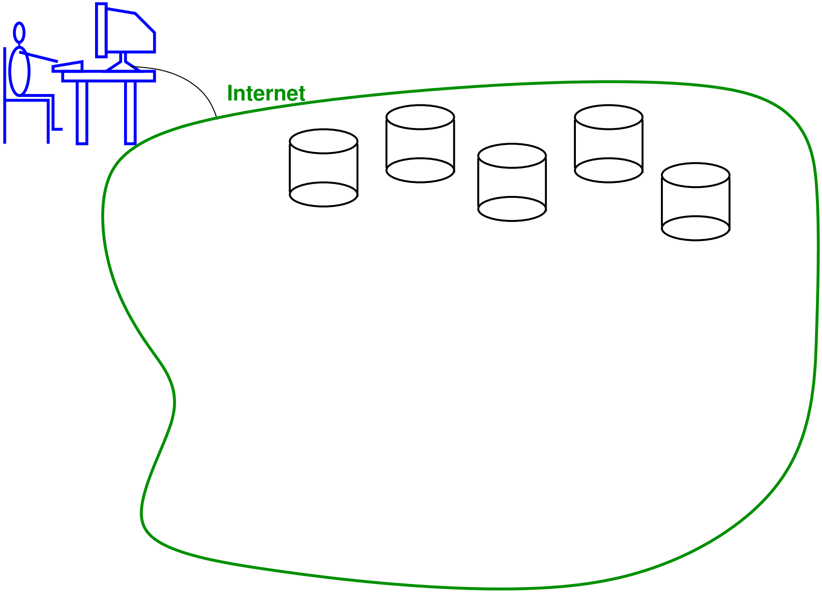
Enough with these sissy keyword searches!

Searching the Internet

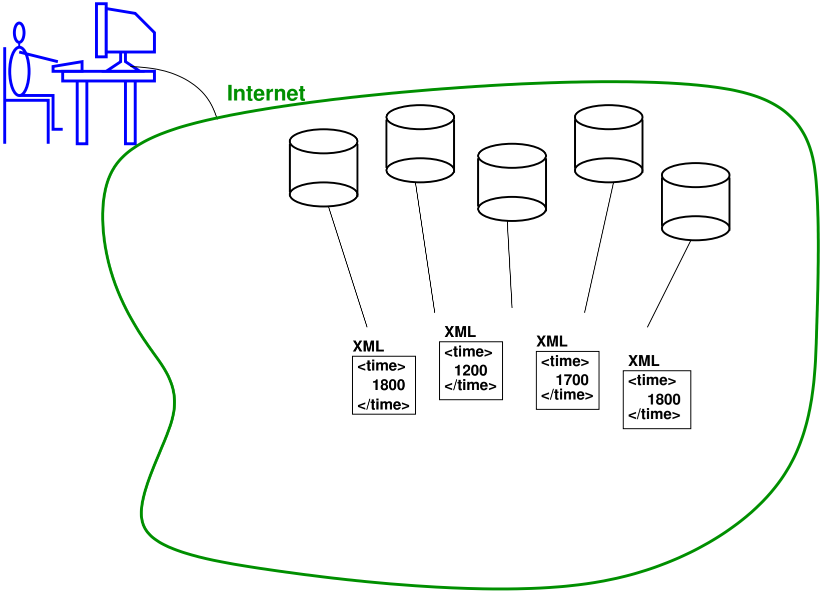
A *real* search

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

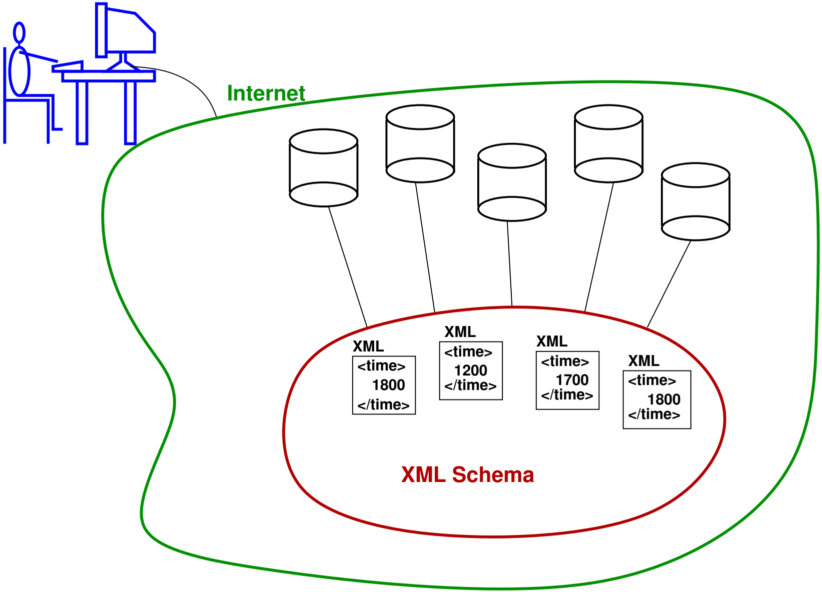
An Example



An Example


Internet
XML
<time> 1800 </time>
XML
<time> 1200 </time>
XML
<time> 1700 </time>
XML
<time> 1800 </time>

An Example



Internet
XML
<time>
1800
</time>
XML
<time>
1200
</time>
XML
<time>
1700
</time>
XML
<time>
1800
</time>
XML Schema

A self-describing data format

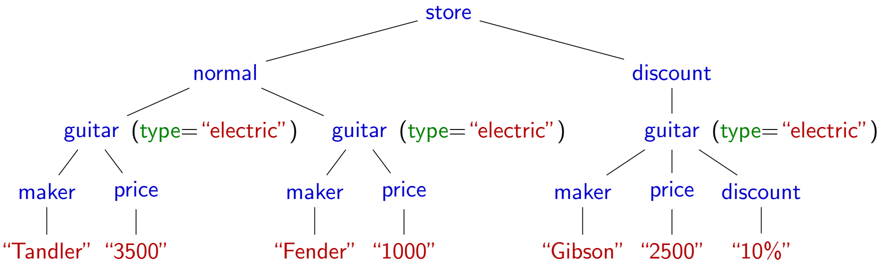
```
<store>
    <normal>
        <guitar type="electric">
            <maker> Tandler </maker>
            <price> 3500 </price>
        </guitar>
        <guitar type="electric">
            <maker> Fender </maker>
            <price> 1000 </price>
        </guitar>
    </normal>
    <discount>
        <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) — CFGs with REs
- XML Schema (W3C) — $\not\approx$ 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 — in this talk)

Outline

Outline

Document Type Definitions (DTDs)

Example

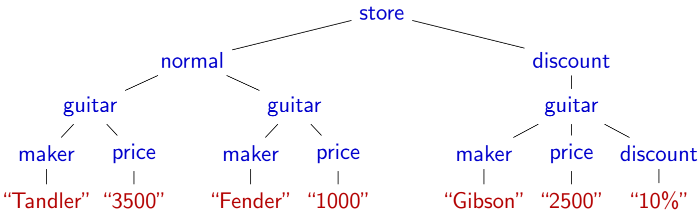
```
<!DOCTYPE store [
    <!ELEMENT store     (normal,discount)>
    <!ELEMENT normal    (guitar*)>
    <!ELEMENT discount  (guitar+)>
    <!ELEMENT guitar    (maker,price,discount?)>
    <!ELEMENT maker     (#PCDATA)>
    <!ELEMENT price     (#PCDATA)>
    <!ELEMENT discount  (#PCDATA)>
]>
```

Corresponding grammar (start symbol store)

$$
\begin{array}{lcl}
\text{store} & \longrightarrow & \text{normal discount} \\
\text{normal} & \longrightarrow & \text{guitar}^* \\
\text{discount} & \longrightarrow & \text{guitar}^+ \\
\text{guitar} & \longrightarrow & \text{maker price discount?} \\
\text{maker} & \longrightarrow & \text{DATA} \\
\text{price} & \longrightarrow & \text{DATA} \\
\text{discount} & \longrightarrow & \text{DATA}
\end{array}
$$

Document Type Definitions (DTDs)

XML Document



Corresponding grammar (start symbol store)

store	$\rightarrow$	normal discount
normal	$\rightarrow$	guitar*
discount	$\rightarrow$	guitar$^+$
guitar	$\rightarrow$	maker price discount?
maker	$\rightarrow$	DATA
price	$\rightarrow$	DATA
discount	$\rightarrow$	DATA

Extended Context-free grammars as a formal abstraction

Definition

A DTD is a triple (Σ, d, s_d) where

- Σ is a finite alphabet
- $s_d \in \Sigma$ is the start symbol
- $d : \Sigma \to \mathrm{RE}(\Sigma)$ 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 node v labeled a the string formed by the children of v belongs to $d(a)$.

Optimization questions: from FLT to XML

Schema containment ($\subseteq$)

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

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

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

(when d_1 and d_2 are reduced).

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

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

(when d_1 and d_2 are reduced).

Theorem (Meyer, Stockmeyer, 1973)

Containment of regular expressions is **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

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

(when d_1 and d_2 are reduced).

Theorem (Meyer, Stockmeyer, 1973)

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

Corollary

DTD containment is **PSPACE**-complete.

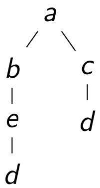
Outline

Queries for XML

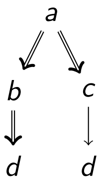
Conjunctive Queries over Trees

XPath

Tree:



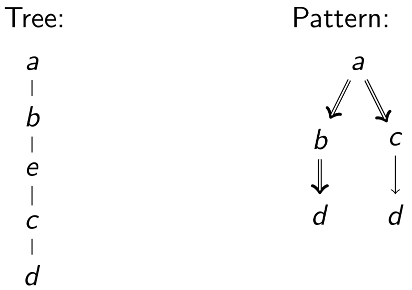
Pattern:



Pattern Matching

- Tree matches Pattern if there is a homomorphism h : Pattern $\longrightarrow$ Tree
- Homomorphism doesn't have to be injective

XPath



Pattern Matching

- Tree matches Pattern if there is a homomorphism h : Pattern $\longrightarrow$ Tree
- Homomorphism doesn't have to be injective

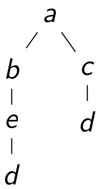
Conjunctive Queries over Trees

Pattern Matching

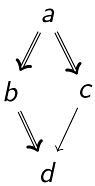
- Tree matches Pattern if there is a homomorphism h : Pattern $\rightarrow$ Tree
- Homomorphism doesn't have to be injective

Conjunctive Queries over Trees

Tree:



Pattern:

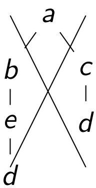


Pattern Matching

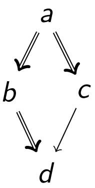
- Tree matches Pattern if there is a homomorphism h : Pattern $\rightarrow$ Tree
- Homomorphism doesn't have to be injective

Conjunctive Queries over Trees

Tree:



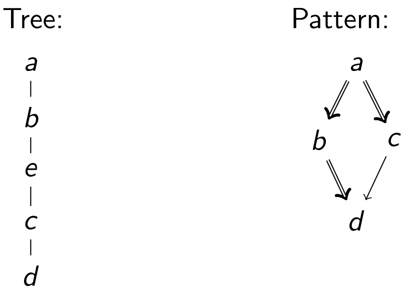
Pattern:



Pattern Matching

- Tree matches Pattern if there is a homomorphism h : Pattern $\rightarrow$ Tree
- Homomorphism doesn't have to be injective

Conjunctive Queries over Trees



Pattern Matching

- Tree matches Pattern if there is a homomorphism h : Pattern $\longrightarrow$ 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 $L(Q_1) \subseteq L(Q_2)$?

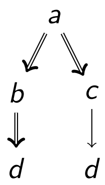
Query Containment w.r.t. a DTD

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

XPath Query Optimization

Formal Language Theory to the Rescue!

XPath Query



Lemma

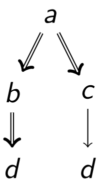
For each XPath query Q there is an Alternating Tree Automaton A s.t.

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

XPath Query Optimization

Formal Language Theory to the Rescue!

XPath Query



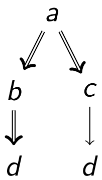
Lemma

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

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

Moreover, $|A|$ *is polynomial in* $|Q|$

XPath Query



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

XPath Query Optimization

Formal Language Theory to the Rescue!

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**

XPath Query Optimization

Formal Language Theory to the Rescue!

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

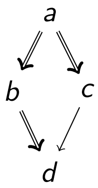
Theorem

- *XPath Containment (tree pattern fragment) is* **NP***-complete [Miklau, Suciu 2002]*
- *XPath Containment (with $\neg$ and $\vee$) 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!

Conjunctive Query



Lemma (Björklund, Mar., Schwentick 2008)

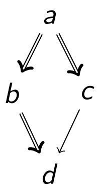
For each Conjunctive Query Q there is an Alternating Tree Automaton A s.t.

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

Conjunctive Query Optimization

Formal Language Theory to the Rescue!

Conjunctive Query



Lemma (Björklund, Mar., Schwentick 2008)

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

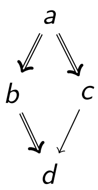
$$L(Q) = L(A)$$

But, $|A|$ is exponential in $|Q|$

Conjunctive Query Optimization

Formal Language Theory to the Rescue!

Conjunctive Query



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

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 s.t.

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

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

Theorem

- *CQ Containment w.r.t. DTDs is* **2EXPTIME**-*complete*

  [Björklund, Mar., Schwentick 2008]

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 s.t.

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

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

Theorem

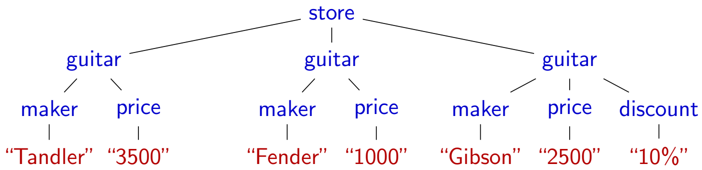
- *CQ Containment is Π_2^P-complete [Björklund, Mar., Schwentick 2007]*
- *CQ Containment w.r.t. DTDs is* **2EXPTIME***-complete [Björklund, Mar., Schwentick 2008]*

Outline

Extended DTDs

Grammar based approach to unranked regular tree languages

tree t



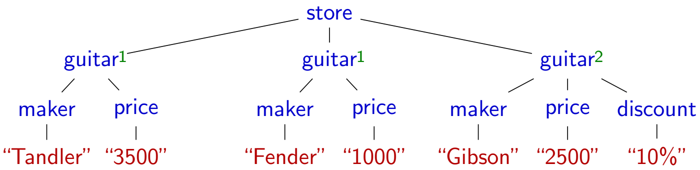
Example

$$
\begin{array}{lcl}
\text{store} & \to & (\text{guitar}^1)^* \, (\text{guitar}^2)^+ \\
\text{guitar}^1 & \to & \text{maker price} \\
\text{guitar}^2 & \to & \text{maker price discount}
\end{array}
$$

Extended DTDs

Grammar based approach to unranked regular tree languages

Typed tree t'



Example

$$\begin{array}{lcl} \text{store} & \rightarrow & (\text{guitar}^1)^* \ (\text{guitar}^2)^+ \\ \text{guitar}^1 & \rightarrow & \text{maker price} \\ \text{guitar}^2 & \rightarrow & \text{maker price discount} \end{array}$$

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

$$
\begin{array}{lcl}
\text{store} & \rightarrow & (\text{guitar}^1)^* \ (\text{guitar}^2)^+ \\
\text{guitar}^1 & \rightarrow & \text{maker price} \\
\text{guitar}^2 & \rightarrow & \text{maker price discount}
\end{array}
$$

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

$$\begin{array}{lcl} \text{store} & \rightarrow & (\text{guitar}^1)^* \ (\text{guitar}^2)^+ \\ \text{guitar}^1 & \rightarrow & \text{maker price} \\ \text{guitar}^2 & \rightarrow & \text{maker price discount} \end{array}$$

NTA

$$\begin{array}{lcl} \delta(\mathit{store}, \mathit{store}) & = & (\text{guitar}^1)^* \ (\text{guitar}^2)^+ \\ \delta(\mathit{guitar}^1, \mathit{guitar}) & = & \text{maker price} \\ \delta(\mathit{guitar}^2, \mathit{guitar}) & = & \text{maker price discount} \end{array}$$

Does XML Schema correspond to EDTDs?

```
<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?

```
<xs:element name="store">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="guitar" type="1"
                                minOccurs="0"
```

Rejected by XML Schema validator

Violates the Element Declarations Consistent Constraint.

```
                                minOccurs="1"
                                maxOccurs="unbounded"/>
    </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

$$\begin{array}{lcl} \text{store} & \to & (\text{guitar}^1)^* \; (\text{guitar}^2)^+ \\ \text{guitar}^1 & \to & \text{maker price} \\ \text{guitar}^2 & \to & \text{maker price discount} \end{array}$$

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	$\longrightarrow$	normal discount
normal	$\longrightarrow$	$(\text{guitar}^1)^*$
discount	$\longrightarrow$	$(\text{guitar}^2)^+$
guitar^1	$\longrightarrow$	maker price
guitar^2	$\longrightarrow$	maker 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

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

smart XML Schema validator

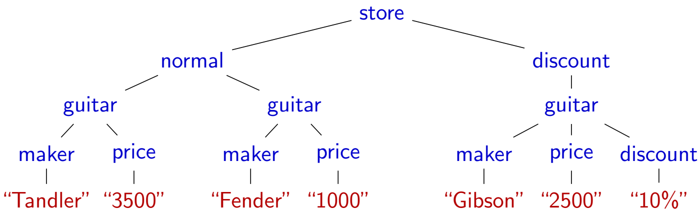
Outline

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

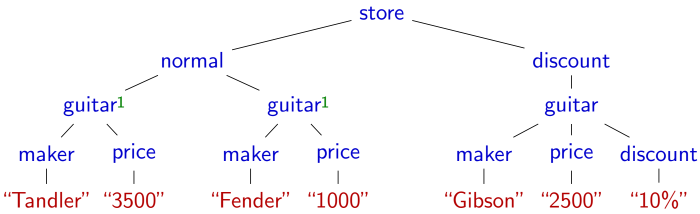
(1) Single-type EDTDs: simple top-down typing



Example

$$
\begin{array}{lcl}
\text{store} & \rightarrow & \text{normal discount} \\
\text{normal} & \rightarrow & (\text{guitar}^1)^* \\
\text{discount} & \rightarrow & (\text{guitar}^2)^+ \\
\text{guitar}^1 & \rightarrow & \text{maker price} \\
\text{guitar}^2 & \rightarrow & \text{maker price discount}
\end{array}
$$

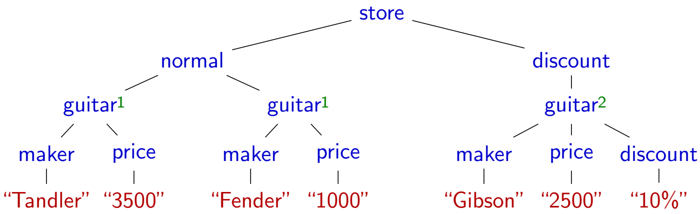
(1) Single-type EDTDs: simple top-down typing



Example

$$
\begin{array}{lcl}
\text{store} & \to & \text{normal discount} \\
\text{normal} & \to & (\text{guitar}^1)^* \\
\text{discount} & \to & (\text{guitar}^2)^+ \\
\text{guitar}^1 & \to & \text{maker price} \\
\text{guitar}^2 & \to & \text{maker price discount}
\end{array}
$$

(1) Single-type EDTDs: simple top-down typing



Example

store	$\rightarrow$	normal discount
normal	$\rightarrow$	$(\text{guitar}^1)^*$
discount	$\rightarrow$	$(\text{guitar}^2)^+$
guitar^1	$\rightarrow$	maker price
guitar^2	$\rightarrow$	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.

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

(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.

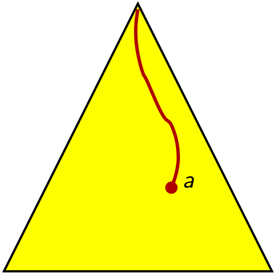
$\mu(a^1 + b^1 c^2) = a + bc$

Corollary

Single-typedness implies unique top-down typing.

(2) An exchange property of single-type EDTDs

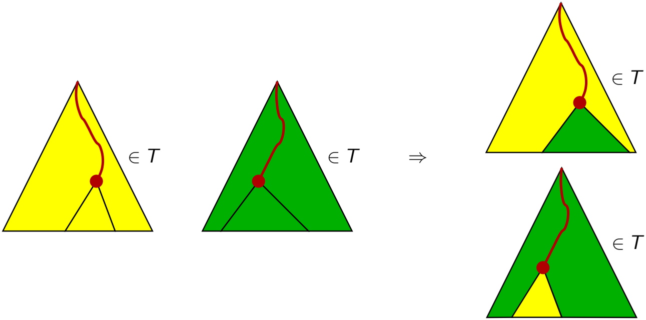
The Ancestor-String



(2) An exchange property for single-type EDTDs

Ancestor-Guarded Subtree Exchange

T is a regular tree language

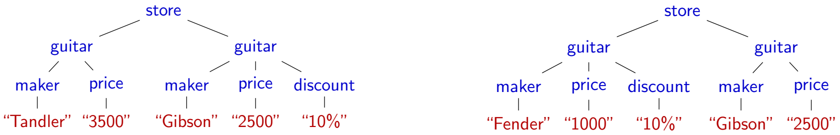


Theorem (Mar., Neven, Schwentick 2005)

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

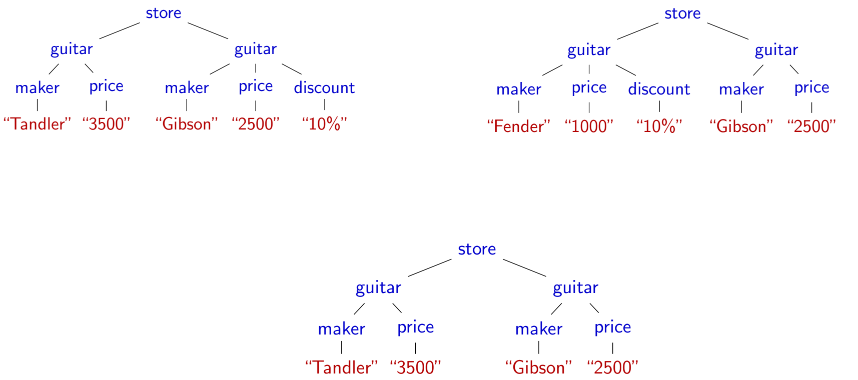
(2) Tool for proving inexpressibility

"At least one discount guitar" is not single-type



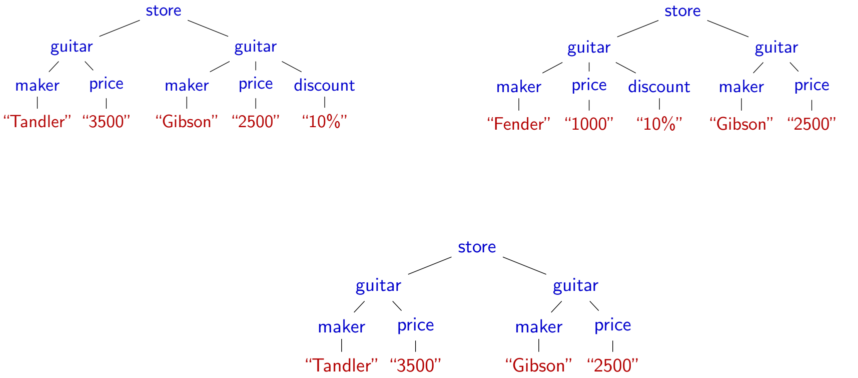
(2) Tool for proving inexpressibility

"At least one discount guitar" is not single-type



(2) Tool for proving inexpressibility

"At least one discount guitar" is not single-type



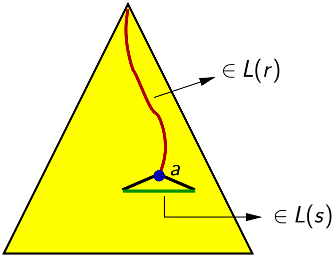
Single-type EDTDs are not closed under union or complement.

(3) Pattern-based Language

Making dependencies explicit

Definition

An ancestor-based DTD A is a set of rules $r \longrightarrow s$ 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 $r \longrightarrow s$ 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

$$
\begin{array}{lcl}
\text{store} & \to & \text{normal discount} \\
\text{normal} & \to & (\text{guitar}^1)^* \\
\text{discount} & \to & (\text{guitar}^2)^+ \\
\text{guitar}^1 & \to & \text{maker price} \\
\text{guitar}^2 & \to & \text{maker price discount}
\end{array}
$$

Ancestor-guarded DTD

$$
\begin{array}{lcl}
\text{store} & \to & \text{normal discount} \\
\text{normal} & \to & \text{guitar}^* \\
\text{discount} & \to & \text{guitar}^+ \\
*\cdot \text{normal} \cdot \text{guitar} & \to & \text{maker price} \\
*\cdot \text{discount} \cdot \text{guitar} & \to & \text{maker price discount}
\end{array}
$$

Smart XML Schema validator

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

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

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

Smart XML Schema validator

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, $a^1 \to b^1 b^2$, $b^1 \to c^1$, $b^2 \to c^2$ becomes

$$a^{\{1\}} \to b^{\{1,2\}} b^{\{1,2\}}$$
$$b^{\{1,2\}} \to c^{\{1,2\}} + c^{\{1,2\}}$$

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

Summary slide

What to remember?

- XML Schema $\approx$ single-type EDTDs $\subsetneq$ 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

Outline

Complexity of basic decision problems

Theorem (Mar., Neven, Schwentick 2004)

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

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

Theorem (Seidl 1990, 1994)

CONTAINMENT *and* **EQUIVALENCE** *are* **EXPTIME***-complete for 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 $\mathscr{C}$ a complexity class. Then the following are equivalent:

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

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 $\mathscr{C}$ a complexity class. Then the following are equivalent:

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

Theorem (Mar., Neven, Schwentick 2004)

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

- **INTERSECTION** *for $\mathscr{X}$ is* **NP***-complete;*
- **INTERSECTION** *for single-type EDTD($\mathscr{X}$) 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 ε, $\emptyset$, 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*

Regular Expression Analysis Revisited

Fragment	CONTAINMENT	EQUIVALENCE	INTERSECTION
a, a^+	in **PTIME** (DFA!)	in **PTIME**	in **PTIME**
a, a^*	**coNP**	in **PTIME**	**NP**
$a, a?$	**coNP**	in **PTIME**	**NP**
$a, (+a)^*$	**PSPACE**	in **PSPACE**	**NP**
$all - \{(+w)^*, (+w)^+\}$	**PSPACE**	in **PSPACE**	**NP**
$a, (+w)^*$	**PSPACE**	in **PSPACE**	**PSPACE** [Bala 2002]
RE	**PSPACE**	**PSPACE**	**PSPACE**

Regular Expression Analysis Revisited

Fragment	CONTAINMENT	EQUIVALENCE	INTERSECTION
a, a^+	in **PTIME** (DFA!)	in **PTIME**	in **PTIME**
a, a^*	**coNP**	in **PTIME**	**NP**
$a, a?$	**coNP**	in **PTIME**	**NP**
$a, (+a)^*$	**PSPACE**	in **PSPACE**	**NP**
$all - \{(+w)^*, (+w)^+\}$	**PSPACE**	in **PSPACE**	**NP**
$a, (+w)^*$	**PSPACE**	in **PSPACE**	**PSPACE** [Bala 2002]
RE	**PSPACE**	**PSPACE**	**PSPACE**

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(#,&)*

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!

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!
- single occurrence
- Kleene star only applied to disjunctions single symbols

Outline

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 $\mathcal{O}(n)$, such that any regular expression defining $\overline{L(r)}$ must be of size $\Omega(2^{2^n})$

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 $\mathcal{O}(n)$ for $\overline{K_{2^n}}$

Outline

Schema 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!

Schema Minimization

Minimization is typically studied on automata models

Schema Minimization

Minimization is typically studied on automata models
and the results look prettier on deterministic automata

Schema Minimization

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?

Schema Minimization

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.

Single-type EDTD Minimization

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

Unranked Tree Automaton Minimization

(Brüggemann-Klein, Murata, Wood 2001)

A bottom-up unranked tree automaton is *deterministic* if for every pair of rules $a(L_1) \to q_1$ and $a(L_2) \to q_2$,

$$L_1 \cap L_2 = \emptyset$$

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:

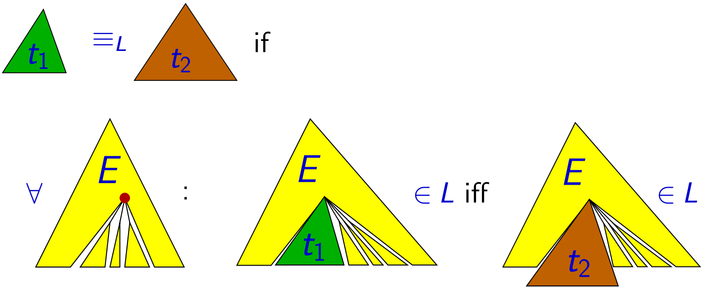
Theorem (Mar., Niehren 2005)

- **MINIMIZATION** *is in* **P** *for bottom-up deterministic tree automata*
- *the Myhill-Nerode theorem for unranked tree languages holds*

Myhill-Nerode for Unranked Tree Automata

For tree language L, define relation $\equiv_L$ on trees

Definition



$\equiv_L$ is an equivalence relation on unranked trees

Myhill-Nerode for Unranked Tree Automata

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

Question

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

Question

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

Theorem (Jiang, Ravikumar 1993)

DFA $\rightarrow$ 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*

Back to the Basics

NFA Minimization

Question

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

Theorem (Jiang, Ravikumar 1993)

DFA $\rightarrow$ *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 Revisited

Can there be any non-determinism at all?

Definition (δNFA)

The class of NFAs that

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

Definition (δNFA)

The class of NFAs that

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

Theorem (Björklund, Mar., ICALP 2008)

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

$DFA \to \mathscr{C}$ **MINIMIZATION** *is* **NP**-*hard*

Conclusion and Outlook

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

Conclusion and Outlook

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