
S. Margherita di Pula
September 2004

Thomas Schwentick
Disclaimer

About this talk
It will be Theory
and it will be a lot of Theory

Welcome to this talk!

Note
The uptodate slides will be available from my homepage
The Big Picture

XML Languages

Known Formal Models

Fragment

New Formal Models
Composer

Vita

Name

Claude Debussy

Born

When

1862

Where

Paris

Married

When

1899

Whom

Rosalie

Married

When

1908

Whom

Emma

Died

When

1918

Where

Paris

Piece

PTitle

La Mer

PYear

1905

Instruments

Large orchestra

Movements

3
XML Processing

Four important kinds of XML processing ............ and their languages

Validation
Check whether an XML document is of a given type

Navigation
Select a set of positions in an XML document

Querying
Extract information from an XML document

Transformation
Construct a new XML document from a given one

Validation: DTD, XML Schema
Navigation: XPath
Querying: XQuery
Transformation: XSLT
Validation: DTD

DTDs describe types of XML documents

Example document

```xml
<Composer>
  <Name> Claude Debussy </Name>
  <Vita>
    <Born> <When> August 22, 1862 </When> <Where> Paris </Where> </Born>
    <Married> <When> October 1899 </When> <Whom> Rosalie </Whom> </Married>
    <Married> <When> January 1908 </When> <Whom> Emma </Whom> </Married>
    <Died> <When> March 25, 1918 </When> <Where> Paris </Where> </Died>
  </Vita>
  <Piece>
    <PTitle> La Mer </PTitle>
    <PYear> 1905 </PYear>
    <Instruments> Large orchestra </Instruments>
    <Movements> 3 </Movements>
    ...
  </Piece>
  ...
</Composer>
```

Example

```xml
<!DOCTYPE Composers [
  <!ELEMENT Composers (Composer*)>
  <!ELEMENT Composer (Name, Vita, Piece*)>
  <!ELEMENT Vita (Born, Married*, Died?)>
  <!ELEMENT Born (When, Where)>
  <!ELEMENT Married (When, Whom)>
  <!ELEMENT Died (When, Where)>
  <!ELEMENT Piece (PTitle, PYear, Instruments, Movements)>
]>
**Navigation: XPath**

XPath expressions select sets of nodes of XML documents by specifying navigational patterns.

**Example query**
```
//Vita/Died/*
```

---

**Example document**
```
<Composer>
  <Name>Claude Debussy</Name>
  <Vita>
    <Born><When>August 22, 1862</When><Where>Paris</Where></Born>
    <Married><When>October 1899</When><Whom>Rosalie</Whom></Married>
    <Married><When>January 1908</When><Whom>Emma</Whom></Married>
    <Died><When>March 25, 1918</When><Where>Paris</Where></Died>
  </Vita>
  <Piece>
    <PTitle>La Mer</PTitle>
    <PYear>1905</PYear>
    <Instruments>Large orchestra</Instruments>
    <Movements>3</Movements>
    ...
  </Piece>
  ...
</Composer>
```
XQuery is a full-fledged XML query language.

Example query:

```xml
for $x in doc('composers.xml')/Composer
where $x/Vita/Died/Where = 'Paris'
return $x/Name
```

Result:

```
(Name) Claude Debussy (/Name)
(Name) Eric Satie (/Name)
(Name) Hector Berlioz (/Name)
(Name) Camille Saint-Saëns (/Name)
(Name) Frédéric Chopin (/Name)
(Name) Maurice Ravel (/Name)
(Name) Jim Morrison (/Name)
(Name) César Franck (/Name)
(Name) Gabriel Fauré (/Name)
(Name) George Bizet (/Name)
```

```
(instrum) Large orchestra (/instrum)
(Movements) 3 (/Movements)
...
(Piece)
...
(Composer)
...
Transformation: XSLT

XSLT transforms documents by means of templates.

Example

XSLT transforms documents by means of templates.

Schwentick XML: Algorithms & Complexity Introduction - XML Processing

Belgium
A Schematic View

**DTD/ XML Schema**

- yes/no

**XPath**

**XQuery**

**XSLT**
Focus of this Talk

### Topics
- Expressive power of XML languages
- Complexity of algorithmic tasks related to XML processing
- Tradeoff between expressiveness and complexity

### Goals of this Research
- Understand expressive power and complexity of XML languages
- Identify interesting fragments with good tradeoff
Algorithmic Tasks

Evaluation

**Evaluation (Combined)**

**I:** Tree \( t \), Query \( q \)

**O:** \( q(t) \)

**Evaluation (Data(\( q \)))**

**I:** Tree \( t \)

**O:** \( q(t) \)

**Incremental Eval. (\( q \))**

**I:** Tree \( t \), Changes of \( t \)

**O:** \( q(t) \)

Static Analysis

**Satisfiability**

**I:** Query \( q \)

**Q:** Is \( q(t) \neq \emptyset \) for some \( t \)?

**Containment**

**I:** Queries \( q_1, q_2 \)

**Q:** Is always \( q_1(t) \subseteq q_2(t) \)?

**Equivalence**

**I:** Queries \( q_1, q_2 \)

**Q:** Is always \( q_1(t) = q_2(t) \)?

**Type Checking**

**I:** Types \( d_1, d_2 \), Transformation \( T \)

**Q:** Does \( t \models d_1 \) imply \( T(t) \models d_2 \)?

**Type Inference**

**I:** Types \( d \), Transformation \( T \)

**O:** Type of \( \{T(t) \mid t \models d\} \)
Expressive power

Question: How do we measure expressive power?

Remarks
- Classes of logical formulas are a good yardstick
  - They provide methods to prove that a query can not be expressed

Recall Relational Databases
- Core of SQL $\equiv$ First-order Logic
- Most frequently asked queries $\equiv$ Conjunctive queries
Contents

Introduction

Background on Tree Automata and Logic
Schema Languages
XPath and Node-selecting Queries
XSLT
XQuery
Conclusion
Background: Complexity Classes

Overview of Complexity Classes

Decidable

EXPSPACE

- Equivalence of reg. expressions with squaring

EXPTIME

- 2-Player Corridor Tiling

PSPACE

- Quantified Boolean Formulas

coNP

- Satisfiability of prop. formulas

NP

- Boolean circuit evaluation

P

- Efficiently parallelizable problems

NC

- Efficiently parallelizable problems

LOGCFL

- Acyclic conjunctive queries

NL

- Reachability in directed graphs

LOGSPACE

- Reachability in directed forests
Question: Why is XML appealing for Theory people?

Years ago...

- Theoretical Computer Science for Database Theorists: Logics, Complexity, Algorithms,...
- Database Theory for Theoretical Computer Scientists: terra incognita

After the advent of XML

Many connections between

- Formal Languages & Automata Theory
- and
- XML & Database Theory
**XML, Trees and Automata**

**Question:** Why trees?

**A Natural Answer**
- Trees reflect the hierarchical structure of XML
- Underlying data model of XML is tree based

**Limitations**
- But trees cannot model all aspects of XML (e.g., IDREFs, data values)
  - Sometimes extensions are needed
- E.g., directed graphs instead of trees

**Nevertheless**
In this tutorial we will concentrate on the tree view at XML

**Example**

---

Schwentick  
XML: Algorithms & Complexity  
Introduction - XML Processing  
16
Question: Why automata?

Ingredients of XML

Concepts from formal languages are obviously present around XML:

- Labelled trees
- DTD: context-free grammars
- DTD: regular expressions
- XPath: regular path expressions

We will see

Automata turn out to be useful as:

- a means to define robust classes with clear semantics
- a tool for proofs
- an algorithmic tool for static analysis
- a tool for query evaluation
Contents

Introduction

**Background on Tree Automata and Logic**
- Parallel Ranked Tree Automata
- Sequential Ranked Tree Automata
- Decision Problems for Ranked Tree Automata
- Parallel Unranked Tree Automata
- Sequential Unranked Tree Automata
- Sequential Document Automata

Schema Languages
- XPath and Node-selecting Queries
- XSLT
- XQuery

Conclusion
From Strings to Trees

A String

abcab

String as Tree

A Ranked Tree

An Unranked Tree
XML and Trees

- XML trees are **unranked**: the number of children of a node is not restricted
- Automata have first been considered on **ranked** trees, where each symbol has a fixed number of children (rank)
- Most important ideas were already developed for ranked trees

→ Let us take a look at this first
Sometimes trees are viewed as terms

Example Tree as Term

$$a^1(b^2(c^1(a^2(b, a)), b^2(a^2(b, c), c)))$$
Question

How do string automata generalize to trees?

Sequential

Parallel
**Bottom-Up Automata**

**Example: Tree-structured Boolean Circuits**

Idea
Tree-structured Boolean circuits
Two states: $q_0, q_1$
Accepting at the root: $q_1$

Transitions

- $\delta(\land, q_1) = \{(q_1, q_1)\}$
- $\delta(\land, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$
- $\delta(\lor, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$
- $\delta(\lor, q_0) = \{(q_0, q_0)\}$
- $\delta(0, q_0) = \{\epsilon\}; \delta(0, q_1) = \emptyset$
- $\delta(1, q_1) = \{\epsilon\}; \delta(1, q_0) = \emptyset$
Idea
Guess the correct values starting from the root
Check at the leaves
Three states: $q_0$, $q_1$, acc
Initial state $q_1$ at the root
Accepting if all leaves end in acc

Transitions
\[
\begin{align*}
\delta(\land, q_1) &= \{(q_1, q_1)\} \\
\delta(\land, q_0) &= \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\} \\
\delta(\lor, q_1) &= \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\} \\
\delta(\lor, q_0) &= \{(q_0, q_0)\} \\
\delta(0, q_0) &= \{\text{acc}\}; \delta(0, q_1) = \emptyset \\
\delta(1, q_1) &= \{\text{acc}\}; \delta(1, q_0) = \emptyset
\end{align*}
\]
Regular Tree Languages

Definition

A bottom-up automaton is **deterministic** if for each $a$ and $p \neq q$: $\delta(a, p) \cap \delta(a, q) = \emptyset$

Theorem

The following are equivalent for a tree language $L$:

(a) $L$ is accepted by a nondeterministic bottom-up automaton
(b) $L$ is accepted by a deterministic bottom-up automaton
(c) $L$ is accepted by a nondeterministic top-down automaton

Proof idea

(a) $\implies$ (b): Powerset construction
(a) $\iff$ (c): Just the same thing, viewed in two different ways
Automata as Tiling Systems

Observation

- \((q_0, q_1) \in \delta(\lor, q_1)\) can be interpreted as an allowed pattern:

- A tree is accepted, iff there is a labelling with states such that
  - all local patterns are allowed
  - the root is labelled with \(q_1\)

Example
Regular tree languages and logic

Definition: (MSO logic)

- **Formulas** talk about
  - edges of the tree (\(E\))
  - node labels (\(Q_0, Q_1, Q_\land, Q_\lor\))
  - the root of the tree (root)

- **First-order-variables** represent nodes

- **Monadic second-order** (MSO) variables represent sets of nodes

Example: Boolean Circuits

Boolean circuit true \(\equiv\) \(\exists X \ X\text{(root)} \land \forall x\)

\((Q_0(x) \rightarrow \neg X(x)) \land (Q_\land(x) \land X(x)) \rightarrow (\forall y [E(x, y) \rightarrow X(y)])) \land (Q_\lor(x) \land X(x)) \rightarrow (\exists y [E(x, y) \land X(y)])\)

Theorem [Doner 70; Thatcher, Wright 68]

MSO \(\equiv\) Regular Tree Languages
Theorem

\[ \text{MSO} \equiv \text{Regular Tree Languages} \]

Proof idea

**Automata $\Rightarrow$ MSO:**

Formula expresses that there exists a correct tiling

**MSO $\Rightarrow$ Automata:** more involved

Basic idea:

Automaton computes for each node $v$ the set of formulas which hold in the subtree rooted at $v$
Regular tree languages and logic (cont.)

**Formula ⇒ automaton**

- Let \( \varphi \) be an MSO-formula, \( k := \) quantifier-depth of \( \varphi \)
- **\( k \)-type** of a tree \( t := (\text{essentially}) \)
  - set of MSO-formulas \( \psi \) of quantifier-depth \( \leq k \) which hold in \( t \)
- \( t_1 \equiv_k t_2 : k\text{-type}(t_1) = k\text{-type}(t_2) \)
- Automaton computes \( k \)-type of tree and concludes whether \( \varphi \) holds

**Crucial fact**
Det. Top-Down Automata

Question
What is the right notion for deterministic top-down automata?

3 Possibilities
State at a node $v$ might depend on

- state and symbol of parent

- state and symbol of parent and symbol of $v$

- state and symbol of parent and symbols at $v$ and its sibling
### Question
What is a good acceptance mechanism for deterministic top-down automata?

### Several possibilities
1. At all leaves states have to be accepting
2. There is a leave with an accepting state

### Observations
2. is problematic for complement and intersection
1. is problematic for complement and union
Definition: (Root-to-frontier automata with regular acceptance condition)

- Tree automata $\mathcal{A}$ are equipped with an additional regular string language $L$ over $Q \times \Sigma$
- $\mathcal{A}$ accepts $t$ if the (state,symbol)-string at the leaves (from left to right) is in $L$ [Jurvanen, Potthoff, Thomas 93]

Illustration

A robust class

- The resulting class is closed under Boolean operations
- Good algorithmic properties
- Does not capture all regular tree languages
## Summary

### Regular tree languages

- Regular tree languages are a robust class
- Characterized by
  - parallel tree automata
  - MSO logic
  - several other models
- They are the natural analog of regular string languages
- Deterministic top-down automata with regular acceptance conditions define a weaker but nevertheless robust class
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduction</strong></td>
</tr>
<tr>
<td><strong>Background on Tree Automata and Logic</strong></td>
</tr>
<tr>
<td>Parallel Ranked Tree Automata</td>
</tr>
<tr>
<td>Sequential Ranked Tree Automata</td>
</tr>
<tr>
<td>Decision Problems for Ranked Tree Automata</td>
</tr>
<tr>
<td>Parallel Unranked Tree Automata</td>
</tr>
<tr>
<td>Sequential Unranked Tree Automata</td>
</tr>
<tr>
<td>Sequential Document Automata</td>
</tr>
<tr>
<td>Schema Languages</td>
</tr>
<tr>
<td>XPath and Node-selecting Queries</td>
</tr>
<tr>
<td>XSLT</td>
</tr>
<tr>
<td>XQuery</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
Tree-Walk Automata

Definition: (Tree-walk automata)

Depending on
- current state
- symbol of current node
- position of current node wrt its siblings

the automaton moves to a neighbor and takes a new state

Question

What is the expressive power of tree-walk automata?
Tree-Walk Automata (cont.)

Fact
- Tree-walk automata can evaluate Boolean circuit trees
- 5 states

Example

Idea

$q^0$

$q^1$
A Recent Result and an Even More Recent Result

**Theorem [Bojanczyk, Colcombet 04]**
Deterministic TWAs are weaker than nondeterministic TWAs

**Corollary**
Deterministic TWAs do not capture all regular tree languages

**Theorem [Bojanczyk, Colcombet 04]**
Nondeterministic TWAs do not capture all regular tree languages
Overview of Models

- Non-det. top-down tree automata
- Non-det. bottom-up tree automata
- Det. bottom-up tree automata
- Non-det. tree walk automata
- Det. top-down tree automata
- Det. tree walk automata
# Decision Problems

## Algorithmic problems
- We consider the following algorithmic problems
- All of them will be useful in the XML context

### Membership test for $\mathcal{A}$
- **Given:** Tree $t$
- **Question:** Is $t \in L(\mathcal{A})$?

### Membership test (combined)
- **Given:** Automaton $\mathcal{A}$, tree $t$
- **Question:** Is $t \in L(\mathcal{A})$?

### Non-emptiness
- **Given:** Automaton $\mathcal{A}$
- **Question:** Is $L(\mathcal{A}) \neq \emptyset$?

### Containment
- **Given:** Automata $\mathcal{A}_1, \mathcal{A}_2$
- **Question:** Is $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$?

### Equivalence
- **Given:** Automata $\mathcal{A}_1, \mathcal{A}_2$
- **Question:** Is $L(\mathcal{A}_1) = L(\mathcal{A}_2)$?
Facts

Time Bounds for the combined complexity of membership test for tree automata:

- Deterministic (parallel) tree automata: $O(|A||t|)$
- Nondeterministic (parallel) tree automata: $O(|A|^2|t|)$
  (Compute, for each node, the set of reachable states)
- Deterministic TWAs: $O(|A|^2|t|)$
  (Compute, for each node $v$, the aggregated behavior of $A$ on its subtree: Behavior function)
- Nondeterministic TWAs: $O(|A|^3|t|)$
  (Compute, for each node $v$, the aggregated behavior of $A$ on its subtree: Behavior relation)
Question: What is the structural complexity for the various models?

<table>
<thead>
<tr>
<th>Model</th>
<th>Time Complexity</th>
<th>Structural Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Det. top-down TA</td>
<td>$O(</td>
<td>A</td>
</tr>
<tr>
<td>Det. bottom-up TA</td>
<td>$O(</td>
<td>A</td>
</tr>
<tr>
<td>Nondet. bottom-up TA</td>
<td>$O(</td>
<td>A</td>
</tr>
<tr>
<td>Nondet. top-down TA</td>
<td>$O(</td>
<td>A</td>
</tr>
<tr>
<td>Det. TWA</td>
<td>$O(</td>
<td>A</td>
</tr>
<tr>
<td>Nondet. TWA</td>
<td>$O(</td>
<td>A</td>
</tr>
</tbody>
</table>

[Reference: Lohrey 01, Segoufin 03]
Non-emptiness

Facts
- Non-emptiness for string automata corresponds to Graph Reachability (complete for NLOGSPACE)
- Non-emptiness for tree automata corresponds to Path Systems

Result
- Non-emptiness for bottom-up tree automata can be checked in linear time
- It is complete for PTIME
Observations

- Of course:
  \[ L(A_1) = L(A_2) \iff [L(A_1) \subseteq L(A_2) \text{ and } L(A_2) \subseteq L(A_1)] \]

- Complexity of containment problem is very different for deterministic and non-deterministic automata

- For deterministic automata: construct product automaton
Reminder: Product automaton

Product of 2 string automata

- "even number of zeros"
- "contains substring 00"

```
0110100
```

Schwentick
XML: Algorithms & Complexity
Introduction - XML Processing
45
Containment: Complexity

Deterministic bottom-up tree automata

- Product automaton analogous as for string automata
  - Set of states: \( Q_1 \times Q_2 \)
  - Transitions component-wise

- To check \( L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2) \):
  - Compute \( \mathcal{B} = \mathcal{A}_1 \times \mathcal{A}_2 \)
  - Accepting states: \( F_1 \times (Q_2 - F_2) \)
  - Check whether \( L(\mathcal{B}) = \emptyset \)
  - If so, \( L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2) \) holds

Theorem

Complexity of Containment for deterministic bottom-up tree automata:

\[
O(|\mathcal{A}_1| \times |\mathcal{A}_2|)
\]
Non-deterministic automata

- Naive approach:
  - Make $A_2$ deterministic (size: $O(2^{|A_2|})$)
  - Construct product automaton

$\Rightarrow$ Exponential time

Unfortunately...
There is essentially no better way

Theorem [Seidl 1990]
Containment for non-deterministic tree automata is complete for EXPTIME
Theorem

Nonemptiness for deterministic top-down automata $A$ can be checked in polynomial time.

Proof idea

Check for each state $p$ of $A$ and each pair $(q, q')$ of the leaves automaton $B$:
Is there a tree $t$ such that $A$ starts from state $p$ and obtains a leave string which brings $B$ from $q$ to $q'$?
Theorem

Containment for deterministic top-down automata \( \mathcal{A} \) can be checked in polynomial time.

Proof idea

- Tree automata \( \mathcal{A}_1, \mathcal{A}_2 \) with leaves automata \( \mathcal{B}_1, \mathcal{B}_2 \)
- Check
  - for each pair \( (p_1, p_2) \) of states of \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \) and
  - for each two pairs \( (q_1, q'_1) \) and \( (q_2, q'_2) \) of \( \mathcal{B}_1 \) and \( \mathcal{B}_2 \), resp.: Is there a tree \( t \) such that for both \( i = 1, \ i = 2 \): \( \mathcal{A}_i \) starts from state \( p_i \) and obtains a leave string which brings \( \mathcal{B}_i \) from \( q_i \) to \( q'_i \)?
### Complexities of basic algorithmic problems

<table>
<thead>
<tr>
<th>Model</th>
<th>Membership</th>
<th>Non-emptiness</th>
<th>Containment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Det. top-down TA</td>
<td>LOGSPACE</td>
<td>PTIME</td>
<td>PTIME</td>
</tr>
<tr>
<td>Det. bottom-up TA</td>
<td>LOGDCFL</td>
<td>PTIME</td>
<td>PTIME</td>
</tr>
<tr>
<td>Nondet. bottom-up TA</td>
<td>LOGCFL</td>
<td>PTIME</td>
<td>EXPTIME</td>
</tr>
<tr>
<td>Nondet. top-down TA</td>
<td>LOGCLFL</td>
<td>PTIME</td>
<td>EXPTIME</td>
</tr>
<tr>
<td>Det. TWA</td>
<td>LOGSPACE</td>
<td>PTIME (*)</td>
<td>PTIME (*)</td>
</tr>
<tr>
<td>Nondet. TWA</td>
<td>NLOGSPACE</td>
<td>PTIME (*)</td>
<td>EXPTIME (*)</td>
</tr>
</tbody>
</table>

(*: upper bounds)
Example Tree

Composer

Name

Born

When

Where

Claude Debussy

1862

Paris

Married

When

Whom

Rosalie

1899

Emma

1908

Died

When

Where

Paris

1918

Piece

PTitle

La Mer

PYear

1905

Instruments

Large orchestra

Movements

3
Now we move from ranked to unranked trees

There is a basic choice:

- Either: we encode unranked trees as binary trees and go on with ranked automata
- Or: we adapt the ranked automata models

In both cases: not many surprises, most results remain
Encoding Unranked Trees as Binary Trees

Example: Unranked Tree

```
   a
  / \  \
 c   e  a  c
 / \  / \  /  \
 a  c c  e a
```

Encoding via ...

- first child
- next sibling

... as Binary Tree
Encoding Unranked Trees as Binary Trees (cont.)

Example: Unranked Tree

... if path expressions matter (Milo, Suciu, Vianu 00)
There are still other ways to encode unranked trees as binary trees,
e.g., [Carme, Niehren, Tommasi 04]. We consider automata for unranked trees next.
**Unranked Trees: Formal Definition**

**Definition**

A *(finite) tree domain* \( V \) over \( \mathbb{N} \) is a *(finite) subset of \( \mathbb{N}^* \), such that if \( v \cdot i \in V \), where \( v \in \mathbb{N}^* \) and \( i \in \mathbb{N} \),

- then \( v \in V \)
- and \( v \cdot (i - 1) \in V \), if \( i > 1 \)

**Note**

\( \varepsilon \) represents the root

**Definition**

A *(labelled tree)* \( t \) is a pair \( (V, \lambda) \), where \( V \) is a tree domain over \( \mathbb{N} \), and \( \lambda \) is a function from \( V \) to the set \( \Sigma \) of labels.

**Remark**

XML tags can be captured by the set \( \Sigma \) of labels. But what about text?

- This depends on the context
- E.g., for type checking, text is irrelevant.
- In many applications, the relevant information about text nodes can be represented by predicates, e.g., whether the name = ’Debussy’.
From Ranked to Unranked Tree Automata

**Ranked trees**

Transitions are described by finite sets:

\[ \delta(\sigma, q) = \{(q_1, q_2), (q_3, q_4), \ldots \} \]

- For unranked trees, \( \delta(\sigma, q) \) is a regular language
- \( \delta(\sigma, q) \) can be specified by regular expression or finite string automaton

[Brüggemann-Klein, Murata, Wood 2001]
Representation of $\delta(\sigma, q)$

Remark

- Representation of $\delta(\sigma, q)$ has influence on complexity
- Natural choice:
  - For nondeterministic tree automata: represent $\delta(\sigma, q)$ by NFAs or regular expressions
  - For deterministic tree automata: represent $\delta(\sigma, q)$ by DFAs

$\Rightarrow$ Same complexity results as for ranked trees
The following are equivalent for a set $L$ of unranked trees:

(a) $L$ is accepted by a nondeterministic bottom-up automaton

(b) $L$ is accepted by a deterministic bottom-up automaton

(c) $L$ is accepted by a nondeterministic top-down automaton

(d) $L$ is characterized by an MSO-formula
## Deterministic Top-Down Automata

<table>
<thead>
<tr>
<th>State at $v$ might depend on ...</th>
<th><img src="image" alt="Diagram" /></th>
</tr>
</thead>
<tbody>
<tr>
<td>state and symbol of parent</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>state and symbol of parent and</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>symbol of $v$</td>
<td>simple</td>
</tr>
<tr>
<td>state and symbol of parent and</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>symbols at $v$ and its left</td>
<td>left-siblings</td>
</tr>
<tr>
<td>siblings</td>
<td>aware</td>
</tr>
<tr>
<td>state and symbol of parent and</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>symbols at $v$ and its siblings</td>
<td></td>
</tr>
</tbody>
</table>
Fact

A simple deterministic top-down automaton can check the existence of vertical paths with regular properties

Construction

- For a node $v$ let $s(v)$ denote the sequence of labels from the root to $v$
- Let $\mathcal{A}$ be a deterministic string automaton
- $\mathcal{A}' :=$ top-down automaton which takes at $v$ state of $\mathcal{A}$ after reading $s(v)$

$\Rightarrow \mathcal{A}'$ is deterministic

- There is a path from the root to a leaf $v$ with $s(v) \in L(\mathcal{A})$ iff $\mathcal{A}'$ assumes at least one state from $F$ at a leave

Streaming XML

Similar construction used for XPath evaluation on streams [Green et al. 2003]
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduction</strong></td>
</tr>
<tr>
<td><strong>Background on Tree Automata and Logic</strong></td>
</tr>
<tr>
<td>Parallel Ranked Tree Automata</td>
</tr>
<tr>
<td>Sequential Ranked Tree Automata</td>
</tr>
<tr>
<td>Decision Problems for Ranked Tree Automata</td>
</tr>
<tr>
<td>Parallel Unranked Tree Automata</td>
</tr>
<tr>
<td><strong>Sequential Unranked Tree Automata</strong></td>
</tr>
<tr>
<td>Sequential Document Automata</td>
</tr>
<tr>
<td>Schema Languages</td>
</tr>
<tr>
<td>XPath and Node-selecting Queries</td>
</tr>
<tr>
<td>XSLT</td>
</tr>
<tr>
<td>XQuery</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
Sequential Automata on Unranked Trees

Generalization of Tree-Walk Automata

Allowed transitions:
- Go up
- Go to first child
- Go to left sibling
- Go to right sibling

→ Caterpillar automata [Brüggemann-Klein, Wood 2000]

Basic design choice

Should a transition to a sibling be aware of the label of the parent?

![Diagram of a tree with a label 'a' and children labeled 'v' and 'w']
## Contents

### Introduction

### Background on Tree Automata and Logic

- Parallel Ranked Tree Automata
- Sequential Ranked Tree Automata
- Decision Problems for Ranked Tree Automata
- Parallel Unranked Tree Automata
- Sequential Unranked Tree Automata

**Sequential Document Automata**

### Schema Languages

- XPath and Node-selecting Queries
- XSLT
- XQuery

### Conclusion
A third kind of automata for XML

- **Document automata** are string automata reading XML documents as text
- Tags are represented by symbols from a given alphabet
- Variants:
  - Finite document automata
  - Pushdown document automata
- Useful especially in the context of streaming XML

**Theorem [Segoufin, Vianu 02]**

- Regular languages of XML-trees can be recognized by deterministic push-down document automata.
- Depth of push-down is bounded by depth of tree
Summary

- Moving from ranked to unranked automata requires some adaptations.
- Transitions can be defined with regular string languages \( \delta(\sigma, q) \).
- By and large, things work smoothly.
- In particular:
  - There is an equally robust notion of regular tree languages.
  - The complexities are the same as for ranked automata (if the sets \( \delta(\sigma, q) \) are represented in a sensible way).
Refined Overview of Models

- Non-det. top-down tree automata
- Non-det. bottom-up tree automata
- Det. bottom-up tree automata
- Pushdown document automata
- Non-det. tree walk automata
- Det. tree walk automata
- Finite document automata
DTDs

Example DTD

```xml
<!DOCTYPE Composers [
  <!ELEMENT Composers (Composer*)>
  <!ELEMENT Composer (Name, Vita, Piece*)>
  <!ELEMENT Vita (Born, Married*, Died?)>
  <!ELEMENT Born (When, Where)>
  <!ELEMENT Married (When, Whom)>
  <!ELEMENT Died (When, Where)>
  <!ELEMENT Piece (PTitle, PYear, Instruments, Movements)>
]>
```

Some Facts

- DTDs \(\equiv\) generalized context-free grammars
  
  \[\text{[Berstel, Boasson 00]}\] provide characterizations

- Additional restriction: **one-unambiguous**
Definition: One-unambiguous Regular Expression

- Let $r$ be a regular expression
- $r \leftrightarrow r'$: number the symbols of $r$ from left to right
- $w \in L(r) \iff$ there is a numbered string $w' \in L(r')$
- $r$ is one-unambiguous if $ux_iv \in L(r'), uy_jw \in L(r'), i \neq j \Rightarrow x \neq y$

Example

- $(a + b)^*ac + c \leftrightarrow (a_1 + b_2)^*a_3c_4 + c_5$
- $babbac \in L(r)$ and $b_2a_1b_2b_2a_3c_4 \in L(r')$
- $(a + b)^*ac + c$ is not one-unambiguous because $b_2b_2 \textcolor{red}{a_3}c_4 \in L(r')$ and $b_2b_2 \textcolor{red}{a_1}a_3c_4 \in L(r')$
- $(b^*a)^*c$ is one-unambiguous

Restriction

- Expressions in DTDs have to be one-unambiguous
- Inherited from SGML
Validation wrt a DTD

Example Tree

Composer
  Name
    Debussy
  Vita
    Born
      When
      1862
      Where
      Paris
    Married
      When
      1899
      Whom
      Rosalie
    Married
      When
      1908
      Whom
      Emma
    Died
      When
      1918
      Where
      Paris
  Piece
    PTitle
    La Mer
    PYear
    1905
    Instruments
    Orch.
    Movements
    3

Example DTD

<!DOCTYPE Composers [  
  <!ELEMENT Composers (Composer*)>  
  <!ELEMENT Composer (Name, Vita, Piece*)>  
  <!ELEMENT Vita (Born, Married*, Died?)>  
  <!ELEMENT Born (When, Where)>  
  <!ELEMENT Married (When, Whom)>  
  <!ELEMENT Died (When, Where)>  
  <!ELEMENT Piece (PTitle, PYear,  
    Instruments, Movements)>  
]>
Validation wrt a DTD (cont.)

Observation

- Validation wrt DTDs is a very simple task
- Can be done by
  - Bottom-up automata
  - Deterministic top-down automata
    (if siblings contribute to new state)
  - Deterministic tree-walk automata:
    Just a depth-first left-to-right traversal
- In particular: Validation possible in linear time during one pass through the document
  (1-pass validation)
**Example**

\[
\begin{align*}
    a & \rightarrow bc \\
    b & \rightarrow cd \\
    c & \rightarrow \epsilon \\
    d & \rightarrow \epsilon
\end{align*}
\]

**Fact**

Satisfiability for DTDs is complete for $\text{PTIME}$
Lemma [Martens, Neven, Sch. 04]

Containment of DTDs with regular expressions from $R$ is in $C$ if and only if

Containsment of regular expressions from $R$ is in $C$

Corollary

Containment of DTDs (with one-unambiguous regular expressions) is in $\text{PTIME}$

Proof sketch

- One-unambiguous regular expressions have deterministic automata of linear size

$\Rightarrow$ Containment of regular expressions $r_1, r_2$ by product automaton of size $O(|r_1||r_2|)$
Containment of DTDs (cont.)

Question
What if the requirement of being one-unambiguous is dropped?

A classical result

Theorem [Stockmeyer, Meyer 71]
Containment and Equivalence for regular expressions on strings are complete for PSPACE

Corollary
Containment of DTDs (with unrestricted regular expressions) is PSPACE-complete

Theorem [Martens, Neven, Sch. 04]
Containment and Equivalence for regular expressions are
- coNP-complete for concatenations of $a, b, c$ and $a^*, b^*, c^*$
- coNP-complete for concatenations of $a, b, c$ and $a?, b?, c?$
- PSPACE-complete for concatenations of $a, b, c$ and $(a^* + b^* + \cdots + c^*)$
Contents

Introduction

Background on Tree Automata and Logic

**Schema Languages**

- DTDs
  - Specialized DTDs
  - 1-pass Preorder Typing

XPath and Node-selecting Queries

XSLT

XQuery

Conclusion
Weakness of DTDs

A classical example

```xml
<!DOCTYPE Dealer [
  <!ELEMENT Dealer (UsedCars, NewCars)>
  <!ELEMENT UsedCars (ad*)>
  <!ELEMENT NewCars (ad*)>
  <!ELEMENT ad ((model, year) | model)>
]>
```

Intention

Intention:

```
Dealer
  UsedCars
    ad
      model
      year
  NewCars
    ad
      model
```

Observation

- Elements with the same name may have different structure in different contexts

→ It would be nice to have types for elements

→ Specialized DTDs
Specialized DTDs

Definition: [Papakonstantinou, Vianu 2000]

A specialized DTD (SDTD) over alphabet \( \Sigma \) is a pair \((d, \mu)\), where

- \( d \) is a DTD over the alphabet \( \Sigma' \) of types
- \( \mu : \Sigma' \rightarrow \Sigma \) maps types to tag names

Note

Concerning the name:
“specialized” refers to types, not to DTDs

Example

Dealer \(\rightarrow\) UsedCars NewCars \quad \mu(\text{Dealer}) = \text{Dealer}

UsedCars \(\rightarrow\) adUsed* \quad \mu(\text{UsedCars}) = \text{UsedCars}

NewCars \(\rightarrow\) adNew* \quad \mu(\text{NewCars}) = \text{NewCars}

adUsed \(\rightarrow\) model year \quad \mu(\text{adUsed}) = \text{ad}

adNew \(\rightarrow\) model \quad \mu(\text{adNew}) = \text{ad}
A Further Example

**Example: SDTD for Boolean circuit trees**

<table>
<thead>
<tr>
<th>Tag</th>
<th>μ(Tag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-AND</td>
<td>AND</td>
</tr>
<tr>
<td>0-AND</td>
<td>AND</td>
</tr>
<tr>
<td>1-OR</td>
<td>OR</td>
</tr>
<tr>
<td>0-OR</td>
<td>OR</td>
</tr>
<tr>
<td>1-leaf</td>
<td>1</td>
</tr>
<tr>
<td>0-leaf</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-AND</td>
<td>→ (1-OR</td>
</tr>
<tr>
<td>1-OR</td>
<td>→ .* (1-OR</td>
</tr>
<tr>
<td>0-AND</td>
<td>→ .* (0-OR</td>
</tr>
<tr>
<td>0-OR</td>
<td>→ (0-OR</td>
</tr>
<tr>
<td>1-leaf</td>
<td>→ ε</td>
</tr>
<tr>
<td>0-leaf</td>
<td>→ ε</td>
</tr>
</tbody>
</table>
Observation
- A naive validation by exhaustively trying all possible functions $\mu$ requires exponential time
- But help comes from automata...
- A tree conforms to a specialized DTD $(d, \mu)$ if there is a labeling of its nodes by types which is valid wrt. $d$
- This reminds us of something...

Theorem
Specialized DTDs capture exactly the regular tree languages
## Validation and Typing

<table>
<thead>
<tr>
<th>Definition: Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given: Specialized DTD $d$, tree $t$</td>
</tr>
<tr>
<td>Question: Is $t$ valid wrt $d$?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Definition: Typing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given: Specialized DTD $d$, tree $t$</td>
</tr>
<tr>
<td>Output: Consistent type assignment for the nodes of $t$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Specialized DTDs $\equiv$ regular tree languages</td>
</tr>
<tr>
<td>$\rightarrow$ Validation in linear time by deterministic push-down automata</td>
</tr>
<tr>
<td>• Typing in linear time (Bottom-up automaton)</td>
</tr>
<tr>
<td>• Satisfiability $\equiv$ Non-emptiness of tree automata: PTIME</td>
</tr>
</tbody>
</table>
Restrictions of Schemas

Restricted Schemas

(Murata, Lee, Mani 2001) introduced* restrictions on specialized DTDs to ensure efficient validation

(*): in a slightly different framework

- Two types $b, b'$ compete if $\mu(b) = \mu(b')$

- A specialized DTD is single-type if no competing types occur in the same rule
  (e.g., $a \rightarrow bcb'$ is not single-type)

- A specialized DTD is restrained-competition if no rule allows strings $wbv, wb'v'$ with competing types $b, b'$
  (e.g., $a \rightarrow c(b + d^*b')$ is not restrained-competition)

- The authors argue that XML-Schema roughly corresponds to single-type SDTDs
## Schema Containment

**Given:** Schemas $d_1, d_2$  

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

### Observations

- **Important,** e.g., for data integration
- **Recall:** Specialized DTDs are essentially non-deterministic tree automata

$\Rightarrow$ Containment of specialized DTDs is in **EXPTIME**

- **But the restricted forms have lower complexity**
- **Complexity of containment depends on the allowed regular expressions**
Schema Containment: Complexity

<table>
<thead>
<tr>
<th>Schema type</th>
<th>unrestricted</th>
<th>deterministic expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DTDs</strong></td>
<td><strong>PSPACE</strong></td>
<td><strong>PTIME</strong></td>
</tr>
<tr>
<td>single-type SDTDs</td>
<td><strong>PSPACE</strong></td>
<td><strong>PTIME</strong></td>
</tr>
<tr>
<td>restrained-competition</td>
<td><strong>PSPACE</strong></td>
<td><strong>PTIME</strong></td>
</tr>
<tr>
<td>SDTDs</td>
<td><strong>EXPTIME</strong></td>
<td><strong>EXPTIME</strong></td>
</tr>
<tr>
<td>unrestricted SDTDs</td>
<td><strong>EXPTIME</strong></td>
<td><strong>EXPTIME</strong></td>
</tr>
</tbody>
</table>

Observations

- For unrestricted SDTDs the complexity is dominated by tree automata containment.
- For the others it is dominated by the sub-task of checking containment for regular expressions.
Schema Containment: Complexity

Observations (cont.)

- ... for the others it is dominated by the sub-task of checking containment for regular expressions
- Actually, this observation can be made more precise

Theorem [Martens, Neven, Sch. 04]
For a class $\mathcal{R}$ of regular expressions and a complexity class $\mathcal{C}$, the following are equivalent

(a) The containment problem for $\mathcal{R}$ expressions is in $\mathcal{C}$.

(b) The containment problem for DTDs with regular expressions from $\mathcal{R}$ is in $\mathcal{C}$.

(c) The containment problem for single-type SDTDs with regular expressions from $\mathcal{R}$ is in $\mathcal{C}$. 
Contents

Introduction

Background on Tree Automata and Logic

**Schema Languages**

- DTDs
- Specialized DTDs
  - 1-pass Preorder Typing

XPath and Node-selecting Queries

XSLT

XQuery

Conclusion
Typing (cont.)

Observations

- Type of a node $\equiv$ state of deterministic bottom-up automaton
- Deterministic push-down automaton can assign types during 1 pass
- But the type of a node $v$ is determined after visiting its subtree

1-pass preorder typing:

determine type of $v$ before visiting the subtree of $v$
### Question
When would it be important to know the type of $v$ before visiting the subtree of $v$?

### Answer
Whenever the processing proceeds in document order, e.g.:

- Streaming XML: Typing as the first operator in a pipeline
- SAX-based processing

### Our next goal
Find out which schemas admit 1-pass preorder typing
1-Pass Preorder Typing (cont.)

**Remarks**

- The definition of “1-pass preorder typing” does not yet restrict the efficiency of determining the type of a node
- Typing could be 1-pass preorder but very time consuming
- It turns out that essentially this never happens
- Clearly, restrained competition is sufficient for 1-pass preorder typing
- Is it also necessary?

**Theorem [Martens, Neven, Sch. 2004]**

For a regular tree language $L$ the following are equivalent

(a) $L$ can be described by a 1-pass preorder typable SDTD
(b) $L$ can be described by a restrained-competition SDTD
(c) $L$ has linear time 1-pass pre-order typing
(d) $L$ can be preorder-typed by a deterministic pushdown document automaton
(e) Types for trees in $L$ can be computed by a left-siblings-aware top-down deterministic tree automaton
Further characterizations

- This class has further interesting characterizations
- E.g., by closure under ancestor-sibling-guarded subtree exchange
A Related Result

Theorem [Martens, Neven, Sch. 2004]

For a regular tree language $L$ the following are equivalent

(a) $L$ can be described by a single-type SDTD

(b) Types for trees in $L$ can be computed by a simple top-down deterministic tree automaton

(c) $L$ is closed under ancestor-guarded subtree exchange
### Summary: Schema Languages

#### Expressive power
- Regular tree languages offer a nice framework (\(\equiv\) MSO logic!)
- Restrained competition \(\equiv\) Deterministic top-down automata

#### Validation
**Linear time**

#### Typing
- **Linear time**
- Efficient 1-pass preorder typing for restrained competition SDTDs

#### Satisfiability
- DTDs: **PTIME**
- SDTDs: **PTIME**

#### Containment
- General SDTDs: **EXPTIME**
- Restrained competition SDTDs: **PTIME**
## Contents

- Introduction
- Background on Tree Automata and Logic
- Schema Languages

### XPath and Node-selecting Queries

- Node-selecting Queries
- XPath: Semantics and Fragments
- XPath: Expressive Power
- XPath: Evaluation
- XPath: Satisfiability
- XPath: Containment

- XSLT
- XQuery
- Conclusion
**Node-Selecting Queries**

**Example document**

```xml
<Composer>
  <Name>Claude Debussy</Name>
  <Vita>
    <Born><When>August 22, 1862</When><Where>Paris</Where></Born>
    <Married><When>October 1899</When><Whom>Rosalie</Whom></Married>
    <Married><When>January 1908</When><Whom>Emma</Whom></Married>
    <Died><When>March 25, 1918</When><Where>Paris</Where></Died>
  </Vita>
  <Piece>
    <PTitle>La Mer</PTitle>
    <PYear>1905</PYear>
    <Instruments>Large orchestra</Instruments>
    <Movements>3</Movements>
    ...
  </Piece>
  ...
</Composer>
```

**Example query**

`//Vita/Died/*`

**Observation**

**XPath** expressions define sets of nodes → **node-selecting queries**
Node-Selecting Queries (cont.)

Question

Is there a class of node-selecting queries, as robust as the regular tree languages?

Observation

- There is a simple way to define node selecting queries by monadic second-order formulas:

  - Simply use one free variable: \( \varphi(x) \)

- Is there a corresponding automaton model?

- It is relatively easy to add node selection to nondeterministic bottom-up automata

Definition: (Nondeterministic bottom-up node-selecting automata)

- Nondeterministic bottom-up automata plus select function:

  \[ s : Q \times \Sigma \rightarrow \{0, 1\} \]

- Node \( v \) is in result set for tree \( t \) if there is an accepting computation on \( t \) in which \( v \) gets a state \( q \) such that \( s(q, \lambda(v)) = 1 \)
Example Automaton

Example query

\texttt{//*[a]//b}

Example automaton

- \( Q = \{q_0, q_a, q_b\} \)
- \( L(q_a, a) = Q^* \)
- \( L(q_b, \sigma) = Q^* \)
- \( L(q_0, \sigma) = \epsilon + q_0^* + Q^* q_a Q^* \)
- all other sets empty
- \( s(q_b, b) = 1 \)
- all others: 0
- Accepting: \( q_0 \)

Example tree: Run 2

Query tree

\[ \star \]

\texttt{a} \hspace{1cm} \texttt{b}
**Fact**

- Existential semantics: a node is in the result if there exists an accepting run which selects it
- Universal semantics: a node is in the result if every accepting run selects it
- Both semantics define the same class of queries

**Result**

A node selecting query is MSO-definable iff it is expressible by a nondeterministic bottom-up node selecting automaton
A node selecting query is MSO-definable iff it is expressible by a nondeterministic bottom-up node selecting automaton.

Proof idea

- Given formula $\varphi(x)$ of quantifier-depth $k$ and tree $t$,
  for each node $v$ the automaton does the following:
  - Compute $k$-type of subtree at $v$
  - Guess $k$-type of "envelope tree" at $v$
  - Conclude whether $v$ is in the output
  - Check consistency upwards towards the root

$\Rightarrow$ one unique accepting run

Crucial fact

$$
\begin{align*}
& e_1 \equiv_k t_1 \\
& e_2 \equiv_k t_2 \\
\Rightarrow & e_1 \equiv_k t_1 \\
& e_2 \equiv_k t_2
\end{align*}
$$
Equivalent Models

More query models

- Unfortunately, the translation from formula to automaton can be prohibitively expensive: number of states \( \sim 2^{2^{2^{|\varphi|}}} \).

- Actually: If \( P \neq NP \) there is no elementary \( f \), such that MSO-formulas can be evaluated in time \( f(|\text{formula}| \times p(|\text{tree}|)) \) with polynomial \( p \) [Frick, Grohe 2002]

\[ \rightarrow \] query languages with better complexity properties needed

- Good candidate: Monadic Datalog [Gottlob, Koch 2002] and its restricted dialects like TMNF

- Further models:
  - Attributed Grammars [Neven, Van den Bussche 1998]
  - \( \mu \)-formulas [Neumann 1998]
  - Context Grammars [Neumann 1999]
  - Deterministic Node-Selecting Automata [Neven, Sch. 1999]
Some facts about query evaluation

- MSO node-selecting queries can be evaluated in two passes through the tree
  - first pass, bottom-up: essentially computes the types of the subtrees
  - second pass, top-down: essentially computes the types of the envelopes and combines it with the subtree information

- This can be implemented by a 2-pass pushdown document automaton which in its first pass attaches information to each node

- In particular: queries can be evaluated in linear time

[Neumann, Seidl 1998; Koch 2003]
Node-selecting Queries: Static Analysis

Facts

- Satisfiability: Non-emptiness of node-selecting automata is $\text{PTIME}$-complete.
- Satisfiability of MSO-queries is non-elementary.
- Containment of node-selecting automata is $\text{EXPTIME}$-complete.
### Summary

- There is a natural notion of **regular node-selecting queries** generalizing regular tree languages
- Probably for most practical purposes too strong
- But it offers a useful framework for the study of other classes of queries
- A robust but weaker class of queries is captured by pebble automata
## Contents

- Introduction
- Background on Tree Automata and Logic
- Schema Languages

### XPath and Node-selecting Queries

- Node-selecting Queries
  - **XPath: Semantics and Fragments**
  - XPath: Expressive Power
  - XPath: Evaluation
  - XPath: Satisfiability
  - XPath: Containment

- XSLT
- XQuery

- Conclusion
XPath Fragments

Many fragments of XPath have been defined

The main fragments we consider are:

- **Full XPath**: XPath 1.0
  (besides the namespace related functions)

- **Navigational XPath** [Gottlob, Koch, Pichler 03, Benedikt, Fan, Kuper 03]:
  Location paths along all axes plus Boolean operations
  (no attributes, no relational operators)

- **Forward XPath**: Navigational XPath restricted to child, descendant, self, descendant-or-self
Main Ingredients of Navigational XPath

- **Location Step**:
  \[ p = \text{Axis} :: \text{Node-Test Predicate}^* \]
- **Predicate**: [Expression]
- **Location Path**:
  \[ \pi = \text{Location Step} / \text{Location Path} \]
  More explicitly: \[ \pi = p_1/\cdots/p_k \]
- **Expression**: basically a Boolean combination of location steps

Example

```
/descendant::*a/
  child::*[descendant::*c and not following-sibling::*b]/
  descendant::*a
```
Example XPath Expression

/desc::a/child::*[desc::c and not foll-sib::b]/desc::a

Example Tree

```
c
  a
  c
  b
  a

b a
  c
  b
  a
  c
  a
  c
  c
```
### XPath Semantics

<table>
<thead>
<tr>
<th>XPath Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Result of an expression is a node set or a single value (Boolean, number or string)</td>
</tr>
<tr>
<td>● Expressions are evaluated relative to a <strong>context</strong>, in particular relative to a <strong>context node</strong></td>
</tr>
<tr>
<td>● Location step: ( p = (a :: n \ q) ) relative to context node ( u ) yields the set (<a href="u">p</a>) of nodes ( v ) such that</td>
</tr>
<tr>
<td>– ((u, v)) are in a-relation</td>
</tr>
<tr>
<td>– ( v ) is labeled according to ( n ) (arbitrary, if ( n = * ))</td>
</tr>
<tr>
<td>– all predicates of ( q ) hold at ( v )</td>
</tr>
<tr>
<td>● Extended to sets ( S ) of nodes: (<a href="S">p</a> = \bigcup_{u \in S} <a href="u">p</a>)</td>
</tr>
<tr>
<td>● Location path: (<a href="S">p/\pi</a> = <a href="%5Bp%5D(S)">\pi</a>)</td>
</tr>
</tbody>
</table>
Example Revisited

Example XPath Expression

/desc::a/child::*[desc::c and not foll-sib::b]/desc::a

Example Tree
Example XPath Expression

/desc::a/child::*[desc::c and not foll-sib::b]/desc::a

Example Query Tree

```
root
  |       
  desc::a
  |       
child::*
     \   
  desc::a  and
         \   
  desc::c  not
            \ 
              foll-sib::b
```
A simplified Notation [Benedikt, Fan, Kuper 03]

**Notation**

- \( \downarrow, \uparrow, \rightarrow, \leftarrow, \circ : \)
  - child, parent, next-sibling, previous-sibling, self

- \( \downarrow^+, \uparrow^+, \rightarrow^+, \leftarrow^+ : \)
  - descendant, ancestor, following-sibling, preceding-sibling

- \( \downarrow^*, \uparrow^*, \rightarrow^*, \leftarrow^* : \)
  - descendant-or-self, ancestor-or-self, following-sibling-or-self, preceding-sibling-or-self

**Example**

- child::a/descendant::c/following-sibling::*//parent::b can be expressed as \( \downarrow/a/\downarrow^+/c/\rightarrow/\uparrow/b \)

- The following-axis can be expressed via \( \uparrow^*/\rightarrow^+/\downarrow^* \)
Contents

Introduction

Background on Tree Automata and Logic

Schema Languages

**XPath and Node-selecting Queries**

  - Node-selecting Queries
  - XPath: Semantics and Fragments
    - **XPath: Expressive Power**
  - XPath: Evaluation
  - XPath: Satisfiability
  - XPath: Containment

XSLT

XQuery

Conclusion
**Xpath and Existential First-order Logic**

**Characterizations of XPath [Benedikt, Fan, Kuper 03]**
- Navigational XPath (without not and and) corresponds to positive existential first-order logic.
- Different XPath axes correspond to different signatures.

**Proof idea**
- Basic idea:
  For each node $v$ of the query tree: guess a node $h(u)$ in the document tree and check that $h$ is a “homomorphism”.
- Main difficulty in proof:
  Deal with conjunctions of conditions.

**Further Results on**
- closure properties
- axiomatizations of equivalence
Elimination of Backward Axes [Olteanu et al. 02]

- In absolute XPath expressions, all backward axes can be eliminated
- Two sets of rewrite rules:
  - with intersection, linear time (and size)
  - without intersection, possibly exponential size
Xpath and First-Order Logic

Reminder
Navigational XPath without negation corresponds to positive existential first-order logic

Question: What is needed to capture full first-order logic?

Conditional axes

Conditional axes:

Expressions of the kind $P^+$, where $P$ is an expression

Example

$\text{(child :: a[desc :: b or child :: c])}^+$

holds between $u$ and $v$ if

- $v$ is a descendant of $u$ and
- all intermediate nodes
  - are labelled with $a$ and
  - have a $c$-child or a $b$-descendant
Xpath and First-Order Logic (cont.)

<table>
<thead>
<tr>
<th>Theorem [Marx 04]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigational XPath with conditional axes corresponds exactly to first-order logic (wrt node-selecting queries)</td>
</tr>
</tbody>
</table>

**Proof idea**

The proof uses a decomposition technique similar to the proof that LTL corresponds to first-order logic over linear structures [Gabbay et al. 80]
**Pebble Automata and XPath**

**Definition: Pebble Automata**
- Extension of tree-walk automata by fixed number of pebbles
- Only pebble with highest number (current pebble) can move, depending on state, number of pebble symbols under pebbles and incidence of pebbles
- Possible pebble movements:
  - stay, go to left sibling, go to right sibling, go to parent
  - lift current pebble or place new pebble at current position
- Nondeterminism possible

**Fact**
Deterministic pebble automata capture navigational **XPath** queries

**Proof idea**
For each node of the query tree:
- cycle through all possible nodes of the document tree
Automata and Logic

Some observations

- On strings, MSO logic and (unary) transitive closure logic (TC-logic) coincide
- On trees
  - MSO $\equiv$ parallel automata
  - TC-logic $\equiv$ pebble automata (i.e., strongest sequential automata)
- Whether on trees MSO $\equiv$ TC-logic is open
- The relationship between logics and automata models between FO and TC-logic is largely unexplored:
  - Tree-walk automata,
  - FO-logic + regular expressions
  - Conditional XPath + arbitrary star operator
  - ...
Contents

Introduction

Background on Tree Automata and Logic

Schema Languages

**XPath and Node-selecting Queries**

- Node-selecting Queries
- XPath: Semantics and Fragments
- XPath: Expressive Power
  - **XPath: Evaluation**
- XPath: Satisfiability
- XPath: Containment

XSLT

XQuery

Conclusion
XPath Query Evaluation

Naive Evaluation

Procedure Eval($p_1/\cdots/p_n,v$)

\[ S := [p_1]v \]

IF $n = 1$ RETURN $S$ ELSE $S' := \emptyset$

FOR $u \in S$ DO $S' := S' \cup$ Eval($p_2/\cdots/p_n,u$)

RETURN $S'$

Complexity

- $T(p_1/\cdots/p_n,t) = O(\text{size of } t) \times T(p_2/\cdots/p_n,t)$

- Could be exponential

- Experiments (reported in [Gottlob, Koch, Pichler 02]) show that available XPath processors had exponential complexity

Example

$/\text{descendant::a}(/\text{child::b}/\text{parent::a})^n$ on document

```
  a
 /\  \\
 b   b
```
Evaluation of Navigational XPath

Basic Idea
Combine top-down evaluation of the “main path” with bottom-up evaluation of predicates [Gottlob, Koch, Pichler 02]

Example Query Tree

```
root
  | desc::a
  |  |
  | child::*
  |    desc::a
  |     and
  |      desc::c
  |      not
  |      foll-sib::b
```

Example Document

```
c
  a b a
  c b a
  b a c b c a c
  c a b
```
Evaluation of Navigational XPath (cont.)

Evaluation Algorithm for Navigational XPath

Procedure NEval\((p_1/\cdots/p_n,v)\)

\(S' \colon= \{v\}\)

FOR \(i \colon= 1 \) TO \(n\)

\((\ast \ p_i = a_i::n_i \ q_i \ast)\)

\(S' \colon= \{u \mid v \in S', (v,u) \text{ in } a_i\text{-relation, } u \text{ matches } n_i\}\)

Compute \(S'' \colon= \{u \mid \llbracket q_i \rrbracket(u) \neq \emptyset\}\) bottom-up

\(S' \colon= S' \cap S''\)

RETURN \(S'\)

Complexity

- For each node of the query tree: \(O(|t|)\) steps
- Overall: \(O(\text{query size } \times |t|)\)
Beyond Navigational XPath

Example expression
/desc::a/child::*[desc::c[position() > 1]]/desc::a

Observations

- In general, a subexpression does not only depend on a context node but also on
  - context position (position())
  - context size (last())

→ predicates can no longer be evaluated in a bottom-up fashion

- Basic idea of [Gottlob, Koch, Pichler 02]: Compute the value of each subexpression for each triple $(v, i, l)$ of
  - a node $v$
  - a position $i$
  - a size $l$
Two Algorithms for XPath Evaluation

Results from [Gottlob, Koch, Pichler 02/03]

- The basic idea can be turned into different algorithms:
  - a bottom-up algorithm:
    * Computing the value for each \( e, (v, i, l) \) in a dynamic programming fashion
    * Time bound: \( O((\text{tree size})^5 \times (\text{query size})^2) \)
  - a (mixed) top-down algorithm:
    * Compute as much information as possible in top-down fashion to evaluate subexpressions only for relevant triples \( (v, i, l) \)
    * Time bound: \( O((\text{tree size})^4 \times (\text{query size})^2) \)
- Further time bound for the “extended Wadler fragment”:
  \( O((\text{tree size})^2 \times (\text{query size})^2) \)
Further Results

- In [Gottlob, Koch, Pichler 03] the complexity of XPath evaluation is considered.

- Data Complexity:
  - Navigational XPath: LOGSPACE-complete (e.g., via pebble automata)
  - Full XPath: also LOGSPACE (?)

- Combined Complexity:
  - Navigational XPath: PTIME-complete
  - Positive Navigational XPath: LOGCFL-complete
  - An even much larger fragment (pXPath) is in LOGCFL
## Contents

Introduction  
Background on Tree Automata and Logic  
Schema Languages

**XPath and Node-selecting Queries**  
  Node-selecting Queries  
  XPath: Semantics and Fragments  
  XPath: Expressive Power  
  XPath: Evaluation  
  **XPath: Satisfiability**  
  XPath: Containment

XSLT  
XQuery  
Conclusion
## XPath satisfiability

### Observation
Not all XPath expressions are satisfiable, e.g.:
\[
\text{child::a/child::b/following-sibling::c/parent::d}
\]

### Question
What is the complexity of checking satisfiability of an XPath expression for different fragments?

### Theorem [Hidders 03]
- Satisfiability for positive navigational XPath expressions is in \( \text{NP} \)
- Even for expressions without Boolean operators it is \( \text{NP-hard} \)
- For relative expressions without Boolean operators it is in \( \text{P} \)

### Remark
As navigational XPath can express star-free regular expressions along a path: Satisfiability of navigational XPath is non-elementary
(Note: this depends on the exact notion of Navigational XPath)
**Theorem [Hidders 03]**

Satisfiability for positive navigational XPath expressions is in **NP**

**Proof idea**

- If an expression $e$ without $\cup$ is satisfiable it has a model of size $\leq |e|$
- For an arbitrary (negation-free) expression guess a disjunct of the disjunctive normal form
**Theorem [Hidders 03]**
Satisfiability for positive navigational XPath expressions without Boolean operators is NP-hard

**Proof idea**
- Reduction from *Bounded Multiple String Matching (BMS)*:
  - Given: Pattern strings $p_1, \ldots, p_n$ over $\{0, 1, *\}$
  - Question: Is there a string over $\{0, 1\}$ of length $|p_1|$ which matches all patterns?
- Example: $*0**1, 00*1, *111$ has solution 00111
- As XPath expression:
  $$/\downarrow/\downarrow/0/\downarrow/\downarrow/\downarrow/1 \uparrow*/1/\uparrow/\uparrow/0/\uparrow/0 \uparrow*/1/\uparrow/1/\uparrow/1$$
Example document

```xml
<Composer>
  <Name>Claude Debussy</Name>
  <Vita>
    <Born><When>August 22, 1862</When><Where>Paris</Where></Born>
    <Married><When>October 1899</When><Whom>Rosalie</Whom></Married>
    <Married><When>January 1908</When><Whom>Emma</Whom></Married>
    <Died><When>March 25, 1918</When><Where>Paris</Where></Died>
  </Vita>
  <Piece>
    <PTitle>La Mer</PTitle>
    <PYear>1905</PYear>
    <Instruments>Large orchestra</Instruments>
    < Movements>3</Movements>
  ...</Piece>
  ...
</Composer>
```

Example query

```
//Vita/Died/*
```
Abbreviated Syntax for Forward XPath

Example document

```
<Composer>
  <Name> Claude Debussy </Name>
  <Vita>
    <Born> August 22, 1862 </Born>
    <Where> Paris </Where>
    <Married> October 1899 <Whom> Rosalie </Whom> </Married>
    <Married> January 1908 <Whom> Emma </Whom> </Married>
    <Died> March 25, 1918 </Died>
  </Vita>
  <Piece>
    <PTitle> La Mer </PTitle>
    <PYear> 1905 </PYear>
    <Instruments> Large orchestra </Instruments>
    <Movements> 3 </Movements>
    ...
  </Piece>
  ...
</Composer>
```

Another example query

```
(//*[Name]/*When) | (//Where)
```

More XPath operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>p/q</td>
<td>child</td>
</tr>
<tr>
<td>p//q</td>
<td>descendant</td>
</tr>
<tr>
<td>p[q]</td>
<td>filter</td>
</tr>
<tr>
<td>*</td>
<td>wildcard</td>
</tr>
<tr>
<td>p</td>
<td>q</td>
</tr>
</tbody>
</table>
XPath containment

Question
Does //Vita/Died/* always select a subset of positions of (*[Name]//When) | (//Where)?

Answer
No!

Counter-example
<Vita>
  <Died>
    <How>Heart disease</How>
  </Died>
</Vita>

Further question
But what if the type of documents is constrained?
Fact

For all XML documents of type

```xml
<!DOCTYPE Composers [  
  <!ELEMENT Composers (Composer*)>  
  <!ELEMENT Composer (Name, Vita, Piece*)>  
  <!ELEMENT Vita (Born, Married*, Died?)>  
  <!ELEMENT Born (When, Where)>  
  <!ELEMENT Married (When, Whom)>  
  <!ELEMENT Died (When, Where)>  
  <!ELEMENT Piece (PTitle, PYear,  
                   Instruments, Movements)>  
]>```

the pattern `//Vita/Died/*` always selects a subset of positions of

`(/*[Name]//When) | (//Where)`
**XPath Containment: Definition**

**Definition: Containment for XPath(S)**

Let $S$ be a set of XPath-operators. The containment problem for $\text{XPath}(S)$ is:

**Given:** $\text{XPath}(S)$-expression $p, q$

**Question:** Is $p(t) \subseteq q(t)$ for all documents $t$?

**Definition: Containment for XPath(S) with DTD**

Let $S$ be a set of XPath-operators. The containment problem for $\text{XPath}(S)$ in the presence of DTDs is:

**Given:** $\text{XPath}(S)$-expression $p, q$, DTD $d$

**Question:** Is $p(t) \subseteq q(t)$ for all documents $t$ satisfying $t \models d$?

**Observation**

These problems are crucial for static analysis and query optimization.

**Question**

For which fragments $S$ are these problems

- decidable?
- efficiently solvable?
Results

General remarks

- The **XPath** containment problem has been considered for various sets of operators
- Focus on Forward **XPath**
- Results vary from **PTIME** to “undecidable”
- Various methods have been used:
  - Canonical model technique
  - Homomorphism technique
  - Chase technique
- More about this in [Miklau, Suciu 2002; Deutsch, Tanen 2001; Sch. 2004]
- We will consider automata based techniques
The Automata Technique

Definition: (Relative Containment for XPath (S) wrt DTD)

Let $S$ be a set of XPath-operators. The containment problem for XPath($S$) relative to a DTD is:

**Given:** XPath($S$)-expression $p, q$, DTD $d$

**Question:** Is $p(D) \subseteq q(D)$ for all documents $D$ satisfying $D \models d$?

A vague plan

- Construct an automaton $A_p$ for $p$
- Construct an automaton $A_q$ for $q$
- Construct an automaton $A_d$ for $d$
- Combine these automata suitably to get an automaton which accepts all counter-example documents
A Simplification

Definition: (Boolean containment)

\[ p \subseteq_b q \quad \iff \quad \text{whenever } p \text{ selects some node in a tree } t \text{ then } q \text{ also selects some node in } t. \]

Useful observation [Miklau, Suciu 2002]

In the presence of [], Boolean containment has the same complexity as containment.

Crucial idea

If and only if

\[ x \quad \subseteq \quad x' \quad \text{if and only if} \]

\[ p_1 \quad p_2 \quad p_1' \quad p_2' \]

\[ p_1 \quad \# \quad p_2 \quad p_1' \quad \# \quad p_2' \]
**Result 1 [Neven, Sch. 2003]**
The Boolean containment problem for $\text{XPath}(//, //)$ in the presence of DTDs is in **PTIME**

**Result 2 [Neven, Sch. 2003]**
The Boolean containment problem for $\text{XPath}(//, //, [], *, |)$ in the presence of DTDs is in **EXPTIME**

**Note**
Both results are optimal wrt complexity: the problems are complete for these classes
Containment for $\text{XPath}(//, //)$ and DTDs

**Result 1 [Neven, Sch. 2003]**

The Boolean containment problem for $\text{XPath}(//, //)$ in the presence of DTDs is in $\text{PTIME}$

**Proof idea**

- $\text{XPath}(//, //)$-expressions can only describe vertical paths in a tree
- Each expression is basically of the form $p_1//p_2//\cdots//p_k$, where each $p_i$ is of the form $l_{i1}/\cdots/l_{im_i}$
- On strings this is a sequence of string matchings corresponding to a regular language $L$

$\Rightarrow$ Deterministic string automaton of linear size

- Recall: there is a deterministic top-down automaton which checks whether a $p$-path exists

$\Rightarrow$ Deterministic top-down automaton $\mathcal{A}_p$

$\Rightarrow$ Deterministic top-down automaton $\mathcal{A}_q$ checking that no $q$-path exists
**Result 1 [Neven, Sch. 2003]**

The containment problem for XPath(\(/, //\)) in the presence of DTDs is in \textbf{PTIME}

---

**Proof idea (cont.)**

- Deterministic top-down automaton $A_p$
- Deterministic top-down automaton $A_{\bar{q}}$ checking that no $q$-path exists
- There is a deterministic top-down automaton $A_d$ checking whether $t$ conforms to $d$
- $p \subseteq_b q$ in the presence of $d \iff L(A_p \times A_{\bar{q}} \times A_d) = \emptyset$
- The latter can be checked in polynomial time
Containment for $\text{XPath}(\text{/}, \text{//}, [], *, |)$ and DTDs

Result 2 [Neven, Sch. 2003]
The containment problem for $\text{XPath}(\text{/}, \text{//}, [], *, |)$ in the presence of DTDs is in $\text{EXPTIME}$

Proof idea

We again represent patterns like 

$\langle/*[\text{Name}]/\text{When}\rangle | (\text{//Where})$

as query trees:

![Example query tree]

Lemma

For each $\text{XPath}(\text{/}, \text{//}, [], *, |)$-expression $p$ there is a deterministic bottom-up automaton $\mathcal{A}_p$ of exponential size which checks whether in a tree $p$ holds
**Lemma**

For each XPath(//, //, [], *, |)-expression \( p \) there is a deterministic bottom-up automaton \( A_p \) of exponential size which checks whether in a tree \( p \) holds.

**Proof idea for Lemma**

- States of \( A_p \) are of the form \((S/, S//)\)
- Both \( S/ \) and \( S// \) are sets of positions of the query tree:
  - \( S/ \): positions matching \( v \)
  - \( S// \): positions matching some node in the subtree of \( v \)
**Containment for** XPath(/, //, [], *, |) **and DTDs**

**Result 2** [Neven, Sch. 2003]
The containment problem for XPath(/, //, [], *, |) in the presence of DTDs is in **EXPTIME**

**Proof idea (cont.)**
- Construct deterministic bottom-up automaton $A_p$ of exponential size
- Construct deterministic bottom-up automaton $A_q$ of exponential size
- Construct deterministic bottom-up automaton $A_d$ of exponential size
- $p \subseteq_b q$ in the presence of $d \iff L(A_p \times A_q \times A_d) = \emptyset$
  $\Rightarrow$ exponential time
Corresponding Lower Bound

Theorem

The containment problem for \( \text{XPath}(\//, //, [], *, |) \) in the presence of DTDs is \textbf{EXPTIME}-hard

Proof sketch

Proof by reduction from \textit{Two-player corridor tiling}

Example

Example:

Top row \( T = \begin{array}{llll}
    \text{c} & \text{a} & \text{a} & \text{c}
\end{array} \)

Bottom row \( B = \begin{array}{llll}
    \text{a} & \text{c} & \text{a} & \text{c}
\end{array} \)

Vertical and horizontal constraints:

\( V = \begin{array}{llll}
    \text{c} & \text{a} & \text{c}
\end{array}, \begin{array}{llll}
    \text{c} & \text{c} & \text{a}
\end{array} \)

\( H = \begin{array}{llll}
    \text{a} & \text{c} & \text{a} & \text{a}
\end{array}, \begin{array}{llll}
    \text{c} & \text{a} & \text{a}
\end{array} \)

Deciding whether player I has a winning strategy is \textbf{EXPTIME}-complete
Strategies as trees

Proof Sketch (cont.)

Tiles over \( \{a, b, c\} \)

This DTD describes all strategy trees:

\[
S \rightarrow (a, \text{I}) + (b, \text{I}) + (c, \text{I})
\]

\[
(\sigma, \text{I}) \rightarrow (a, \text{II})(b, \text{II})(c, \text{II}) + \# + $^{\text{II}}$
\]

\[
(\sigma, \text{II}) \rightarrow (a, \text{I}) + (b, \text{I}) + (c, \text{I}) + \# + $^{\text{I}+!}$
\]

\[
$^{\text{II}} \rightarrow (a, \text{II})(b, \text{II})(c, \text{II})$
\]

\[
$^{\text{I}} \rightarrow (a, \text{I}) + (b, \text{I}) + (c, \text{I})$
\]

\$^{\text{I}} = \text{line separator} \quad \# = \text{terminal symbol} \quad ! = \text{indicates misplaced tile}$

One path corresponds to one game
Winning strategies and paths

Proof Sketch (cont.)

There are various kinds of paths in a game tree:

(a) Legal tilings $\implies$ Player I wins

(b) Syntactically wrong: some row of wrong length

(c) II places a wrong tile $\implies$ Player I wins

(d) I places a wrong tile $\implies$ Player II wins

Player I has a winning strategy $\iff$

there is a tree in which all paths are of the form (a) or (c)

We want to construct $q$ such that all paths of the form (b) or (d) are selected

Then: Player I wins iff $/S \nsubseteq q$ wrt DTD

Problem: if II places a wrong tile, I might be forced to place a wrong tile, too

$\implies$ We let player I mark wrong tiles of II by $\text{!}$

$\implies$ We have to check that I does this correctly
Path conditions

Proof Sketch (cont.)

Player I has winning strategy \( \iff \ /S \not\subseteq q \)

\( q \) expresses that one of the following holds

- Player I violates a horizontal constraint:
  \( \text{For each } (x, y) \not\in H: \ (/(x, II)/(y, I)) \)

- Player I violates a vertical constraint:
  \( \text{For each } (x, y) \not\in V: \ (/(x, I)/*/^{n+1} /(y, I)) \)

- Some row does not contain exactly \( n \) tiles
  \[
  D^{n+1} \upharpoonright \bigcup_{i=0}^{n-1} (S^i|S)\ D^i/(S^i|\#) \]

- Player I wrongly claims a mistake of II:
  \( \text{For each } (x, y) \in V, (x', y) \in H: \ (/(x, II)*^n / (x', I)/(y, II)/[] \)

- Some more conditions on \( B \) and \( T \)
  \( \ast = \text{OR of all symbols, } \sigma^i = \sigma / \cdots / \sigma \ (i \text{ times}) \)
  \( D = (d_1, I)|\cdots|(d_m, I)|(d_1, II)|\cdots|(d_m, II) \)
Related work on XPath containment

**More Results**

- Containment of XPath with `/` and a subset of `{///, [], *}` was studied in [Miklau and Suciu 2002]:
  - Containment of XPath(`///, [], *`) is coNP-complete even if the number of `*` or the number of `[]` is bounded
  - If the number of `///` is bounded then it is in polynomial time
- XPath containment in the presence of DTDs and simple integrity constraints was investigated in [Deutsch and Tanen 2001]:
  - In general (`unbounded` constraints): undecidable
- More complexity results between coNP and undecidable for other fragments and extensions in [Neven and S. 2003]

**Some Open Questions**

- What’s the exact borderline between fragments of XPath with decidable and undecidable containment problem?
- To what extent can the presented result be extended to other axes (siblings, backward)?
**Summary: XPath**

**Expressive Power**
Closely related to first-order logic

**Evaluation**
- In general: Polynomial time
- Large fragments in linear time
- Structural complexity between \text{LOGSPACE} and \text{PTIME}

**Satisfiability**
- Without negation: \text{PTIME} or \text{NP}
- With negation: non-elementary

**Containment**
- Varying from \text{PTIME} to undecidable
- Upper bound for positive navigational XPath?
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
</tr>
<tr>
<td>Background on Tree Automata and Logic</td>
</tr>
<tr>
<td>Schema Languages</td>
</tr>
<tr>
<td>XPath and Node-selecting Queries</td>
</tr>
<tr>
<td>XQuery</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
## XSLT Typechecking

**Definition: Transformation typechecking**

Given: DTDs $d_1$ and $d_2$ and a transformation $T$

Result: Is $T(t)$ valid wrt. $d_2$, for each document $t$ valid wrt. $d_1$?

**Question:** Is XSLT typechecking decidable?

**Question:** What is the complexity?

### Outline of the Following

- Provide an automata model for XSLT transformations
- Show that the behaviour of these automata can be captured by MSO logic
- Use manipulation of regular tree languages to solve type checking problem

→ This part is based on [Milo, Suciu, Vianu 01]
**XSLT in more detail**

**How XSLT Roughly Works**

**Templates:**

\[
\langle \text{xsl:template name=TName match=pattern mode=MName} \rangle
\]

**Template application:**

\[
\langle \text{xsl:apply-templates select=Expression mode=MName} \rangle
\]

**XSLT Processing** Whenever \text{xsl:apply-templates} is called at a node \( v \) the following happens:

- Compute set \( S(v) \) of nodes, reachable from \( v \) via \text{Expression} (if \text{select} is not present, \( S(v) = \) children of \( v \))
- For each \( w \in S(v) \) compute which templates that can be applied to \( w \):
  - \( w \) has to match pattern of a template
  - the mode of the template has to be the same as the mode of \text{xsl:apply-templates}
- If no template matches, take the default template
- For each \( w \in S(v) \) select the best template and apply it.

The process starts at the root of the tree.
**XSLT: Example**

Example Transformation

*Remove everything below a c. Translate a below b into d*

Example XSLT (Abbreviated)

```
<... match="a"> <a> <xsl:apply-templates> </a> </...>
<... match="a" mode="below"> <d> <xsl:apply-templates> </d> </...>
<... match="b"> <b> <xsl:apply-templates mode="below"> </b> </...>
<... match="b" mode="below"> <b> <xsl:apply-templates mode="below"> </b> </...>
<... match="c"> <c> </c> </...>
<... match="c" mode="below"> <c> </c> </...>
```

Example Trees

```
  a
 / \
 a   b
   / \  
  a   a
 /  \
 a   a
```

⇒

```
  a
 / \
 a   b
   / \  
  a   a
 /  \
  d   a
```
**Remark**

The previous example corresponds to top-down tree transducers.

### Example XSLT

```xml
<xsl:template match="/b">
  <b>
    <xsl:apply-templates select='child[1]' mode="acopy" />
  </b>
</xsl:template>

<xsl:template match="a" mode="acopy">
  <a>
    <xsl:apply-templates select='child[1]' mode="acopy" />
    <xsl:copy-of select='/child[last()]' />
  </a>
</xsl:template>
```

### Example Trees

Original tree:
```
  b
  |
  a
  |
  a
  |
  a
  |
  b
```

Transformed tree:
```
  a
  |
  c
  |
  d
  |
  e
```

⇒
```
  a
  |
  c
  |
  d
  |
  e
```

```xml
Schwentick
XML: Algorithms & Complexity
Introduction – XPath
155```
An automaton model for XSLT

**Definition: $k$-pebble Transducer**

- Work on binary tree encodings of unranked trees
- Up to $k$ pebbles can be placed on the tree
- Only pebble with highest number (*current pebble*) can move, depending on state, number of pebbles symbols under pebbles and incidence of pebbles
- Possible pebble movements:
  - stay
  - go to left child, right child or parent
  - lift current pebble
  - place new pebble on the root
- Nondeterminism allowed
- If current pebble stays it is possible to produce output:
  - a node with two (forthcoming) subtrees; in this case two independent subcomputations (*branches*) are started, which construct the left subtree and right subtree, respectively
  - a leaf; in this case the computation branch stops
Computing XSLT transformations by $k$-pebble transducers

**Fact**

$k$-pebble transducers can evaluate most XPath expressions (and produce as output an encoded version of the result list)
- even with other axes than the forward axis

**Proof idea**

- Whenever `xsl:apply-templates` is called at a node $v$ the following happens:
  - Cycle through the set $S(v)$ of nodes, reachable from $v$ via Expression (if `select` is not present, $S(v) =$ children of $v$)
  - For each $w \in S(v)$ check which templates can be applied to $w$:
    * $w$ has to match pattern of a template
    * the mode of `xsl:apply-templates` is stored in the state of the automaton
  - For each $w \in S(v)$ select the best template and branch into
    * a subcomputation which handles the next node in $S(v)$ (via the right child)
    * a subcomputation which applies the template to the current node

- The computation starts at the root of the tree
Question: Is XSLT typechecking decidable?

Proof idea

- How can we check that $T(t) \in L(d_2)$, for each $t \in L(d_1)$?

- Obvious approach:
  - Compute $T(L(d_1))$
  - Check that $T(L(d_1)) \subseteq L(d_2)$

- Problem: $T(L(d_1))$ does not need to be regular:

  Transform

  ![Tree transformation diagram]

- Alternative approach:
  - Compute $T^{-1}(L(d_2))$
  - Check $L(d_1) \cap T^{-1}(L(d_2)) = \emptyset$
### Pebble acceptors

#### Definition: $k$-pebble acceptors
- Basically the same as $k$-pebble transducers
- Instead of output producing steps:
  - `accept`
  - branch into two independent subcomputations
- A tree is accepted if all subcomputations accept

#### Main Steps of the Proof
1. $T^{-1}(L(d_2))$ is accepted by a $k$-pebble acceptor
2. $k$-pebble acceptors only accept regular tree languages
Step (i)

Lemma

$T^{-1} \left( L(d_2) \right)$ is accepted by a $k$-pebble acceptor

Proof

- Let $B$ be a nondeterministic top-down tree automaton which accepts $L(d_2)$
- Let $T$ be a $k$-pebble tree transducer
- We construct $k$-pebble acceptor $A$ for $T^{-1} \left( L(d_2) \right)$, i.e., an automaton which on input $t$ decides whether there is a tree in $T(t)$ which is accepted by $B$:
  - Simulate $T$ on $t$ and $B$
  - Simulate at the same time the behaviour of $B$ on the (virtual) output tree
    - this is possible as the output tree is produced top-down and can be instantly consumed by $B$
  - The simulation involves branching, whenever $T$ branches, and produces two new subtrees
Lemma

$k$-pebble acceptors only accept regular tree languages

Proof idea

Show that the language of a $k$-pebble acceptor can be expressed by an MSO-formula:

1. Reduce $k$-pebble automaton acceptance to AGAP (Alternating Graph Accessibility)
2. Show that AGAP can be expressed in MSO
3. Some adjustments necessary
Alternating Graph Accessibility

Definition: Accessible Nodes

Let $G = (V, E)$, $V = V_\wedge \cup V_\lor$. A node $w$ is accessible if

- $w \in V_\wedge$ and all successors of $w$ are accessible, or
- $w \in V_\lor$ and at least one successor of $w$ is accessible.

Example

Definition: Alternating Graph Accessibility Problem (AGAP)

Given: Graph $G = (V, E)$, $V = V_\wedge \cup V_\lor$, and $v \in V$

Question: Is $v$ accessible?
Alternating Graph Accessibility (cont.)

Construction of $G_{A,t}$ from Automaton $A$ and Tree $t$

- Nodes in $V_\lor$ are the configurations of $A$ on $t$: tuples $[i, q, \theta]$, where $\theta : \{1, \ldots, i\} \rightarrow t$
- Nodes in $V_\land$ are $\epsilon$ and pairs $(\gamma_1, \gamma_2)$ of configurations with ”the same $\theta$”
- Edges:
  - $(\gamma_1, \gamma_2) \rightarrow \gamma_1$, $(\gamma_1, \gamma_2) \rightarrow \gamma_2$
  - $\gamma \rightarrow \gamma'$, if this is a step of $A$
  - $\gamma \rightarrow \epsilon$, if $A$ can get into the accept state from $\gamma$
  - $\gamma \rightarrow (\gamma_1, \gamma_2)$ if this is a branching step of $A$

Fact

A $k$-pebble acceptor $A$ accepts a tree $t \iff \gamma$ is accessible in $(G_{A,t})$
AGAP is MSO-expressible

Definition: Reverse-closed Sets of Nodes

A set $S$ of nodes is **reverse-closed** if the following holds:

- if $v$ is in $V_\wedge$ and $w \in S$, for all nodes $w$ with $(v, w) \in E$, then $v \in S$.
- if $v$ is in $V_\lor$ and $w \in S$, for some node $w$ with $(v, w) \in E$, then $v \in S$.

**Example**

**Fact**

Node $v$ is accessible iff it is in every reverse-closed set of nodes.

**...as MSO-Formula**

$v$ accessible $\equiv \forall S \left( \text{reverse-closed}(S) \rightarrow S(v) \right)$, where

$$\text{reverse-closed}(S) \equiv \forall x \left( \left[ V_\wedge(x) \land \forall y \left( E(x, y) \rightarrow S(y) \right) \right] \rightarrow S(x) \right) \land \left( \left[ V_\lor(x) \land \exists y \left( E(x, y) \land S(y) \right) \right] \rightarrow S(x) \right)$$
Unfortunately, $G_{A,t}$ has too many nodes to use this directly:

- MSO can only quantify over sets of linear size in the given structure (i.e., $t$)

- $G_{A,t}$ has $\Omega(|t|^k)$ configurations

- But $G_{A,t}$ has a special structure:
  Nodes are only connected if their number of pebbles is the same $\pm 1$ and they agree in all but at most the last pebble
\( k \)-pebble acceptors and MSO (cont.)

Proof (cont.)

- Wlog assume that each state of \( A \) is only used for a fixed number of pebbles: 
  \( Q = Q_1 \cup \cdots \cup Q_k \), where the states in \( Q_i \) are only used, when \( i \) pebbles are present.

- Further assume that all sets \( Q_i \) are of equal size \( m \): 
  \( Q_i = \{ q_{i1}, \ldots, q_{im} \} \)

- \( k = 1 \):
  - Use one relation \( S^1_i \) for each state \( q_{1i} \)
  - Intended meaning of \( v \in S^1_i \):
    - there is an accepting subcomputation of \( A \) starting at \( v \) in state \( q_{1i} \)
  - \( \varphi = \forall S^1_1 \cdots \forall S^1_m \) (reverse-closed \( \rightarrow S^1_1(\text{root}) \))
  - reverse-closed is a conjunction of subformulas, induced by the transitions of \( A \), e.g.:
    * if \( (q_{1i}, a) \rightarrow \text{accept} \) then \( \forall x \ Q_a(x) \rightarrow S^1_i(x) \)
    * if \( (q_{1i}, a) \rightarrow (q_{1j}, \text{down-right}) \) then
      \[ \forall x \ \forall y (Q_a(x) \land E_r(x, y) \land S^1_j(y)) \rightarrow S^1_i(x) \]
$k$-pebble acceptors and MSO (cont.)

Proof (cont.)

$k = 2$:

- reverse-closed$^1$ and reverse-closed$^2$ describe reverse closure for configurations with one and two pebbles, respectively
- reverse-closed$^2$ expresses the same as reverse-closed before, but with the (immobile) pebble 1 represented by variable $x_1$
- reverse-closed$^1$ also refers to subcomputations with a second pebble
- Conjuncts corresponding to simple movements are essentially the same
- Conditions which check whether pebbles are at the same node have to be added
- The following conjuncts are added for pebble placement and lifting:
  - $(q_{2i}, a) \rightarrow (q_{1j}, \text{lift})$ adds $\forall x_2 (Q_a(x_2) \land S^1_j(x_1)) \rightarrow S^2_i(x_2)$ to reverse-closed$^2$
  - $(q_{1i}, a) \rightarrow (q_{2j}, \text{place})$ adds $\forall x_1 (Q_a(x_1) \land \varphi^2) \rightarrow S^1_i(x_1)$ to reverse-closed$^1$, where $\varphi^2$ is $\forall S^2_1 \cdots \forall S^2_m (\text{reverse-closed}^2 \rightarrow S^2_j(\text{root}))$
Proof (cont.)

- To solve the type checking problem, given $d_1$, $d_2$ and $T$, we can proceed as follows.
  1. Construct the $k$-pebble acceptor $A$ for $T^{-1}(\overline{L(d_2)})$
  2. Transform $A$ into an equivalent MSO formula $\Phi$
  3. $\Phi$ holds for all trees $t$ for which $T(t) \not\subseteq L(d_2)$
  4. Construct a nondeterministic bottom-up automaton $A'$ equivalent to $\neg \Phi$
  5. Check that $L(d_1) \subseteq L(A')$

- Hence, the type-checking problem is decidable

- Steps (1) and (4) can be done in poly-time

- Step (2) is exponential in $k$, FO-quantifier depth of $\Phi$ is $k$, MSO-quantifier depth of $\Phi$ is $|Q|$

- Step (3) is non-elementary (exponentiation tower of height $k$)

- Hence, the algorithm for the type-checking problem has a very bad complexity
Summary: Typechecking

Related Work

- If transformations are allowed to compare data values in the input document, type checking becomes undecidable very quickly, even for restricted types and transformations [Alon et al. 2001]

- Typechecking for deterministic top-down tree transducers is more tractable. Complexity depends on exact representation of DTDs and restrictions on the transducers: between **PTIME** and **EXPTIME** [Martens and Neven 2003]

- If **P ≠ NP** there is no elementary $f$, such that MSO-formulas can be evaluated in time $f(|\text{formula}|) \times p(|\text{tree}|)$ with polynomial $p$ [Frick and Grohe 2002]

Open

- Find (more) transformations with a tractable typechecking problem

- In particular, with data values
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
</tr>
<tr>
<td>Background on Tree Automata and Logic</td>
</tr>
<tr>
<td>Schema Languages</td>
</tr>
<tr>
<td>XPath and Node-selecting Queries</td>
</tr>
<tr>
<td>XSLT</td>
</tr>
<tr>
<td><strong>XQuery</strong></td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
In general, the theoretical foundations of XQuery have to be developed.

Clearly: XQuery is Turing-complete and therefore static analysis is generally impossible.

What about important fragments with better properties?

E.g., Tree pattern queries

Here, we concentrate on:
- Conjunctive queries for trees
- Some questions related to automata for XQuery
Contents

Introduction
Background on Tree Automata and Logic
Schema Languages
XPath and Node-selecting Queries
XSLT
**XQuery**
  Conjunctive Queries
  Automata and XQuery
Conclusion
Conjunctive Queries

Introduction

- Navigational XPath expressions (without or and not) can be written as **conjunctive queries**
- `/child::a/desc::*[child::c]/parent::*d` corresponds to

\[
Q(x) = \text{root}(x_1) \land \text{child}(x_1, x_2) \land L_a(x_2) \land \text{desc}(x_2, x_3) \land \\
\text{child}(x_3, x_4) \land L_c(x_4) \land \text{child}(x, x_3) \land L_d(x)
\]

- Conjunctive Queries can express queries of higher arity:

\[
Q(x, y) = \text{child}(x, x_1) \land \text{child}(x_1, y)
\]

- What is the complexity of evaluating conjunctive queries on trees?

- Data complexity is in **PTIME** (even in **LOGSPACE**):
  - Cycle through all valuations of the variables

- What about combined complexity?
**A Generic Algorithm**

**Definition**

- **Pre-valuation**: mapping from variables to non-empty sets of nodes
- For a conjunctive query $Q$ a pre-valuation $\theta$ is **consistent** if:
  - for each atom $L_\sigma(x)$: $v \in \theta(x) \Rightarrow L_\sigma(v)$
  - for each atom $R(x, y)$:
    - $v \in \theta(x) \Rightarrow \exists u \in \theta(y) R(v, u)$
    - $v \in \theta(y) \Rightarrow \exists u \in \theta(x) R(u, v)$

---

**Example Query**

$Q(x, y) = \text{child}(x, y) \land L_\alpha(x)$

**Example Pre-Valuation**

$\theta(x) = \ldots$

$\theta(y) = \ldots$
Fact
A maximal consistent pre-valuation \( \theta \) can be computed in time
\[ O(\text{query size} \times \text{tree size}) \]

Algorithmic Idea
- Let \( < \) be a total order on the nodes
- For query \( Q \) and tree \( t \):
  - Compute maximal consistent pre-valuation \( \theta \)
  - Define \( < \)-minimal valuation \( h \) via:
  - For each variable \( x \):
    \( h(x) := \text{minimal node in } \theta(x) \text{ wrt } < \)

Example Document

Example Document

Question: Is \( h \) always a solution?
Example Query

\[ Q(x, y) = \text{child}(x, y) \land L_a(x) \]

Example Pre-Valuation

\[ \theta(x) = \ldots \]
\[ \theta(y) = \ldots \]

Question: Is \( h \) always a solution?

Observations

- Let \( u = h(x), v = h(y) \)
- As \( u \in \theta(x) \) there is \( v' \in \theta(y) \) such that \( \text{child}(u, v') \)
- As \( v \in \theta(y) \) there is \( u' \in \theta(x) \) such that \( \text{child}(u', v) \)
- As \( u \leq u' \) and \( v \leq v' \) we get \( \text{child}(u, v) \)
### Definition

A binary relation $R$ is **$<$-hemichordal** if for all $u, u', v, v'$ with $u < u'$ and $u \leq v \leq v'$

- $R(u, v') \land R(u', v) \rightarrow R(u, v)$ and
- $R(v', u) \land R(v, u') \rightarrow R(v, u)$

### Theorem [Gottlob, Koch, Schulz 04]

If the relations of a query $Q$ are $<$-hemichordal and $\theta$ is a consistent pre-valuation for $Q$

then the $<$-minimal valuation for $\theta$ is a solution for $Q$

### Corollary

If the axes used in a conjunctive query $Q$ are $<$-hemichordal then $Q$ can be evaluated in time $O(\text{query size} \times \text{tree size})$
Combined Complexity of Conjunctive Queries

Observation
It is sufficient to consider the axes $\text{child}$, $\text{child}^+$, $\text{child}^*$, $\text{NextSibling}$, $\text{NextSibling}^+$, $\text{NextSibling}^*$, $\text{Following}$.

Theorem [Gottlob, Koch, Schulz 04]
- $\text{child}^+$ and $\text{child}^*$ are preorder-hemichordal
- $\text{Following}$ is postorder-hemichordal
- $\text{child}$, $\text{NextSibling}$, $\text{NextSibling}^+$, $\text{NextSibling}^*$ are breadth-first-left-to-right-hemichordal

Corollary
For each of these sets of axes conjunctive queries can be evaluated in time $O(\text{query size} \times \text{tree size})$.

Amazing Result
For sets of axes not contained in those, the combined complexity of conjunctive query evaluation is $\text{NP}$-complete.
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
</tr>
<tr>
<td>Background on Tree Automata and Logic</td>
</tr>
<tr>
<td>Schema Languages</td>
</tr>
<tr>
<td>XPath and Node-selecting Queries</td>
</tr>
<tr>
<td>XSLT</td>
</tr>
<tr>
<td><strong>XQuery</strong></td>
</tr>
<tr>
<td>Conjunctive Queries</td>
</tr>
<tr>
<td>Automata and XQuery</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
Automata and XQuery

So far...

- We have seen that automata are useful for
  - Validation, Typing
  - Navigation
  - Transformation

- What about more general queries?
  - results of higher arity?
  - joins, i.e., comparisons of data values
  - counting

- Are automata useful for XQuery?
- ... for tree pattern queries?
Higher arity

- Nonemptiness and containment questions can be handled by automata: tuples can be encoded by additional labels
- What about query evaluation for higher arity?

Data values

- When data values in XML documents are taken into account, things become more complicated, e.g.:
  - Even First-order logic becomes undecidable
  - Pebble automata become undecidable
  - Automata with data registers become undecidable when they are allowed to move up and down
- What is the right notion for regular (string) languages over infinite alphabets?
- What are sensible decidable restrictions of logics and automata in the context of data values?
Counting

- Automata can be equipped with counting facilities, e.g.:
  - Presburger tree automata: \( \delta(\sigma, q) \) is Boolean combination of
    - regular expressions and
    - quantifier-free Presburger formulas like
      “number of children in state \( q_1 \) = number of children in state \( q_2 \)”

- Nondet. Presburger automata:
  - \( \equiv \) MSO logic
  - Whether automaton accepts all trees is undecidable

- Det. Presburger automata:
  - \( \equiv \) Presburger \( \mu \)-formulas
  - Membership test: \( O(|A||t|) \)
  - Non-emptiness: \textbf{PSPACE}
  - Containment: \textbf{PSPACE}

[Seidl, Sch., Muscholl, Habermehl 2004]
## Contents

- Introduction
- Background on Tree Automata and Logic
- Schema Languages
- XPath and Node-selecting Queries
- XSLT
- XQuery

**Conclusion**
Conclusion

Summary
- Schema languages and XPath are well understood
- There are some nice results on transformations
- Theory for XQuery still has to be developed

Finally...
Thanks for your patience