
S. Margherita di Pula
August 2004

Thomas Schpentick
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>
```
XML Processing

Four important kinds of XML processing

**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
## XML Processing

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation</td>
<td>DTD, XML Schema</td>
</tr>
<tr>
<td></td>
<td>Check whether an XML document is of a given type</td>
</tr>
<tr>
<td>Navigation</td>
<td>XPath</td>
</tr>
<tr>
<td></td>
<td>Select a set of positions in an XML document</td>
</tr>
<tr>
<td>Querying</td>
<td>XQuery</td>
</tr>
<tr>
<td></td>
<td>Extract information from an XML document</td>
</tr>
<tr>
<td>Transformation</td>
<td>XSLT</td>
</tr>
<tr>
<td></td>
<td>Construct a new XML document from a given one</td>
</tr>
</tbody>
</table>
Example document

```xml
<Composer>
  <Name>Claude Debussy</Name>
  <Vita>
    <Born>August 22, 1862</Born>
    <Where>Paris</Where>
    <When>Born</When>
    <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>
```
**Validation: DTD**

**Example document**

```xml
<Composer>
  <Name>Claude Debussy</Name>
  <Vita>
    <Born>August 22, 1862</Born> <Where>Paris</Where> <Born/>
    <Married>October 1899</When> <Whom>Rosalie</Whom> <Married/>
    <Married>January 1908</When> <Whom>Emma</Whom> <Married/>
    <Died>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>
```

**DTD**

DTDs describe types of XML documents.
Validation: DTD

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)>
]>`
Example document

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

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

Example document:

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

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

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

Example query:
```
//Vita/Died/*
```
Navigation: XPath

XPath

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

Example document:

```xml
<Data>
  <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>
</Data>
```

Example query:

```
//Vita/Died/*
```
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 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>
```

XQuery is a full-fledged XML query language.
**Querying: XQuery**

**XQuery**

XQuery is a full-fledged XML query language.

**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**

```xml
for $x in doc('composers.xml')/Composer
where $x/Vita/Died/Where = 'Paris'
return $x/Name
```
XQuery is a full-fledged XML query language. Example query for \( \text{$x$} \) in \( \text{doc('composers.xml')/Composer} \) where \( \text{$x$/Vita/Died/Where} = \text{’Paris’} \) return \( \text{$x$/Name} \).
Transformation: XSLT

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>
```
Transformation: XSLT

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>
```

XSLT transforms documents by means of templates.
XSLT transforms documents by means of templates.

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
<xsl:template match="Composer[Vita//Where='Paris']">
  <ParisComposer>
    <xsl:copy-of select="Name"/>
    <xsl:copy-of select="Vita/Born"/>
  </ParisComposer>
</xsl:template>
```
### Transformation: XSLT

<table>
<thead>
<tr>
<th>Result</th>
</tr>
</thead>
</table>
| ⟨ParisComposer⟩  
  ⟨Name⟩ Claude Debussy ⟨/Name⟩  
  ⟨Born⟩  
    ⟨When⟩ August 22, 1862 ⟨/When⟩  
    ⟨Where⟩ Paris ⟨/Where⟩  
  ⟨/Born⟩  
⟨/ParisComposer⟩  
 ⟨ParisComposer⟩  
  ⟨Name⟩ Frédéric Chopin ⟨/Name⟩  
  ⟨Born⟩  
    ⟨When⟩ March 1, 1810 ⟨/When⟩  
    ⟨Where⟩ Żelazowa ⟨/Where⟩  
  ⟨/Born⟩  
⟨/ParisComposer⟩  
 ⟨ParisComposer⟩  
  ⟨Name⟩ Camille Saint-Saëns ⟨/Name⟩  
  ⟨Born⟩  
    ⟨When⟩ October 9, 1835 ⟨/When⟩  
    ⟨Where⟩ Paris ⟨/Where⟩  
  ⟨/Born⟩  
⟨/ParisComposer⟩ |

### XSLT

XSLT transforms documents by means of templates.

Example:

```xml
<xsl:template match="Composer[Vita//Where='Paris']">  
  ⟨ParisComposer⟩  
    ⟨Name⟩ Rosalie ⟨/Name⟩  
    ⟨/Name⟩  
    ⟨/ParisComposer⟩  
  ⟨ParisComposer⟩  
    ⟨Name⟩ Emma ⟨/Name⟩  
    ⟨/Name⟩  
    ⟨/ParisComposer⟩  
  ⟨ParisComposer⟩  
    ⟨Name⟩ Camille Saint-Saëns ⟨/Name⟩  
    ⟨/Name⟩  
    ⟨/ParisComposer⟩  
</xsl:template>
```
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
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 \( T \)
Q: Type of \( f \)
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
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
Background: Complexity Classes

Overview of Complexity Classes

Decidable

EXPSPACE  
...... Equivalence of reg. expressions with squaring

EXPTIME  
........................... 2-Player Corridor Tiling

PSPACE  
............................ Quantified Boolean Formulas

P  
............................... Satisfiability of prop. formulas

coNP

NP  

P

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?
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:
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
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
- XML & Database Theory
Question: Why trees?
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
## 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
**XML, Trees and Automata**

**Question:** Why trees?

<table>
<thead>
<tr>
<th>A Natural Answer</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Trees reflect the hierarchical structure of XML</td>
<td>● But trees cannot model all aspects of XML (e.g., IDREFs, data values)</td>
</tr>
<tr>
<td>● Underlying data model of XML is tree based</td>
<td>⇒ Sometimes extensions are needed</td>
</tr>
<tr>
<td></td>
<td>● E.g., directed graphs instead of trees</td>
</tr>
</tbody>
</table>

**Example**

![Tree Diagram](image)
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

Example
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 can not 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
XML, Trees and Automata

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
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
**XML, Trees and Automata**

**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
XML, Trees and Automata

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
## XML, Trees and Automata

**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
From Strings to Trees

A String

abcab

String as Tree

a
  b
c
  a
b
From Strings to Trees

A String
abcab

String as Tree
a
  b
  c
  a
  b

A Ranked Tree
a
  b
   c
   a
   b
   a
   b
   c

Schwentick XML: Algorithms & Complexity
Introduction - XML Processing
From Strings to Trees

A String

abcab

String as Tree

A Ranked Tree

An Unranked Tree

Schwentick XML: Algorithms & Complexity
Introduction - XML Processing
**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
Trees as Terms

Remark
Sometimes trees are viewed as terms

Example Tree as Term
\[ a^1(b^2(c^1(a^2(b, a)), b^2(a^2(b, c), c))) \]
From String Automata to Tree Automata

Question

How do automata generalize to trees?

Sequential

Parallel
How do automata generalize to trees?

Sequential

Parallel
Question

How do automata generalize to trees?

Sequential

Parallel
How do automata generalize to trees?

Sequential

Parallel
How do automata generalize to trees?

**Sequential**

- **a**
- **e**
- **c**
- **a**
- **e**
- **c**
- **e**
- **a**
- **e**
- **c**

**Parallel**

- **a**
- **e**
- **c**
- **a**
- **e**
- **c**
- **a**
- **e**
- **c**
Question
How do automata generalize to trees?

Sequential
Parallel
From String Automata to Tree Automata

Question

How do automata generalize to trees?

Sequential

Parallel
How do automata generalize to trees?

**Sequential**

```
  a
 / \
c e \\
/    |    \
|    |    |
|    |    |    
|    |    |    |
ea  a  a  c
```

**Parallel**

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

Schwentick
How do automata generalize to trees?

Sequential

Parallel
Question
How do automata generalize to trees?

Sequential

Parallel
From String Automata to Tree Automata

**Question**

How do automata generalize to trees?

**Sequential**

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

**Parallel**

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

Schwentick XML: Algorithms & Complexity

Introduction - XML Processing
From String Automata to Tree Automata

Question

How do automata generalize to trees?

Sequential

Parallel
How do automata generalize to trees?

### Sequential

1. a
2. e
3. c
4. e
5. a
6. a
7. c
8. e
9. a
10. e
11. c

### Parallel

1. a
2. e
3. c
4. e
5. a
6. e
7. c
8. e
9. a
10. c
How do automata generalize to trees?

Sequential

Parallel
Question

How do automata generalize to trees?

Sequential

Parallel
From String Automata to Tree Automata

Question

How do automata generalize to trees?

Sequential

Parallel
How do automata generalize to trees?

**Sequential**

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

**Parallel**

```
a
 e
 / \
 c  e
 |   |
a  a  c
  |  |
e  a  c
```
**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

$$
\delta(\land, q_1) = \{(q_1, q_1)\}
$$

$$
\delta(\land, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}
$$

$$
\delta(\lor, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}
$$

$$
\delta(\lor, q_0) = \{(q_0, q_0)\}
$$

$$
\delta(0, q_0) = \{\epsilon\}; \delta(0, q_1) = \emptyset
$$

$$
\delta(1, q_1) = \{\epsilon\}; \delta(1, q_0) = \emptyset
$$
Bottom-Up Automata

Example: Tree-structured Boolean Circuits

Transitions
\[
\begin{align*}
\delta(\wedge, q_1) &= \{(q_1, q_1)\} \\
\delta(\wedge, q_0) &= \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\} \\
\delta(\vee, q_1) &= \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\} \\
\delta(\vee, q_0) &= \{(q_0, q_0)\} \\
\delta(0, q_0) &= \{\epsilon\}; \delta(0, q_1) = \emptyset \\
\delta(1, q_1) &= \{\epsilon\}; \delta(1, q_0) = \emptyset
\end{align*}
\]

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

Tree-structured Boolean circuits

Two states: $q_0, q_1$

Accepting at the root: $q_1$

Transitions

$\delta(\land, q_1) = \{(q_1, q_1)\}$

$\delta(\land, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$

$\delta(\lor, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$

$\delta(\lor, q_0) = \{(q_0, q_0)\}$

$\delta(0, q_0) = \{\epsilon\}; \delta(0, q_1) = \emptyset$

$\delta(1, q_1) = \{\epsilon\}; \delta(1, q_0) = \emptyset$
**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**

\[
\begin{align*}
\delta(\land, q_1) &= \{(q_1, q_1)\} \\
\delta(\land, q_0) &= \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\} \\
\delta(\lor, q_1) &= \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\} \\
\delta(\lor, q_0) &= \{(q_0, q_0)\} \\
\delta(0, q_0) &= \{\varepsilon\}; \delta(0, q_1) = \emptyset \\
\delta(1, q_1) &= \{\varepsilon\}; \delta(1, q_0) = \emptyset
\end{align*}
\]
**Bottom-Up Automata**

Example: Tree-structured Boolean Circuits

![Diagram of tree-structured Boolean circuits]

**Idea**

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

**Transitions**

\[
\begin{align*}
\delta(\land, q_1) &= \{(q_1, q_1)\} \\
\delta(\land, q_0) &= \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\} \\
\delta(\lor, q_1) &= \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\} \\
\delta(\lor, q_0) &= \{(q_0, q_0)\} \\
\delta(0, q_0) &= \{\epsilon\}; \delta(0, q_1) = \emptyset \\
\delta(1, q_1) &= \{\epsilon\}; \delta(1, q_0) = \emptyset
\end{align*}
\]
Non-det. Top-Down Automata

Example

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
\[
\begin{align*}
\delta(\&, q_1) &= \{(q_1, q_1)\} \\
\delta(\&, q_0) &= \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\} \\
\delta(\|, q_1) &= \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\} \\
\delta(\|, q_0) &= \{(q_0, q_0)\} \\
\delta(0, q_0) &= \{acc\}; \delta(0, q_1) = \emptyset \\
\delta(1, q_1) &= \{acc\}; \delta(1, q_0) = \emptyset
\end{align*}
\]
Non-det. Top-Down Automata

Example

```
\[ q_1 \]
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Idea

Guess the correct values starting from the root
Check at the leaves
Three states: \( q_0, q_1, \text{acc} \)
Initial state \( q_1 \) at the root
Accepting if all leaves end in \( \text{acc} \)

Transitions

\[
\begin{align*}
\delta(\land, q_1) &= \{(q_1, q_1)\} \\
\delta(\land, q_0) &= \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\} \\
\delta(\lor, q_1) &= \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\} \\
\delta(\lor, q_0) &= \{(q_0, q_0)\} \\
\delta(0, q_0) &= \{\text{acc}\}; \delta(0, q_1) = \emptyset \\
\delta(1, q_1) &= \{\text{acc}\}; \delta(1, q_0) = \emptyset 
\end{align*}
\]
Non-det. Top-Down Automata

Example

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
$\delta(\land, q_1) = \{(q_1, q_1)\}$
$\delta(\land, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$
$\delta(\lor, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$
$\delta(\lor, q_0) = \{(q_0, q_0)\}$
$\delta(0, q_0) = \{\text{acc}\}; \delta(0, q_1) = \emptyset$
$\delta(1, q_1) = \{\text{acc}\}; \delta(1, q_0) = \emptyset$
Non-det. Top-Down Automata

Example

Idea
Guess the correct values starting from the root
Check at the leaves
Three states: $q_0, q_1, \text{acc}$
Initial state $q_1$ at the root
Accepting if all leaves end in acc

Transitions
\[
\begin{align*}
\delta(\land, q_1) &= \{(q_1, q_1)\} \\
\delta(\land, q_0) &= \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\} \\
\delta(\lor, q_1) &= \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\} \\
\delta(\lor, q_0) &= \{(q_0, q_0)\} \\
\delta(0, q_0) &= \{\text{acc}\}; \delta(0, q_1) = \emptyset \\
\delta(1, q_1) &= \{\text{acc}\}; \delta(1, q_0) = \emptyset
\end{align*}
\]
Non-det. Top-Down Automata

Example

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

$\delta(\land, q_1) = \{(q_1, q_1)\}$
$\delta(\land, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$
$\delta(\lor, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$
$\delta(\lor, q_0) = \{(q_0, q_0)\}$
$\delta(0, q_0) = \{\text{acc}\}; \delta(0, q_1) = \emptyset$
$\delta(1, q_1) = \{\text{acc}\}; \delta(1, q_0) = \emptyset$
Non-det. Top-Down Automata

Example

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

$\delta(\land, q_1) = \{(q_1, q_1)\}$
$\delta(\land, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$
$\delta(\lor, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$
$\delta(\lor, q_0) = \{(q_0, q_0)\}$
$\delta(0, q_0) = \{\text{acc}\}; \delta(0, q_1) = \emptyset$
$\delta(1, q_1) = \{\text{acc}\}; \delta(1, q_0) = \emptyset$
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
Observation

• \((q_0, q_1) \in \delta(\vee, q_1)\) 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 \(q_1\)
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
Observation

- \((q_0, q_1) \in \delta(\vee, q_1)\) 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]**

$\text{MSO} \equiv \text{Regular Tree Languages}$
Regular tree languages and logic (cont.)

**Theorem**

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

**Proof idea**

**Automata }\Rightarrow \text{ MSO}:**

Formula expresses that there exists a correct tiling.

**MSO }\Rightarrow \text{ Automata}:** more involved

Basic idea:

Automaton computes for each node \( v \) the set of formulas which hold in the subtree rooted at \( v \).
Formula $\Rightarrow$ automaton

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

Crucial fact
### 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

![Diagram](image)

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

![Diagram](image)

- state and symbol of parent and symbols at $v$ and its sibling

![Diagram](image)
### Question
What is a good acceptance mechanism for deterministic top-down automata?

<table>
<thead>
<tr>
<th>Several possibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) At all leaves states have to be accepting</td>
</tr>
<tr>
<td>(2) There is a leave with an accepting state</td>
</tr>
<tr>
<td>(2) is problematic for complement and intersection</td>
</tr>
<tr>
<td>(1) is problematic for complement and union</td>
</tr>
</tbody>
</table>
Definition: (Root-to-frontier automata with regular acceptance condition)

- Tree automata $A$ are equipped with an additional regular string language $L$ over $Q \times \Sigma$
- $A$ 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
### 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><strong>Schema Languages</strong></td>
</tr>
<tr>
<td>XPath and Node-selecting Queries</td>
</tr>
<tr>
<td>XSLT</td>
</tr>
<tr>
<td>XQuery</td>
</tr>
<tr>
<td><strong>Conclusion</strong></td>
</tr>
</tbody>
</table>
Definition: (Tree-walk automata)
Depending on
- current state
- symbol of current node
- position of current node with respect to 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

$q^0$

$q^1$
Tree-Walk Automata (cont.)

Fact

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

Example

Idea
Tree-Walk Automata (cont.)

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

**Example**

**Idea**

```
q^0
\(\land\)  \(\lor\)  \(\land\)  \(\lor\)  \(\land\)
\(0\)  \(1\)  \(1\)  \(0\)  \(0\)  \(1\)  \(1\)  \(1\)
```

```
q^1
\(\lor\)  \(\land\)  \(\lor\)  \(\land\)
\(1\)  \(0\)
```
Tree-Walk Automata (cont.)

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

Example

Idea
Tree-Walk Automata (cont.)

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

Example

Idea
Tree-Walk Automata (cont.)

Fact

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

Example

Idea

Schwentick XML: Algorithms & Complexity

Introduction - XML Processing 35
Tree-Walk Automata (cont.)

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

Example

Idea

\[
\begin{array}{c}
q^0 \\
\wedge \\
0 \\
\wedge \\
1 \\
\vee \\
0 \\
\wedge \\
1 \\
\vee \\
0 \\
\wedge \\
1 \\
\wedge \\
0 \\
\wedge \\
1 \\
\vee \\
1 \\
\wedge \\
1 \\
\vee \\
0 \\
\end{array}
\]
Tree-Walk Automata (cont.)

Fact

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

Example

Idea
Tree-Walk Automata (cont.)

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

Example

Idea
Tree-Walk Automata (cont.)

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

Example

Idea

q^0
\[ \land \]

q^1
\[ \lor \]

0

1

0

1

0

1

1

1

1

1

1

1

1
Tree-Walk Automata (cont.)

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

Example

Idea
Tree-Walk Automata (cont.)

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

Example

Idea

Schwentick XML: Algorithms & Complexity
Introduction - XML Processing 35
Tree-Walk Automata (cont.)

**Fact**

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

**Example**

```
\[ \text{Diagram of a tree-walk automaton with states and inputs} \]
```

**Idea**

```
\[ \text{Diagram showing the idea of the tree-walk automaton} \]
```
Tree-Walk Automata (cont.)

Fact

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

Example

Idea
Tree-Walk Automata (cont.)

Fact

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

Example

Idea

\[ q^0 \]
\[ q^1 \]
Tree-Walk Automata (cont.)

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

**Example**

**Idea**

```
\[ q^0 \]
\[ q^1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```

```
\[ \land \]
\[ \lor \]
\[ 0 \]
\[ 1 \]
```
Tree-Walk Automata (cont.)

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

Example

Idea

$q^0$

$q^1$
Fact

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

Example

Idea
Tree-Walk Automata (cont.)

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

Example

\[ \begin{array}{c}
0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
\end{array} \]

Idea

\[ \begin{array}{c}
q^0 \\
q^1 \\
\end{array} \]
Tree-Walk Automata (cont.)

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

Example

Idea
Tree-Walk Automata (cont.)

Fact

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

Example

Idea

$q^0$

$q^1$

0

1

0

1

0

1

1

1
Fact
- Tree-walk automata can evaluate Boolean circuit trees
- 5 states

Example

Idea
Tree-Walk Automata (cont.)

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

Example

Idea

Schwentick
**Tree-Walk Automata (cont.)**

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

**Example**

```
\( q^0 \)
```

```
\( q^1 \)
```

```
^\wedge^\wedge^\wedge^\wedge
\downarrow\downarrow\downarrow\downarrow
\wedge\wedge\wedge\wedge
\downarrow\downarrow\downarrow\downarrow
\vee\vee\vee\vee
\downarrow\downarrow\downarrow\downarrow
0110011111
```

**Idea**

```
^\wedge^\wedge^\wedge
\downarrow\downarrow\downarrow
\vee\vee\vee
\downarrow\downarrow\downarrow
1010
```

```
^\wedge^\wedge^\wedge
\downarrow\downarrow\downarrow
\vee\vee\vee
\downarrow\downarrow\downarrow
1100
```
Tree-Walk Automata (cont.)

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

Example

Idea

\( q^0 \)

\( q^1 \)
Tree-Walk Automata (cont.)

Fact

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

Example

Idea
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

Det. top-down tree automata

Non-det. tree walk automata

Det. tree walk automata
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
## Decision Problems

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

<table>
<thead>
<tr>
<th>Membership test for $\mathcal{A}$</th>
<th>Membership test (combined)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Given:</strong> Tree $t$</td>
<td><strong>Given:</strong> Automaton $\mathcal{A}$, tree $t$</td>
</tr>
<tr>
<td><strong>Question:</strong> Is $t \in L(\mathcal{A})$?</td>
<td><strong>Question:</strong> Is $t \in L(\mathcal{A})$?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-emptiness</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Given:</strong> Automaton $\mathcal{A}$</td>
<td><strong>Question:</strong> Is $L(\mathcal{A}) \neq \emptyset$?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Containment</th>
<th>Equivalence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Given:</strong> Automata $\mathcal{A}_1, \mathcal{A}_2$</td>
<td><strong>Given:</strong> Automata $\mathcal{A}_1, \mathcal{A}_2$</td>
</tr>
<tr>
<td><strong>Question:</strong> Is $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$?</td>
<td><strong>Question:</strong> Is $L(\mathcal{A}_1) = L(\mathcal{A}_2)$?</td>
</tr>
</tbody>
</table>
Time Bounds for the combined complexity of membership test for tree automata:

- Deterministic (parallel) tree automata: $O(|\mathcal{A}| |t|)$
- Nondeterministic (parallel) tree automata: $O(|\mathcal{A}|^2 |t|)$
  (Compute, for each node, the set of reachable states)
- Deterministic TWAs: $O(|\mathcal{A}|^2 |t|)$
  (Compute, for each node $v$, the aggregated behavior of $\mathcal{A}$ on its subtree: **Behavior function**)
- Nondeterministic TWAs: $O(|\mathcal{A}|^3 |t|)$
  (Compute, for each node $v$, the aggregated behavior of $\mathcal{A}$ on its subtree: **Behavior relation**)

---

### Membership Test

#### Facts

- **Deterministic (parallel) tree automata:** $O(|\mathcal{A}| |t|)$
- **Nondeterministic (parallel) tree automata:** $O(|\mathcal{A}|^2 |t|)$
  (Compute, for each node, the set of reachable states)
- **Deterministic TWAs:** $O(|\mathcal{A}|^2 |t|)$
  (Compute, for each node $v$, the aggregated behavior of $\mathcal{A}$ on its subtree: **Behavior function**)
- **Nondeterministic TWAs:** $O(|\mathcal{A}|^3 |t|)$
  (Compute, for each node $v$, the aggregated behavior of $\mathcal{A}$ on its subtree: **Behavior relation**)

---
Membership Test

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)
Membership Test

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

- Deterministic (parallel) tree automata:
  \[ O(|A|^2|t|) \]
  (Compute, for each node, the set of reachable states)

- Nondeterministic (parallel) tree automata:
  \[ O(|A|^3|t|) \]
  (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)
Membership Test

**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**)

Schwentick XML: Algorithms & Complexity

Introduction - XML Processing
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>$O(</td>
<td>\mathcal{A}</td>
</tr>
<tr>
<td>Det. bottom-up TA</td>
<td>$O(</td>
<td>\mathcal{A}</td>
</tr>
<tr>
<td>Nondet. bottom-up TA</td>
<td>$O(</td>
<td>\mathcal{A}</td>
</tr>
<tr>
<td>Nondet. top-down TA</td>
<td>$O(</td>
<td>\mathcal{A}</td>
</tr>
<tr>
<td>Det. TWA</td>
<td>$O(</td>
<td>\mathcal{A}</td>
</tr>
<tr>
<td>Nondet. TWA</td>
<td>$O(</td>
<td>\mathcal{A}</td>
</tr>
</tbody>
</table>

[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
Non-emptiness

Facts

- Non-emptiness for string automata corresponds to Graph Reachability (complete for NLOGSPACE)

- Non-emptiness for tree automata corresponds to Path Systems
Non-emptiness

Facts

- Non-emptiness for string automata corresponds to Graph Reachability (complete for \textbf{NLOGSPACE})

- Non-emptiness for tree automata corresponds to \textbf{Path Systems}
Non-emptiness

Facts

- Non-emptiness for string automata corresponds to Graph Reachability (complete for NLOGSPACE)

Non-emptiness for tree automata corresponds to Path Systems
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
Containment/Equivalence

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

- Deterministic automata: construct product automaton
Reminder: Product automaton

Product of 2 string automata
- "even number of zeros"
- "contains substring 00"

```
0,1
0,1
```

```
1 0 0,1
1 0 0,1
```

```
1 0 1 1 0,1
1 0 0,1
```

```
1
0 0
1
```

```
a b
1
```

```
0110100
```

Schwentick
XML: Algorithms & Complexity
Introduction - XML Processing
Reminder: Product automaton

Product of 2 string automata

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

|0110100|
Reminder: Product automaton

Product of 2 string automata

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

```
0,1
0
1
```

```
e
1
0
```

```
c
1
0
```

```
d
0
1
```

```
e
0
1
```

```
f
0,1
```

0|110100
Reminder: Product automaton

Product of 2 string automata

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

\[
\begin{align*}
&0 \quad 1 \\
&\downarrow \quad \downarrow \\
&a & b
\end{align*}
\]

\[
\begin{align*}
&0 \quad 1 \\
&\downarrow \quad \downarrow \\
&c & d & e & f
\end{align*}
\]

\[
\begin{align*}
&0,1 \\
&\downarrow \quad \downarrow \\
&0,1
\end{align*}
\]
Reminder: Product automaton

Product of 2 string automata

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

Graph:

- States: a, b, c, d, e, f
- Edges:
  - a to a: 0
  - b to b: 0
  - c to d: 0
  - d to e: 1
  - e to f: 0
- Accepting state: f

Strings:

- 011
- 0100
Reminder: Product automaton

Product of 2 string automata

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

\[ \text{Diagram of automata} \]

\[
\begin{array}{c}
\text{a} \quad 0110 \quad \text{b} \quad \text{1}
\end{array}
\]

\[
\begin{array}{c}
\text{c} \quad 0 \quad \text{d} \quad 1 \quad \text{e} \quad 0 \quad \text{f}
\end{array}
\]
Reminder: Product automaton

Product of 2 string automata

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

---

Schwentick

XML: Algorithms & Complexity

Introduction - XML Processing
Reminder: Product automaton

Product of 2 string automata

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

01101010
Reminder: Product automaton

Product of 2 string automata

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

\[a\quad b\quad c\quad d\quad e\quad f\]

State transitions:

- From state 'c' to 'd' on input 0
- From state 'c' on self-loop 1
- From state 'd' to 'e' on input 1
- From state 'e' to 'f' on input 0
- From state 'f' on self-loop 0,1

Input sequence: 0110100
Reminder: Product automaton

Product of 2 string automata

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

```
0110100
```
Reminder: Product automaton

Product of 2 string automata

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

|0110100|
Reminder: Product automaton

Product of 2 string automata

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

\[
\begin{align*}
\text{a} & \rightarrow \text{b} \\
\text{c} & \rightarrow \text{d} \rightarrow \text{e} \rightarrow \text{f} \\
\text{ac} & \rightarrow \text{ad} \rightarrow \text{ae} \rightarrow \text{af} \\
\text{bc} & \rightarrow \text{bd} \rightarrow \text{be} \rightarrow \text{bf} \\
\end{align*}
\]
Reminder: Product automaton

Product of 2 string automata

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

### Automaton Diagram

[Diagram of two automata connected to form a product automaton]

- States: a, b, c, d, e, f
- Edges: 0, 1
- Accepting states: f

### Transition Table

<table>
<thead>
<tr>
<th>Input</th>
<th>State</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
<td>a, b</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>a, b</td>
</tr>
<tr>
<td>0</td>
<td>b</td>
<td>b, c</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>b, c</td>
</tr>
<tr>
<td>0</td>
<td>c</td>
<td>c, d</td>
</tr>
<tr>
<td>1</td>
<td>c</td>
<td>c, d</td>
</tr>
<tr>
<td>0</td>
<td>d</td>
<td>d, e</td>
</tr>
<tr>
<td>1</td>
<td>d</td>
<td>d, e</td>
</tr>
<tr>
<td>0</td>
<td>e</td>
<td>e, f</td>
</tr>
<tr>
<td>1</td>
<td>e</td>
<td>e, f</td>
</tr>
<tr>
<td>0</td>
<td>f</td>
<td>f, e</td>
</tr>
<tr>
<td>1</td>
<td>f</td>
<td>f, e</td>
</tr>
</tbody>
</table>

### Example Input

01|10100
Reminder: Product automaton

Product of 2 string automata

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

Schwentick
XML: Algorithms & Complexity
Introduction - XML Processing
Reminder: Product automaton

Product of 2 string automata

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

Schwentick XML: Algorithms & Complexity
Introduction - XML Processing
Reminder: Product automaton

Product of 2 string automata

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

---

Schwentick XML: Algorithms & Complexity
Introduction - XML Processing

01101|00
Reminder: Product automaton

Product of 2 string automata

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

011010|0
Reminder: Product automaton

Product of 2 string automata
- "even number of zeros"
- "contains substring 00"

schwentickxml: Algorithms & Complexity Introduction - XML Processing
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 $L(A_1) \subseteq L(A_2)$:
  - Compute $B = A_1 \times A_2$
  - Accepting states: $F_1 \times (Q_2 - F_2)$
  - Check whether $L(B) = \emptyset$
  - If so, $L(A_1) \subseteq L(A_2)$ holds

Theorem

Complexity of Containment for deterministic bottom-up tree automata:

$$O(|A_1| \times |A_2|)$$
Non-deterministic automata

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

⇒ Exponential time
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
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 $\text{EXPTIME}$
Det. Top-Down Automata: Non-Emptiness

Theorem
Nonemptiness for deterministic top-down automata $\mathcal{A}$ can be checked in polynomial time

Proof idea
Check for each state $p$ of $\mathcal{A}$ and each pair $(q, q')$ of the leaves automaton $\mathcal{B}$:
Is there a tree $t$ such that $\mathcal{A}$ starts from state $p$ and obtains a leave string which brings $\mathcal{B}$ from $q$ to $q'$?
Det. Top-Down Automata: Containment

**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 $i = 1, i = 2$:
  - $T_i$ starts from state $p_i$ and obtains a leave string which brings $\mathcal{B}_i$ from $q_i$ to $q'_i$?
# Summary

## 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>LOGCFL</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

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
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
 /|
/|
/|
a c e c c e a
```

Encoding via ...

<table>
<thead>
<tr>
<th></th>
<th>first child</th>
<th>next sibling</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Encoding Unranked Trees as Binary Trees

Example: Unranked Tree

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

```
Encoding via ...

<table>
<thead>
<tr>
<th>first child</th>
<th>next sibling</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="first child" /></td>
<td><img src="image2" alt="next sibling" /></td>
</tr>
</tbody>
</table>

... as Binary Tree

```
  a
 /   \
|    |
|    |
|    |
|    |
  c   e
 /   / \
|   |   |
|   |   |
|   |   |
|   |   |
a   c
 /   / \
|   |   |
|   |   |
|   |   |
a   e
 /   / \
|   |   |
|   |   |
|   |   |
a   e
 /   / \
|   |   |
|   |   |
|   |   |
a   e
```
Encoding Unranked Trees as Binary Trees (cont.)

Example: Unranked Tree

\[ b \]
\[ a \quad a \quad a \quad n \quad c \]
\[ o \quad e \]

... if path expressions matter (Milo, Suciu, Vianu 00)
Remark

- 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 $N$ is a (finite) subset of $N^*$, such that if $v \cdot i \in V$, where $v \in N^*$ and $i \in N$,

- 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 $N$, 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:
$$\delta(\sigma, q) = \{(q_1, q_2), (q_3, q_4), \ldots\}$$
Transitions are described by finite sets:
\[ \delta(\sigma, q) = \{(q_1, q_2), (q_3, q_4), \ldots\} \]
From Ranked to Unranked Tree Automata

Ranked trees

Transitions are described by finite sets:
\( \delta(\sigma, q) = \{(q_1, q_2), (q_3, q_4), \ldots \} \)

Unranked trees

\( \in \delta(\sigma, q) ? \)
From Ranked to Unranked Tree Automata

Ranked trees

Transitions are described by finite sets:
\[ \delta(\sigma, q) = \{(q_1, q_2), (q_3, q_4), \ldots\} \]

Unranked trees

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

State at $v$ 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 left siblings
- State and symbol of parent and symbols at $v$ and its siblings
Checking Existence of Paths

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 $A$ be a deterministic string automaton
- $A' :=$ top-down automaton which takes at $v$ state of $A$ after reading $s(v)$
- $A'$ is deterministic
- There is a path from the root to a leaf $v$ with $s(v) \in L(A)$ iff $A'$ assumes at least one state from $F$ at a leave

Illustration

Streaming XML
Similar construction used for XPath evaluation on streams [Green et al. 2003]
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 labels a, v, and w]
Document Automata

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
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

Det. top-down tree automata

Non-det. tree walk automata

Det. tree walk automata

Finite document automata
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
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

$\rightarrow$ [Berstel, Boasson 00] provide characterizations

- Additional restriction: **one-unambiguous**
One-unambiguous Regular Expressions

**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
  \[ ux_i v \in L(r'), uy_j w \in L(r'), i \neq j \implies x \neq y \]

**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
  \( b_2b_2a_3c_4 \in L(r') \) and \( b_2b_2a_1a_3c_4 \in L(r') \)
- \((b^*a)^*c \) is one-unambiguous
**One-unambiguous Regular Expressions**

**Definition: One-unambiguous Regular Expression**

- Let $r$ be a regular expression
- $r \mapsto r'$: 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**

- $(a + b)^*ac + c \mapsto (a_1 + b_2)^*a_3c_4 + c_5$
- $babbac \in L(r)$ and $b_2a_1b_2b_2a_3c_4 \in L(r')$
- $(a + b)^*ac + c$ is not one-unambiguous because $b_2b_2\color{green}a_3c_4 \in L(r')$ and $b_2b_2\color{red}a_1a_3c_4 \in L(r')$
- $(b^*a)^*c$ is one-unambiguous
One-unambiguous Regular Expressions

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 $b_2a_1b_2a_3c_4 \in L(r')$
- $(a + b)^*ac + c$ is not one-unambiguous because $b_2b_2 a_3 c_4 \in L(r')$ and $b_2b_2 a_1 a_3 c_4 \in L(r')$
- $(b^*a)^*c$ 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 Algorithm

For each node:
Check that the children are ok wrt the parent’s rule
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)
- Further: DTDs are always satisfiable
Lemma [Martens, Neven, Sch. 04]

Containment of DTDs with regular expressions from $\mathcal{R}$ is in $\mathsf{C}$

$\iff$

Containment of regular expressions from $\mathcal{R}$ is in $\mathsf{C}$

Corollary

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

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.)

<table>
<thead>
<tr>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>What if the requirement of being one-unambiguous is dropped?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A classical result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Theorem [Stockmeyer, Meyer 71]</strong></td>
</tr>
<tr>
<td>Containment and Equivalence for regular expressions on strings are complete for <strong>PSPACE</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Corollary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Containment of DTDs (with unrestricted regular expressions) is <strong>PSPACE</strong>-complete</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theorem [Martens, Neven, Sch. 04]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Containment and Equivalence for regular expressions are</td>
</tr>
<tr>
<td>- <strong>coNP</strong>-complete for concatenations of $a, b, c$ and $a^<em>, b^</em>, c^*$</td>
</tr>
<tr>
<td>- <strong>coNP</strong>-complete for concatenations of $a, b, c$ and $a?, b?, c?$</td>
</tr>
<tr>
<td>- <strong>PSPACE</strong>-complete for concatenations of $a, b, c$ and $(a^* + b^* + \cdots + c^*)$</td>
</tr>
</tbody>
</table>
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**

- $\text{Dealer} \rightarrow \text{UsedCars} \quad \text{NewCars}$
  - $\mu(\text{Dealer}) = \text{Dealer}$
- $\text{UsedCars} \rightarrow \text{adUsed}^*$
  - $\mu(\text{UsedCars}) = \text{UsedCars}$
- $\text{NewCars} \rightarrow \text{adNew}^*$
  - $\mu(\text{NewCars}) = \text{NewCars}$
- $\text{adUsed} \rightarrow \text{model year}$
  - $\mu(\text{adUsed}) = \text{ad}$
- $\text{adNew} \rightarrow \text{model}$
  - $\mu(\text{adNew}) = \text{ad}$
### 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>Tag</th>
<th>μ(Tag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-AND</td>
<td>(1-OR</td>
</tr>
<tr>
<td>0-AND</td>
<td>(0-OR</td>
</tr>
<tr>
<td>1-OR</td>
<td>.* (1-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>
Specialized DTDs (cont.)

Observation

- A naive validation by exhaustively trying all possible functions \( \mu \) requires exponential time
- But help comes from automata...

- 
- 
-
Specialized DTDs (cont.)

<table>
<thead>
<tr>
<th>Observation</th>
</tr>
</thead>
<tbody>
<tr>
<td>• A naive validation by exhaustively trying all possible functions $\mu$ requires exponential time</td>
</tr>
<tr>
<td>• But help comes from automata...</td>
</tr>
<tr>
<td>• A tree conforms to a specialized DTD $(d, \mu)$ if there is a labeling of its nodes by types which is valid wrt. $d$</td>
</tr>
</tbody>
</table>
Specialized DTDs (cont.)

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...
**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**

**Definition: (Validation)**
Given: Specialized DTD $d$, tree $t$
Question: Is $t$ valid wrt $d$?

**Definition: (Typing)**
Given: Specialized DTD $d$, tree $t$
Output: Consistent type assignment for the nodes of $t$

**Facts**
- Specialized DTDs $\equiv$ regular tree languages
- Validation in linear time by deterministic push-down automata
- Typing in linear time (Bottom-up automaton) the document
- Satisfiability $\equiv$ Non-emptiness of tree automata: PTIME
(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
(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
Restrictions of Schemas

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

\[ \mu(b) = \mu(b') \]

- 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
(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
(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

## Results (partly from [Martens, Neven, Sch. 04])

<table>
<thead>
<tr>
<th>Schema type</th>
<th>unrestricted</th>
<th>deterministic expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTDs</td>
<td>PSPACE</td>
<td>PTIME</td>
</tr>
<tr>
<td>single-type SDTDs</td>
<td>PSPACE</td>
<td>PTIME</td>
</tr>
<tr>
<td>restrained-competition SDTDs</td>
<td>PSPACE</td>
<td>PTIME</td>
</tr>
<tr>
<td>unrestricted SDTDs</td>
<td>EXPTIME</td>
<td>EXPTIME</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.
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 \( \mathcal{C} \).
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
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 $v$
1-Pass Preorder Typing

Question

When would it be important to know the type of \( v \) before visiting the subtree of \( v \)?
1-Pass Preorder Typing

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
1-Pass Preorder Typing

**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
## 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?
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
A Very Robust Class

Further characterizations

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

Illustration
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: trivial
- SDTDs: **PTIME**

## Containment
- General SDTDs: **EXPTIME**
- Restrained competition SDTDs: **PTIME**
<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><strong>XPath and Node-selecting Queries</strong></td>
</tr>
<tr>
<td>Node-selecting Queries</td>
</tr>
<tr>
<td>XPath: Semantics and Fragments</td>
</tr>
<tr>
<td>XPath: Expressive Power</td>
</tr>
<tr>
<td>XPath: Evaluation</td>
</tr>
<tr>
<td>XPath: Satisfiability</td>
</tr>
<tr>
<td>XPath: Containment</td>
</tr>
<tr>
<td>XSLT</td>
</tr>
<tr>
<td>XQuery</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
Node-Selecting Queries

Example document

Example query
//Vita/Died/*

〈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〉
...
Node-Selecting Queries

Example document

Example query

//Vita/Died/*

Observation

XPath expressions define sets of nodes ➔ node-selecting queries
<table>
<thead>
<tr>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is there a class of node-selecting queries, as robust as the regular tree languages?</td>
</tr>
</tbody>
</table>
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
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
//*[a]//b

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

Example tree: Run 1

Query tree

\( \star \)

a
b
Example Automaton

Example query

//*[a]//b

Query tree

Example automaton

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

Example tree: Run 1
Example Automaton

Example query
\/*/ \texttt{a} /\texttt{/b}

Query tree

Example automaton

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

Example query

//*[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 1

<table>
<thead>
<tr>
<th>e</th>
<th>q_0</th>
<th>c</th>
<th>q_0</th>
<th>e</th>
<th>q_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>q_a</td>
<td>c</td>
<td>q_b</td>
<td>b</td>
<td>q_0</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>q_b</td>
<td>c</td>
<td>q_0</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b</td>
<td>q_0</td>
<td>c</td>
<td>q_0</td>
</tr>
</tbody>
</table>

Query tree

```text
query = \( \star \) and \( a \) and \( b \)
```
Example Automaton

Example query

\ /* [a] //b */

Query tree

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 1
Example Automaton

Example query

// *[a] // b

Query tree

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
Example Automaton

Example query
\/**/[a]/**/b

Query tree

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

Example tree: Run 2
Example Automaton

Example query

// *[a]//b

Query tree

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
Example Automaton

Example query
//*[a]//b

Query tree

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
Example Automaton

Example query
\/**/[a]/**/b

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

Query tree

Example tree: Run 2
Node-Selecting Automata

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
### Result

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

$\implies$ one unique accepting run

### Crucial fact

\begin{align*}
  e_1 & \equiv_k e_2 \\
  t_1 & \equiv_k t_2 \\
  e_1 & \equiv_k e_2 \\
  t_1 & \equiv_k t_2
\end{align*}
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]

→ 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
Satisfiability: Non-emptiness of node-selecting automata is \textbf{PTIME}-complete

Satisfiability of MSO-queries is non-elementary

Containment of node-selecting automata is \textbf{EXPTIME}-complete
Summary: Node-Selecting Queries

- 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
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: \[ \pi = p_1/\ldots/p_k \]

- **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  b  a
  |
 c  b  a  c  b  a
 |
b  a  c  b  c  a  c
 |
   c  a  b
```
XPath Example

Example XPath Expression

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

Example Tree
XPath Semantics

- Result of an expression is a node set or a single value (Boolean, number or string)
- Expressions are evaluated relative to a context, in particular relative to a context node
- Location step: $p = (a :: n \ q)$ relative to context node $u$ yields the set $\mathbb{[p]}(u)$ of nodes $v$ such that
  - $(u, v)$ are in a-relation
  - $v$ is labeled according to $n$ (arbitrary, if $n = *$)
  - all predicates of $q$ hold at $v$
- Extended to sets $S$ of nodes: $\mathbb{[p]}(S) = \bigcup_{u \in S} \mathbb{[p]}(u)$
- Location path: $\mathbb{[p/\pi]}(S) = \mathbb{[\pi]}(\mathbb{[p]}(S))$
Example Revisited

Example XPath Expression

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

Example Tree
Example Revisited

Example XPath Expression

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

Example Tree

```
  c
 /\  
 a  b  a
 / \ / \ / 
 c b a c b a
 / \ / \ / 
 a b c a a
 / \ / \ 
 b a c c
 / 
 c
```
Example Revisited

Example XPath Expression

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

Example Tree
Example XPath Expression

/\text{desc::a/child::*}[\text{desc::c and not foll-sib::b}]/\text{desc::a}

Example Tree
Example XPath Expression

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

Example Tree
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
```
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 \( \uparrow^*/\rightarrow^+/\downarrow^* \)
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduction</strong></td>
</tr>
<tr>
<td>Background on Tree Automata and Logic</td>
</tr>
<tr>
<td><strong>Schema Languages</strong></td>
</tr>
<tr>
<td><strong>XPath and Node-selecting Queries</strong></td>
</tr>
<tr>
<td>Node-selecting Queries</td>
</tr>
<tr>
<td>XPath: Semantics and Fragments</td>
</tr>
<tr>
<td><strong>XPath: Expressive Power</strong></td>
</tr>
<tr>
<td>XPath: Evaluation</td>
</tr>
<tr>
<td>XPath: Satisfiability</td>
</tr>
<tr>
<td>XPath: Containment</td>
</tr>
<tr>
<td><strong>XSLT</strong></td>
</tr>
<tr>
<td><strong>XQuery</strong></td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
**Xpath and Existential First-order Logic**

<table>
<thead>
<tr>
<th>Characterizations of XPath [Benedikt, Fan, Kuper 03]</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Navigational XPath (without not and and) corresponds to positive existential first-order logic</td>
</tr>
<tr>
<td>• Different XPath axes correspond to different signatures</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proof idea</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 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”</td>
</tr>
<tr>
<td>• Main difficulty in proof: Deal with conjunctions of conditions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Further Results on</th>
</tr>
</thead>
<tbody>
<tr>
<td>• closure properties</td>
</tr>
<tr>
<td>• axiomatizations of equivalence</td>
</tr>
</tbody>
</table>
Elimination of Backward Axes [Olteanu et al. 02]

- In **absolute** XPath expressions, all backward axes can be eliminated
- Two sets of rewrite rules:
  - introducing equality expressions, linear time (and size)
  - without equality expressions, possibly exponential size
## Reminder

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

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

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

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
Navigational XPath with conditional axes corresponds exactly to first-order logic (wrt node-selecting queries)

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]
**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 pebbles 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
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
  - ...

Schwentick XML: Algorithms & Complexity

Introduction - XPath
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 $S' := S' \cup \text{Eval}(p_2/ \cdots /p_n, u)$

RETURN $S'$
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 $S' := S' \cup \text{Eval}(p_2/\cdots/p_n,u)$

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

$/\text{descendant::a}(//\text{child::b/parent::a})^n$ 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
  c b c a c
  c a b
```
Basic Idea

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

Example Document

Schwentick XML: Algorithms & Complexity Introduction - XPath 119
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
              - c
                - b
                  - a
                    - c
                      - a
                        - b
                          - a
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

Example Document
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

Example Document

Schwentick XML: Algorithms & Complexity Introduction - XPath
Evaluation of Navigational XPath

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

Example Query Tree

Example Document
Evaluation of Navigational XPath

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

Example Document
Evaluation of Navigational XPath (cont.)

Evaluation Algorithm for Navigational XPath

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

$S' := \{v\}$

FOR $i := 1$ TO $n$

(* $p_i = a_i::n_i q_i *$)

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

Compute $S'' := \{u \mid [q_i](u) \neq \emptyset\}$ bottom-up

$S' := S' \cap S''$

RETURN $S'$
Evaluation Algorithm for Navigational XPath

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

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

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

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

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

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

\(S' := S' \cap S''\)

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: $\text{LOGSPACE}$-complete (e.g., via pebble automata)
  - Full XPath: also $\text{LOGSPACE}$ (?)

- Combined Complexity:
  - Navigational XPath: $\text{PTIME}$-complete
  - Positive Navigational XPath: $\text{LOGCFL}$-complete
  - An even much larger fragment (pXPath) is in $\text{LOGCFL}$
<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><strong>Schema Languages</strong></td>
</tr>
<tr>
<td><strong>XPath and Node-selecting Queries</strong></td>
</tr>
<tr>
<td>Node-selecting Queries</td>
</tr>
<tr>
<td>XPath: Semantics and Fragments</td>
</tr>
<tr>
<td>XPath: Expressive Power</td>
</tr>
<tr>
<td>XPath: Evaluation</td>
</tr>
<tr>
<td>XPath: Satisfiability</td>
</tr>
<tr>
<td>XPath: Containment</td>
</tr>
<tr>
<td><strong>XSLT</strong></td>
</tr>
<tr>
<td><strong>XQuery</strong></td>
</tr>
<tr>
<td><strong>Conclusion</strong></td>
</tr>
</tbody>
</table>
### XPath satisfiability

**Observation**

Not all *XPath* expressions are satisfiable, e.g.:

```
child::a/child::b/following-sibling::c/parent::d
```

**Question**

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

**Observation**

Not all *XPath* expressions are satisfiable, e.g.:

\[
\text{child}::a/\text{child}::b/\text{following-sibling}::c/\text{parent}::d
\]

**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 *NP*
- Even for expressions without Boolean operators it is *NP*-hard
- For relative expressions without Boolean operators it is in *P*
**Observation**

Not all XPath expressions are satisfiable, e.g.:

```
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 NP
- Even for expressions without Boolean operators it is NP-hard
- For relative expressions without Boolean operators it is in P

**Remark**

As navigational XPath can express star-free regular expressions along a path:

Satisfiability of navigational XPath is non-elementary
## 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 \(\textbf{NP}\)-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:
  \[
  /\downarrow/\downarrow/0/\downarrow/\downarrow/\downarrow/1\ \uparrow*/1/\uparrow/\uparrow/0/\uparrow/0\ [\uparrow*/1/\uparrow/1/\uparrow/1]\]
<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><strong>Schema Languages</strong></td>
</tr>
<tr>
<td><strong>XPath and Node-selecting Queries</strong></td>
</tr>
<tr>
<td>Node-selecting Queries</td>
</tr>
<tr>
<td>XPath: Semantics and Fragments</td>
</tr>
<tr>
<td>XPath: Expressive Power</td>
</tr>
<tr>
<td>XPath: Evaluation</td>
</tr>
<tr>
<td>XPath: Satisfiability</td>
</tr>
<tr>
<td>XPath: Containment</td>
</tr>
<tr>
<td><strong>XSLT</strong></td>
</tr>
<tr>
<td><strong>XQuery</strong></td>
</tr>
<tr>
<td><strong>Conclusion</strong></td>
</tr>
</tbody>
</table>
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>

Example query
//Vita/Died/*
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 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>
```
Another example query

```xml
(//*[Name]/When) | (//Where)
```
Abbreviated Syntax for Forward XPath

Example document:

```xml
<Composer>
  <Name>Claude Debussy</Name>
  <Vita>
    <Born>August 22, 1862</Born>
    <Where>Paris</Where>
    <Married>October 1899</Married>
    <Whom>Rosalie</Whom>
    <Married>January 1908</Married>
    <Whom>Emma</Whom>
    <Died>March 25, 1918</Died>
    <Where>Paris</Where>
  </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>
Abbreviated Syntax for Forward XPath

Example document

```xml
<Composer>
  <Name>Claude Debussy</Name>
  <Vita>
    <Born>August 22, 1862</Born>
    <Married>October 1899</Married>
    <Married>January 1908</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)?
Question

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

Answer
No!
**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>
```
**XPath containment**

**Question**
Does \(/\text{Vita}/\text{Died}/\ast\) always select a subset of positions of \((\ast[\text{Name}]/\text{When}) \mid (\ast/\text{Where})\)?

**Answer**
No!

**Counter-example**

\[
\langle \text{Vita} \rangle \\
\langle \text{Died} \rangle \\
\quad \langle \text{How} \rangle \text{ Heart disease } \langle /\text{How} \rangle \\
\langle /\text{Died} \rangle \\
\langle /\text{Vita} \rangle
\]

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

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

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

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

Useful observation [Miklau, Suciu 2002]
In the presence of [], Boolean containment has the same complexity as containment.

Crucial idea

\[
\begin{align*}
\begin{array}{c}
x \\
\| \\
p_1 & \subseteq & p'_{1} \\
p_2 & \subseteq & p'_{2} \\
\end{array}
\end{align*}
\]

if and only if

\[
\begin{align*}
\begin{array}{c}
x \\
\# \\
p_{1} & \subseteq_b & p'_{1} \\
p_{2} & \subseteq_b & p'_{2} \\
\end{array}
\end{align*}
\]
**XPath Containment: 2 Examples**

---

**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
Contents for XPath(/, //) and DTDs

Result 1 [Neven, Sch. 2003]
The Boolean containment problem for XPath(/, //) in the presence of DTDs is in \textbf{PTIME}

Proof idea

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

- Deterministic top-down automaton $A_p$
- Deterministic top-down automaton $A_{\overline{q}}$ 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 $d \iff L(A_p \times A_{\overline{q}} \times A_d) = \emptyset$
- The latter can be checked in polynomial time
**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**

We again represent patterns like

\((/*[\text{Name}]//\text{When}) \mid (//\text{Where})\)

as query trees:

![Example query tree](image)

**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
**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 $S/_v$ and $S//_v$ are sets of positions of the query tree:
  - $S/_v$: positions matching $v$
  - $S//_v$: 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 \( \text{xPath}(/, //, [], *, \mid) \) in the presence of DTDs is **EXPTIME**-hard

Proof sketch
Proof by reduction from *Two-player corridor tiling*

Example

Example:
Top row \( T = \text{\begin{tabular}{ccc} c & a & a & c \end{tabular}} \)
Bottom row \( B = \text{\begin{tabular}{ccc} a & c & a & c \end{tabular}} \)

Vertical and horizontal constraints:
\( V = \text{\begin{tabular}{ccc} c & a & c \\ c & c & a \end{tabular}} \)
\( H = \text{\begin{tabular}{ccc} a & c \\ a & a \\ c & a \end{tabular}} \)

Player I to move
\( \text{\begin{tabular}{ccc} c & a & a & c \\ : & : & : & : \end{tabular}} \)
\( \text{\begin{tabular}{ccc} a & c & a & c \end{tabular}} \)
**Theorem**

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

**Proof sketch**

Proof by reduction from *Two-player corridor tiling*

**Example**

Example:

Top row $T = \begin{array}{c|c|c|c} 
\text{c} & \text{a} & \text{a} & \text{c} \\
\end{array}$

Bottom row $B = \begin{array}{c|c|c|c} 
\text{a} & \text{c} & \text{a} & \text{c} \\
\end{array}$

Vertical and horizontal constraints:

$V = \begin{array}{c|c|c|c} 
\text{c} & \text{a} & \text{c} \\
\text{c} & \text{c} & \text{a} \\
\end{array}$

$H = \begin{array}{c|c|c|c} 
\text{a} & \text{c} & \text{a} & \text{c} \\
\text{a} & \text{a} & \text{c} \\
\end{array}$

Player II to move
Corresponding Lower Bound

Theorem
The containment problem for XPath(//, //, [], *, |) in the presence of DTDs is EXPTIME-hard

Proof sketch
Proof by reduction from Two-player corridor tiling

Example

Example:
Top row \( T = \begin{array}{cccc} c & a & a & c \end{array} \)
Bottom row \( B = \begin{array}{cccc} a & c & a & c \end{array} \)
Vertical and horizontal constraints:
\( V = \begin{array}{ccc} c & a & c \\ c & c & a \end{array} \)
\( H = \begin{array}{cccc} a & c & a & a \\ a & a & c & a \end{array} \)
Player I to move
\( \begin{array}{cccc} c & a & a & c \\ \vdots & \vdots & \vdots & \vdots \end{array} \)
Theorem

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

Proof sketch

Proof by reduction from $Two$-$player$ $corridor$ $tiling$

Example:

Top row $T = \begin{array}{ccc} c & a & a & c \end{array}$

Bottom row $B = \begin{array}{ccc} a & c & a & c \end{array}$

Vertical and horizontal constraints:

$V = \begin{array}{ccc} c & a & c \\ c & c & a \end{array}$

$H = \begin{array}{ccc} a & c, & a, a, c, a \end{array}$

Player II to move
Corresponding Lower Bound

Theorem

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

Proof sketch

Proof by reduction from *Two-player corridor tiling*

Example

Example:
Top row $T = [c, a, a, c]
Bottom row $B = [a, c, a, c]
Vertical and horizontal constraints:
$V = [c, a, c, c, a]
$H = [a, c, a, a, c, a]

Player I to move

$[c, a, a, c, a, a, c, a, c, a]$
Corresponding Lower Bound

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

Proof sketch
Proof by reduction from $\text{Two-player corridor tiling}$

Example:

Example:
Top row $T = \begin{bmatrix} c & a & a & c \end{bmatrix}$
Bottom row $B = \begin{bmatrix} a & c & a & c \end{bmatrix}$

Vertical and horizontal constraints:
$V = \begin{bmatrix} c & a & c \\ c & c & a \end{bmatrix}$
$H = \begin{bmatrix} a & c, a & a, c & a \end{bmatrix}$

Player II to move
$\begin{bmatrix} c & a & a & c \\ \vdots & \vdots & \vdots & \vdots \\ c & c & c & c \end{bmatrix}$
Corresponding Lower Bound

Theorem

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

Proof sketch

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

Example

Example:

Top row $T = \begin{bmatrix} c & a & a & c \end{bmatrix}$

Bottom row $B = \begin{bmatrix} a & c & a & c \end{bmatrix}$

Vertical and horizontal constraints:

$V = \begin{bmatrix} c & a & c \end{bmatrix}$

$H = \begin{bmatrix} a & c, & a & a, & c & a \end{bmatrix}$

Player I to move

Player II lost

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

Proof Sketch (cont.)

Tiles over \( \{a, b, c\} \)
Strategies as trees

Proof Sketch (cont.)

Tiles over \{a, b, c\}

This DTD describes all strategy trees:

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

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

\[
(\sigma, II) \rightarrow (a, I) + (b, I) + (c, I) + \# + S^I + !
\]

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

\[
S^I \rightarrow (a, I) + (b, I) + (c, I)
\]

\[
\$ = \text{line separator} \quad \# = \text{terminal symbol} \\
! = \text{indicates misplaced tile}
\]
Strategies as trees

Tiles over \{a, b, c\}

This DTD describes all strategy trees:

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

$ = line separator  # = terminal symbol  ! 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 $/S \not\subseteq q$ wrt DTD
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 $/S \not\subseteq q$ 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 $!$

$\implies$ We have to check that I does this correctly
Player I has winning strategy $\iff /S \not\subseteq q$

$q$ expresses that one of the following holds

- Player I violates a horizontal constraint:
- Player I violates a vertical constraint:
- Some row does not contain exactly $n$ tiles
- Player I wrongly claims a mistake of II:
- Some more conditions on $B$ and $T$
Path conditions

Proof Sketch (cont.)

Player I has winning strategy $\iff /S \not\subset q$

$q$ expresses that one of the following holds

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

- Player I violates a vertical constraint:

  - Some row does not contain exactly $n$ tiles
  - Player I wrongly claims a mistake of II:

    - Some more conditions on $B$ and $T$

  $\ast = \text{OR of all symbols, } \sigma^i = \sigma/\cdots/\sigma$ ($i$ times)
Player I has winning strategy $\iff \neg S \subseteq q$

$q$ expresses that one of the following holds

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

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

- Some row does not contain exactly $n$ tiles

- Player I wrongly claims a mistake of II:

- Some more conditions on $B$ and $T$

  $\ast = \text{OR of all symbols}, \sigma^i = \sigma/\cdots/\sigma$ ($i$ times)
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:
  \[
  \text{For each } (x, y) \not\in H: \quad // (x, II)/ (y, I)
  \]

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

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

- Player I wrongly claims a mistake of II:

- Some more conditions on \( B \) and \( T \)

\( * = \text{OR of all symbols, } \sigma^i = \sigma / \cdots / \sigma \ (i \text{ times}) \)

\[
\mathcal{D} = (d_1, I) / \cdots / (d_m, I) / (d_1, II) / \cdots / (d_m, II)
\]
**Path conditions**

**Proof Sketch (cont.)**

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

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

1. Player I violates a horizontal constraint:
   
   \[
   \forall (x, y) \notin H: \quad \langle / (x, II)/(y, I) \rangle
   \]

2. Player I violates a vertical constraint:
   
   \[
   \forall (x, y) \notin V: \quad \langle / (x, I)/(y, I) \rangle
   \]

3. Some row does not contain exactly \( n \) tiles
   
   \[
   D^{n+1} \mid \bigcup_{i=0}^{n-1} \langle (S^I|S^II|S)/D^i/(S^I|S^II|\#) \rangle
   \]

4. Player I wrongly claims a mistake of II:
   
   \[
   \forall (x, y) \in V, (x', y) \in H: \quad \langle / (x, II)/(y, II) \rangle
   \]

5. Some more conditions on \( B \) and \( T \)
   
   \* \( = \) OR of all symbols, \( \sigma^i = \sigma/\cdots/\sigma \) (\( i \) 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 LOGSPACE and PTIME

Satisfiability
- Without negation: PTIME or NP
- With negation: non-elementary

Containment
- Varying from PTIME to undecidable
- Upper bound for positive navigational XPath?
Contents

Introduction

Background on Tree Automata and Logic

Schema Languages

XPath and Node-selecting Queries

XSLT

XQuery

Conclusion
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. $d_1$?
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?
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 `xsl:apply-templates` is called at a node `v` the following happens:

- Compute set \( S(v) \) of nodes, reachable from `v` via `Expression` (if `select` is not present, \( S(v) = \text{children of } v \))
- 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
    `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
**Example XSLT**

```xml
<xsl:template match="a">
  <a>
    <xsl:apply-templates/>
  </a>
</xsl:template>

<xsl:template match="a" mode="below">
  <d>
    <xsl:apply-templates/>
  </d>
</xsl:template>

<xsl:template match="b">
  <b>
    <xsl:apply-templates mode="below"/>
  </b>
</xsl:template>

<xsl:template match="b" mode="below">
  <b>
    <xsl:apply-templates mode="below"/>
  </b>
</xsl:template>

<xsl:template match="c">
  <c/>
</xsl:template>

<xsl:template match="c" mode="below">
  <c/>
</xsl:template>
```

*Remove everything below a c. Translate a below b into d*
XSLT: Example

Example Transformation

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

Example XSLT (Abbreviated)

```xml
〈... 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〉〈/...〉
```
**XSLT: Example**

Example Transformation

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

Example XSLT (Abbreviated)

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

Example Trees
XSLT: Example

Example Transformation

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

Example XSLT (Abbreviated)

```xml
<... 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
```
XSLT: Example

Example Transformation

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

Example XSLT (Abbreviated)

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

Example Trees
XSLT: Example

Example Transformation

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

Example XSLT (Abbreviated)

```xml
<... 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
 / \
 a b   a   a c
     |   |   |
    a a a b
  a  a  b
```

⇒

```
  a               a
 / \
 a b   a   a c
     |   |   |
    a a a b
  a  a  b
```
XSLT: Example

Example Transformation
Remove everything below a c. Translate a below b into d.

Example XSLT (Abbreviated)

\[
\langle \ldots \text{match}="a" \rangle \langle a \rangle \langle \text{xsl:apply-templates} \rangle \langle /a \rangle \langle /\ldots \rangle \\
\langle \ldots \text{match}="a" \text{ mode}="\text{below}" \rangle \langle d \rangle \langle \text{xsl:apply-templates} \rangle \langle /d \rangle \langle /\ldots \rangle \\
\langle \ldots \text{match}="b" \rangle \langle b \rangle \langle \text{xsl:apply-templates \text{ mode}="\text{below}"} \rangle \langle /b \rangle \langle /\ldots \rangle \\
\langle \ldots \text{match}="b" \text{ mode}="\text{below}" \rangle \langle b \rangle \langle \text{xsl:apply-templates \text{ mode}="\text{below}"} \rangle \langle /b \rangle \langle /\ldots \rangle \\
\langle \ldots \text{match}="c" \rangle \langle c \rangle \langle /c \rangle \langle /\ldots \rangle \\
\langle \ldots \text{match}="c" \text{ mode}="\text{below}" \rangle \langle c \rangle \langle /c \rangle \langle /\ldots \rangle 
\]

Example Trees

Schwentick XML: Algorithms & Complexity

Introduction - XPath

152
XSLT: Example

Example Transformation

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

Example XSLT (Abbreviated)

```xml
... 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
XSLT: More involved example

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
**XSLT: More involved example**

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

**Example Trees**

```
   b
  /  \
 a    a
  /  \
 a    b
     /  \n    d    c
     /    \
     e
```
**XSLT: More involved example**

**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

```
  a  a  a  b  c
  \
  \
  d  e

⇒

  b
```
XSLT: More involved example

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

Example Trees
**XSLT: More involved example**

**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'">
  
    <xsl:apply-templates select='child[1]' mode="acopy"/>

    <xsl:copy-of select=''/child[last()]'/>

  </a>
</xsl:template>
```

**Example Trees**

Original tree: `a` `b` `c` `d` `e`

Transformed tree: `a` `b` `c` `d` `e`
XSLT: More involved example

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

Example Trees

```
(a)   b
  |____ a
     |___ b
        |____ a
```

⇒
```
(a)   a
   /____ c
      /____ d
          /____ e
```

```
**XSLT: More involved example**

**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 / 
 c
 d e
```

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

Schwentick

XML: Algorithms & Complexity

Introduction – XPath
**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
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 ($current$) can move, depending on state, number of pebbles under pebbles, and incidence of pebbles.
- Possible pebble movements:
  - stay
  - go to left child, right child, or parent
  - lift current
  - place new pebble on 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.

Example: Unranked Tree

... as binary tree
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
Back to the Typechecking Question

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 $T(L(d_1)) \subseteq L(d_2)$
Back to the Typechecking Question

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

\[ \begin{aligned}
    &a &a &a &a \\
    &b
\end{aligned} \]

into

\[ \begin{aligned}
    &a &a \\
    &b \\
    &a &a \\
    &a &a \\
    &a &a
\end{aligned} \]
Back to the Typechecking Question

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

  \[
  \begin{array}{c}
  & b \\
  a & a & a & a \\
  a & a & a & a & a
  \end{array}
  \]
  into

  \[
  \begin{array}{c}
  & b \\
  a & a \\
  a & a & a & a & a
  \end{array}
  \]

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

Definition: $k$-pebble acceptors

- Basically the same as $k$-pebble transducers
- Instead of output producing steps:
  - \textit{accept}
  - branch into two independent subcomputations
- A tree is accepted if all subcomputations accept

Main Steps of the Proof

(i) $T^{-1}(L(d_2))$ is accepted by a $k$-pebble acceptor
(ii) $k$-pebble acceptors only accept regular tree languages
Step (i)

**Lemma**

$T^{-1}(L(d_2))$ 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 $T^{-1}(L(d_2))$, 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 $G = (V, E)$, $V = V_\land \cup V_\lor$. A node $w$ is accessible if

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

**Example**

![Diagram showing the accessibility of nodes in an alternating graph.](attachment:diagram.png)
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
**Alternating Graph Accessibility**

**Definition: Accessible Nodes**

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

1. $w \in V_\wedge$ and all successors of $w$ are accessible, or
2. $w \in V_\lor$ and at least one successor of $w$ is accessible.
Definition: Accessible Nodes

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

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

Definition: Accessible Nodes

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

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

Example
Alternating Graph Accessibility

Definition: Accessible Nodes

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

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

Example
Alternating Graph Accessibility

**Definition: Accessible Nodes**

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

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

**Example**

![Graph Diagram]

**Definition: Alternating Graph Accessibility Problem (AGAP)**

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

**Question:** Is $v$ accessible?
### 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 $V_\land$ 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**

![Graph Example]

**Fact**

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

**...as MSO-Formula**

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

\[
\text{reverse-closed}(S) \equiv \forall x \ ([V_\land (x) \land \forall y \ (E(x, y) \rightarrow S(y))] \rightarrow S(x)) \land\\
([V_\lor (x) \land \exists y \ (E(x, y) \land S(y))] \rightarrow S(x))
\]
Proof idea
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 $S_i^1$ for each state $q_{1i}$
  - Intended meaning of $v \in S_i^1$:
    there is an accepting subcomputation of $A$ starting at $v$ in state $q_{1i}$
  - $\varphi = \forall S_1^1 \cdots \forall S_m^1$ (reverse-closed $\rightarrow S_1^1$(root))
  - reverse-closed is a conjunction of subformulas, induced by the transitions of $A$, e.g.:
    - if $(q_{1i},a) \rightarrow$ accept then $\forall x \ Q_a(x) \rightarrow S_i^1(x)$
    - if $(q_{1i},a) \rightarrow (q_{1j},\text{down-right})$ then $\forall x \ \forall y \ (Q_a(x) \land E_r(x,y) \land S_j^1(y)) \rightarrow S_i^1(x)$
$k$-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 $\forall S^2_1 \cdots \forall S^2_m$ (reverse-closed$^2 \rightarrow S^2_j$(root))
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 \( T^{-1}(L(d_2)) \)
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 \text{PTIME} and \text{EXPTIME} [Martens and Neven 2003]

- 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 and Grohe 2002]

Open

- Find (more) transformations with a tractable typechecking problem

- In particular, with data values
Introduction

Background on Tree Automata and Logic

Schema Languages

XPath and Node-selecting Queries

XSLT

XQuery

Conclusion
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.
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

$\theta(x) = \ldots$
$\theta(y) = \ldots$
**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**

$\theta(x) = \ldots$
$\theta(y) = \ldots$
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 $x$:
      - $h(x) :=$ minimal node in $\theta(x)$ wrt $<$.

Example Document:

Let $<$ be the breadth-first left-to-right order.
### A Generic Algorithm (cont.)

<table>
<thead>
<tr>
<th>Fact</th>
</tr>
</thead>
<tbody>
<tr>
<td>A maximal consistent pre-valuation $\theta$ can be computed in time $O(\text{query size} \times \text{tree size})$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithmic Idea</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Let $&lt;$ be a total order on the nodes</td>
</tr>
</tbody>
</table>
| • For query $Q$ and tree $t$:
  - Compute maximal consistent pre-valuation $\theta$
  - Define $<$-minimal valuation $h$ via:
    - For each variable $x$:
    - $h(x) :=$ minimal node in $\theta(x)$ wrt $<$ |

**Example Document**

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

Let $<$ be the breadth-first left-to-right order
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 $<\text{-minimal valuation } h$ via:
    - For each variable $x$:
      - $h(x) := \text{minimal node in } \theta(x) \text{ wrt } <$

**Example Document**

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

Let $<$ be the breadth-first left-to-right order.

**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?
A Generic Algorithm (cont.)

Example Query

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

Example Pre-Valuation

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

Example Document

![Example Document Diagram]

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

Observations

- Let \( u = h(x) \), \( v = h(y) \)
- As \( u \in \theta(x) \) there is \( v' \) such that \( \text{child}(u, v') \)
- As \( v \in \theta(y) \) there is \( u' \) 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 \( Q \)
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 $O(query \ size \times \ tree \ size)$
Combined Complexity of Conjunctive Queries

**Observation**
It is sufficient to consider the axes `child, child⁺, child*, NextSibling, NextSibling⁺, NextSibling*, Following`

**Theorem [Gottlob, Koch, Schulz 04]**
- `child⁺` and `child*` are preorder-hemichordal
- `following` is postorder-hemichordal
- `child, NextSibling, NextSibling⁺, NextSibling*, Following` are breadth-first-left-to-right-hemichordal

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

Observation
It is sufficient to consider the axes child, child\(^+\), child\(^*\), NextSibling, NextSibling\(^+\), NextSibling\(^*\), Following

Theorem [Gottlob, Koch, Schulz 04]
- child\(^+\) and child\(^*\) are preorder-hemichordal
- following is postorder-hemichordal
- child, NextSibling, NextSibling\(^+\), 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><strong>Introduction</strong></td>
</tr>
<tr>
<td><strong>Background on Tree Automata and Logic</strong></td>
</tr>
<tr>
<td><strong>Schema Languages</strong></td>
</tr>
<tr>
<td><strong>XPath and Node-selecting Queries</strong></td>
</tr>
<tr>
<td><strong>XSLT</strong></td>
</tr>
<tr>
<td><strong>XQuery</strong></td>
</tr>
<tr>
<td><strong>Conjunctive Queries</strong></td>
</tr>
<tr>
<td><strong>Automata and XQuery</strong></td>
</tr>
<tr>
<td><strong>Conclusion</strong></td>
</tr>
</tbody>
</table>
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: $O(|\mathcal{A}||t|)$
  - Non-emptiness: PSPACE
  - Containment: PSPACE

[Seidl, Sch., Muscholl, Habermehl 2004]
<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><strong>Schema Languages</strong></td>
</tr>
<tr>
<td><strong>XPath and Node-selecting Queries</strong></td>
</tr>
<tr>
<td><strong>XSLT</strong></td>
</tr>
<tr>
<td><strong>XQuery</strong></td>
</tr>
<tr>
<td><strong>Conclusion</strong></td>
</tr>
</tbody>
</table>
Conclusion

Summary

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