# Contents

**Introduction**

- XML: Tasks
- Logic on Trees

**MSO Logics**

**Weaker Logics**

**Extensions**

**Conclusion**
Composer

Vita

Name

Claude Debussy

Born

When
1862
Where
Paris

Married

When
1899
Whom
Rosalie

When
1908
Whom
Emma

Died

When
1918
Where
Paris

Piece

PTitle

La Mer

PYear

1905

Instruments

Large orchestra

Movements

3
## XML Processing

<table>
<thead>
<tr>
<th>Kind</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>
Validation: DTD

DTDs describe types of XML documents.

Example document

```xml
<Composer>
  <Name>Claude Debussy</Name>
  <Vita>
    <Born>
      <When>August 22, 1862</When>
      <Where>Paris</Where>
    </Born>
    <Married>
      <When>October 1899</When>
      <Whom>Rosalie</Whom>
    </Married>
    <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)>
]> M¨unchen May 05 Logic and XML 4 Thomas Schwentick
```
Navigation: **XPath**

**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>
```

**XPath**

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

**Example query**

```xml
//Vita/Died/*
```

**Remark**

XPath expressions define sets of nodes: *node-selecting queries*.
XQuery is a full-fledged XML query language

Example query

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

Result

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

...
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

```
<xsl:template match="Composer[Vita//Where='Paris']">
  <ParisComposer>
    <Name> Claude Debussy </Name>
    <When> August 22, 1862 </When>
    <Where> Paris </Where>
  </ParisComposer>
</xsl:template>
```

Result

```
<ParisComposer>
  <Name> Claude Debussy </Name>
  <When> August 22, 1862 </When>
  <Where> Paris </Where>
</ParisComposer>
```

XSLT transforms documents by means of templates

```
<xsl:template match="Composer[Vita//Where='Paris']">
  <ParisComposer>
    <Name> Frédéric Chopin </Name>
    <When> March 1, 1810 </When>
    <Where> Żelazowa </Where>
  </ParisComposer>
</xsl:template>
```

```
<ParisComposer>
  <Name> Camille Saint-Saëns </Name>
  <When> October 9, 1835 </When>
  <Where> Paris </Where>
</ParisComposer>
```
A Schematic View

DTD/XML Schema

XPath

XQuery

XSLT
The Big Picture

XML Languages

Known Formal Models

Suitable Fragments

New Formal Models
<table>
<thead>
<tr>
<th>Algorithmic Tasks</th>
<th>Static Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Evaluation</strong></td>
<td><strong>Static Analysis</strong></td>
</tr>
<tr>
<td>Evaluation (Combined)</td>
<td>Satisfiability</td>
</tr>
<tr>
<td>I: Tree $t$, Query $q$</td>
<td>I: Query $q$</td>
</tr>
<tr>
<td>O: $q(t)$</td>
<td>Q: Is $q(t) \neq \emptyset$ for some $t$?</td>
</tr>
<tr>
<td>Evaluation (Data($q$))</td>
<td></td>
</tr>
<tr>
<td>I: Tree $t$</td>
<td></td>
</tr>
<tr>
<td>O: $q(t)$</td>
<td></td>
</tr>
<tr>
<td>Incremental Eval. ($q$)</td>
<td></td>
</tr>
<tr>
<td>I: Tree $t$, Changes of $t$</td>
<td></td>
</tr>
<tr>
<td>O: $q(t)$</td>
<td></td>
</tr>
<tr>
<td><strong>Containment</strong></td>
<td></td>
</tr>
<tr>
<td>I: Queries $q_1, q_2$</td>
<td></td>
</tr>
<tr>
<td>Q: Is always $q_1(t) \subseteq q_2(t)$?</td>
<td></td>
</tr>
<tr>
<td><strong>Equivalence</strong></td>
<td></td>
</tr>
<tr>
<td>I: Queries $q_1, q_2$</td>
<td></td>
</tr>
<tr>
<td>Q: Is always $q_1(t) = q_2(t)$?</td>
<td></td>
</tr>
<tr>
<td><strong>Type Checking</strong></td>
<td></td>
</tr>
<tr>
<td>I: Types $d_1, d_2$,</td>
<td></td>
</tr>
<tr>
<td>Transformation $T$</td>
<td></td>
</tr>
<tr>
<td>Q: Does $t \models d_1$ imply $T(t) \models d_2$?</td>
<td></td>
</tr>
<tr>
<td><strong>Type Inference</strong></td>
<td></td>
</tr>
<tr>
<td>I: Types $d$,</td>
<td></td>
</tr>
<tr>
<td>Transformation $T$</td>
<td></td>
</tr>
<tr>
<td>O: Type of ${T(t) \mid t \models d}$</td>
<td></td>
</tr>
</tbody>
</table>
A Look Back

Relational Databases
- SQL
- Relational Algebra
- FO-Logic

Basic Properties of these Formalisms
**SQL**
- Declarative, easy to use
- Queries, data definition, updates

**FO-logic**
- Formal framework for investigations
- Clear Semantics
- Expressive power well understood

**Relational Algebra**
- Operational model
- Flexible, optimizable
- Automatic translation

Further Properties
- Satisfiability undecidable
- Fragments like conjunctive queries
- Evaluation for conjunctive queries NP-hard
- but works well in practice
- SQL can count and group
- Can be added to FO

Goal
- XML-Languages
- Automata
- ???-Logic

[Hella et al. 01]
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduction</strong></td>
</tr>
<tr>
<td>XML: Tasks</td>
</tr>
<tr>
<td>Logic on Trees</td>
</tr>
<tr>
<td>MSO Logics</td>
</tr>
<tr>
<td>Weaker Logics</td>
</tr>
<tr>
<td>Extensions</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
Example Document

\[\langle a \rangle \langle b \rangle \langle d \rangle 12 \langle /d \rangle \langle e \rangle 22 \langle /e \rangle \langle /b \rangle \langle b \rangle \langle d \rangle 4 \langle /d \rangle \langle /b \rangle \langle b \rangle \langle d \rangle 11 \langle /d \rangle \langle e \rangle 8 \langle /e \rangle 18 \langle e \rangle 5 \langle /e \rangle \langle /b \rangle \langle c \rangle \langle b \rangle \langle f \rangle 2 \langle /f \rangle \langle /b \rangle 7 \langle /c \rangle \langle /a \rangle\]

...as Unranked Tree

...as Binary Tree
### XML Trees as Finite Models

<table>
<thead>
<tr>
<th>Data Values</th>
<th>Signatures</th>
</tr>
</thead>
<tbody>
<tr>
<td>For many investigations, data values can and have to be ignored</td>
<td>There are various ways to represent unordered trees as finite relational models</td>
</tr>
<tr>
<td>Example</td>
<td>Possible relations</td>
</tr>
<tr>
<td><img src="image" alt="XML Tree Example" /></td>
<td>– Local orders: →, ↓ ↔, ↑</td>
</tr>
<tr>
<td><img src="image" alt="XML Tree Example" /></td>
<td>– Their transitive closures: →+, ↓+, ↔+, ↑+</td>
</tr>
<tr>
<td><img src="image" alt="XML Tree Example" /></td>
<td>– Reflexive and transitive: →<em>, ↓</em>, ↔<em>, ↑</em></td>
</tr>
<tr>
<td><img src="image" alt="XML Tree Example" /></td>
<td>– Document order: ↑* →↓*</td>
</tr>
<tr>
<td><img src="image" alt="XML Tree Example" /></td>
<td>Frequent combinations:</td>
</tr>
<tr>
<td><img src="image" alt="XML Tree Example" /></td>
<td>– →, ↓</td>
</tr>
<tr>
<td><img src="image" alt="XML Tree Example" /></td>
<td>– →+, ↓+</td>
</tr>
<tr>
<td><img src="image" alt="XML Tree Example" /></td>
<td>– →, ↓, →+, ↓+</td>
</tr>
<tr>
<td><img src="image" alt="XML Tree Example" /></td>
<td>– →, ↓</td>
</tr>
</tbody>
</table>

### Labels

- Usually, XML trees are modeled as labeled trees over a finite alphabet
- For schema languages this is ok
- For queries, the actual alphabet might depend on the query
- Unary predicates on data values can be also modeled this way
## Contents

Introduction

**MSO Logics**

- Automata and MSO-logic on Trees
- Schema Languages
- Node-selecting Queries
- XML Transformations

**Weaker Logics**

**Extensions**

**Conclusion**
Automata for Ranked Trees

**Bottom-up Automaton for (Tree-) Boolean Circuits**

- **Idea**
  - Two states: $q_0, q_1$
  - $q_1 \equiv$ subtree evaluates to 1

- **Transitions**
  - $\delta(\land, q_1) = \{(q_1, q_1)\}$
  - $\delta(\land, q_0) = \{(q_0, q_1), (q_1, q_0), (q_0, q_0)\}$
  - $\delta(\lor, q_1) = \{(q_0, q_1), (q_1, q_0), (q_1, q_1)\}$
  - $\delta(\lor, q_0) = \{(q_0, q_0)\}$
  - $\delta(0, q_0) = \{\epsilon\}; \delta(0, q_1) = \emptyset$
  - $\delta(1, q_1) = \{\epsilon\}; \delta(1, q_0) = \emptyset$

**Top-down Automaton for Boolean Circuits**

- **Idea**
  - Three states: $q_0, q_1, \text{acc}$
  - $q_1 \equiv$ subtree will be 1

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

(a) \( \implies \) (b): Powerset construction

(a) \( \iff \) (c): Just the same thing, viewed in two different ways

**Definition**

Such an \( L \) is called **regular**

**Observation**

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

  ![Example Tree](image)

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

- **MSO-formulas** talk about
  - node labels \((Q_0, Q_1, Q_\land, Q_\lor)\)
  - Children and neighbors: \(\to, \downarrow\)
  - the root of the tree (root)

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

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

**Remark**

Exact signature does not matter

---

**Example: Boolean Circuits**

\[
\exists X \ X (\text{root}) \land \forall x
\]

\[
(Q_0(x) \to \neg X(x)) \land
((Q_\land(x) \land X(x)) \to (\forall y[(x \downarrow y) \to X(y)]) \land
((Q_\lor(x) \land X(x)) \to (\exists y[(x \downarrow y) \land X(y)])
\]

---

**Theorem (Doner 70; Thatcher, Wright 68)**

On ranked trees:

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

**Proof idea**

**Automata \(\Rightarrow\) MSO:**

Formula expresses that there exists a correct tiling

**MSO \(\Rightarrow\) Automata:** more involved

Basic idea:

Automaton computes for each node \(v\) the set of formulas which hold in the subtree rooted at \(v\)
**MSO \Rightarrow Automata: more precisely**

- 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**

```
  t_1 \equiv_k t'_1  t_2 \equiv_k t'_2
  ↓
  l
  t_1 \equiv_k t'_1  t_2 \equiv_k t'_2
```
From Ranked to Unranked Trees

**Ranked trees**

On 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

[Bruggemann-Klein, Murata, Wood 2001]
### Theorem

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

### Definition

Again: such an $L$ is called **regular**

### Complexity issues for MSO on trees

**Data Complexity:**

Query evaluation is possible in time $O(|t|)$

**Combined Complexity:**

- Query evaluation is complete for $\text{PSPACE}$
- Query evaluation is possible in time $f(|\varphi|)|t|$, where $f$ is $\sim 2^{2^{\ldots 2^{\varphi}}}$

- No elementary $f$ possible unless $P = NP$ [Frick, Grohe 2002]

- Satisfiability: $f(|\varphi|)$ (same $f$)

- In practice much better: MONA [Klarlund et al.]
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
</tr>
<tr>
<td><strong>MSO Logics</strong></td>
</tr>
<tr>
<td>Automata and MSO-logic on Trees</td>
</tr>
<tr>
<td><strong>Schema Languages</strong></td>
</tr>
<tr>
<td>Node-selecting Queries</td>
</tr>
<tr>
<td>XML Transformations</td>
</tr>
<tr>
<td>Weaker Logics</td>
</tr>
<tr>
<td>Extensions</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
## DTDs and their Weakness

<table>
<thead>
<tr>
<th>DTDs</th>
<th>Weakness of DTDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>• DTDs are essentially generalized context-free grammars</td>
<td>• Elements with the same name may have different structure in different contexts</td>
</tr>
<tr>
<td>→ [Berstel,Boasson 00] provide characterizations</td>
<td>→ It would be nice to have types for elements</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td><strong>Extended DTDs</strong></td>
</tr>
</tbody>
</table>

```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)>
]>```

**Intention**

```
Dealer
  UsedCars
    ad
    model
  NewCars
    ad
    year
    model
```
**Extended DTDs**

**Definition [Papakonstantinou, Vianu 2000]**

An **extended DTD** (EDTD) 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

**Example**

<table>
<thead>
<tr>
<th>Dealer</th>
<th>UsedCars NewCars</th>
<th>$\mu$(Dealer) = Dealer</th>
</tr>
</thead>
<tbody>
<tr>
<td>UsedCars</td>
<td>adUsed*</td>
<td>$\mu$(UsedCars) = UsedCars</td>
</tr>
<tr>
<td>NewCars</td>
<td>adNew*</td>
<td>$\mu$(NewCars) = NewCars</td>
</tr>
<tr>
<td>adUsed</td>
<td>model year</td>
<td>$\mu$(adUsed) = ad</td>
</tr>
<tr>
<td>adNew</td>
<td>model</td>
<td>$\mu$(adNew) = ad</td>
</tr>
</tbody>
</table>

**Note**

Extended DTDs are often called *specialized DTDs*

**EDTD for Boolean circuits**

<table>
<thead>
<tr>
<th>Operator</th>
<th>EDTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-AND</td>
<td>(1-OR</td>
</tr>
<tr>
<td>1-OR</td>
<td>.* (1-OR</td>
</tr>
<tr>
<td>0-AND</td>
<td>.* (0-OR</td>
</tr>
<tr>
<td>0-OR</td>
<td>(0-OR</td>
</tr>
<tr>
<td>1-leaf</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>0-leaf</td>
<td>$\epsilon$</td>
</tr>
</tbody>
</table>

- **Tag**
- **$\mu$(Tag)**

<table>
<thead>
<tr>
<th>Tag</th>
<th>$\mu$(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>
**Extended DTDs and MSO**

**Observation**
- A tree conforms to an extended DTD \((d, \mu)\) if there is a labeling of its nodes by types which is valid wrt. \(d\).
- This reminds us of something...

**Theorem**
Extended DTDs capture exactly the regular tree languages

**Remarks**
- Regular tree languages and MSO-logic are a convenient framework for the study of XML schema languages.
- Practical languages as XML Schema usually correspond to subclasses.
- Full MSO power: Relax NG.
- What about queries?
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
</tr>
<tr>
<td><strong>MSO Logics</strong></td>
</tr>
<tr>
<td>Automata and MSO-logic on Trees</td>
</tr>
<tr>
<td>Schema Languages</td>
</tr>
<tr>
<td>Node-selecting Queries</td>
</tr>
<tr>
<td>XML Transformations</td>
</tr>
<tr>
<td><strong>Weaker Logics</strong></td>
</tr>
<tr>
<td><strong>Extensions</strong></td>
</tr>
<tr>
<td><strong>Conclusion</strong></td>
</tr>
</tbody>
</table>
## Node-selecting Queries

<table>
<thead>
<tr>
<th>Question</th>
<th>Existential vs. universal semantics</th>
</tr>
</thead>
</table>
| Is there an equally robust class of node-selecting queries? | - 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 |

### Observations

- 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 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 $s(q, \lambda(v)) = 1$
Node-selecting queries (cont.)

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>A node selecting query is MSO-definable iff it is expressible by a nondeterministic bottom-up node selecting automaton</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proof idea</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Given formula ( \varphi(x) ) of quantifier-depth ( k ) and tree ( t ), for each node ( v ) the automaton does the following:</td>
</tr>
<tr>
<td>– Compute ( k )-type of subtree at ( v )</td>
</tr>
<tr>
<td>– Guess ( k )-type of &quot;envelope tree&quot; at ( v )</td>
</tr>
<tr>
<td>– Conclude whether ( v ) is in the output</td>
</tr>
<tr>
<td>– Check consistency upwards towards the root</td>
</tr>
<tr>
<td>( \Rightarrow ) one unique accepting run</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Crucial fact</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

\[ e_1 \quad \equiv_k \quad e_2 \]

\[ \& \]

\[ \downarrow \]

\[ t_1 \quad \equiv_k \quad t_2 \]
Node-selecting queries (cont.)

More query models

- Same combined complexity as for language recognition (\textsc{PSPACE}-complete)

→ query languages with better complexity properties needed

- Good candidate: Monadic Datalog \cite{GottlobKoch2002} and its restricted dialects like TMNF

- Further models:
  - Attributed Grammars \cite{NevenVanBussche1998}
  - $\mu$-formulas \cite{Neumann1998}
  - Context Grammars \cite{Neumann1999}
  - Deterministic Node-Selecting Automata \cite{NevenSch1999}

Data complexity

- MSO node-selecting queries can be evaluated in two passes

  \textbf{First pass, bottom-up:} Compute the types of the subtrees

  \textbf{Second pass, top-down:} Compute the types of the envelopes and combines it with the subtree information

→ Can be implemented by a 2-pass pushdown document automaton which in its first pass attaches information to each node \cite{NeumannSeidl1998;Koch2003}

- In particular: Data complexity is linear time
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
</tr>
<tr>
<td><strong>MSO Logics</strong></td>
</tr>
<tr>
<td>Automata and MSO-logic on Trees</td>
</tr>
<tr>
<td>Schema Languages</td>
</tr>
<tr>
<td>Node-selecting Queries</td>
</tr>
<tr>
<td><strong>XML Transformations</strong></td>
</tr>
<tr>
<td>Weaker Logics</td>
</tr>
<tr>
<td>Extensions</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
### XML Transformations and MSO-Logic

**Definition: XSLT TYPECHECKING**

<table>
<thead>
<tr>
<th>Given:</th>
<th>DTDs $d_1$ and $d_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformation $T$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result:</th>
<th>Is $T(t) \models d_2$ for each document $t$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>with $t \models d_1$?</td>
</tr>
</tbody>
</table>

**Questions**

- Is XSLT TYPECHECKING decidable?
- What is the complexity?

**Outline**

- 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
- [Milo,Suciu,Vianu 01]
XSLT in more detail

How XSLT roughly works

**Templates:** 
\[
\text{\texttt{\{xsl:template name=\textasciitilde TName} \text{match=\textasciitilde pattern} \\
\text{mode=\textasciitilde MName\}}}
\]

**Template application:**
\[
\text{\texttt{\{xsl:apply-templates select=\textasciitilde Expression} \\
\text{mode=\textasciitilde MName\}}}
\]

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

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

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

Example Transformation

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

Example XSLT (Abbreviated)

```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   c
```

⇒

```
        a
       /|
      a  b  a  c
     /  |  |
   a   d  b  a
```

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
  (Proof presented here: deterministic case)
- 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
Back to the Typechecking Question

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:

![Diagram of a tree with labels a, a, a, a, b]

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

Definition: $k$-pebble acceptors

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

Main Steps of the Proof

(i) $T$ computed by $k$-pebble transducer
   $\Rightarrow T = T_1 \circ \cdots \circ T_{k+1}$
   with $0$-pebble transducers $T_i$
   [Engelfriet, Maneth 03]
(ii) $L$ regular, $T_i$ $0$-pebble transducer
    $\Rightarrow T_i^{-1}(L)$ accepted by $0$-pebble acceptor
(iii) $0$-pebble acceptors only accept regular tree languages
**Step (ii)**

<table>
<thead>
<tr>
<th>Lemma</th>
</tr>
</thead>
</table>
| $L$ regular, $T_i$ 0-pebble transducer  
$\Rightarrow T_i^{-1}(L)$ accepted by 0-pebble acceptor |

<table>
<thead>
<tr>
<th>Proof</th>
</tr>
</thead>
</table>
| Let $B$ be a nondeterministic top-down tree automaton which accepts $\overline{L}$  
We construct a 0-pebble acceptor $A$ for $T_i^{-1}(\overline{L})$, i.e., an automaton which on input $t$ decides whether $T(t)$ 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 |
### Step (iii)

<table>
<thead>
<tr>
<th><strong>Lemma</strong></th>
<th><strong>Definition: AGAP</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0-pebble acceptors only accept regular tree languages</td>
<td>Given: Graph ( G = (V, E) ), ( V = V_\wedge \cup V_\lor ), and ( v \in V )</td>
</tr>
<tr>
<td>Proof idea</td>
<td>Question: Is ( v ) accessible?</td>
</tr>
</tbody>
</table>

**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:

![Diagram of AGAP with nodes and arrows]
0-Pebble Acceptors, AGAP and MSO

**Construction of $G_{A,t}$**

- Nodes in $V_\lor$ are the configurations of $A$ on $t$: pairs $[q,v]$, state $q$, node $v$ of $t$
- Nodes in $V_\land$ are $\epsilon$ and certain pairs $(\gamma_1, \gamma_2)$ of configurations
- 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 $A$

**Definition: Reverse-closed Sets of Nodes**

A set $S$ of nodes is reverse-closed if:

- 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**

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

**Facts**

- $A$ accepts $t \iff [q, \text{root}]$ is accessible in $G_{A,t}$ with $q \in F$
- $|G_{A,t}| = O(|t|)$

**Reverse-closed as MSO-Formula**

$$\forall S \ (\text{rc}(S) \rightarrow S(v)), \text{ where rc}(S) \text{ is}$$

$$\forall x \([V_\land(x) \land \forall y \ (E(x,y) \rightarrow S(y))] \rightarrow S(x)) \land$$

$$([V_\lor(x) \land \exists y \ (E(x,y) \land S(y))] \rightarrow S(x))$$

München May 05 Logic and XML 38 Thomas Schwentick
### Summary of proof

- 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 $L(d_1) \subseteq L(A')$

- Complexity: non-elementary

### Related work

- **TYPECHECKING** is decidable for compositions of macro tree transducers [Engelfriet, Maneth 03]

- 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. 01]

- 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, Neven 03]
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
</tr>
<tr>
<td>MSO Logics</td>
</tr>
<tr>
<td><strong>Weaker Logics</strong></td>
</tr>
<tr>
<td>Temporal Logics</td>
</tr>
<tr>
<td>First-Order Logics</td>
</tr>
<tr>
<td>Transitive-Closure Logics</td>
</tr>
<tr>
<td>Extensions</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
MSO-logic offers a framework for dealing with schemas
- Close connection to automata
- Helpful in proofs
- Extends to (node-selecting) queries
- But: for many querying tasks MSO-power is not needed

**XPath**: Navigation along paths

→ Similarity to temporal logics

- Presentation follows [Libkin 05]
Navigation and Temporal Logics (cont.)

**Definition: An LTL-like logic: TL\textsuperscript{tree}**

- $A, \varphi \lor \psi, \neg \varphi$
- $X^* \varphi, X^- \varphi, \varphi U^* \psi, \varphi U^- \psi$
  \((\ast = \rightarrow \text{ or } \downarrow)\)

**Theorem [Marx 04]**

A unary or binary query over unordered trees is FO-definable (with $\downarrow^*, \rightarrow^*$) iff it is definable in TL\textsuperscript{tree}

**Proof idea**

Proof similar as equivalence of LTL and FO on orders [Kamp 68]

---

**Definition: A CTL*-like logic: CTL*\textsubscript{past}**

- **Node formulas:**
  - $A, \alpha \lor \alpha', \neg \alpha$
  - $E \beta^*$
- **Path formulas:**
  - $\alpha, \neg \beta^*, \beta \lor \beta'$
  - $X^* \beta, X^- \beta, \beta U^* \beta', \beta U^- \beta'$
  \((\ast = \rightarrow \text{ or } \downarrow)\)

**Theorem [Barcelo, Libkin 05]**

A unary or binary query over unordered trees is FO-definable (with $\downarrow^*, \rightarrow^*$) iff it is definable in CTL*\textsubscript{past}

**Proof idea**

Proof similar as equivalence of CTL* and FO on binary trees [Hafer, Thomas 87]
A Related Result

Theorem [Marx 04]

Containment of Navigational $\text{XPath}$ queries in the presence of DTDs is in $\text{EXPTIME}$

Proof idea

- Navigational $\text{XPath}$ can be translated into propositional dynamic logic (PDL) (over structures with $\downarrow$, $\rightarrow$)
- DTDs can also be expressed by PDL-formulas
- Implication for PDL is $\text{EXPTIME}$-complete

Remark

For much weaker fragments, containment is already $\text{EXPTIME}$-hard
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
</tr>
<tr>
<td>MSO Logics</td>
</tr>
<tr>
<td><strong>Weaker Logics</strong></td>
</tr>
<tr>
<td>Temporal Logics</td>
</tr>
<tr>
<td>First-Order Logics</td>
</tr>
<tr>
<td>Transitive-Closure Logics</td>
</tr>
<tr>
<td>Extensions</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
**XPath and Fragments of First-order Logic**

<table>
<thead>
<tr>
<th>Characterizations of XPath</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>
<tr>
<td>[Benedikt, Fan, Kuper 03]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proof idea</th>
</tr>
</thead>
</table>
| ● Basic idea:  
  For each node $u$ of the query tree: guess a node $h(u)$ in the document tree and check that $h$ is a “homomorphism” |
| ● Main difficulty in proof:  
  Deal with conjunctions of conditions |

<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>

<table>
<thead>
<tr>
<th>Theorem [Marx, de Rijke 05]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node-selecting navigational XPath queries correspond to unary two-variable first-order formulas over $\downarrow, \downarrow^<em>, \rightarrow, \rightarrow^</em>$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proof idea</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proof follows the same lines as characterization of unary temporal logic by two-variable logic</td>
</tr>
<tr>
<td>[Etessami, Vardi, Wilke]</td>
</tr>
</tbody>
</table>
Question

What is needed to capture full first-order logic?

Conditional XPath

- **Conditional axes**: Expressions of the kind $P^+$, where $P$ is a location step
- **Conditional XPath**: Navigational XPath plus conditional axes

Example

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

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

Theorem [Marx 04,05]

Conditional XPath corresponds exactly to first-order logic over $\downarrow^*$, $\rightarrow^*$
(wrt node-selecting and binary queries)
First-Order Logic Plus Regular Expressions

+ vertical regular expressions (over paths)

+ horizontal regular expressions (over children)

+ Nesting of formulas and regular expressions

\[ [\varphi_1(s,t) \cdot \varphi_2(s,t)^* \cdot \varphi_3(s,t)]_{s,t}(x,y) \]
**Definition: Tree-walk automaton**

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

the automaton moves to a neighbor and takes a new state.

**Fact [Neven, Sch. 00]**

FO-sentences over binary trees with ↓ and → can be evaluated by deterministic Tree-Walk automata.

**Proof idea**
- Simple application of Gaifman’s Theorem
- Does not hold for unranked trees

**Remark**

To capture the expressive power of Tree-walk automata one needs a bit more...

**Theorem [Neven, Sch. 00]**

- Nondet. TWA $\equiv TC^1[FO[\downarrow, \mod]]$
- Det. TWA $\equiv DTC^1[FO[\downarrow, \mod]]$

over binary trees with ↓ and →.
### MSO-logic vs. Unary Transitive Closure Logic

<table>
<thead>
<tr>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>- On strings: MSO $\equiv$ FO $+$ unary TC</td>
</tr>
<tr>
<td>- On (binary) trees???</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theorem [Engelfriet, Hoogeboom 05]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nondeterministic pebble automata correspond to FO plus positive unary TC on binary trees</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Corollary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive, unary TC-logic is weaker than MSO on binary trees iff pebble automata are weaker than parallel automata</td>
</tr>
</tbody>
</table>

### Inclusion structure

\[
\begin{align*}
\text{MSO} &= \text{Pebble-Marble-TWA} = \text{ATWA} \\
\text{FO}(\text{posTC}^1) &= \text{Pebble-TWA} \\
\text{FOREG} &\neq \text{NTWA} = \text{TC}^1[\text{FO}[\downarrow, \text{mod}]] \\
\text{FO}[\downarrow^*] &\neq \text{DTWA} = \text{DTC}^1[\text{FO}[\downarrow, \text{mod}]] \\
\text{FO}[\downarrow] &\neq \text{FO}[\downarrow]
\end{align*}
\]
## Contents

- Introduction
- MSO Logics
- Weaker Logics
  - **Extensions**
    - Counting
    - Data Values
- Conclusion
## Extensions

### So far...

- We have seen logics for
  - Validation, Typing
  - Navigation
  - Transformation
- What about more general queries?
  - results of higher arity?
  - joins, i.e., comparisons of data values
  - counting

### 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:
  - \( \equiv \) EMSO logic
  - Whether automaton accepts all trees is undecidable
- Det. Presburger automata:
  - \( \equiv \) Presburger \( \mu \)-formulas
  - Membership test: \( O(|A| |t|) \)
  - Non-emptiness: \( \text{PSPACE} \)
  - Containment: \( \text{PSPACE} \)

[Seidl, Sch., Muscholl, Habermehl 2004]
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
</tr>
<tr>
<td>MSO Logics</td>
</tr>
<tr>
<td>Weaker Logics</td>
</tr>
<tr>
<td><strong>Extensions</strong></td>
</tr>
<tr>
<td>Counting</td>
</tr>
<tr>
<td>Data Values</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
Data Values

Remark

- In (database) queries or constraints comparisons of data values are very common.
- Most XML theory concentrates on structure of trees instead.

Example queries/constraints

- Did we reserve a room for every participant?
  \[ \forall x \text{ Partic.Name}(x) \rightarrow \exists y \text{Room.Name}(y) \land x.\text{data} = y.\text{data} \]

- Does every participant give at most one talk?
  \[ \forall x \forall y \ [x \neq y \land \text{Talk.Speaker}(x) \land \text{Talk.Speaker}(y)] \rightarrow x.\text{data} \neq y.\text{data} \]

The setting

- We concentrate on data strings.
- Access to data values only via equality tests.

Example: data string

\[\begin{array}{cccccccccccccccc}
2 & 3 & 3 & 3 & 2 & 2 & 7 & 17 & 17 & 3 & 4 & 5 & 2 & 3 & 3 & 4 & 4 \\
c & b & c & a & a & b & b & b & c & a & b & a & c & b & a & a \\
\end{array}\]

Definition

- **Data string**: Finite sequence over \( \Sigma \times D \), where
  - \( \Sigma \) finite
  - \( D \) infinite

  Logical language:
  - \( a(x) \): Letter position \( x \) is \( a \in \Sigma \)
  - order relation \(<\), successor relation \(+1\)
  - \( \sim \): \( x \sim y \) if positions \( x \) and \( y \) have the same \( D \)-value

  \[\rightarrow \text{ Equivalence relation}\]
## Some Known Results about $\text{FO}^2$

<table>
<thead>
<tr>
<th>Over arbitrary relational structures</th>
<th>Over strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Finite model property [Mortimer 75]</td>
<td>● Satisfiability $\text{NEXPTIME}$-complete</td>
</tr>
<tr>
<td>● Satisfiability $\text{NEXPTIME}$-complete [Grädel et al. 97]</td>
<td>● Expressive power:</td>
</tr>
<tr>
<td>● On structures with 1 or 2 equivalence relations: decidable [Kieroński, Otto 05]</td>
<td>– unary LTL and $\Sigma^2 \cap \Pi^2$ [Etessami, Vardi, Wilke 97]</td>
</tr>
<tr>
<td>● On structures with 3 equivalence relations: undecidable [Kieroński, Otto 05]</td>
<td>– variety DA [Thérien, Wilke 98]</td>
</tr>
<tr>
<td>● On structures with a linear order: in $\text{coNEXPTIME}$ [Otto]</td>
<td>– two way, partially-ordered DFA [Sch., Thérien, Vollmer 01]</td>
</tr>
<tr>
<td>● On structures with several well-orderings: undecidable [Otto]</td>
<td></td>
</tr>
</tbody>
</table>
# Automata on Data Strings

## Models
- **Register-Automata**: finitely many registers for data values
- **Pebble-Automata**: finitely many pebbles with stack discipline
- **Variations**: 1-way or 2-way, Deterministic or Nondeterministic

## Example

![Example Diagram](image)

## Facts
- Non-emptiness undecidable for most models
- Exception: 1N-register automata
- This holds also for FO logic

→ [Kaminski, Francez 94; Neven, Sch., Vianu 01]
Overview of Automata Models

Inclusion Structure

- 1D-RA
- 2D-RA
- 1N-RA
- 2A-RA
- 2A-PA
- 2ID-PA
- 2ID-PA
- FO*
- MSO*
- W1D-PA
- W1N-PA
- 1N-RA
Decidability Results

**Theorem**

Satisfiability of $\text{FO}^2(+1, <)$ on data strings is decidable

**Further results**

- Emptiness of multicounter automata can be reduced to satisfiability of $\text{FO}^2(+1, <)$ on data strings
- $\text{FO}^2(<)$ on data strings is complete for $\text{NEXPTIME}$
- $\text{FO}^2(+1)$ on data strings is hard for $\text{NEXPTIME}$ (and in $\text{2NEXPTIME}$)
- $\text{FO}^3(+1)$ on data strings is undecidable

**Some Definitions**

- **Data string** $s$:

- **Class**: all positions with the same data value
- **Interval**: (maximal set of) contiguous positions of a class
- **String projection** $P(s)$: $\text{cbcaabbbcabacbbbaa}$
What $\text{FO}^2$ Can Express

Example properties

- Let $\alpha$ and $\beta$ denote unary types
- $\text{FO}^2$ can express
  - data-blind properties, i.e., properties not using $\sim$
  - All occurrences of a type $\alpha$ are in the same class:
    $$\theta = \forall x \forall y ((\alpha(x) \land \alpha(y)) \rightarrow x \sim y)$$
  - Each class contains at most one occurrence of $\alpha$:
    $$\theta = \forall x \forall y ((\alpha(x) \land \alpha(y) \land x \sim y) \rightarrow x = y)$$
  - In each class, every $\alpha$ occurs before every $\beta$:
    $$\theta = \forall x \forall y ((\alpha(x) \land \beta(y) \land x \sim y) \rightarrow x < y)$$
  - Each class with an $\alpha$ also has a $\beta$:
    $$\theta = \forall x \exists y (\alpha(x) \rightarrow (\beta(y) \land x \sim y))$$
  - It turns out: That’s basically all!
Proof Structure

Main steps of the proof

- \( F O^2 \) formula \( \varphi \)
  - Scott normal form
  - Intermediate normal form
  - Data normal form \( \psi \)

- Construct multicounter automaton \( A_\psi \) such that:
  - \( A_\psi \) accepts a string \( w \) if and only if
  - there is a data string \( s \) with
    * \( s \models \varphi \)
    * \( P(s) = w \)

- Check whether \( L(A_\psi) \neq \emptyset \)

Definition: Multicounter-automaton

- Nondeterministic string automaton (not: data string!)
- Finite number of counters
- Counter values \( \geq 0 \) (or reject)
- No intermediate test whether counter is 0
- Acceptance if finally all counters are 0

Remarks

- Closely related to Petri nets
- Non-emptiness: decidable
  \[ \text{[Sacerdote, Tenney 77]} \]
- Lower bound: \( \text{EXPSPACE} \)
- No elementary upper bound known
We transform into equivalent EMSO formulas

**Scott normal form**: \( \exists R_1, \ldots, R_k \ \forall x \forall y \chi \land \land_i \forall x \exists y \chi_i \)

**Intermediate normal form**: \( \exists R_1 \cdots R_m \theta_1 \land \cdots \land \theta_n \)

**\( \theta_i \)**:

1. \( \forall x \forall y \ (\delta(x,y) \geq 2 \land \alpha(x) \land \beta(y) \land \begin{cases} x \sim y \\ x \not\sim y \end{cases} \) \( \rightarrow \) \( \begin{array}{l} x < y \\ x > y \end{array} \)

2. \( \forall x \exists y \ \alpha(x) \rightarrow (\beta(y) \land \begin{cases} x \sim y \\ x \not\sim y \end{cases} \land \begin{array}{l} x + 1 < y \\ x + 1 = y \\ x = y \\ x = y + 1 \\ x > y + 1 \end{array} \) \)

**Note**: \( \forall \forall \) without \( +1 \)
Normalization (cont.)

Normal forms (cont.)

- **Data normal form**: Disjunction of formulas
  \[ \exists R_1 \cdots R_n R^\# \theta_1 \land \cdots \land \theta_n \]

- **\( \theta_i \)**:
  
  (a) data-blind
  
  (b) All \( \alpha \) are in the same class
  
  (c) Each class contains at most one \( \alpha \)
  
  (d) In each class, every \( \alpha \) occurs before every \( \beta \)
  
  (e) Each class with an \( \alpha \) also has a \( \beta \)
  
  (f) \( R^\# \) marks the first position of each interval:
  \[ \forall x R^\#(x) \leftrightarrow \forall y(x = y + 1 \rightarrow x \neq y) \]

Normalization steps

- \( \text{FO}^2 \rightarrow \text{Scott normal form}: \) standard

  Scott normal form
  \[ \rightarrow \text{intermediate normal form:} \]
  relatively straightforward

  Intermediate normal form
  \[ \rightarrow \text{data normal form:} \]

  - For each type \( \alpha \) we capture the two left-most classes with \( \alpha \) and the two rightmost classes with \( \alpha \) by unary relations \( R^\alpha_1, \ldots, R^\alpha_4 \)
  
  - Case distinction on possible formulas (1) and (2)
    \[ \rightarrow \) in each case \( \theta_i \) can be replaced by some “data normal” formulas
## Construction of Multicounter automaton

### Proof (cont.)

- Recall ingredients of data normal form:
  - (a) data-blind
  - (b) All $\alpha$ are in the same class
  - (c) Each class contains at most one $\alpha$
  - (d) In each class, every $\alpha$ occurs before every $\beta$
  - (e) Each class with an $\alpha$ also has a $\beta$
  - (f) $R_\#$ marks the first position of each interval:
    \[
    \forall x R_\#(x) \iff \forall y(x = y + 1 \rightarrow x \not< y)
    \]

- (a), (f): straightforward

- (c), (d), (e) induce regular conditions for each class: $L$

- (b) specifies regular conditions for some special classes: $L_1, \ldots, L_k$

- Multicounter automaton $\mathcal{A}$ accepts basically shuffle of $L, L_1, \ldots, L_k$. 
Construction of Multicounter automaton (cont.)

<table>
<thead>
<tr>
<th>Proof (cont.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• $A$ accepts string projections of models of disjunctions of formulas $\exists R_1 \cdots R_n R_# \theta_1 \land \cdots \land \theta_n$</td>
</tr>
<tr>
<td>• $A$ guesses a disjunct</td>
</tr>
<tr>
<td>• $A$ guesses, for each position $R_1, \ldots, R_n, R_#$</td>
</tr>
<tr>
<td>→ In particular: guesses intervals</td>
</tr>
<tr>
<td>• But: $A$ does not know which intervals belong to the same class</td>
</tr>
<tr>
<td>• Special classes $(L_1, \ldots, L_k)$ can be checked directly (if $\alpha$ occurs in only one class this class is $R_1^\alpha$)</td>
</tr>
<tr>
<td>• To check that all other class strings are in $L$:</td>
</tr>
<tr>
<td>– Let $B$ be a string automaton for $L$</td>
</tr>
<tr>
<td>– $A$ has one counter per state of $B$</td>
</tr>
<tr>
<td>– Counter $C_q$ counts how many (non-special) class strings seen so far led to a state $q$</td>
</tr>
<tr>
<td>– Complication: At interval border $A$ can proceed from a state $q$ that was just “reached” only if $C_q \geq 2$</td>
</tr>
<tr>
<td>Contents</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>Introduction</td>
</tr>
<tr>
<td>MSO Logics</td>
</tr>
<tr>
<td>Weaker Logics</td>
</tr>
<tr>
<td>Extensions</td>
</tr>
<tr>
<td><strong>Conclusion</strong></td>
</tr>
</tbody>
</table>
## Conclusion

### What we have seen

Logic is useful for the theory of XML languages:

- MSO offers framework for schema languages
- MSO \( \equiv \) regular node selecting queries
- Two-variable logic \( \equiv \) XPath
- FO-logic \( \equiv \) natural extension of XPath
- MSO helpful in the context of transformations
- Two-variable logic with data is decidable

### Open

There remains a lot to be done, e.g.

- XQuery
- Automata in the presence of data values
- Practical relevance of logic-automata approach?

### Finally

Thank You!