Two variable logics in the presence of an equivalence relation.

Bonn
September 2006

Thomas Schwentick

Joint work with
Mikołaj Bojańczyk, Claire David, Anca Muscholl, Luc Segoufin
Toy examples from Computer Science

<table>
<thead>
<tr>
<th>A printer and a process</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Printer, 1 Process</td>
</tr>
<tr>
<td>Process can</td>
</tr>
<tr>
<td>- stay idle</td>
</tr>
<tr>
<td>- wait to print</td>
</tr>
<tr>
<td>- print</td>
</tr>
<tr>
<td>➔ Propositions $i, w, p$</td>
</tr>
<tr>
<td>Actions of the printer:</td>
</tr>
<tr>
<td>- req: User submits print request</td>
</tr>
<tr>
<td>- beg: Start printing</td>
</tr>
<tr>
<td>- end: End printing</td>
</tr>
<tr>
<td>A property: $G(w \rightarrow Fp)$</td>
</tr>
</tbody>
</table>
A printer and a process

• 1 Printer, 1 Process
• Process can
  – stay idle
  – wait to print
  – print

→ Propositions $i, w, p$
• Actions of the printer:
  – req: User submits print request
  – beg: Start printing
  – end: End printing
• A property: $G(w \rightarrow Fp)$
Toy examples from Computer Science

A printer and a process

- 1 Printer, 1 Process
- Process can
  - stay idle
  - wait to print
  - print

→ Propositions \( i, w, p \)

- Actions of the printer:
  - req: User submits print request
  - beg: Start printing
  - end: End printing
- A property: \( G(w \rightarrow Fp) \)

A printer and two processes

\[
\begin{align*}
0 & \rightarrow req_1 \rightarrow beg_2 \\
1 & \rightarrow req_1 \rightarrow end_2 \\
2 & \rightarrow req_1 \rightarrow beg_2 \\
3 & \rightarrow req_1 \rightarrow beg_1 \\
4 & \rightarrow req_2 \rightarrow beg_2 \\
5 & \rightarrow req_2 \rightarrow beg_1 \\
6 & \rightarrow beg_1 \\
7 & \rightarrow end_2
\end{align*}
\]
A printer and a process

- 1 Printer, 1 Process
- Process can
  - stay idle
  - wait to print
  - print

→ Propositions \( i, w, p \)

- Actions of the printer:
  - req: User submits print request
  - beg: Start printing
  - end: End printing

- A property: \( G(w \rightarrow Fp) \)

A printer and two processes

\[
G[(w_1 \rightarrow Fp_1) \land (w_2 \rightarrow Fp_2)]
\]
To toy examples from Computer Science:

- **A printer and a process**
  - 1 Printer, 1 Process
  - Process can
    - stay idle
    - wait to print
    - print
  - Propositions \( i, w, p \)
  - Actions of the printer:
    - `req`: User submits print request
    - `beg`: Start printing
    - `end`: End printing
  - A property: \( G(w \rightarrow Fp) \)

- **A printer and two processes**

A flowchart illustrating the interactions between processes and the printer, with states and actions labeled accordingly.

- **Propositions**
  \( G[(w_1 \rightarrow Fp_1) \land (w_2 \rightarrow Fp_2)] \)
- **K processes**: \( 3^k \) states, growing formula size
- **Arbitrary number of processes?**

Two variable logics...

Thomas Schwentick
Model checking:
- System: $M$
- Property: $\varphi$
- Does $M$ fulfil property $\varphi$?
The Model Checking Approach to Verification

- Model checking:
  - System: $M$
  - Property: $\varphi$
  - Does $M$ fulfil property $\varphi$?

- Model a "real life" system as a transition system with finite state space
  - Abstract away data values, process numbers, ...

- Model executions of the system as infinite strings or trees

- Specify properties in a logic that allows translation into automata

- Use decidability of non-emptiness for automata to obtain decidability of model checking
The Model Checking Approach to Verification

- Model checking:
  - System: $M$
  - Property: $\varphi$
  - Does $M$ fulfil property $\varphi$?

- Model a "real life" system as a transition system with finite state space
  - Abstract away data values, process numbers,...

- Model executions of the system as infinite strings or trees

- Specify properties in a logic that allows translation into automata

- Use decidability of non-emptiness for automata to obtain decidability of model checking

- But sometimes this approach does not work, e.g., if number of processes is not bounded a priori
Example Document

<Composer>
  <Name>Claude Debussy</Name>
</Composer>

<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

Composer

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
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
Four important kinds of XML processing and their languages

**Validation**
- Check whether an XML document is of a given type
  - DTD, XML Schema

**Navigation**
- Select a set of positions in an XML document
  - XPath

**Querying**
- Extract information from an XML document
  - XQuery

**Transformation**
- Construct a new XML document from a given one
  - XSLT
### Example Document

```
⟨a⟩⟨b⟩
  ⟨d⟩12⟨/d⟩⟨e⟩22⟨/e⟩
⟩
⟨b⟩
⟨b⟩⟨d⟩4⟨/d⟩⟨/b⟩
⟨b⟩⟨d⟩11⟨/d⟩
  ⟨e⟩8⟨/e⟩18⟨e⟩5⟨/e⟩
⟩
⟨b⟩
⟨c⟩⟨b⟩⟨f⟩2⟨/f⟩
⟩
⟨b⟩7⟨/c⟩
⟨/a⟩
```
XML as Tree

Example Document

⟨a⟩⟨b⟩
  ⟨d⟩12⟨/d⟩⟨e⟩22⟨/e⟩
  ⟨/b⟩
⟨b⟩⟨d⟩4⟨/d⟩⟨/b⟩
⟨b⟩⟨d⟩11⟨/d⟩
  ⟨e⟩8⟨/e⟩18⟨e⟩5⟨/e⟩
  ⟨/b⟩18⟨e⟩5⟨/e⟩
  ⟨/b⟩
⟨c⟩⟨b⟩⟨f⟩2⟨/f⟩
  ⟨/b⟩7⟨/c⟩
⟨/a⟩

...as Unranked Tree

<table>
<thead>
<tr>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>12</td>
<td>22</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
XML as Tree

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

- For many investigations, data values can and have to be ignored
  ➞ Validation, Navigation
For many investigations, data values can and have to be ignored

→ Validation, Navigation
For many investigations, data values can and have to be ignored.

Validation, Navigation

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.

XML as Tree
Many tasks concentrate on structure of XML documents and disregard data values

→ Model XML documents as unranked trees with a finite set of labels

→ Validation can be modelled by tree automata

→ Transformation can be modelled by tree transducers

→ Make use of connection between FO or MSO logic and automata

→ Decidable static analysis
• Many tasks concentrate on structure of XML documents and disregard data values

→ Model XML documents as unranked trees with a finite set of labels

→ Validation can be modelled by tree automata

→ Transformation can be modelled by tree transducers

• Make use of connection between FO or MSO logic and automata

→ Decidable static analysis

• But sometimes this approach does not work, e.g., key constraints can not be modelled
Summing up

- In verification and XML processing, finite alphabet strings and trees are used as approximations for the "real world"
- This is sufficient for many applications
- It allows to exploit the logic-automata connection to obtain decidable Model Checking and static analysis
In verification and XML processing, finite alphabet strings and trees are used as approximations for the "real world"

- This is sufficient for many applications
- It allows to exploit the logic-automata connection to obtain decidable Model Checking and static analysis

- Sometimes this modelling becomes messy
Summing up

- In verification and XML processing, finite alphabet strings and trees are used as approximations for the "real world"
- This is sufficient for many applications
- It allows to exploit the logic-automata connection to obtain decidable Model Checking and static analysis
- Sometimes this modelling becomes messy
- Sometimes it is simply not good enough
In verification and XML processing, finite alphabet strings and trees are used as approximations for the "real world"

This is sufficient for many applications

It allows to exploit the logic-automata connection to obtain decidable Model Checking and static analysis

Sometimes this modelling becomes messy

Sometimes it is simply not good enough

Goal of this research: Find logics and automata that allow for decidability even if data values are present
In verification and XML processing, finite alphabet strings and trees are used as approximations for the "real world". This is sufficient for many applications. It allows to exploit the logic-automata connection to obtain decidable Model Checking and static analysis. Sometimes this modelling becomes messy. Sometimes it is simply not good enough. Goal of this research: Find logics and automata that allow for decidability even if data values are present. Here: consider classical logics.


Preliminaries

Background on two variable logics
Two variable logics on data strings
Two variable logics on data trees
Conclusion
### Example: data string

<table>
<thead>
<tr>
<th>17</th>
<th>5</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>2</th>
<th>2</th>
<th>7</th>
<th>17</th>
<th>17</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>4</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

### Definition

- **Data string**: Finite sequence over $\Sigma \times D$, where
  - $\Sigma$ finite
  - $D$ infinite
- **Our 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
  - Equivalence relation
### Our setting

- **Example: data string**

<table>
<thead>
<tr>
<th>17</th>
<th>5</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>2</th>
<th>2</th>
<th>7</th>
<th>17</th>
<th>17</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>4</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

- **Definition**

  - **Data string**: Finite sequence over $\Sigma \times D$, where
    - $\Sigma$ finite
    - $D$ infinite
  - **Our 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
  - Equivalence relation

- **Definition**

  - **Data tree**: Tree with labels from $\Sigma$ and one data value from $D$ per node
  - **Logical language:**
    - $\sim$: as for strings
    - $E\rightarrow$: horizontal neighbor ("next sibling")
    - $E\downarrow$: parent-child
    - $E\Rightarrow$, $E\downarrow$: the transitive closures of $E\rightarrow$ and $E\downarrow$
Our setting

**Example: data string**

<table>
<thead>
<tr>
<th>17</th>
<th>5</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>2</th>
<th>2</th>
<th>7</th>
<th>17</th>
<th>17</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>4</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

**Definition**

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

- **Our 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
  - $\rightarrow$: Equivalence relation

- **Data tree**: Tree with labels from $\Sigma$ and one data value from $D$ per node

- **Logical language**:
  - $\sim$: as for strings
  - $E_{\rightarrow}$: horizontal neighbor (”next sibling”)
  - $E_{\downarrow}$: parent-child
  - $E_{\Rightarrow}, E_{\downarrow}$: the transitive closures of $E_{\rightarrow}$ and $E_{\downarrow}$

Note: no other operations on data values, in particular no arithmetic
A first attempt

- We know:
  - First-order logic is undecidable in general
  - First-order logic is decidable on strings
- What about First-order logic on data strings?
A first attempt

- We know:
  - First-order logic is undecidable in general
  - First-order logic is decidable on strings
- What about First-order logic on data strings?

**Theorem**

Satisfiability of First-Order formulas on data strings is undecidable, even for formulas with 3 variables

**Proof idea**

- Reduction from PCP:
  - Given: \((u_1, v_1), \ldots, (u_k, v_k)\), pairs of strings
  - Question: is there a sequence \(i_1, \ldots, i_n\) such that
    \[ u_{i_1} \cdots u_{i_n} = v_{i_1} \cdots v_{i_n} \]
A first attempt

- We know:
  - First-order logic is undecidable in general
  - First-order logic is decidable on strings
- What about First-order logic on data strings?

**Theorem**

Satisfiability of First-Order formulas on data strings is undecidable, even for formulas with 3 variables

**Proof idea**

- Reduction from PCP:
  - Given: \((u_1, v_1), \ldots, (u_k, v_k)\), pairs of strings
  - Question: is there a sequence \(i_1, \ldots, i_n\) such that 
    \[ u_{i_1} \cdots u_{i_n} = v_{i_1} \cdots v_{i_n} \]?

- Encode solution candidates as data strings over \(\{a, b, \#, 1, \ldots, n\}\) of the form 
  \[ u \# v \]

- Each occurrence of a \(u_i\) is prefixed by \(i\), 
i.e.:
  If \(u_1 = aba\) and \(u_2 = bb\) then 121 induces 1aba2bb1aba

- Each data value occurs exactly twice, once in \(u\) and once in \(v\)

- Corresponding positions should have the same data value (and same number/symbol)

- Crucial: check that sequence of data values is the same on both sides for number positions and letter positions

- Important subformula:
  \[
  x \sim y \rightarrow \exists z \ (x + 1 = z \land \\
  \exists x \ y + 1 = x \land z \sim x)
  \]

"if \(x\) and \(y\) are equivalent then their right neighbors are also equivalent"
Two variables: a useful restriction?

- Classical approach: restrict to 2 variables
- Does this give us anything useful?
Two variables: a useful restriction?

- Classical approach: restrict to 2 variables
- Does this give us anything useful?
  1. We do not have free choice...
  2. We can express a lot of useful things with only two variables
Two variables: a useful restriction?

- Classical approach: restrict to 2 variables
- Does this give us anything useful?
  1. We do not have free choice...
  2. We can express a lot of useful things with only two variables

...in the verification context

- Whenever a print is requested by a process, it is granted at some point:
  \[ \forall x \exists y \, \text{req}(x) \rightarrow (x < y \land \text{beg}(y) \land x \sim y) \]
- No printer job is requested twice:
  \[ \forall x \forall y \, (\text{req}(x) \land \text{req}(y) \land x \sim y) \rightarrow x = y \]
**Two variables: a useful restriction?**

- Classical approach: restrict to 2 variables
- Does this give us anything useful?
  1. We do not have free choice...
  2. We can express a lot of useful things with only two variables

...in the verification context

- Whenever a print is requested by a process, it is granted at some point:
  \[ \forall x \exists y \rightreq(x) \rightarrow (x < y \land \text{beg}(y) \land x \sim y) \]
- No printer job is requested twice:
  \[ \forall x \forall y (\rightreq(x) \land \rightreq(y) \land x \sim y) \rightarrow x = y \]

...in the XML context

- With 2 variables many integrity constraints can be expressed:
  - Did we reserve a room for every participant?
  - \[ \forall x \rightpartic.name(x) \rightarrow \exists y \rightroom.name(y) \land x.data = y.data \]
  - Does every participant give at most one talk?
  - \[ \forall x \forall y [\righttalk.speaker(x) \land \righttalk.speaker(y)] \rightarrow x.data = y.data \]
Two variables: a useful restriction?

- Classical approach: restrict to 2 variables
- Does this give us anything useful?
  1. We do not have free choice...
  2. We can express a lot of useful things with only two variables

...in the verification context

- Whenever a print is requested by a process, it is granted at some point:
  \( \forall x \exists y \text{req}(x) \rightarrow (x < y \land \text{beg}(y) \land x \sim y) \)
- No printer job is requested twice:
  \( \forall x \forall y (\text{req}(x) \land \text{req}(y) \land x \sim y) \rightarrow x = y \)

...in the XML context

- With 2 variables many integrity constraints can be expressed:
  - Did we reserve a room for every participant?
  - \( \forall x \text{Partic.Name}(x) \rightarrow \exists y \text{Room.Name}(y) \land x \cdot \text{data} = y \cdot \text{data} \)
  - Does every participant give at most one talk?
  - \( \forall x \forall y [\text{Talk.Speaker}(x) \land \text{Talk.Speaker}(y)] \rightarrow x \cdot \text{data} = y \cdot \text{data} \)
- Close relationship with navigation: XPath
- Regular languages require only two variables (plus ex. quantification of sets)
Preliminaries

Background on two variable logics

Two variable logics on data strings
Two variable logics on data trees
Conclusion
Some Known results about Two-Variable Logics

- Two-variable logics has the finite model property [Mortimer 75]
- It actually has the exponential-size model property [Grädel, Otto 97]

→ Satisfiability in **NEXPTIME**
Some Known results about Two-Variable Logics

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>♦ Two-variable logics has the finite model property [Mortimer 75]</td>
</tr>
<tr>
<td>♦ It actually has the exponential-size model property [Grädel, Otto 97]</td>
</tr>
</tbody>
</table>

→ Satisfiability in **NEXPTIME**

Proof idea: Step 1

- **Scott Normal Form**: With respect to satisfiability each $\text{FO}^2$-formula $\varphi$ is equivalent to a formula $\varphi'$ of the form:
  \[
  \forall x \forall y \chi \wedge \bigwedge_i \forall x \exists y \chi_i
  \]
  $\chi, \chi_i$ quantifier-free

- $\varphi'$ might have enlarged signature: more unary relations

- $\varphi'$ can be computed in polynomial time
Some Known results about Two-Variable Logics

- Two-variable logics has the finite model property [Mortimer 75]
- It actually has the exponential-size model property [Grädel, Otto 97]

→ Satisfiability in **NEXPTIME**

**Proof idea: Step 1**

- **Scott Normal Form**:
  - With respect to satisfiability each $\text{FO}^2$-formula $\varphi$ is equivalent to a formula $\varphi'$ of the form:
    $$\forall x \forall y \chi \land \bigwedge_i \forall x \exists y \chi_i$$
  - $\chi, \chi_i$ quantifier-free
  - $\varphi'$ might have enlarged signature: more unary relations
  - $\varphi'$ can be computed in polynomial time

**Proof idea: Step 2**

- Assume there is a model $\mathcal{A}$ of $\varphi'$
  -> Construct a model $\mathcal{B}$ for $\varphi'$ of exp. size
- Consider basic 1-types and 2-types
- **King**: 1-type that occurs exactly once
- Construct $\mathcal{B}$ as follows:
  - Take the kings
  - Take $\chi_i$-witnesses for the kings (Court)
  - Take three sets of witnesses and combine them in a circular fashion

(Proof and picture: [Grädel, Otto 97])
• Satisfiability of $\mathbf{FO}^2$ decidable

• Is this applicable to our setting?

• Unfortunately not: on structures with particular properties the situation is more complicated
Is this the end of the story?

- Satisfiability of $\text{FO}^2$ decidable
- Is this applicable to our setting?
- Unfortunately not: on structures with particular properties the situation is more complicated

More Results on general structures

- On structures with 1 or 2 equivalence relations: decidable [Kieroński, Otto 05]
- On structures with 3 equivalence relations: undecidable [Kieroński, Otto 05]
- On structures with a linear order: in $\text{coNEXPTIME}$ [Otto]
- On structures with several well-orderings: undecidable [Otto]
Is this the end of the story?

Satisfiability of $\mathbf{FO}^2$ decidable

Is this applicable to our setting?

Unfortunately not: on structures with particular properties the situation is more complicated

More Results on general structures

On structures with 1 or 2 equivalence relations: decidable [Kieroński, Otto 05]

On structures with 3 equivalence relations: undecidable [Kieroński, Otto 05]

On structures with a linear order: in $\text{coNEXPTIME}$ [Otto]

On structures with several well-orderings: undecidable [Otto]

...on strings without data

Satisfiability $\text{NEXPTIME}$-complete

Expressive power:

- unary LTL and $\Sigma^2 \cap \Pi^2$ [Etessami, Vardi, Wilke 97]
- variety DA [Thérien, Wilke 98]
Is this the end of the story?

- Satisfiability of $\text{FO}^2$ decidable
- Is this applicable to our setting?
- Unfortunately not: on structures with particular properties the situation is more complicated

More Results on general structures

- On structures with 1 or 2 equivalence relations: decidable [Kieroński, Otto 05]
- On structures with 3 equivalence relations: undecidable [Kieroński, Otto 05]
- On structures with a linear order: in $\text{coNEXPTIME}$ [Otto]
- On structures with several well-orderings: undecidable [Otto]

...on strings without data

- Satisfiability $\text{NEXPTIME}$-complete
- Expressive power:
  - unary LTL and $\Sigma^2 \cap \Pi^2$ [Etessami, Vardi, Wilke 97]
  - variety DA [Thérien, Wilke 98]

More known results

- Regular string languages: EMSO with two FO-variables (in the presence of $\bot$)
- Regular tree languages: EMSO with two FO-variables (in the presence of $E_{\rightarrow}, E_{\downarrow}$)
Preliminaries
Background on two variable logics

Two variable logics on data strings
Two variable logics on data trees
Conclusion
Some notation

**Some Definitions**

- **Data string** $s$:

  $17 \ a \ b \ c \ a \ a \ b \ b \ c \ a \ a \ b \ b \ 17 \ a \ b \ c \ a \ b \ a \ a \ 17 \ a \ b \ 17 \ a \ b \ 17 \ a \ b \ a \ b \ 17 \ a \ b \ b \ a \ a$

- **Class**: all positions with the same data value

- **Interval**: (maximal set of) contiguous positions of a class

- **String projection** $P(s)$:

- **type** $\alpha(x)$: any conjunction of unary literals
Some notation

Some Definitions

• Data string $s$:

- Class: all positions with the same data value
- Interval: (maximal set of) contiguous positions of a class
- String projection $P(s)$:
- type $\alpha(x)$: any conjunction of unary literals
Some notation

- **Data string** $s$:

- **Class**: all positions with the same data value
- **Interval**: (maximal set of) contiguous positions of a class
- **String projection** $P(s)$: abcbaabbbcabacbbaa
- **type** $\alpha(x)$: any conjunction of unary literals
On the expressive power of $\text{FO}^2$ on data strings

Example

- $\varphi_a$:
  - $\varphi_a = \forall x \forall y (x \neq y \land a(x) \land a(y)) \rightarrow x \not\sim y$
  - all $a$’s are in different classes

- Similarly: $\varphi_b$

- $\psi_{a,b}$:
  - $\psi_{a,b} = \forall x \exists y (a(x) \rightarrow (b(y) \land x \sim y))$
  - each class with an $a$ also contains a $b$

- Similarly: $\psi_{b,a}$. 
On the expressive power of \( \text{FO}^2 \) on data strings

**Example**

- **\( \varphi_a \):**
  
  \[
  \varphi_a = \forall x \forall y (x \neq y \land a(x) \land a(y)) \rightarrow x \not\sim y
  \]

  - all \( a \)'s are in different classes

- **Similarly: \( \varphi_b \)**

- **\( \psi_{a,b} \):**
  
  \[
  \psi_{a,b} = \forall x \exists y (a(x) \rightarrow (b(y) \land x \sim y))
  \]

  - each class with an \( a \) also contains a \( b \)

- **Similarly: \( \psi_{b,a} \).**

- **Thus:**
  
  \[
  \varphi = \varphi_a \land \varphi_b \land \psi_{a,b} \land \psi_{b,a}
  \]

  - the numbers of \( a \) and \( b \)-labeled positions are equal

- **In a similar fashion:** number of \( a \)'s, \( b \)'s and \( c \)'s are equal

\( \rightarrow \) non-regular language
More example properties

- Let $\alpha$ and $\beta$ denote unary types
- $\text{FO}^2$ can express
  - 
  - 
  - 
  - 
  - 
  - 
  -
On the expressive power of $\text{FO}^2$ on data strings (cont.)

<table>
<thead>
<tr>
<th>More example properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Let $\alpha$ and $\beta$ denote unary types</td>
</tr>
<tr>
<td>• $\text{FO}^2$ can express</td>
</tr>
<tr>
<td>- data-blind properties, i.e., properties not using $\sim$</td>
</tr>
</tbody>
</table>
More 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:
    \[ \theta = \forall x \forall y \left( (\alpha(x) \land \alpha(y)) \rightarrow x \sim y \right) \]
More 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:
    
    $$\theta = \forall x \forall y \left( (\alpha(x) \land \alpha(y)) \rightarrow x \sim y \right)$$

  - Each class contains at most one occurrence of $\alpha$:
    
    $$\theta = \forall x \forall y \left( (\alpha(x) \land \alpha(y) \land x \sim y) \rightarrow x = y \right)$$
More 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:
    \[
    \theta = \forall x \forall y \left((\alpha(x) \land \alpha(y)) \rightarrow x \sim y\right)
    \]
  - Each class contains at most one occurrence of \( \alpha \):
    \[
    \theta = \forall x \forall y \left((\alpha(x) \land \alpha(y) \land x \sim y) \rightarrow x = y\right)
    \]
  - In each class, every \( \alpha \) occurs before every \( \beta \):
    \[
    \theta = \forall x \forall y \left((\alpha(x) \land \beta(y) \land x \sim y) \rightarrow x < y\right)
    \]
More example properties

- Let $\alpha$ and $\beta$ denote unary types

- $\text{FO}^2$ can express
  
  - \textbf{data-blind} properties, i.e., properties not using $\sim$
    
    - All occurrences of a type $\alpha$ are in the same class:
      
      \[ \theta = \forall x \forall y \left( (\alpha(x) \land \alpha(y)) \rightarrow x \sim y \right) \]
    
    - Each class contains at most one occurrence of $\alpha$:
      
      \[ \theta = \forall x \forall y \left( (\alpha(x) \land \alpha(y) \land x \sim y) \rightarrow x = y \right) \]
    
    - In each class, every $\alpha$ occurs before every $\beta$:
      
      \[ \theta = \forall x \forall y \left( (\alpha(x) \land \beta(y) \land x \sim y) \rightarrow x < y \right) \]
    
    - Each class with an $\alpha$ also has a $\beta$:
      
      \[ \theta = \forall x \exists y \left( \alpha(x) \rightarrow (\beta(y) \land x \sim y) \right) \]
More 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:
    \[
    \theta = \forall x \forall y \left( (\alpha(x) \land \alpha(y)) \rightarrow x \sim y \right)
    \]
  - Each class contains at most one occurrence of $\alpha$:
    \[
    \theta = \forall x \forall y \left( (\alpha(x) \land \alpha(y) \land x \sim y) \rightarrow x = y \right)
    \]
  - In each class, every $\alpha$ occurs before every $\beta$:
    \[
    \theta = \forall x \forall y \left( (\alpha(x) \land \beta(y) \land x \sim y) \rightarrow x < y \right)
    \]
  - Each class with an $\alpha$ also has a $\beta$:
    \[
    \theta = \forall x \exists y \left( \alpha(x) \rightarrow (\beta(y) \land x \sim y) \right)
    \]
- That’s all!
### Main result regarding data strings

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satisfiability of $\text{FO}^2(+1, \lt)$ on data strings is decidable</td>
</tr>
</tbody>
</table>
Main result regarding data strings

**Theorem**

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

**Main steps of the proof**

- $\text{FO}^2$ formula $\varphi$
- Scott normal form
- Intermediate normal form
- Data normal form $\psi$
- Data automaton $D_\psi$
- Multicounter automaton $A_\psi$ (on strings)

- Finally: $A_\psi$ accepts a string $w$ if and only if there is a data string $s$ with
  - $s \models \varphi$
  - $P(s) = w$
- Thus: $\psi$ satisfiable $\iff L(A_\psi) \neq \emptyset$
Steps 1 and 2

Scott and intermediate normal form

• We transform into satisfiability 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**: 
    \[
    \exists R_1 \cdot \cdot \cdot R_m \theta_1 \land \cdot \cdot \cdot \land \theta_n
    \]

• \( \theta_i \):

  (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{cases} 
    x < y \\
    x > y
  \end{cases} \)

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

  (3) \( \forall x \forall y \ \theta \) (quantifier-free, DNF, no \( \sim \))

• Both steps are straightforward
## Step 3

### Data normal form

- **Data normal form**: Disjunction of formulas

\[ \exists R_1 \cdots R_n \, \theta_1 \land \cdots \land \theta_n \]

- **\( \theta_i \)**:
  1. Data-blind
  2. All \( \alpha \) are in the same class
  3. Each class contains at most one \( \alpha \)
  4. In each class, every \( \alpha \) occurs before every \( \beta \)
  5. Each class with an \( \alpha \) also has a \( \beta \)
  6. If \( x \) is in a different class than its successor it has type \( \alpha \)
### Data normal form

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

- **\( \theta_i \)**:
  
  - (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) If \( x \) is in a different class than its successor it has type \( \alpha \)

- For each type \( \alpha \) we capture the two left-most classes with \( \alpha \) and the two rightmost classes with \( \alpha \) by unary relations \( R_1^\alpha, \ldots, R_4^\alpha \)

- Case distinction on possible formulas (1) and (2)

  → in each case \( \theta_i \) can be replaced by some “data normal” formulas
Step 4

Data automata

- A data automaton $D = (A, B)$ consists of a
  - non-deterministic letter-to-letter string transducer $A$ (base automaton) and a
  - string automaton $B$ (class automaton)
- A data word $w$ is accepted by $D$ if
  - $A$ on input $P(w)$ accepts with some output $b_1 \cdots b_n$ such that
  - for each class $\{x_1, \ldots, x_k, \ldots, \subseteq\}\{1, \ldots, n, \ldots\}$, $x_1 < \cdots < x_k$, $B$ accepts $b_{x_1}, \ldots, b_{x_k}$.
Step 4

Data automata

- A **data automaton** $D = (A, B)$ consists of a
  - non-deterministic letter-to-letter string transducer $A$ (base automaton) and a
  - string automaton $B$ (class automaton)

- A data word $w$ is **accepted** by $D$ if
  - $A$ on input $P(w)$ accepts with some output $b_1 \cdots b_n$ such that
  - for each class $\{x_1, \ldots, x_k, \ldots, \subseteq\}\{1, \ldots, n, \ldots, \}$, $x_1 < \cdots < x_k$, $B$ accepts $b_{x_1}, \ldots, b_{x_k}$.

---

**Proposition**

For each two-variable formula over data strings there is an equivalent data automaton.
A multicounter-automaton is a finite automaton with counters $C_1, \ldots, C_k$.

Counters can be incremented and decremented (if a counter is zero and decremented, computation stops).

A run is accepting, if at the end, all counters are zero.

Theorem [Mayr 84, Kosaraju 84]

Emptiness for multicounter automata is decidable (complexity-wise equivalent to Petri net reachability).
### Step 5

#### Multicounter-automata

- A **multicounter-automaton** is a finite automaton with counters $C_1, \ldots, C_k$.
- Counters can be incremented and decremented (if a counter is zero and decremented, computation stops).
- A run is accepting, if at the end, all counters are zero.

**Theorem [Mayr 84, Kosaraju 84]**

Emptiness for multicounter automata is decidable (complexity-wise equivalent to Petri net reachability).

#### Proposition

For each data automaton $D = (A, B)$ there is a multicounter automaton $M$ such that $L(M) = P(L(D))$.

**Proof idea**

- Plan: $M$ checks whether a given string $w$ can be extended to a data string $w'$ accepted by $D$.
- Clearly: $M$ can simulate transducer $A$ (which does not see the data anyway).
- Idea: $M$ uses one counter $C_p$, for each state $p$ of $B$.
- After reading a prefix $u$ of $v$, counter $C_p$ tells in how many class strings state $p$ is reached.
- At the end: all $C_p$ for non-accepting $p$ must be zero.
Related results

- Complexity of reductions:

Satisfiability for $\text{FO}(\leq, +1)$ on data words

$\text{2NEXPTIME} \downarrow$

Emptiness for data automata

$\text{PTIME} \downarrow$

Reachability for Petri nets
Related results

- Complexity of reductions:
  - Satisfiability for $\text{FO}(\lt, +1)$ on data words
    - $2\text{NEXPTIME} \equiv \text{PTIME}$
  - Emptiness for data automata
    - $\text{PTIME} \equiv \text{PTIME}$
  - Reachability for Petri nets
    - $\text{PTIME} \equiv \text{PTIME}$
Complexity of reductions:

- Satisfiability for $\text{FO}(\langle, +1)\rangle$ on data words:
  - 2NEXPTIME \uparrow \downarrow PTIME
- Emptiness for data automata:
  - PTIME \uparrow \downarrow PTIME
- Reachability for Petri nets

Complexity of Satisfiability for $\text{FO}(\langle, +1)\rangle$ on data words closely related to the unknown complexity of Reachability for Petri nets
(no elementary upper bound known)
Related results

- Complexity of reductions:
  - Satisfiability for \( \text{FO}(\prec, +1) \) on data words
    - \( 2\text{NEXPPTIME} \uparrow \text{PTIME} \)
    - Emptiness for data automata
    - \( \text{PTIME} \uparrow \text{PTIME} \)
    - Reachability for Petri nets
  - Complexity of Satisfiability for \( \text{FO}(\prec, +1) \) on data words closely related to the unknown complexity of Reachability for Petri nets
    (no elementary upper bound known)

- **Class successor**: \( i \equiv 1 = j \) iff
  - \( i < j, i \sim j \), and
  - no \( i < k < j, i \sim k \)
Related results

- Complexity of reductions:
  - Satisfiability for $\mathbf{FO}(<, +1)$ on data words
    
    $\mathbf{2NEXPTIME} \uparrow \mathbf{PTIME}$
  - Emptiness for data automata
    
    $\mathbf{PTIME} \uparrow \mathbf{PTIME}$
  - Reachability for Petri nets

  Complexity of Satisfiability for $\mathbf{FO}(<, +1)$ on data words closely related to the unknown complexity of Reachability for Petri nets (no elementary upper bound known)

- **Class successor**: $i \pm 1 = j$ iff
  - $i < j, i \sim j$, and
  - no $i < k < j, i \sim k$

**Proposition**

A language is recognizable by a data automaton iff it is definable in $\mathbf{EMSO}^2(\sim, <, +1, \pm 1)$
Related results

- Complexity of reductions:

  - Satisfiability for $\text{FO}(<, +1)$ on data words
    
    $2\text{NEXPTIME} \uparrow \downarrow \text{PTIME}$
  
  - Emptiness for data automata
    
    $\text{PTIME} \uparrow \downarrow \text{PTIME}$
  
  - Reachability for Petri nets

  Complexity of Satisfiability for $\text{FO}(<, +1)$ on data words closely related to the unknown complexity of Reachability for Petri nets (no elementary upper bound known)

- Proposition

  A language is recognizable by a data automaton iff it is definable in $\text{EMSO}^2(\sim, <, +1, \pm 1)$

- Proposition

  - Satisfiability of $\text{FO}^2(\sim, <)$ on data words is $\text{NEXPTIME}$-complete
  
  - Satisfiability of $\text{FO}^2(\sim, +1)$ on data words is $2\text{NEXPTIME}$-complete
Related results

<table>
<thead>
<tr>
<th>Proposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Satisfiability of $\text{FO}^2(\sim, &lt;)$ on data words is $\text{NEXPTIME}$-complete</td>
</tr>
<tr>
<td>- Satisfiability of $\text{FO}^2(\sim, +1)$ on data words is $2^{-\text{NEXPTIME}}$-complete</td>
</tr>
<tr>
<td>- Decidability remains if predicates $+2, +3, \ldots$ are added</td>
</tr>
<tr>
<td>- Results generalize to $\omega$-strings</td>
</tr>
</tbody>
</table>

- **Complexity of reductions:**
  - Satisfiability for $\text{FO}(<, +1)$ on data words
    - $2\text{NEXPTIME} \Downarrow \text{PTIME}$
  - Emptiness for data automata
    - $\text{PTIME} \Downarrow \text{PTIME}$
  - Reachability for Petri nets
    - Complexity of Satisfiability for $\text{FO}(<, +1)$ on data words closely related to the unknown complexity of Reachability for Petri nets (no elementary upper bound known)

- **Class successor:** $i \not\equiv 1 = j$ iff
  - $i < j, i \sim j$, and
  - no $i < k < j, i \sim k$

- **Proposition:**
  A language is recognizable by a data automaton iff it is definable in $\text{EMSO}^2(\sim, <, +1, \pm 1)$

Two variable logics... Thomas Schwentick Slide 24
### Related results

- **Complexity of reductions:**
  - Satisfiability for $\mathbf{FO}(\prec, +1)$ on data words:
    - $2\text{NEXPTIME} \downarrow \text{PTIME}$
  - Emptiness for data automata:
    - $\text{PTIME} \downarrow \text{PTIME}$
  - Reachability for Petri nets
  
  Complexity of Satisfiability for $\mathbf{FO}(\prec, +1)$ on data words closely related to the unknown complexity of Reachability for Petri nets (no elementary upper bound known)

- **Class successor:** $i \sim 1 = j$ iff
  - $i < j$, $i \sim j$, and
  - no $i < k < j$, $i \sim k$

- **Proposition:**
  A language is recognizable by a data automaton iff it is definable in $\mathbf{EMSO}^2(\sim, \prec, +1, \pm 1)$

### Proposition

- Satisfiability of $\mathbf{FO}^2(\sim, \prec)$ on data words is $\text{NEXPTIME}$-complete

- Satisfiability of $\mathbf{FO}^2(\sim, +1)$ on data words is $2^2 \text{NEXPTIME}$-complete

- Decidability remains if predicates $+2$, $+3$, ... are added

- Results generalize to $\omega$-strings

- Undecidable, e.g. with:
  - 3 variables,
  - 2 equivalence classes, or
  - linear order on data values
### Related results

#### Complexity of reductions:

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satisfiability for $\text{FO}(&lt;, +1)$ on data words</td>
<td>(2\text{NEXPTIME} \leftrightarrow \text{PTIME})</td>
</tr>
<tr>
<td>Emptiness for data automata</td>
<td>(\text{PTIME} \leftrightarrow \text{PTIME})</td>
</tr>
<tr>
<td>Reachability for Petri nets</td>
<td></td>
</tr>
</tbody>
</table>

→ Complexity of Satisfiability for $\text{FO}(<, +1)$ on data words closely related to the unknown complexity of Reachability for Petri nets (no elementary upper bound known)

- **Class successor**: \(i \overset{\pm 1}{=} j\) iff
  - \(i < j, i \sim j\), and
  - no \(i < k < j, i \sim k\)

#### Proposition

- A language is recognizable by a data automaton iff it is definable in $\text{EMSO}^2(\sim, <, +1, \pm 1)$

#### Proposition

- Satisfiability of $\text{FO}^2(\sim, <)$ on data words is $\text{NEXPTIME}$-complete
- Satisfiability of $\text{FO}^2(\sim, +1)$ on data words is $2^{-\text{NEXPTIME}}$-complete
- Decidability remains if predicates $+2, +3, \ldots$ are added
- Results generalize to $\omega$-strings

#### Undecidable, e.g. with:

- 3 variables,
- 2 equivalence classes, or
- linear order on data values

- Temporal logics for data strings:
  - Freeze operator [Demri, Lazić 06]
  - ongoing work: other data extensions of temporal logics
A related issue

- Regular string and tree languages play a key role for decidability results
- Natural question: Is there a sensible notion of regular languages for data strings?
A related issue

- Regular string and tree languages play a key role for decidability results
- Natural question: Is there a sensible notion of regular languages for data strings?
  - Register automata [Kaminski, Francez 94]
  - Regular expressions [Kaminski, Tan 04]
  - Pebble automata [Neven, Sch., Vianu 01]
- General picture: most models pairwise different and/or undecidable
Inclusion structure of Automata Models

- MSO*
- 2A-RA
- 2N-RA
- 2D-RA
- 1N-RA
- 1D-RA
- FO*
- DatA
- W1D-PA
- W1N-PA
- S1N-PA
- S1D-PA
- 2N-PA
- 2D-PA

Two variable logics... Thomas Schwentick
Regular data languages?

- Maybe there is not THE class of regular data languages
- But: the class of languages accepted by data automata is pretty strong and robust
Regular data languages?

• Maybe there is not THE class of regular data languages
• But: the class of languages accepted by data automata is pretty strong and robust
  
  A different view: just a finite automaton reading \( P(w) \), but transitions depend on
  – state at left neighbor, and
  – state at class predecessor

→ allows for deterministic automata model (weaker than non-deterministic)

• (joint work with Henrik Björklund)
Contents

Preliminaries
Background on two variable logics
Two variable logics on data strings

Two variable logics on data trees
Conclusion
Two variable logics...
13. Jahrestagung der GI-Fachgruppe Logik in der Informatik
Dortmund, 12.-13. Oktober 2006

Invited speaker:
Anuj Dawar

More infos:
loginf.cs.uni-dortmund.de
## Data trees: notation

### Definition

- **Data tree**: Tree with labels from $\Sigma$ and one data value from $D$ per node

- **Logical language**:
  - $\sim$: as for strings
  - $E_{\rightarrow}$: horizontal neighbor ("next sibling")
  - $E_{\downarrow}$: parent-child
  - $E_{\Rightarrow}, E_{\Downarrow}$: the transitive closures of $E_{\rightarrow}$ and $E_{\downarrow}$

- **$+1$**: $E_{\rightarrow}$ and $E_{\downarrow}$

- **$<$**: $E_{\Rightarrow}$ and $E_{\Downarrow}$

- **$\text{FO}^2(\sim, +1)$**: only next sibling and child

- **$\text{FO}^2(\sim, +1, <)$**: also: descendant and following sibling
Example

Composer

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

Two variable logics...
XML integrity constraints

- **Key constraint**: \( \tau[X] \rightarrow \tau \)
  \( X \)-attributes determine a node

- **Inclusion constraint**: \( \tau[X] \subseteq \tau'[Y] \)
  \( X \)-values of \( \tau \)-nodes occur as \( Y \)-values of \( \tau' \)-nodes

- Important reasoning tasks:
  - Consistency of integrity constraints
  - Implication of integrity constraints

- Unary key constraints and inclusion constraints can be expressed with 2 variables

\[ \rightarrow \] From our decidability results we can conclude a wide range of decidability results for integrity constraints

- Related work: [Arenas, Fan, Libkin 05; Buneman et al., 03]
**XML integrity constraints**

- **Key constraint**: \( \tau[X] \rightarrow \tau \)
  - \( X \)-attributes determine a node

- **Inclusion constraint**: \( \tau[X] \subseteq \tau'[Y] \)
  - \( X \)-values of \( \tau \)-nodes occur as \( Y \)-values of \( \tau' \)-nodes

- **Important reasoning tasks**:
  - Consistency of integrity constraints
  - Implication of integrity constraints

- **Unary key constraints and inclusion constraints** can be expressed with 2 variables

- **From our decidability results we can conclude a wide range of decidability results for integrity constraints**

- **Related work**: [Arenas, Fan, Libkin 05; Buneman et al., 03]

- **Furthermore**: regular tree languages can be captured by \( \text{EMSO}^2(\sim, +1) \)

- **XML schema languages** correspond to regular tree languages

- **Reasoning on integrity constraints relative to schema descriptions possible**

- **in particular**: stronger notions of types can be handled

---

**Theorem**

The consistency and implication problems for unary keys and unary inclusion constraints relative to a regular tree language are decidable
XPath containment

- XPath is a navigation language for XML
- In its basic form it selects a set of nodes in a tree, which can be reached from the root on a path with certain properties
- Navigation along different axes
- The core of XPath (ignoring data values) corresponds exactly to $\mathbf{FO}^2(\sim,+1,\prec)$ [Marx 05]
- Query-Containment for XPath is in EXPTIME [Marx 05]

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Containment for special XPath is decidable</td>
</tr>
</tbody>
</table>
**XPath containment**

- **XPath** is a navigation language for **XML**
- In its basic form it selects a set of nodes in a tree, which can be reached from the root on a path with certain properties
- Navigation along different axes
- The core of **XPath** (ignoring data values) corresponds exactly to \( FO^2(\sim, +1, <) \) [Marx 05]
- Query-Containment for **XPath** is in **EXPTIME** [Marx 05]
- Our setting allows to consider a limited form of tests on attribute values, e.g.:
  
  \[
  \text{Child:a@A} = \text{/Child:b@C}
  \]
**XPath containment**

- **XPath** is a navigation language for XML.
- In its basic form, it selects a set of nodes in a tree, which can be reached from the root on a path with certain properties.
- Navigation along different axes.
- The core of **XPath** (ignoring data values) corresponds exactly to $\text{FO}^2(\sim, +1, \prec)$ [Marx 05].
- Query-Containment for **XPath** is in EXPTIME [Marx 05].
- Our setting allows to consider a limited form of tests on attribute values, e.g.:
  \[
  \text{Child:a@A} = /\text{Child:b@C}
  \]

**A larger example:**

\[
\text{Child :: a/Child :: b/@B}_1 =
\\
/\text{Child :: c/NextSibling :: d/}
\]

translates to

\[
\exists y E \downarrow(x, y) \land a(y) \land
\exists x E \downarrow(y, x) \land b(x) \land
\exists y E \downarrow(x, y) \land B_1(y) \land
\exists x x \sim y \land B_2(x) \land
\exists y E \downarrow(y, x) \land d(y) \land
\exists x E \rightarrow(x, y) \land c(x) \land
\exists y E \downarrow(y, x) \land \lnot \exists x E \downarrow(x, y)
\]

**Theorem**

Query Containment for special **XPath** is decidable.
The general case: $\text{FO}^2(\sim, +1, <)$ on data trees

- **Vector addition tree automata**: automata for binary trees
- Run induces a state and a vector of natural numbers, for each node
- Transitions $(q, l, q_0, q_1, \bar{a}, \bar{b}, \bar{c})$: automaton can take configuration $(q, \bar{z})$ at $v$ if
  - $v$ has label $l$
  - children have configurations $(q_0, \bar{x})$ and $(q_1, \bar{y})$
  - $\bar{x} - \bar{a} \geq \bar{0}$ and $\bar{y} - \bar{b} \geq \bar{0}$,
  - $\bar{z} = \bar{x} - \bar{a} + \bar{y} - \bar{b} + \bar{c}$
The general case: $\mathbf{FO}^2(\sim, +1, <)$ on data trees

- **Vector addition tree automata**: automata for binary trees
- Run induces a state and a vector of natural numbers, for each node
- Transitions $(q, l, q_0, q_1, \vec{a}, \vec{b}, \vec{c})$: automaton can take configuration $(q, \vec{z})$ at $v$ if
  - $v$ has label $l$
  - children have configurations $(q_0, \vec{x})$ and $(q_1, \vec{y})$
  - $\vec{x} - \vec{a} \geq \vec{0}$ and $\vec{y} - \vec{b} \geq \vec{0}$,
  - $\vec{z} = \vec{x} - \vec{a} + \vec{y} - \vec{b} + \vec{c}$
- Decidability of emptiness of vector addition tree automata is an open problem
- Equivalent to decidability of Multiplicative Exponential Linear Logic
The general case: $\mathbf{FO}^2(\sim, +1, <)$ on data trees

- **Vector addition tree automata**: automata for binary trees
- Run induces a state and a vector of natural numbers, for each node
- Transitions $(q, l, q_0, q_1, \vec{a}, \vec{b}, \vec{c})$: automaton can take configuration $(q, \vec{z})$ at $v$ if
  - $v$ has label $l$
  - children have configurations $(q_0, \vec{x})$ and $(q_1, \vec{y})$
  - $\vec{x} - \vec{a} \geq \vec{0}$ and $\vec{y} - \vec{b} \geq \vec{0}$,
  - $\vec{z} = \vec{x} - \vec{a} + \vec{y} - \vec{b} + \vec{c}$

- Decidability of emptiness of vector addition tree automata is an open problem
- Equivalent to decidability of Multiplicative Exponential Linear Logic

**Theorem**

For any vector addition tree automaton $A$, a formula $\varphi_A \in \mathbf{FO}^2(\sim, <, +1)$ can be computed such that $L(A) \neq \emptyset$ iff $\varphi_A$ has a model.

$\rightarrow$ We concentrate on $\mathbf{FO}^2(\sim, +1)$
## Main result and proof structure

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satisfiability of $\text{FO}^2(+1, &lt;)$ on data strings is decidable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satisfiability of $\text{FO}^2(+1)$ on data trees is decidable</td>
</tr>
<tr>
<td>Main result and proof structure</td>
</tr>
<tr>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>Theorem</strong></td>
</tr>
<tr>
<td>Satisfiability of $\text{FO}^2(+1, &lt;)$ on data strings is decidable</td>
</tr>
</tbody>
</table>

**Main steps of the string proof**

- $\text{FO}^2$ formula $\varphi$
- Scott normal form
- Intermediate normal form
- Data normal form $\psi$
- Data automaton $\mathcal{D}_\psi$
- Multicounter automaton $\mathcal{A}_\psi$ (on strings)

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satisfiability of $\text{FO}^2(+1)$ on data trees is decidable</td>
</tr>
</tbody>
</table>
## Main result and proof structure

<table>
<thead>
<tr>
<th>Theorem</th>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satisfiability of $\text{FO}^2(,+1, &lt;,)$ on data strings is decidable</td>
<td>Satisfiability of $\text{FO}^2(,+1,)$ on data trees is decidable</td>
</tr>
</tbody>
</table>

### Main steps of the string proof

1. $\text{FO}^2$ formula $\varphi$
2. Scott normal form
3. Intermediate normal form
4. Data normal form $\psi$
5. Data automaton $D_\psi$
6. Multicounter automaton $A_\psi$ (on strings)

### Main steps of the tree proof

1. $\text{FO}^2$ formula $\varphi$
2. Scott normal form
3. Intermediate normal form
4. Data normal form $\psi$
5. Puzzle $P_\psi$
6. Small model property
7. Linear constraint tree automaton $A_\psi$ (on trees)
Steps 1 and 2

- Scott normal form
- Intermediate normal form:
  \[ \exists R_1 \cdots R_m \theta_1 \land \cdots \land \theta_n \]
- \( \theta_i \):
  1. \( \forall x \forall y \ [(\alpha(x) \land \beta(y) \land \gamma(x, y)) \rightarrow (\delta(x, y) \leq 1)] \)
  2. \( \forall x \exists y \ \alpha(x) \rightarrow [\beta(y) \land \gamma(x, y) \land \epsilon(x, y)] \)
- where
  - \( \alpha, \beta \) are types.
  - \( \gamma(x, y) \) is either \( x \sim y \) or \( x \not\sim y \), and
  - \( \epsilon(x, y) \) is one of \( E_{\downarrow}(x, y), E_{\downarrow}(y, x), E_{\rightarrow}(x, y), E_{\rightarrow}(y, x), x = y \), or \( (\delta(x, y) > 1) \).
Step 3 and 4

- **Data normal form**: 
  \[ \exists R_1 \cdots R_n \, \theta_1 \land \cdots \land \theta_n \]

- **\( \theta_i \):**
  (a) data-blind
  (b) “Each class contains at most one node of type \( \alpha \).”
  (c) “Each class with at least one node of type \( \alpha \) has no \( \beta \).”
  (d) “Each class with at least one node of type \( \alpha \) also has a \( \beta \).”
  (e) “Each node of type \( \alpha \) has profile \( p \).”

- (Profile: data equality type wrt. parent, left and right sibling)
Step 3 and 4

- **Data normal form**:
  \[ \exists R_1 \cdots R_n \, \theta_1 \land \cdots \land \theta_n \]

- \( \theta_i \):
  - (a) data-blind
  - (b) “Each class contains at most one node of type \( \alpha \).”
  - (c) “Each class with at least one node of type \( \alpha \) has no \( \beta \).”
  - (d) “Each class with at least one node of type \( \alpha \) also has a \( \beta \).”
  - (e) “Each node of type \( \alpha \) has profile \( p \).”

- (Profile: data equality type wrt. parent, left and right sibling)

---

- **Puzzle**: regular data-blind condition \( R \) and pairs
  \( (D_1, S_n), \ldots, (D_k, S_k) \) of types

- A data tree is a **solution** to a puzzle, if
  - its data-free projection fulfils \( P \), and
  - for each class \( C \) there is some \( i \) such that
    - each type from \( D_i \) occurs exactly once in \( C \), and
    - all other types in \( C \) are from \( S_i \)
  - From a \( \textbf{FO}^2(\sim, +1) \) formula \( \phi \) one can construct a satisfiability equivalent puzzle \( P_\phi \)
Step 5

- **Zone**: maximally connected sub-tree within one (data) class
- **Degree** of a zone: number of neighboring classes
Step 5

- **Zone**: maximally connected sub-tree within one (data) class
- **Degree of a zone**: number of neighboring classes

**Proposition**

For every puzzle $\mathcal{P}$, one can compute constants $M, N$ such that $\mathcal{P}$ has a solution only if it has a solution in which at most $M$ zones have degree more than $N$.

**Proof strategy**

- We first show a local version of the proposition which talks about each class:  
  → local pruning
- For the general proposition:
  → global pruning
- We use (tons of) cut-and-paste arguments
Step 5

- **Zone**: maximally connected sub-tree within one (data) class
- **Degree** of a zone: number of neighboring classes

### Proposition
For every puzzle $P$, one can compute constants $M, N$ such that $P$ has a solution only if it has a solution in which at most $M$ zones have degree more than $N$.

### Proof strategy
- We first show a local version of the proposition which talks about each class:
  - local pruning
- For the general proposition:
  - global pruning
- We use (tons of) cut-and-paste arguments
Step 6

• Linear constraint tree automaton (LCTA) \( A \): unranked tree automaton \( A \) with state set \( Q \) and linear constraints over variables \( \#_q \)

\( (\#_q : \text{number of nodes with state } q) \)

Proposition

Non-emptiness of LCTA is \textbf{NP}-complete

Proposition

Let \( P \) be a puzzle over \( \Sigma \) and let \( M, N \in \mathbb{N} \). There is an LCTA that recognizes the data erasure of solutions of \( P \) where at most \( M \) zones have degree more than \( N \).

• This completes the proof that \( \textbf{FO}^2(\sim, +1) \) on data trees is decidable

• Complexity:
  – upper bound: \textbf{3-NEXPTIME}
  – lower bound: \textbf{NEXPTIME}
Two variable logics on data strings
Two variable logics on data trees

Conclusion
Two-variable logic on data strings and data trees offers an interesting framework to obtain decidability results in verification and XML reasoning.

Two main results:
- $\mathbf{FO}^2(\sim, <, +1)$ decidable on data strings
- $\mathbf{FO}^2(\sim, +1)$ decidable on data trees
Two-variable logic on data strings and data trees offers an interesting framework to obtain decidability results in verification and XML reasoning.

Two main results:
- \( \text{FO}^2(\sim, <, +1) \) decidable on data strings
- \( \text{FO}^2(\sim, +1) \) decidable on data trees

Open problems:
- \( \text{FO}^2(\sim, <, +1) \) on data trees
- \( \text{FO}^2(\sim, +\omega) \) on data trees
- Settle the open complexity issues
- Find tractable fragments
- What are regular data string languages / regular data tree languages
- Explore applicability of results in verification
Conclusion

- Two-variable logic on data strings and data trees offers an interesting framework to obtain decidability results in verification and XML reasoning.
- Two main results:
  - $\text{FO}^2(\sim, <, +1)$ decidable on data strings
  - $\text{FO}^2(\sim, +1)$ decidable on data trees

- Open problems:
  - $\text{FO}^2(\sim, <, +1)$ on data trees
  - $\text{FO}^2(\sim, +\omega)$ on data trees
  - Settle the open complexity issues
  - Find tractable fragments
  - What are regular data string languages / regular data tree languages
  - Explore applicability of results in verification

- References:
  - Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, Claire David: Two-Variable Logic on Words with Data. LICS 2006: 7-16
  - Mikolaj Bojanczyk, Claire David, Anca Muscholl, Thomas Schwentick, Luc Segoufin: Two-variable logic on data trees and XML reasoning. PODS 2006: 10-19
Finally...

Thank you!