

# 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

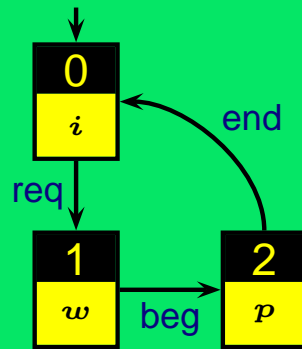
## A printer and a process

- 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:  $\mathbf{G}(w \rightarrow \mathbf{F}p)$

# 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:  $\mathbf{G}(w \rightarrow \mathbf{F}p)$



# Toy examples from Computer Science

## A printer and a process

- 1 Printer, 1 Process

- Process can

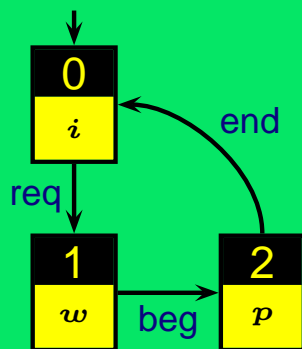
- stay idle
- wait to print
- print

→ Propositions  $i, w, p$

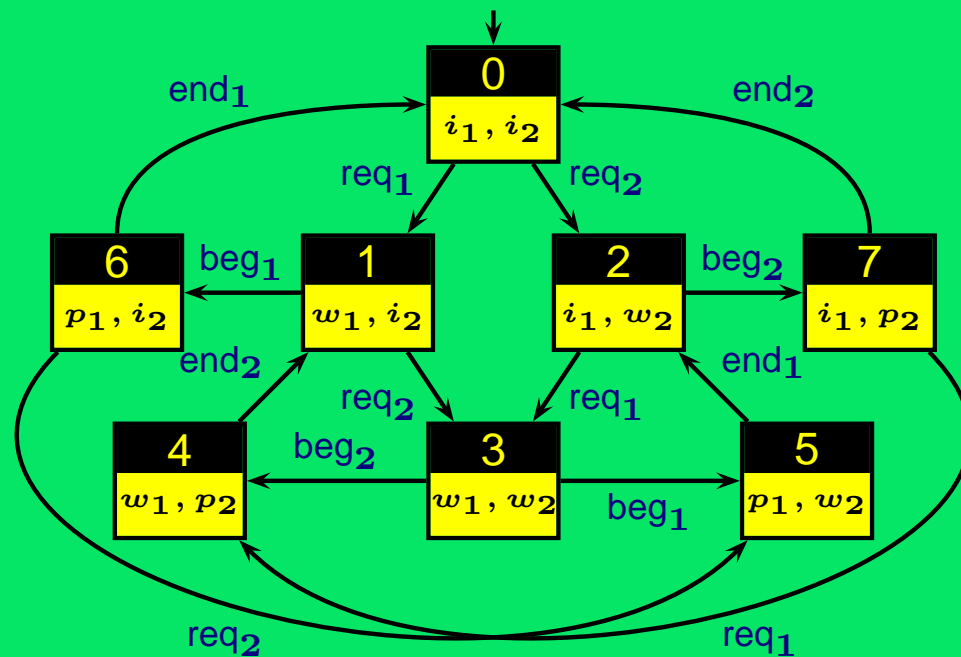
- Actions of the printer:

- req: User submits print request
- beg: Start printing
- end: End printing

- A property:  $\mathbf{G}(w \rightarrow \mathbf{F}p)$



## A printer and two processes



# Toy examples from Computer Science

## A printer and a process

- 1 Printer, 1 Process

- Process can

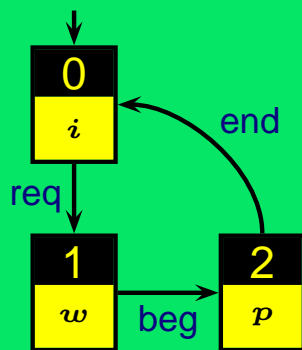
- stay idle
- wait to print
- print

→ Propositions  $i, w, p$

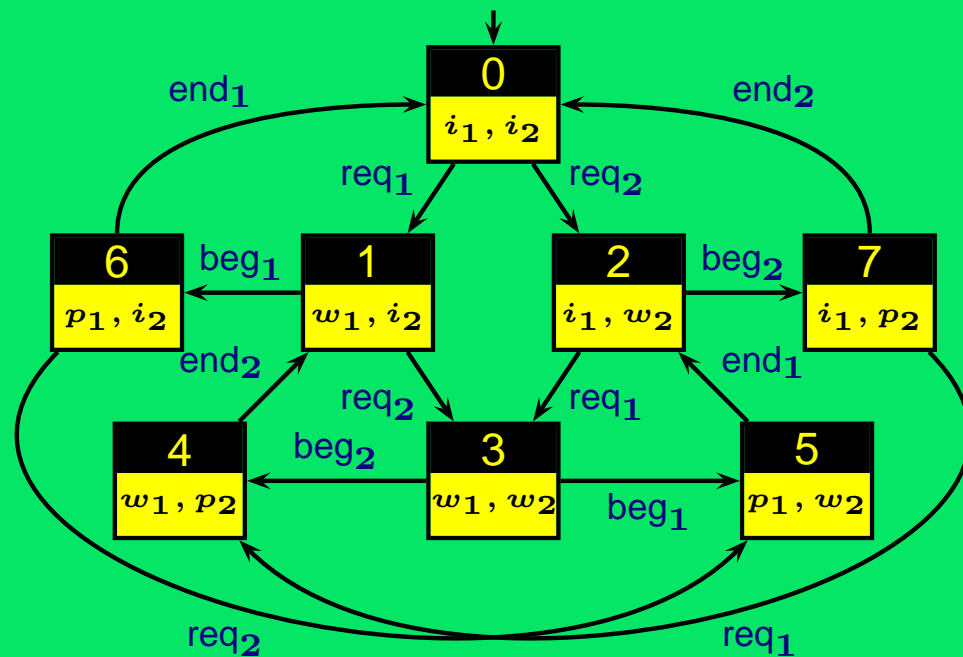
- Actions of the printer:

- req: User submits print request
- beg: Start printing
- end: End printing

- A property:  $\mathbf{G}(w \rightarrow \mathbf{F}p)$



## A printer and two processes



$$\mathbf{G}[(w_1 \rightarrow \mathbf{F}p_1) \wedge (w_2 \rightarrow \mathbf{F}p_2)]$$

# Toy examples from Computer Science

## A printer and a process

- 1 Printer, 1 Process

- Process can

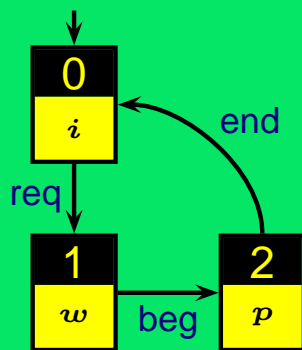
- stay idle
- wait to print
- print

→ Propositions  $i, w, p$

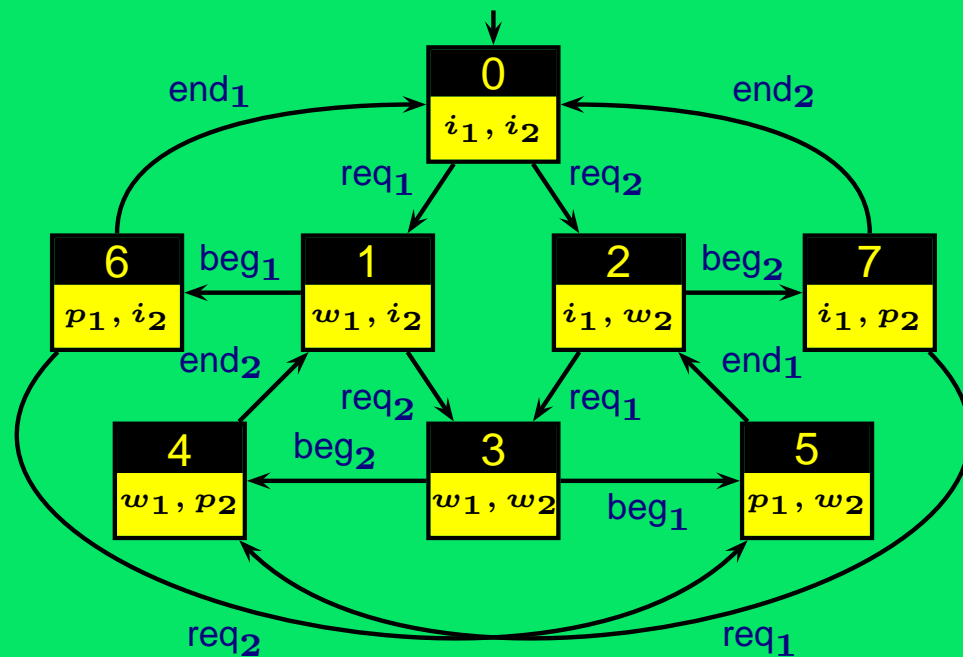
- Actions of the printer:

- req: User submits print request
- beg: Start printing
- end: End printing

- A property:  $\mathbf{G}(w \rightarrow \mathbf{F}p)$



## A printer and two processes



$$\mathbf{G}[(w_1 \rightarrow \mathbf{F}p_1) \wedge (w_2 \rightarrow \mathbf{F}p_2)]$$

- $k$  processes:  $3^k$  states, growing formula size
- arbitrary number of processes?

# The Model Checking Approach to Verification

---

- 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

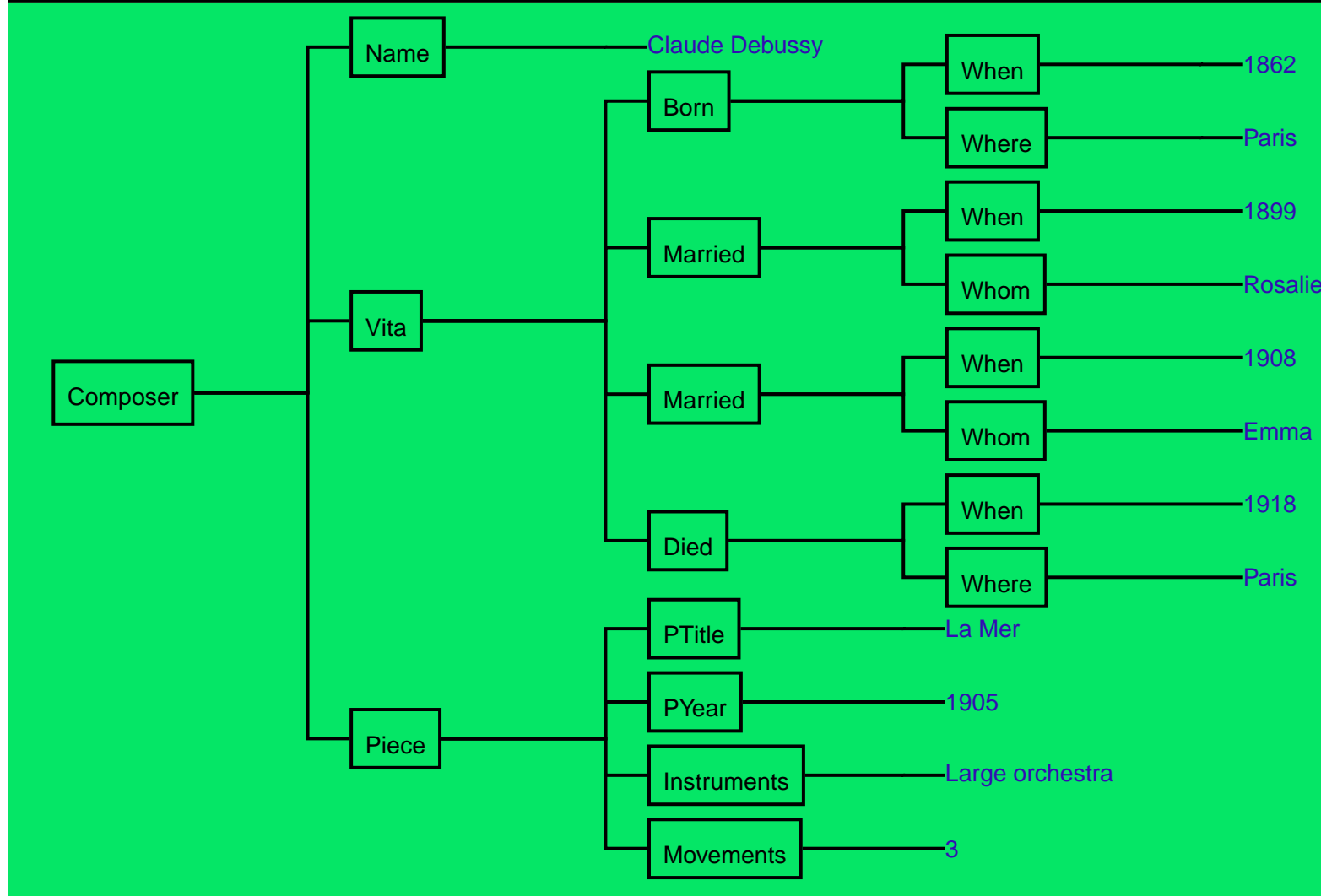
# XML

## Example Document

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

# XML

## Example Tree



# XML Processing

---

Four important kinds of XML processing

## **Validation**

Check whether an XML document is of a given type

## **Navigation**

Select a set of positions in an XML document

## **Querying**

Extract information from an XML document

## **Transformation**

Construct a new XML document from a given one

# XML Processing

---

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

## **Validation**

DTD, XML Schema

Check whether an XML document is of a given type

## **Navigation**

XPath

Select a set of positions in an XML document

## **Querying**

XQuery

Extract information from an XML document

## **Transformation**

XSLT

Construct a new XML document from a given one

# XML as Tree

## Example Document

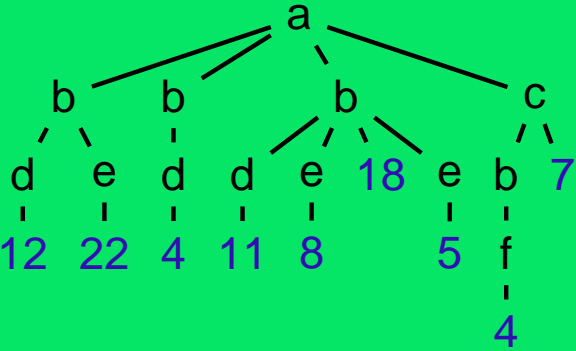
```
<a><b>  
  <d>12</d><e>22</e>  
  </b>  
  <b><d>4</d></b>  
  <b><d>11</d>  
    <e>8</e>18<e>5</e>  
  </b>  
  <c><b><f>2</f>  
  </b>7</c>  
</a>
```

# 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>  
<c><b><f>2</f>  
</b>7</c>  
</a>
```

## ...as Unranked Tree

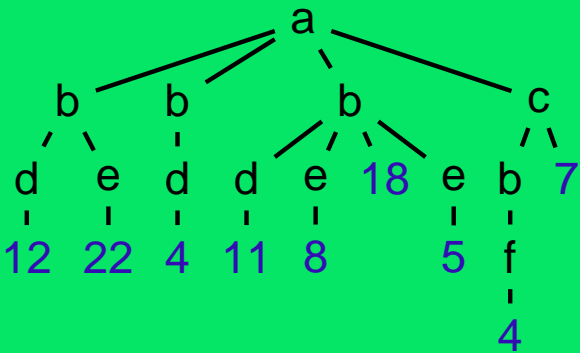


# XML as Tree

## 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>  
<c><b><f>2</f>  
</b>7</c>  
</a>
```

## ...as Unranked Tree



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

→ Validation, Navigation

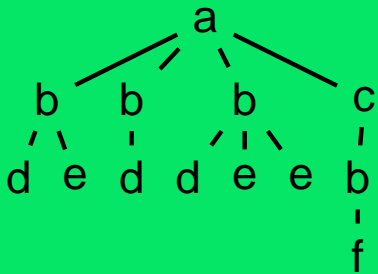


# XML as Tree

## 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>  
<c><b><f>2</f>  
</b>7</c>  
</a>
```

## ...as Unranked Tree



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

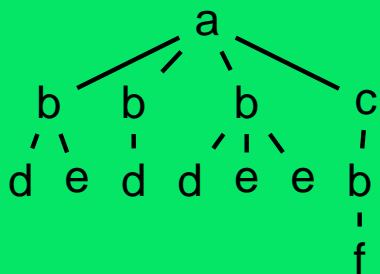
→ Validation, Navigation

# 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>  
<c><b><f>2</f>  
</b>7</c>  
</a>
```

## ...as Unranked Tree



- 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

# The Automata Approach to XML Processing

---

- 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

# The Automata Approach to XML Processing

---

- 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

# 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

# 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

# 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
- 
- Goal of this research: Find logics and automata that allow for decidability even if data values are present



# 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
- 
- Goal of this research: Find logics and automata that allow for decidability even if data values are present
  - Here: consider classical logics

# Contents

---

## ▷ **Preliminaries**

Background on two variable logics

Two variable logics on data strings

Two variable logics on data trees

Conclusion

# Our setting

Example: data string

17 5 2 3 3 3 2 2 7 17 17 3 4 5 2 3 3 4 4  
*a b c b c a a b b b c a b a c b b a a*

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

17 5 2 3 3 3 2 2 7 17 17 3 4 5 2 3 3 4 4  
*a b c b c a a b b b c a b a c b b a a*

## 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

17 5 2 3 3 3 2 2 7 17 17 3 4 5 2 3 3 4 4  
a b c b c a a b b b c a b a c b b a a

## 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  **$E_{\downarrow}$**

---

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:  $(u_1, v_1), \dots, (u_k, v_k)$ , pairs of strings
  - Question: is there a sequence  $i_1, \dots, i_n$  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:  $(u_1, v_1), \dots, (u_k, v_k)$ , pairs of strings
  - Question: is there a sequence  $i_1, \dots, i_n$  such that  $u_{i_1} \cdots u_{i_n} = v_{i_1} \cdots v_{i_n}$ ?

## A bit more detail

- Encode solution candidates as data strings over  $\{a, b, \#, 1, \dots, n\}$  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:
 
$$x \sim y \rightarrow \exists z (x + 1 = z \wedge \exists x y + 1 = x \wedge z \sim x)$$
- ”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:

$$\forall x \exists y \text{ req}(x) \rightarrow (x < y \wedge \text{beg}(y) \wedge x \sim y)$$

- No printer job is requested twice:

$$\forall x \forall y (\text{req}(x) \wedge \text{req}(y) \wedge x \sim y) \rightarrow x = y$$

# 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:

$$\forall x \exists y \text{ req}(x) \rightarrow (x < y \wedge \text{beg}(y) \wedge x \sim y)$$

- No printer job is requested twice:

$$\forall x \forall y (\text{req}(x) \wedge \text{req}(y) \wedge x \sim y) \rightarrow x = y$$

...in the XML context

- With 2 variables many integrity constraints can be expressed:
  - Did we reserve a room for every participant?
  - $\forall x \text{ Partic.Name}(x) \rightarrow \exists y \text{ Room.Name}(y) \wedge x.\text{data} = y.\text{data}$
  - Does every participant give at most one talk?
  - $\forall x \forall y [\text{Talk.Speaker}(x) \wedge \text{Talk.Speaker}(y)] \rightarrow x.\text{data} = y.\text{data}$

# 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:

$$\forall x \exists y \text{ req}(x) \rightarrow (x < y \wedge \text{beg}(y) \wedge x \sim y)$$

- No printer job is requested twice:

$$\forall x \forall y (\text{req}(x) \wedge \text{req}(y) \wedge x \sim y) \rightarrow x = y$$

...in the XML context

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

# Contents

---

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

## Theorem

- 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

## Theorem

- 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

**FO<sup>2</sup>**-formula  $\varphi$  is equivalent to a formula

$\varphi'$  of the form:

$$\forall x \forall y \chi \vee \bigwedge_i \forall x \exists y \chi_i$$

$\chi, \chi_i$  quantifier-free

- $\varphi'$  might have enlarged signature: more unary relations
- $\varphi'$  can be computed in polynomial time



# Some Known results about Two-Variable Logics

## Theorem

- 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

**FO**<sup>2</sup>-formula  $\varphi$  is equivalent to a formula

$\varphi'$  of the form:

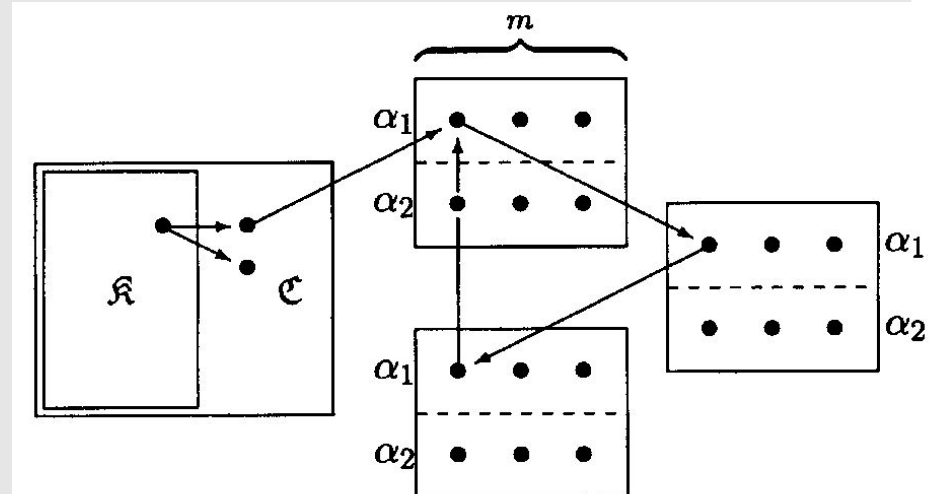
$$\forall x \forall y \chi \vee \bigvee_i \forall x \exists y \chi_i$$

$\chi, \chi_i$  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



# 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

# Is this the end of the story?

- Satisfiability of **FO<sup>2</sup>** 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 **coNEXPTIME** [Otto]
- On structures with several well-orderings: undecidable [Otto]

# Is this the end of the story?

- Satisfiability of **FO<sup>2</sup>** 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 **coNEXPTIME** [Otto]
- On structures with several well-orderings: undecidable [Otto]

## ...on strings without data

- Satisfiability **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 **FO<sup>2</sup>** 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 **coNEXPTIME** [Otto]
- On structures with several well-orderings: undecidable [Otto]

## ...on strings without data

- Satisfiability **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 **+1**)
- Regular tree languages: EMSO with two FO-variables (in the presence of  $E_{\rightarrow}, E_{\downarrow}$ )

# Contents

---

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  $s$ :

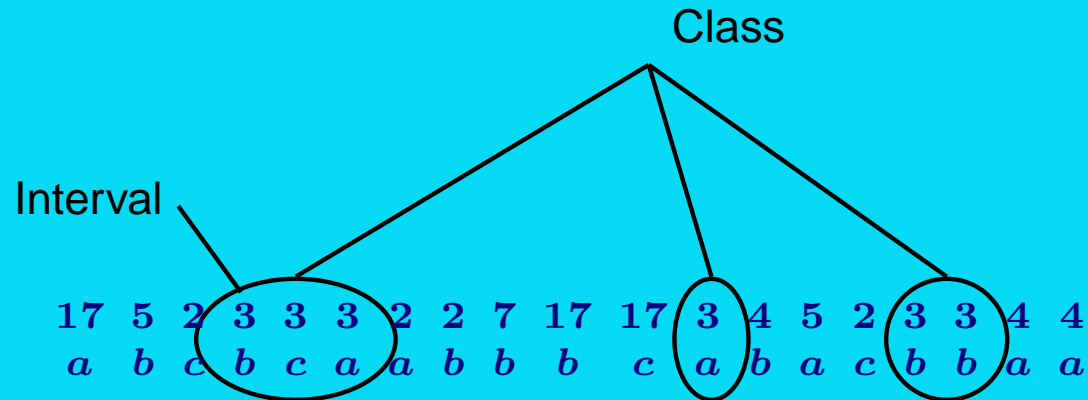
17 5 2 3 3 3 2 2 7 17 17 3 4 5 2 3 3 4 4  
 $a$   $b$   $c$   $b$   $c$   $a$   $a$   $b$   $b$   $b$   $c$   $a$   $b$   $a$   $c$   $b$   $b$   $a$   $a$

- **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  $s$ :



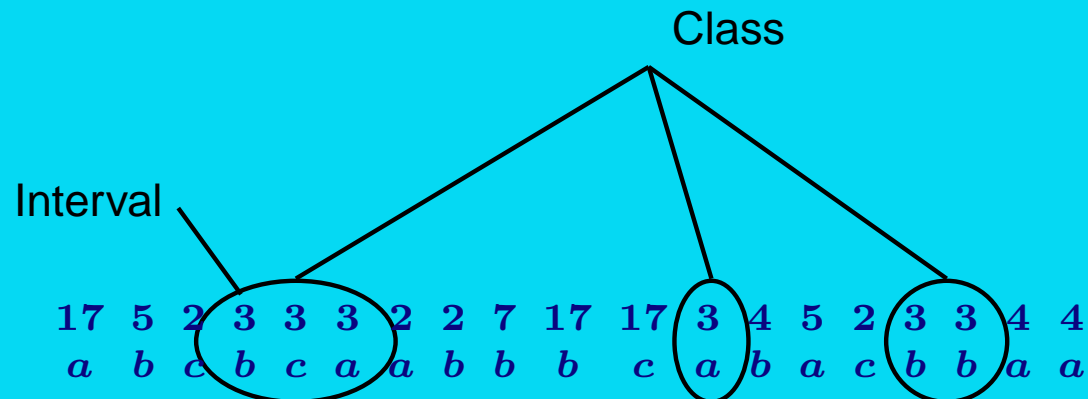
- **Class** : all positions with the same data value
  - **Interval** : (maximal set of) contiguous positions of a class
  - **String projection  $P(s)$**  :
- 
- **type  $\alpha(x)$**  : any conjunction of unary literals



# Some notation

## Some Definitions

- Data string  $s$ :



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

# On the expressive power of $\text{FO}^2$ on data strings

## Example

- $\varphi_a$ :
  - $\varphi_a = \forall x \forall y (x \neq y \wedge a(x) \wedge a(y)) \rightarrow x \not\sim y$
  - all  $a$ 's are in different classes
- Similarly:  $\varphi_b$
- $\psi_{a,b}$ :
  - $\psi_{a,b} = \forall x \exists y (a(x) \rightarrow (b(y) \wedge x \sim y))$
  - 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

- $\varphi_a$ :
    - $\varphi_a = \forall x \forall y (x \neq y \wedge a(x) \wedge a(y)) \rightarrow x \not\sim y$
    - all  $a$ 's are in different classes
  - Similarly:  $\varphi_b$
  - $\psi_{a,b}$ :
    - $\psi_{a,b} = \forall x \exists y (a(x) \rightarrow (b(y) \wedge x \sim y))$
    - each class with an  $a$  also contains a  $b$
  - Similarly:  $\psi_{b,a}$ .
- 
- Thus:
    - $\varphi = \varphi_a \wedge \varphi_b \wedge \psi_{a,b} \wedge \psi_{b,a}$
    - 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
- non-regular language

# On the expressive power of $\text{FO}^2$ on data strings (cont.)

## More example properties

- Let  $\alpha$  and  $\beta$  denote unary types
- $\text{FO}^2$  can express
  - 
  - 
  - 
  - 
  - 
  -

# On the expressive power of $FO^2$ on data strings (cont.)

## More example properties

- Let  $\alpha$  and  $\beta$  denote unary types
- $FO^2$  can express
  - data-blind properties, i.e., properties not using  $\sim$
  - 
  - 
  - 
  - 
  -

# On the expressive power of $\text{FO}^2$ on data strings (cont.)

## 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:

$$\theta = \forall x \forall y ((\alpha(x) \wedge \alpha(y)) \rightarrow x \sim y)$$

- 
- 
- 
-

# On the expressive power of $\text{FO}^2$ on data strings (cont.)

## 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:

$$\theta = \forall x \forall y ((\alpha(x) \wedge \alpha(y)) \rightarrow x \sim y)$$

- Each class contains at most one occurrence of  $\alpha$ :

$$\theta = \forall x \forall y ((\alpha(x) \wedge \alpha(y) \wedge x \sim y) \rightarrow x = y)$$

- 
- 
-

# On the expressive power of $\text{FO}^2$ on data strings (cont.)

## 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:

$$\theta = \forall x \forall y ((\alpha(x) \wedge \alpha(y)) \rightarrow x \sim y)$$

- Each class contains at most one occurrence of  $\alpha$ :

$$\theta = \forall x \forall y ((\alpha(x) \wedge \alpha(y) \wedge x \sim y) \rightarrow x = y)$$

- In each class, every  $\alpha$  occurs before every  $\beta$ :

$$\theta = \forall x \forall y ((\alpha(x) \wedge \beta(y) \wedge x \sim y) \rightarrow x < y)$$

- 
-



# On the expressive power of $\text{FO}^2$ on data strings (cont.)

## 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:

$$\theta = \forall x \forall y ((\alpha(x) \wedge \alpha(y)) \rightarrow x \sim y)$$

- Each class contains at most one occurrence of  $\alpha$ :

$$\theta = \forall x \forall y ((\alpha(x) \wedge \alpha(y) \wedge x \sim y) \rightarrow x = y)$$

- In each class, every  $\alpha$  occurs before every  $\beta$ :

$$\theta = \forall x \forall y ((\alpha(x) \wedge \beta(y) \wedge x \sim y) \rightarrow x < y)$$

- Each class with an  $\alpha$  also has a  $\beta$ :

$$\theta = \forall x \exists y (\alpha(x) \rightarrow (\beta(y) \wedge x \sim y))$$

–

# On the expressive power of $\text{FO}^2$ on data strings (cont.)

## 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:

$$\theta = \forall x \forall y ((\alpha(x) \wedge \alpha(y)) \rightarrow x \sim y)$$

- Each class contains at most one occurrence of  $\alpha$ :

$$\theta = \forall x \forall y ((\alpha(x) \wedge \alpha(y) \wedge x \sim y) \rightarrow x = y)$$

- In each class, every  $\alpha$  occurs before every  $\beta$ :

$$\theta = \forall x \forall y ((\alpha(x) \wedge \beta(y) \wedge x \sim y) \rightarrow x < y)$$

- Each class with an  $\alpha$  also has a  $\beta$ :

$$\theta = \forall x \exists y (\alpha(x) \rightarrow (\beta(y) \wedge x \sim y))$$

- That's all!

# Main result regarding data strings

---

## Theorem

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

# 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  $\mathcal{D}_\psi$



Multicounter automaton  $\mathcal{A}_\psi$  (on strings)

- Finally:  $\mathcal{A}_\psi$  accepts a string  $w$  if and only if there is a data string  $s$  with
  - $s \models \varphi$
  - $P(s) = w$
- Thus:  $\psi$  satisfiable  $\iff L(\mathcal{A}_\psi) \neq \emptyset$

# Steps 1 and 2

## Scott and intermediate normal form

- We transform into satisfiability equivalent EMSO formulas
- **Scott normal form** :  $\exists R_1, \dots, R_k \forall x \forall y \chi \wedge \bigwedge_i \forall x \exists y \chi_i$

- **Intermediate normal form** :

$$\exists R_1 \dots R_m \theta_1 \wedge \dots \wedge \theta_n$$

- $\theta_i$ :

$$(1) \forall x \forall y (\delta(x, y) \geq 2 \wedge \alpha(x) \wedge \beta(y) \wedge \begin{array}{|l} x \sim y \\ x \not\sim y \end{array}) \rightarrow \begin{array}{|l} x < y \\ x > y \\ \text{ff} \end{array}$$

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

$$(3) \forall x \forall y \theta \quad (\text{quantifier-free, DNF, no } \sim)$$

- Both steps are straightforward

# Step 3

## Data normal form

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

$$\exists R_1 \cdots R_n \theta_1 \wedge \cdots \wedge \theta_n$$

- $\theta_i$ :

- (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  $\alpha$

# Step 3

## Data normal form

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

$$\exists R_1 \cdots R_n \theta_1 \wedge \cdots \wedge \theta_n$$

- $\theta_i$ :

- (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  $R_1^\alpha, \dots, R_4^\alpha$

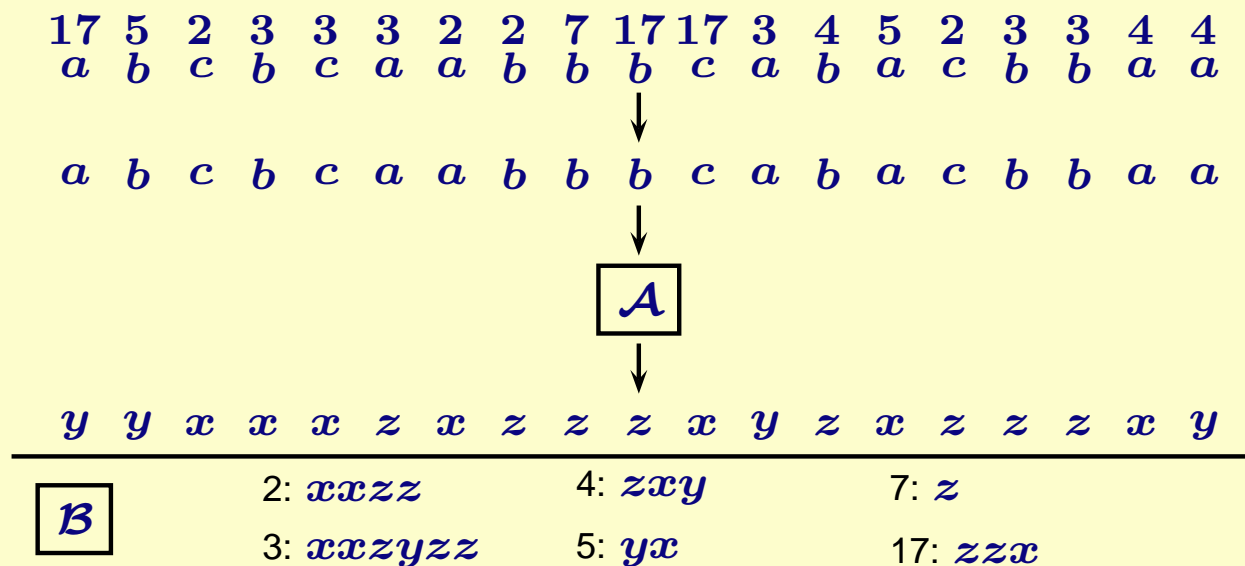
- Case distinction on possible formulas (1) and (2)

→ in each case  $\theta_i$  can be replaced by some “data normal” formulas

# Step 4

## Data automata

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

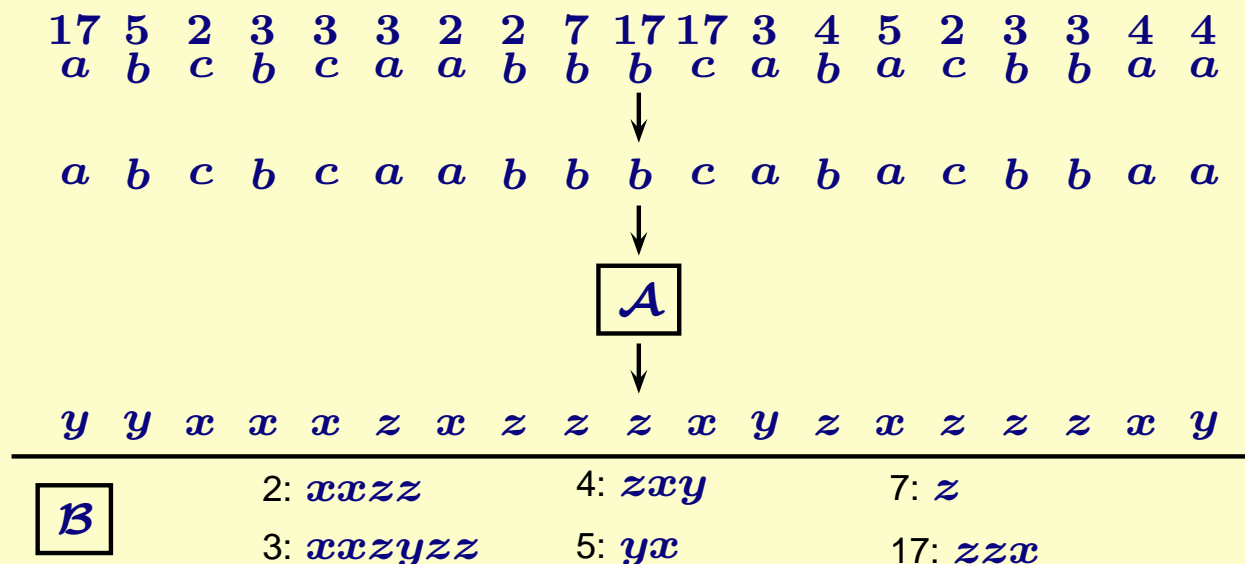




# Step 4

## Data automata

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



## Proposition

For each two-variable formula over data strings there is an equivalent data automaton.

# Step 5

## Multicounter-automata

- A multicounter-automaton is a finite automaton with counters  $C_1, \dots, C_k$
- 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  $C_1, \dots, C_k$
- 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  $\mathcal{D} = (\mathcal{A}, \mathcal{B})$  there is a multicounter automaton  $\mathcal{M}$  such that  $L(\mathcal{M}) = P(L(\mathcal{D}))$

## Proof idea

- Plan:  $\mathcal{M}$  checks whether a given string  $w$  can be extended to a data string  $w'$  accepted by  $\mathcal{D}$
- Clearly:  $\mathcal{M}$  can simulate transducer  $\mathcal{A}$  (which does not see the data anyway)
- Idea:  $\mathcal{M}$  uses one counter  $C_p$ , for each state  $p$  of  $\mathcal{B}$
- 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 **FO**( $<, +1$ ) on data words

2NEXPTIME ↓

Emptiness for data automata

PTIME ↓

Reachability for Petri nets

# Related results

- Complexity of reductions:

Satisfiability for **FO**( $<, +1$ ) on data words

2NEXPTIME  $\Downarrow \Uparrow$  PTIME

Emptiness for data automata

PTIME  $\Downarrow \Uparrow$  PTIME

Reachability for Petri nets

# Related results

- Complexity of reductions:

Satisfiability for  $\mathbf{FO}(<, +1)$  on data words

$2\text{NEXPTIME} \downarrow \uparrow \text{PTIME}$

Emptiness for data automata

$\text{PTIME} \downarrow \uparrow \text{PTIME}$

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)

# Related results

- Complexity of reductions:

Satisfiability for  $\mathbf{FO}(<, +1)$  on data words

$2\text{NEXPTIME} \downarrow \uparrow \text{PTIME}$

Emptiness for data automata

$\text{PTIME} \downarrow \uparrow \text{PTIME}$

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  $i < k < j, i \sim k$

# Related results

- Complexity of reductions:

Satisfiability for  $\mathbf{FO}(<, +1)$  on data words

$2\text{NEXPTIME} \downarrow \uparrow \text{PTIME}$

Emptiness for data automata

$\text{PTIME} \downarrow \uparrow \text{PTIME}$

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  $\mathbf{FO}(<, +1)$  on data words

$2\text{NEXPTIME} \downarrow \uparrow \text{PTIME}$

Emptiness for data automata

$\text{PTIME} \downarrow \uparrow \text{PTIME}$

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  $\mathbf{EMSO}^2(\sim, <, +1, \pm 1)$

## Proposition

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

# Related results

- Complexity of reductions:

Satisfiability for  $\mathbf{FO}(<, +1)$  on data words

$2\text{NEXPTIME} \downarrow \uparrow \text{PTIME}$

Emptiness for data automata

$\text{PTIME} \downarrow \uparrow \text{PTIME}$

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  $\mathbf{EMSO}^2(\sim, <, +1, \pm 1)$

## Proposition

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

# Related results

- Complexity of reductions:

Satisfiability for  $\mathbf{FO}(<, +1)$  on data words

$2\text{NEXPTIME} \downarrow \uparrow \text{PTIME}$

Emptiness for data automata

$\text{PTIME} \downarrow \uparrow \text{PTIME}$

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  $\mathbf{EMSO}^2(\sim, <, +1, \pm 1)$

## Proposition

- Satisfiability of  $\mathbf{FO}^2(\sim, <)$  on data words is **NEXPTIME**-complete
- Satisfiability of  $\mathbf{FO}^2(\sim, +1)$  on data words is **2 – NEXPTIME**-complete
- Decidability remains if predicates  $+2, +3, \dots$  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:

Satisfiability for  $\mathbf{FO}(<, +1)$  on data words

$2\text{NEXPTIME} \downarrow \uparrow \text{PTIME}$

Emptiness for data automata

$\text{PTIME} \downarrow \uparrow \text{PTIME}$

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  $\mathbf{EMSO}^2(\sim, <, +1, \pm 1)$

## Proposition

- Satisfiability of  $\mathbf{FO}^2(\sim, <)$  on data words is **NEXPTIME**-complete
- Satisfiability of  $\mathbf{FO}^2(\sim, +1)$  on data words is **2 – NEXPTIME**-complete
- Decidability remains if predicates  $+2, +3, \dots$  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

---

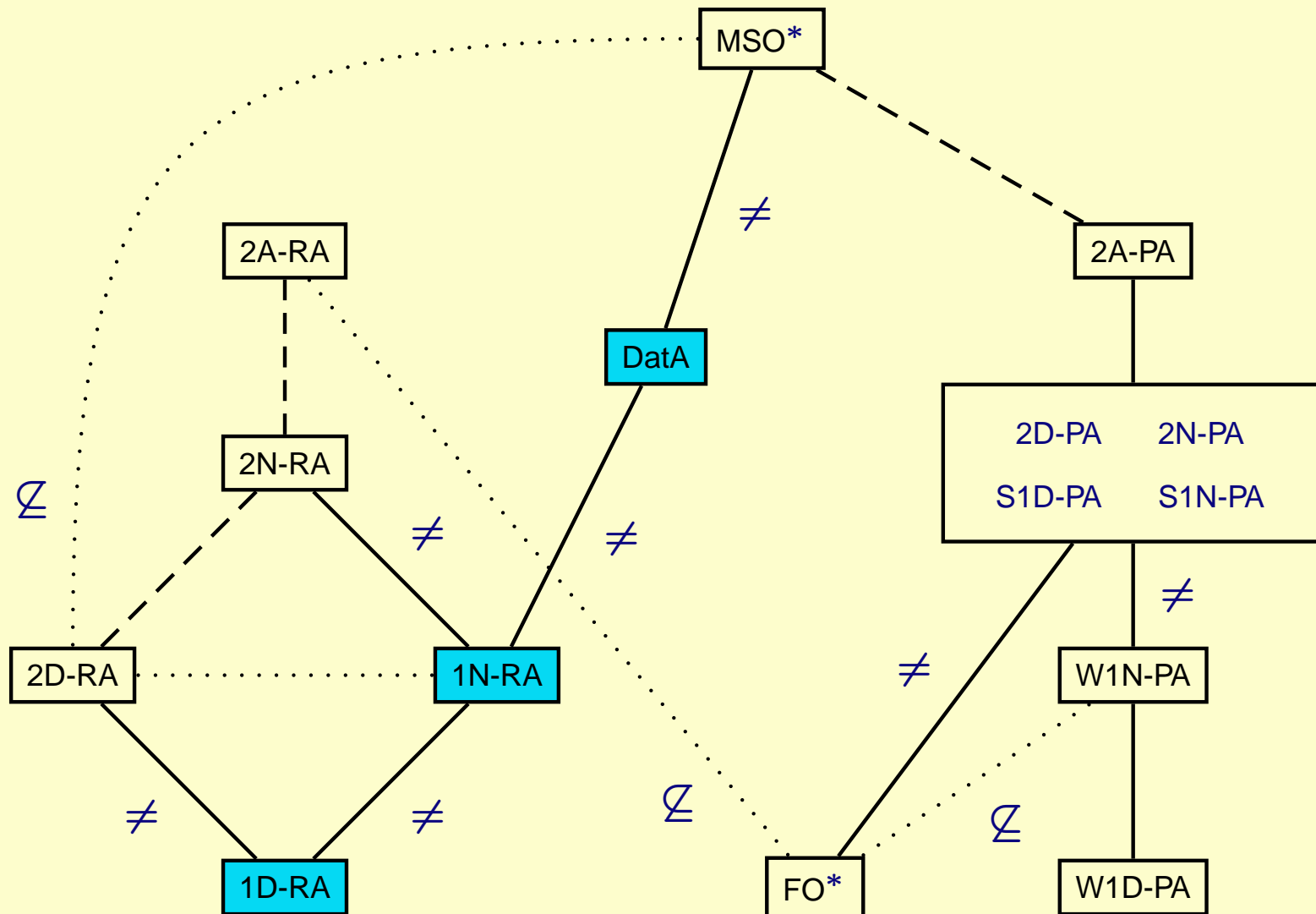
- Regular string and tree languages play a key role for decidability results
- Natural question: Is there a sensible notion of regular languages for data strings?

# 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



# 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

# Coming soon...

---

# Coming soon...

## 13. Jahrestagung der GI-Fachgruppe



### Logik in der Informatik

Dortmund, 12.-13. Oktober 2006



**Invited speaker:**

Anuj Dawar

**More infos:**

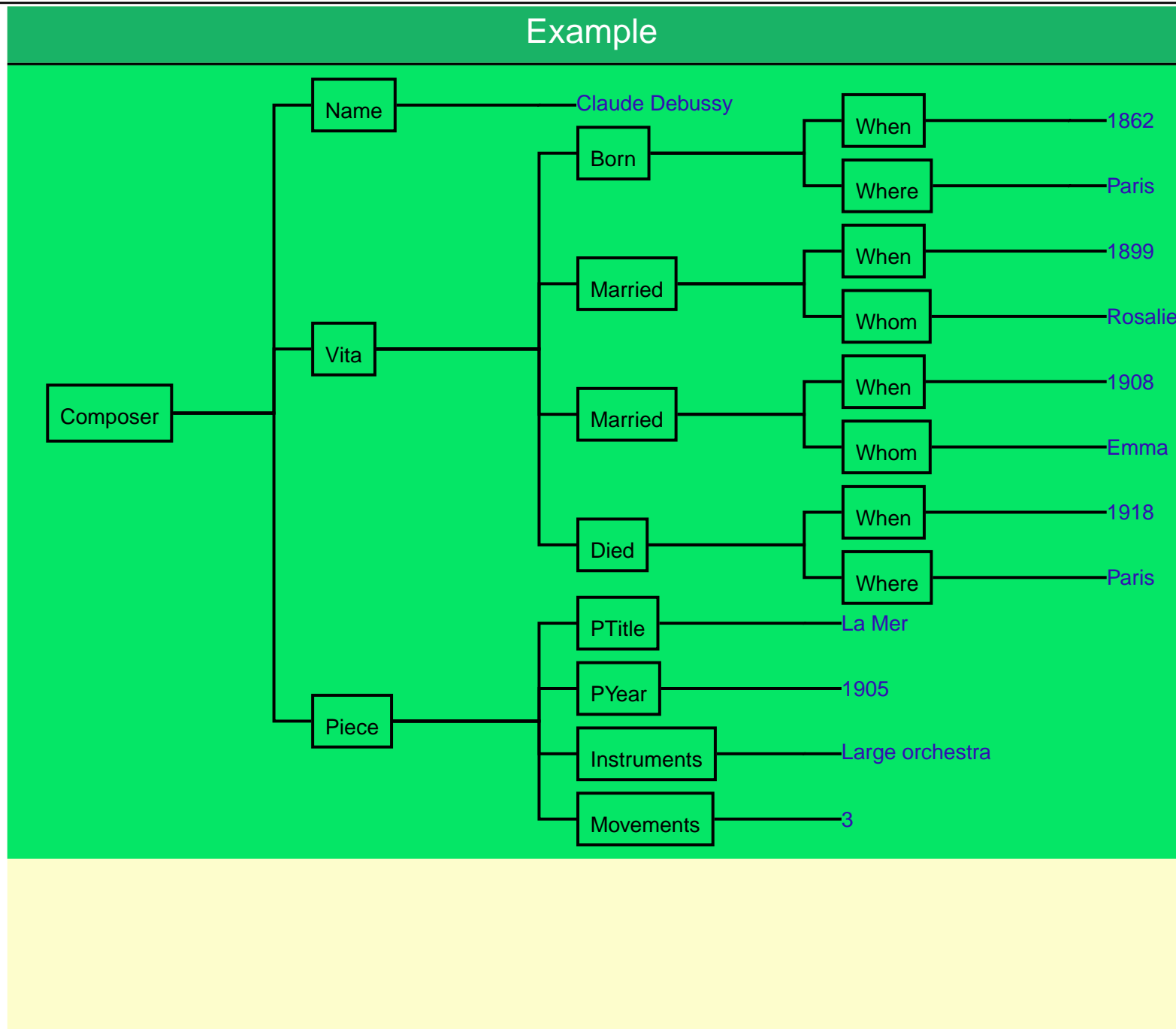
[loginf.cs.uni-dortmund.de](http://loginf.cs.uni-dortmund.de)

# Data trees: notation

## Definition

- **Data tree**: Tree with labels from  $\Sigma$  and one data value from  $\mathcal{D}$  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  $E_{\Downarrow}$
- **FO<sup>2</sup>( $\sim, +1$ )**: only next sibling and child
- **FO<sup>2</sup>( $\sim, +1, <$ )**: also: descendant and following sibling

# Back to XML applications



# 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]

# 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  $\text{FO}^2(\sim, +1, <)$  [Marx 05]
- Query-Containment for XPath is in **EXPTIME** [Marx 05]

## Theorem

Query Containment for special XPath is 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  $\text{FO}^2(\sim, +1, <)$  [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.:  
Child:a@A = /Child:b@C

## Theorem

Query Containment for special XPath is 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  $\text{FO}^2(\sim, +1, <)$  [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.:  
Child:a@A = /Child:b@C

- A larger example:

Child :: a/Child :: b/@B<sub>1</sub> =  
/Child :: c/NextSibling :: d/

translates to

$$\begin{aligned} & \exists y E_{\downarrow}(x, y) \wedge a(y) \wedge \\ & \exists x E_{\downarrow}(y, x) \wedge b(x) \wedge \\ & \exists y E_{\downarrow}(x, y) \wedge B_1(y) \wedge \\ & \exists x x \sim y \wedge B_2(x) \wedge \\ & \exists y E_{\downarrow}(y, x) \wedge d(y) \wedge \\ & \exists x E_{\rightarrow}(x, y) \wedge c(x) \wedge \\ & \exists y E_{\downarrow}(y, x) \wedge \neg \exists x E_{\downarrow}(x, y) \end{aligned}$$

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  $(q, l, q_0, q_1, \vec{a}, \vec{b}, \vec{c})$ :  
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  $\vec{y} - \vec{b} \geq \vec{0}$ ,
  - $\vec{z} = \vec{x} - \vec{a} + \vec{y} - \vec{b} + \vec{c}$

# 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  $(q, l, q_0, q_1, \vec{a}, \vec{b}, \vec{c})$ : 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: $\text{FO}^2(\sim, +1, <)$ 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  $(q, l, q_0, q_1, \vec{a}, \vec{b}, \vec{c})$ : 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  $\varphi_A \in \text{FO}^2(\sim, <, +1)$  can be computed such that  $L(A) \neq \emptyset$  iff  $\varphi_A$  has a model.

→ We concentrate on  $\text{FO}^2(\sim, +1)$

# Main result and proof structure

Theorem

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

Theorem

Satisfiability of  $\mathbf{FO}^2(+1)$  on data trees is  
decidable

# Main result and proof structure

## Theorem

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

### Main steps of the string proof

$\mathbf{FO}^2$  formula  $\varphi$



Scott normal form



Intermediate normal form



Data normal form  $\psi$



Data automaton  $\mathcal{D}_\psi$



Multicounter automaton  $\mathcal{A}_\psi$  (on strings)

## Theorem

Satisfiability of  $\mathbf{FO}^2(+1)$  on data trees is decidable



# Main result and proof structure

Theorem	Theorem
Satisfiability of $\mathbf{FO}^2(+1, <)$ on data strings is decidable	Satisfiability of $\mathbf{FO}^2(+1)$ on data trees is decidable
Main steps of the string proof	Main steps of the tree proof
<p style="text-align: center;"><math>\mathbf{FO}^2</math> formula <math>\varphi</math></p> <p style="text-align: center;">↓</p> <p style="text-align: center;">Scott normal form</p> <p style="text-align: center;">↓</p> <p style="text-align: center;">Intermediate normal form</p> <p style="text-align: center;">↓</p> <p style="text-align: center;">Data normal form <math>\psi</math></p> <p style="text-align: center;">↓</p> <p style="text-align: center;">Data automaton <math>\mathcal{D}_\psi</math></p> <p style="text-align: center;">↓</p> <p style="text-align: center;">Multicounter automaton <math>\mathcal{A}_\psi</math> (on strings)</p>	<p style="text-align: center;"><math>\mathbf{FO}^2</math> formula <math>\varphi</math></p> <p style="text-align: center;">↓</p> <p style="text-align: center;">Scott normal form</p> <p style="text-align: center;">↓</p> <p style="text-align: center;">Intermediate normal form</p> <p style="text-align: center;">↓</p> <p style="text-align: center;">Data normal form <math>\psi</math></p> <p style="text-align: center;">↓</p> <p style="text-align: center;">Puzzle <math>P_\psi</math></p> <p style="text-align: center;">↓</p> <p style="text-align: center;">Small model property</p> <p style="text-align: center;">↓</p> <p style="text-align: center;">Linear constraint tree automaton <math>\mathcal{A}_\psi</math> (on trees)</p>

# Steps 1 and 2

- Scott normal form

- Intermediate normal form :

$$\exists R_1 \cdots R_m \theta_1 \wedge \cdots \wedge \theta_n$$

- $\theta_i$ :

(1)  $\forall x \forall y [\alpha(x) \wedge \beta(y) \wedge \gamma(x, y)] \rightarrow (\delta(x, y) \leq 1)$

(2)  $\forall x \exists y \alpha(x) \rightarrow [\beta(y) \wedge \gamma(x, y) \wedge \epsilon(x, y)]$

- 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 :

$$\exists R_1 \cdots R_n \theta_1 \wedge \cdots \wedge \theta_n$$

- $\theta_i$ :

(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** :

$$\exists R_1 \cdots R_n \theta_1 \wedge \cdots \wedge \theta_n$$

- $\theta_i$ :

- (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  $(D_1, S_1), \dots, (D_k, S_k)$  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  $\text{FO}^2(\sim, +1)$  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

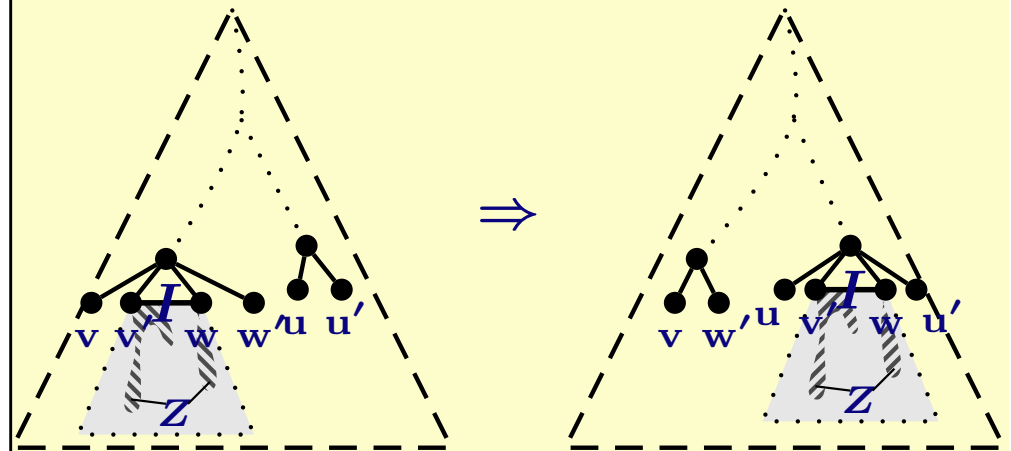
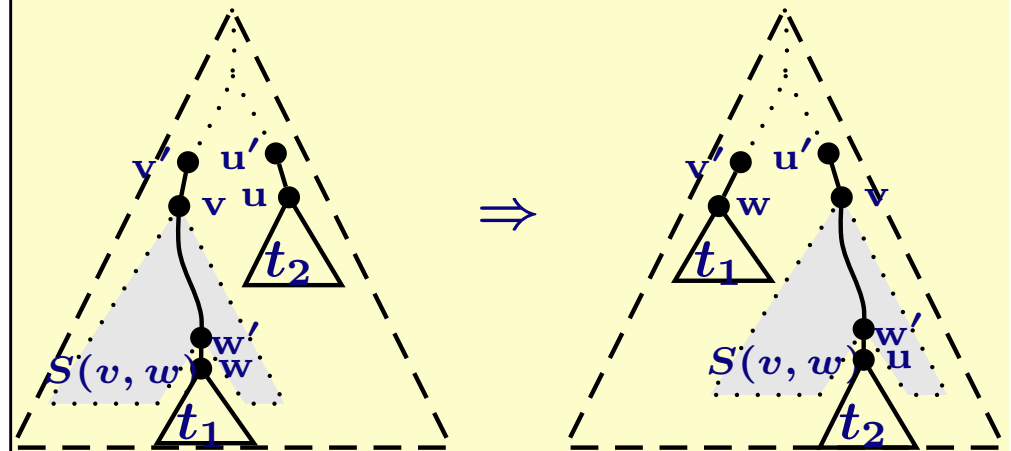
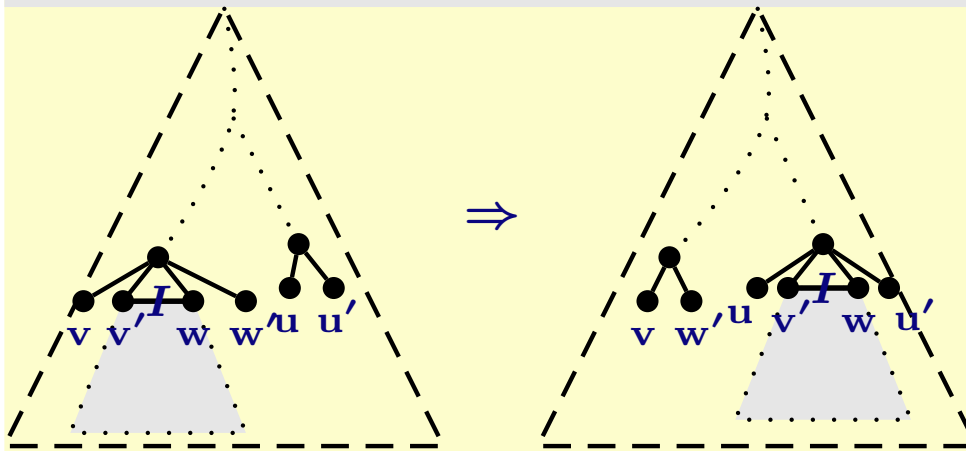
- **Zone**: maximally connected sub-tree within one (data) class
- **degree** of a zone: number of neighboring classes

## Proposition

For every puzzle  $\mathcal{P}$ , 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 6

- **Linear constraint tree automaton (LCTA)**: unranked tree automaton  $\mathcal{A}$  with state set  $Q$  and linear constraints over variables  $\#_q$

( $\#_q$ : number of nodes with state  $q$ )

## Proposition

Non-emptiness of LCTA is **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 **FO<sup>2</sup>( $\sim, +1$ )** on data trees is decidable
- Complexity:
  - upper bound: **3-NEXPTIME**
  - lower bound: **NEXPTIME**



# Contents

---

Preliminaries

Background on two variable logics

Two variable logics on data strings

Two variable logics on data trees

▷ **Conclusion**

# 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

# 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

# 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!