In the Maze of Data Languages

Loris D'Antoni

WPE II
05/08/2012
Data Languages

- **Motivation**
- Data Model
- Data Strings
  - Automata and Logics
  - Regularity
- Data Trees
- Conclusion
Introduction

- Most analysis techniques over programs and data consider the domain to be finite in order to achieve decidability.

- Often this restriction is too strong:
  - XML documents and languages over XML use data comparison.
  - Interesting properties about programs compare values of variables at different points in the program.
**Motivation 1: XML Processing**

- An XML document can be seen as an unranked tree in which
  - Inner nodes correspond to **elements** (tags)
  - Leaves correspond to **data** (attributes, text content)
Motivation 1: XML Processing

- For many useful tasks data values can be ignored
  - we can consider the tree to be over a finite alphabet
  - good for navigation, validation, transformation...

WHAT ABOUT TASKS IN WHICH WE WANT TO SPECIFY CONSTRAINTS OVER DATA?
Motivation 1: XML Processing

- A concrete example: *XPath query optimization*

- **SCHEMA:** can define XML language and can also specify constraints on data

- **XPath:** query language for XML that also allows data comparison
  
  - **Q1:** select all notes someone sent to himself
  
  - **Q2:** select people who sent more than 3 notes

- **QUERY OPTIMIZATION:** given two XPath queries $q_1, q_2$ and a Schema $S$, decide whether,

  $$\text{for each valid document } x \text{ in } S, \ \ q_1(x) \subseteq q_2(x)$$
Motivation 2: Verification

- **Model Checking:** checking properties about programs that can have possibly infinite reachable states
  - Represent system as a finite structure
  - Define a transition relation
  - Use algorithm for reachability of some particular state
- Several **ad-hoc solutions** for particular cases of infinite alphabets and infinite states
  - Timed Automata [Alur90]
  - Regular model checking [Bouajjani00]
Motivation 2: Verification

- No model considers inter-state properties such as

  *the same resource is never granted twice (with infinitely many resources)*

- A run of the transition system can be seen as a string/list of the form

  \[
  q_0 \rightarrow r_1 \rightarrow q_1 \rightarrow r_4 \rightarrow q_3 \rightarrow r_1 \rightarrow \ldots \rightarrow q_f \rightarrow r_1
  \]

  where the states are from a finite alphabet and the resources are from an infinite domain and

- Now we can ask:

  *Is there a list with the same resource appearing twice*
Some Models for Infinite

● Several models have been proposed to work with **infinite alphabets**:
  - LTL with Freeze Quantifiers (LTL with storing registers)
  - Timed Automata (can reason about Time)
  - Symbolic Automata and Transducers (theory over input)

● Most of these models are quite domain specific even though they come with nice properties

● We want a **general theory** for

**structures over infinite alphabets**
Data Languages

- Motivation
- **Data Model**
- Data Strings
  - Automata and Logics
  - Regularity
- Data Trees
- Conclusion
Data Model: Design Principles

- We need a simple model with some decidable features
- The model should be useful
- Possibly it should be guided by some practical applications

**DATA STRINGS and DATA TREES**
Data Languages

- We take languages of words and trees over finite alphabets.
- Then, **one data element** from an **infinite domain** is allowed for every position/node.
- The **only operation** that can be performed over data is checking for **equality**.
- It is a bit restrictive but **easy to study** and **useful** in practice.
- Moreover, most extensions immediately lead to undecidability.
**Data Strings**

- In a data string each position carries
  - a **label** from a *finite alphabet* and
  - a **data value** from an *infinite alphabet*

<table>
<thead>
<tr>
<th>r</th>
<th>w</th>
<th>w</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>34</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

- For example in the data string above
  - the finite alphabet is \{r,w,s\}
  - the infinite alphabet is the natural numbers
Data Trees

- Similarly, in a data tree each NODE carries
  - a **label** from a *finite alphabet* and
  - a **data value** from an *infinite alphabet*

```
Id  From  To  Body
501  Mary  Tom  Prepare dinner!
502  Tom  Mary  Will do!
```
And now?

- Data languages seem to nicely extend regular languages
- The framework is set but now:
  - How do we define data languages?
  - What is the best/right model for data string languages?
  - What is the best/right model for data tree languages?
  - What is a regular data language?
Regularity

- Ideally we are looking for a model for data languages with all the nice properties of regular string languages
  - Good tradeoff between expressiveness and decidability
  - Efficiency of the membership problem
  - Good closure properties
  - Robustness: clear counterpart in logic and several characterizations

DOES A MODEL LIKE THAT EVEN EXIST?
Data Languages

- Motivation
- Data Model
- Data Strings
  - Automata and Logics
  - Regularity
- Data Trees
- Conclusion
Models for Data Strings

- Several models have been proposed for data strings and they are mainly of two kinds:
  - **Automata** based models
  - **Logic** based models
- Usually an automata model is good when it has an equivalent logic model
- Here we present the models that are considered more relevant in the `treasure hunt' for regular data string languages
Register Automata 1/4

- Finite state automaton + finite set of registers that can store data values and test for equality

STATE \( q \)
R1=4, R2=1

\[(q,R1,r) \text{ steps to } (q',L)\]

STATE \( q' \)
R1=4, R2=1
Register Automata 2/4

STATE q
R1=4, R2=1

if no R contains current value (q,r) steps to (q',R2,R)

STATE q'
R1=4, R2=5
Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - \((q0, \{r, w, s\}) \rightarrow (q1, R1, R)\)
  - \((q1, R1, \{r, w, s\}) \rightarrow (qf, S)\)
  - \((q1, \{r, w, s\}) \rightarrow (q1, R1, R)\)

STATE q0

\[
\begin{array}{cccccccc}
 r & w & w & r & s & r & r & s \\
 1 & 4 & 1 & 1 & 4 & 34 & 4 & 5 \\
\end{array}
\]

R1=
Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - \((q_0,\{r,w,s\}) \rightarrow (q_1,R_1,R)\)
  - \((q_1,R_1,\{r,w,s\}) \rightarrow (q_f,S)\)
  - \((q_1,\{r,w,s\}) \rightarrow (q_1,R_1,R)\)

STATE q0
R1=

<table>
<thead>
<tr>
<th></th>
<th>r</th>
<th>w</th>
<th>w</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>34</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - \((q0,\{r,w,s\}) \rightarrow (q1,R1,R)\)
  - \((q1,R1,\{r,w,s\}) \rightarrow (qf,S)\)
  - \((q1,\{r,w,s\}) \rightarrow (q1,R1,R)\)

<table>
<thead>
<tr>
<th>STATE</th>
<th>(q1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1=1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r</th>
<th>w</th>
<th>w</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>34</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - \((q_0, \{r, w, s\}) \rightarrow (q_1, R_1, R)\)
  - \((q_1, R_1, \{r, w, s\}) \rightarrow (q_f, S)\)
  - \((q_1, \{r, w, s\}) \rightarrow (q_1, R_1, R)\)

STATE q1

\[
\begin{array}{cccccccc}
  r & w & w & r & s & r & r & s \\
  1 & 4 & 1 & 1 & 4 & 34 & 4 & 5 \\
\end{array}
\]

R1=1
Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value

- $(q_0, \{r, w, s\}) \rightarrow (q_1, R_1, R)$
- $(q_1, R_1, \{r, w, s\}) \quad (q_f, S)$
- $(q_1, \{r, w, s\}) \quad (q_1, R_1, R)$

\[
\begin{array}{cccccccc}
  r & w & w & r & s & r & r & s \\
  1 & 4 & 1 & 1 & 4 & 34 & 4 & 5 \\
\end{array}
\]
Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - \((q_0, \{r, w, s\}) \rightarrow (q_1, R_1, R)\)
  - \((q_1, R_1, \{r, w, s\}) \quad (q_f, S)\)
  - \((q_1, \{r, w, s\}) \quad (q_1, R_1, R)\)

STATE q1

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th></th>
<th>1</th>
<th>1</th>
<th>4</th>
<th>34</th>
<th>4</th>
<th>5</th>
<th>5</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td></td>
<td></td>
<td>w</td>
<td></td>
<td>w</td>
<td></td>
<td>r</td>
<td>s</td>
<td>r</td>
<td></td>
<td>r</td>
</tr>
<tr>
<td>w</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>34</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

R1=4
Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - \((q_0, \{r, w, s\}) \rightarrow (q_1, R_1, R)\)
  - \((q_1, R_1, \{r, w, s\}) (q_f, S)\)
  - \((q_1, \{r, w, s\}) (q_1, R_1, R)\)

**STATE**

- \(q_1\)

**R1=1**

<table>
<thead>
<tr>
<th></th>
<th>r</th>
<th>w</th>
<th>w</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>34</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - \((q0,\{r,w,s\}) \rightarrow (q1,R1,R)\)
  - \((q1,R1,\{r,w,s\}) \ (qf,S)\)
  - \((q1,\{r,w,s\}) \ (q1,R1,R)\)

STATE q1

R1=1

<table>
<thead>
<tr>
<th></th>
<th>r</th>
<th>w</th>
<th>w</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>34</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Register Automata 3/4

Language of data strings were two adjacent positions contain the same data value

- \((q_0,\{r,w,s\}) \rightarrow (q_1,R_1,R)\)
- \((q_1,R_1,\{r,w,s\}) (q_f,S)\)
- \((q_1,\{r,w,s\}) (q_1,R_1,R)\)

STATE \(q_f\)

\[ R_1=1 \]

DATA STRING ACCEPTED
The **Register Automata (RA)** in the previous example is called one-way deterministic.

Based on the restrictions we impose RAs can be:
- Deterministic, Nondeterministic or Alternating
- One way or Two way

Each of the above choices affects the expressiveness and the decidability of the model.
### What do we have so far?

<table>
<thead>
<tr>
<th></th>
<th>Emptiness</th>
<th>Universality</th>
<th>Inclusion</th>
<th>Equivalence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1Way–ND</strong></td>
<td><strong>Decidable</strong></td>
<td>Undecidable</td>
<td>Undecidable</td>
<td>Undecidable</td>
</tr>
<tr>
<td>Register</td>
<td>Automata</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>2Way–D</strong></td>
<td></td>
<td>Undecidable</td>
<td>Undecidable</td>
<td>Undecidable</td>
</tr>
<tr>
<td>Register</td>
<td>Automata</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Maybe we should slow down a little bit...

Let's put PAs aside and try to improve **1Way-ND RAs**
Generalizing a Little Bit...

- **1Way-ND-RAs** are limited in expressiveness: they can't even represent the language of strings where all the data values are different!!

- We need a slightly more expressive model...but not too expressive otherwise we hit undecidability

- **Weakness of RAs**: they can only talk about global properties but not about local properties:
  - **GLOBAL**: property of the whole string
  - **LOCAL**: all label of position with same data have some property
Class Memory Automata 1/3

- More expressive than Register Automata :)  
- Single pass and one way!! :(  
- Non-deterministic :(  
- At every point transitions depend on class history  
- Let's see an example...
Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. **For every q**,
  - \((q,\{r,w,s\},-) \rightarrow q1\)
  - \((q,\{r,w,s\},q1) \rightarrow q2\)
  - \((q,\{r,w,s\},q2) \rightarrow q2\)

- Global acceptance \(\{q0,q1,q2\}\), local acceptance \(\{q1\}\)
Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. For every q,
  - \((q,\{r,w,s\},-) \rightarrow q1\)
  - \((q,\{r,w,s\},q1) \rightarrow q2\)
  - \((q,\{r,w,s\},q2) \rightarrow q2\)

- Global acceptance \(\{q0,q1,q2\}\), local acceptance \(\{q1\}\)
Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. For every q,
  - \((q,\{r,w,s\},-) \rightarrow q1\)
  - \((q,\{r,w,s\},q1) \rightarrow q2\)
  - \((q,\{r,w,s\},q2) \rightarrow q2\)
- Global acceptance \(\{q0,q1,q2\}\), local acceptance \(\{q1\}\)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>q1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>r</th>
<th>w</th>
<th>w</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>34</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. For every q,
  - \((q, \{r, w, s\}, -) \rightarrow q1\)
  - \((q, \{r, w, s\}, q1) \rightarrow q2\)
  - \((q, \{r, w, s\}, q2) \rightarrow q2\)

- Global acceptance \{q0, q1, q2\}, local acceptance \{q1\}

<table>
<thead>
<tr>
<th></th>
<th>r</th>
<th>w</th>
<th>w</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>34</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>q1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>q1</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>34</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
</tr>
</tbody>
</table>
Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. For every $q$,
  - $(q,\{r,w,s\},-) \rightarrow q_1$
  - $(q,\{r,w,s\},q_1) \rightarrow q_2$
  - $(q,\{r,w,s\},q_2) \rightarrow q_2$

- Global acceptance $\{q_0,q_1,q_2\}$, local acceptance $\{q_1\}$
Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. For every q,
  - $(q,\{r,w,s\},-) \rightarrow q1$
  - $(q,\{r,w,s\},q1) \rightarrow q2$
  - $(q,\{r,w,s\},q2) \rightarrow q2$

- Global acceptance $\{q0,q1,q2\}$, local acceptance $\{q1\}$
Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. For every q,
  - \((q,\{r,w,s\},-) \rightarrow q_1\)
  - \((q,\{r,w,s\},q_1) \rightarrow q_2\)
  - \((q,\{r,w,s\},q_2) \rightarrow q_2\)
- Global acceptance \(\{q_0,q_1,q_2\}\), local acceptance \(\{q_1\}\)
Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. For every q,
  - \((q,\{r,w,s\},-) \rightarrow q_1\)
  - \((q,\{r,w,s\},q_1) \rightarrow q_2\)
  - \((q,\{r,w,s\},q_2) \rightarrow q_2\)

- Global acceptance \(\{q_0,q_1,q_2\}\), local acceptance \(\{q_1\}\)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>34</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>q2</td>
<td>q2</td>
<td>q1</td>
<td>q2</td>
<td>q2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r</th>
<th>w</th>
<th>w</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>34</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

**DATA STRING REJECTED**
This example can't be computed by a 1WAY-ND-RA and in general

- 1WAY-ND-RA are strictly included in CMA's for what concern expressiveness

- Emptiness is still decidable for CMAs!!!

- Membership was easy now is NP-Complete :(
Why is this model right?

- Class memory automata are usable :) 
- More expressive than 1Way-ND-Register Automata :) 
- Still decidable emptiness :) 
- But who tells us there is no better model?? :( 
- Regular string languages are equivalent to MSO
  - Models equivalent to a logic are more robust,
  - They have closure properties for free

- We need a declarative model equivalent to CMA...
Data Languages

- Motivation
- Data Model
- **Data Strings**
  - Automata and **Logics**
  - Regularity
- Data Trees
- Conclusion
First-Order Logic for Data Strings

- **Variables** range over positions of the data string
- **Formulae** of the form

\[ \Phi ::= a(x) \mid x=y \mid \exists x. \Phi \mid \Phi \lor \Phi \mid \neg \Phi \mid x=y+1 \mid x<y \mid x\sim y \]

- We call this logic \(\text{FO}(+1,<,\sim)\)
- Here is a simple formula for the language of data strings with all different data values

\[ \neg \exists x. \exists y. \neg(x=y) \land x\sim y \]
Monadic Second-Order Logic for Data Strings

- **Variables** range over **positions** of the data string
- **Second-order variables** range over **sets of positions**
- **Formulae** of the form

\[ \Phi ::= a(x) \mid x=y \mid \exists x.\Phi \mid \Phi \lor \Phi \mid \neg \Phi \mid x<y \mid x=y+1 \mid x \sim y \mid \exists X.\Phi \mid x \in X \]

- We call this logic **MSO(+1,<,~)**
- Here is a formula for the language of data strings with at least one position with label \( a \)

\[ \exists X. \exists x. x \in X \land a(x) \]
Bad news...

- **Emptiness**: Given a formula $\Phi$ in FO(+1,\,<,\,\sim) or MSO(+1,\,<,\,\sim) checking whether there exists a data string $s$ that is a model of $\Phi$ is undecidable.

- Too much expressiveness, we need something weaker.

- Maybe we can consider a logic where the number of variables that can be used is limited...
Two-Variable Logics 1/3

• We call $\text{FO}_2$ first-order logic with only two variables $x,y$

• In general adding variables adds expressiveness
  
  – $\text{FO}_K(+1,<,\sim)$ is less expressive than $\text{FO}(K+1)(+1,<,\sim)$

• And...
  
  – Emptiness for a formula in $\text{FO}_2(...)$ is decidable
  
  – Emptiness for a formula in $\text{FO}_3(...)$ is undecidable

• This is a nice cut-off!!
Two-Variable Logics 2/3

• We can actually do a bit better

• We call $\text{EMSO2}(+1, <, \sim)$, monadic second order logic where all the set variables $X$ are existentially quantified and they appear at the beginning of a formula

  - $\exists X_1 \ldots \exists X_n. \Phi$ and $\Phi$ only contains $\text{FO2}(\ldots)$

• And..

  - Emptiness for a formula in $\text{EMSO2}(\ldots)$ is decidable

  - $\text{EMSO2}(\ldots)$ is strictly more expressive than $\text{FO2}(\ldots)$
Two-Variable Logics 3/3

- We can push the decidability even a bit further
- Consider the operator $\oplus 1$ such that:
  - $x = y \oplus 1$ iff $y$ is the next position after $x$ with the same data value of $x$
- We call $\text{EMSO2}(+1, <, \sim, \oplus 1)$, $\text{EMSO2}(+1, <, \sim)$ enriched with the $\oplus 1$ operator
- Emptiness for a formula in $\text{EMSO2}(+1, <, \sim, \oplus 1)$ is decidable
- $\text{EMSO2}(+1, <, \sim, \oplus 1)$ is strictly more expressive than $\text{EMSO2}(+1, <, \sim)$
Incredible but true

CLASS MEMORY AUTOMATA

EMSO2(+1,<,~,⊕1)

all have the same expressiveness
Data Languages

- Motivation
- Data Model
- **Data Strings**
  - Automata and Logics
  - **Regularity**
- Data Trees
- Conclusion
It seems that class memory automata are a good model but it still not regular in the standard sense.

<table>
<thead>
<tr>
<th></th>
<th>Regular String Languages</th>
<th>Class Memory Automata</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalence</td>
<td>Decidable</td>
<td>Undecidable</td>
</tr>
<tr>
<td>Emptiness</td>
<td>Decidable</td>
<td>Decidable</td>
</tr>
<tr>
<td>Deterministic</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Closure under intersection, union, star</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Closure under complement</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Logical counter part</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Membership</td>
<td>Linear</td>
<td>NP-Complete</td>
</tr>
</tbody>
</table>
Regularity 2/2

• Unfortunately this is the best we have so far... so:
  
  – We can look for a **less expressive** model maybe **equivalent to** some form of FO so that we have closure under complement
  
  – We accept that Data Languages are **hard to work with** and we give up on some properties

• In any case, what defines a **regular string data language** is still and **open problem**
Data Languages

- Motivation
- Data Model
- Data Strings
  - Automata and Logics
  - Regularity
- Data Trees
- Conclusion
Data Trees

- Similarly, in a data tree each **node** carries
  - a **label** from a *finite alphabet* and
  - a **data value** from an *infinite alphabet*

Messages

```
<table>
<thead>
<tr>
<th>Id</th>
<th>From</th>
<th>To</th>
<th>Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>501</td>
<td>Mary</td>
<td>Tom</td>
<td>Prepare dinner!</td>
</tr>
<tr>
<td>502</td>
<td>Tom</td>
<td>Mary</td>
<td>Will do!</td>
</tr>
</tbody>
</table>
```
Models for Data Trees

- The **state of the art** for automata and logics over data trees is **embryonic**.
- Very few studied models with decidable properties:
  - Some extensions of **register automata**, but without intuitive semantics and limited expressiveness.
  - Extensions of **FO2(+1, <, ~)**, **EMSO2(+1, <, ~)** to data trees.
- Logic is more relevant for regularity, so let's concentrate on that.
In a tree the predicates +1 and < don't have a clear interpretation...

We replace them with two new predicates

- $x = y \downarrow 1$ parent relation and
Two-Variable Logics 1/3

- In a tree the predicates +1 and < don't have a clear interpretation...
- We replace them with two new predicates
  - \( x = y \downarrow \) 1 parent relation and
  - \( x = y \rightarrow 1 \) next sibling relation
Two-Variable Logics 2/3

• Now the \(+1\) predicate corresponds to ↓ and →

• The \(<\) predicate corresponds to the transitive closures of ↓ and →, ↓* and →*

• We now have an interpretation for
  – FO2(+1,<,~) and EMSO2(+1,<,~) over data trees

• But we can also consider simpler logics where we drop the \(<\) operator
  – FO2(+1,~) and EMSO2(+1,~) over data trees

• But why would we do that??
Two-Variable Logics 3/3

- **Emptiness** for
  - $\text{FO}_2(+1,\sim)$ and $\text{EMSO}_2(+1,\sim)$ over data trees is **decidable**
  - We will see a proof sketch

- **Emptiness** for
  - $\text{FO}_2(+1,<,\sim)$ and $\text{EMSO}_2(+1,<,\sim)$ over data trees is a very **hard open problem**
  - **Emptiness** for **vector addition tree automata** reduces to it...
Vector Addition Tree Automata

- **Bottom up** tree automata over binary trees in which transitions have three vectors
  - Every leaf is assigned a vector of values over Nat
  - Transitions of the form $q_1, q_2, a, b, c, l \rightarrow q$

- Is there a run in which root is labeled with the vector 0

- Very hard open problem reduces to emptiness for $\text{FO2}(+1, <, \sim)$
Emptiness for FO(+1,~) is Decidable 1/4

- Proof outline, given a formula $F$ in FO(+1,~)
  - Compute a "puzzle" $P$ that has solutions iff $F$ is satisfiable
  - If a "puzzle" $P$ has a solution, then there also exists a "small" solution
  - Find if there exists a "small" solution of $P$ using some extended tree automata
Emptiness for FO(+1,~) is Decidable 2/4

- Compute a "puzzle" P that has solutions iff F is satisfiable
  - Reduce F to a normal form F'
  - For every formula F' we can compute
- A puzzle over Σ is a pair (L,F)
  - L is a tree automata over an extension of Σ
  - F is a set of accepting pairs (D,S) where D,S are disjoint subsets of Σ
  - Solution: a data tree "in" L where for each class: there exists pair (D,S) such that all labels are from D U S and each label in D appears at most once

Class:
Maximal set of connected nodes with the same data value
Emptiness for FO(+1,~) is Decidable 3/4

- If a "puzzle" P has a solution, then there exists a "small" solution
  - Let's assume there is a solution
- A (M,N)-reduced solution is a solution
  - At most M classes are of size greater than N
  - There are at most M sibilinghoods with more than N classes
- Given a solution we can compute a (M,N)-reduced solution and
  - M and N are effectively computable numbers from the size of the puzzle
Emptiness for FO(+1,~) is Decidable 4/4

- **Find** if there exists a "small" solution of P using some extended tree automata

- Given a tree automata A and a tree t, a run of A can be seen as a labeling t with the states Q of A
  - Linear constraint tree automata (LCTA) are equipped with a linear constraint K over Q
  - A run accepts if K is satisfied when instantiated with the number of states in the run
  - Emptiness of LCTA is decidable

- We can compute a LCTA that is empty iff the puzzle P does not have (M,N)-reduced solutions
Regularity

- Talking about regularity for data tree languages doesn’t make quite sense
- Only one very limited logic with decidable emptiness
- No automata equivalent model
- No proof of undecidability for more expressive logic
- Very little research so far
Conclusion

- Motivation
- Data Model
- Data Strings
  - Automata and Logics
  - Regularity
- Data Trees

Conclusion
We have seen...

- First proposals for automata models for string data languages (RA)
- Improved automata models with equivalent logical counterpart (CMA)
- Most relevant logics for data languages (FO2, EMSO2)
- Basic (only) result on decidability of FO2 for data trees
- Some discussion on what regularity might be for data languages (no hope for full package)
Open Problems

- Is there a variant of RA closed under complementation?
- Is there a variant of FO2 equivalent to 1WAY-ND-RA?
- Is there a logic closed under complementation with decidable equivalence?
- Can we extend current data strings models to data trees preserving good properties?
- Is FO2(+1,<,~) decidable for data trees?
References

- Frank Neven... Finite state machines for strings over infinite alphabets. 2004
- Mikolaj Bojanczyk... Two-variable logic on words with data. 2006
- Mikolaj Bojanczyk... Two-variable logic on data trees and XML reasoning. 2006
- Henrik Bjorklund... On notions of regularity for data languages. 2007
- .....
Thank you...
Questions?