A little bit infinite?
Adding data to finitely labelled structures

Thomas Schwentick
Grantown-on-Spey
July 2008
Contents

Introduction
- Motivation from XML
- Motivation from Verification
- Data Model
- Automata
- Logic
- Other Models
- Conclusion
Contents

Introduction
  ▶ Motivation from XML
    Motivation from Verification
  Data Model
  Automata
  Logic
  Other Models
  Conclusion
### Relational Databases

#### Composers from Southwest

<table>
<thead>
<tr>
<th>COMPOSERS</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Birth</td>
<td>Death</td>
</tr>
<tr>
<td>Ravel</td>
<td>Ciboure</td>
<td>Paris</td>
</tr>
<tr>
<td>Tournemire</td>
<td>Bordeaux</td>
<td>Arcachon</td>
</tr>
</tbody>
</table>

#### Pieces

<table>
<thead>
<tr>
<th>PIECES</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Comp</td>
<td>Year</td>
<td>Instr</td>
</tr>
<tr>
<td>Boléro</td>
<td>Ravel</td>
<td>1928</td>
<td>Orch.</td>
</tr>
<tr>
<td>Douze Préludes</td>
<td>Tournemire</td>
<td>1932</td>
<td>Piano</td>
</tr>
<tr>
<td>La Valse</td>
<td>Ravel</td>
<td>1920</td>
<td>Orch.</td>
</tr>
</tbody>
</table>

#### SELECT

```sql
SELECT B.Name, B.Comp
FROM Composers A, Pieces B
WHERE A.Name = B.Comp AND A.Birth = "Bordeaux"
```

#### Relational data: flat structure & data

- Queries rely on **structure** and **equality of data items**:
  
  $$Q(x_1, x_2) \equiv \exists x_3, \ldots, x_5, y_1 \ldots, y_3$$
  
  $$\text{Pieces}(x_1, x_2, x_3, x_4, x_5) \land$$
  
  $$\text{Composers}(y_1, y_2, y_3) \land$$
  
  $$y_1 = x_2 \land y_3 = \text{"Bordeaux"}$$

#### Integrity Constraints rely on **structure** and **equality of data items**:

$$\forall x_1, \ldots, x_5, y_1, \ldots, y_5$$

$$(x_1 = y_1 \land x_2 = y_2) \rightarrow$$

$$(x_3 = y_3 \land x_4 = y_5 \land x_5 = y_5)$$

A little bit infinite? Thomas Schwentick
Example Tree

Composer

Name
Maurice Ravel

Vita
Born

When
1875

Where
Ciboure

Died

When
1937

Where
Paris

Piece

PTitle
Boléro

PYear
1928

Instruments
Orchestra

Movements
1

Piece

PTitle
La Valse

PYear
1920

Instruments
Orchestra

Movements
1

Composer

Name
Charles Tournemire

Vita

Born

When
1870

Where
Bordeaux

Died

When
1939

Where
Arcachon

Piece

PTitle
Douze préludes poèmes

PYear
1932

Instruments
Piano

Movements
12
XML: hierarchical structure & data

Data model: an XML document can be viewed as an unranked tree in which
- inner nodes correspond to elements
- leaves correspond to data (attributes, text content)

For many investigations,
- the set of tags is restricted
- data values can be ignored

Abstraction: labeled trees over a finite alphabet

Works well for foundational studies on many aspects of
- Validation
- Navigation
- Transformation

Foundational research on XML has largely ignored data but concentrated on finitely labeled trees
There is a need for data-aware foundational XML research:

- **Schemas:**
  - Schemas for XML describe the allowed structure of documents and can specify constraints on the data
  - Structure constraints can be captured by regular tree languages (automata & logics available)
  - Data constraints include uniqueness, keys, foreign keys

- **XPath:**
  - The core of XPath allows to specify navigational queries (automata & logics available)
  - But: it also allows comparisons between data

- **Other data-aware processing tasks:**
  - Querying: XQuery
  - Transformations: XSLT
  - Data Exchange [Arenas, Libkin 05]
An example scenario: **XML Query optimization**

- Algorithmic problem:
  - Given XPath expressions $q_1$, $q_2$ and a schema $S$
  - Decide whether, for each valid document $d$ (wrt $S$):
    $$q_1(d) \subseteq q_2(d)$$

- The XPath queries might combine navigation with conditions on data values:
  - $q_1$: select all composers who wrote a piece in the year they died
  - $q_2$: select all composers whose name is unique

- The schema $S$ might consist of
  - structural constraints $\rightarrow$ regular tree language $L$
  - and data integrity constraints (e.g.: each composer name occurs at most once)

- Most of XPath navigation can be modelled by two-variable logic

- **How to deal with data?**
Contents

**Introduction**
- Motivation from XML
- Motivation from Verification

Data Model
Automata
Logic
Other Models
Conclusion
A Toy Example from Verification

A printer and two processes

- Example properties that might to be checked:
  - "Local property": processes never request a new print job before the last one has terminated, i.e.: for each $i$ the subrun is of the form $(r_i s_i t_i)^*$,
  - "Global property": a print job must be finished before the next one is started, i.e.: between a $s_i$ and the subsequent $t_i$ there is no $s_j$ or $t_j$, $j \neq i$

Memory Allocation

- "Local property": A memory location should only be accessed after it is allocated and before it is freed
- $k$ processes give rise to $3^k$ states (→ “state explosion”)

What if the number of processes is unknown?
What if the number of processes changes during the computation?
The Automata Approach to Model Checking

- **Model checking**:
  - System: $M$
  - Property: $\varphi$
  - Does $M \models \varphi$?

- **The automata approach**:
  - 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 (e.g., LTL/CTL) that allows translation into automata
  - Use decidability of non-emptiness for automata to obtain decidability of model checking

- **But sometimes the finite state space approach does not really work**

- **Sources of infinity in software systems**:
  - **Data manipulation**: integers, lists, trees, more general pointer structures
  - **Control structures**: procedures, process creation
  - **Asynchronous communication**: unbounded FIFO queues
  - **Parameters**: number of processes, duration of delays
  - **Real-time**: discrete or dense domains

- [Esparza]

- There is a huge need for **Model Checking of infinite-state systems**
Current Approaches to Infinite-State Model Checking

- Infinite-State Model Checking has been an active and successful research area for many years.

- **Typical approach (in a nutshell):**
  - Describe system states by some finite objects (strings, tuples of parameters).
  - Describe possible transitions from state to state.
  - Device algorithms for checking reachability and/or repeated reachability.

- **Examples:**
  - Timed automata [Alur, Dill 90]
  - Mutual exclusion protocols [Abdulla et al. 07]
  - Regular model checking [Bouajjani et al. 00]

- **Achievements:**
  - Model checking of linear time properties is in many cases possible.

- **Still missing:**
  - Inter-state reasoning about data from infinite domains (e.g., for each $i$, each $r_i$ is followed by some $s_i$, for an unlimited number of processes).
  - A generic framework for branching-time properties.
A unifying approach

- There are obvious similarities between the XML and the infinite-state model checking scenario:
  - Traditional modeling uses finitely labeled structures:
    - strings, trees, Kripke structures
  - There is a need to add data from infinite domains to the positions/nodes of such structures
  - It should be possible to reason about inter-node relationships between data items

- A possible unifying approach:
  - Enhance finitely labeled structures by data
  - Various possibilities:
    * One (or more) relations per node
    * A vector of data values per node
    * One data item per node
    * ...and many more

- Parameters to choose:
  1. Underlying finitely labeled structures
  2. Amount and structure of data per node
  3. Operations and predicates on data
  4. Expressiveness of specification language

- Limitations:
  - To avoid undecidability of reasoning, parameters (1) - (4) have to be chosen very carefully

- Related work:
  - [Autebert et al. 80]
  - [Otto 85]: Regular and context-free languages over infinite alphabets (Symbols have structure)
  - [Henzinger 90]: Kripke structures with one data value per word
  - [Kaminski, Francez 90]: Strings over an infinite alphabet
  - More related work will be mentioned later
Data Strings and Data Trees

• In this talk:
  – We fix the structure and data parameters:
    (1) Finite or infinite strings or trees as underlying finitely labeled structure
    (2) One data item per node/position
    (3) Only equality tests between data items
  – We try to find (4) expressive and decidable reasoning/specification mechanisms

Example: data string

```
  r  r  s  r  r  t  r  s  t  s  r  t  s  t  s  t
  2  5  5  3  8  5  5  2  2  8  4  8  3  3  4  4  5  5
```

Definition [Bouyer et al. 03]

• Data string: Finite sequence over $\Sigma \times D$, where
  – $\Sigma$ finite (here: $\{r, s, t\}$)
  – $D$ infinite (here: $\mathbb{N}$)
Regular String Languages

- Data strings extend strings
- **Regular string languages** are a very powerful concept:
  1. **Expressiveness**: They capture the desired languages for many kinds of applications
  2. **Decidability**: Automated semantic analysis possible through automata
  3. **Efficiency**: Model checking in linear time.
  4. **Closure properties**: It is hard to find a simple natural operation under which they are not (effectively) closed
  5. **Robustness**: Tons of characterizations

→ Regular string languages offer an ideal framework to deal with string languages:
  - Declarative specifications...
  - ..can be translated into automata...
  - ...which can be efficiently
    * evaluated,
    * manipulated and
    * analyzed semantically

- **Furthermore**: There exist canonical generalizations of regular languages for a variety of data types:
  - Infinite strings, (infinite) trees, pictures,...

→ **Obvious question**: 
  - Is there a corresponding canonical concept of “regular data languages”?

A little bit infinite? Thomas Schwentick tu.
Regular Data Languages?

- **Bad news:** There does **not** seem to be a canonical notion of regular data languages

- **Good news:** We can mimic the regular languages framework:
  - Declarative specifications...
  - ...can be translated into automata...
  - ...which can be **effectively**
    * evaluated,
    * manipulated
    * analyzed semantically

- **This talk is about the search for a good framework to deal with (string or tree) data languages:**
  - Automata for data languages
  - Logic-based specification languages
  - Their (potential) use for XML and Model Checking
  - Other approaches
### Example properties of data strings

#### Example

<table>
<thead>
<tr>
<th>r</th>
<th>r</th>
<th>s</th>
<th>r</th>
<th>t</th>
<th>r</th>
<th>s</th>
<th>t</th>
<th>s</th>
<th>t</th>
<th>s</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

A **class** with **class string** \( rstrst \)

#### Examples

(L1) No two \( a \)-positions do have the same data value  
(*unary key constraint*)

(L2) There are two \( a \)-positions with the same data value

(L3) For each \( a \)-position there is a \( b \)-position with the same data value  
(*unary inclusion constraint*)

(L4) A print job of a user has to be printed before the next one can be requested  
(“local safety”)

(L5) Each print request of a user is eventually followed by a print  
(“local liveness”)

\[ \rightarrow \]  (L1) - (L5) are **“local properties”** of the class strings

(L6) Between two successive print jobs of the same user some other user’s job has to be printed  
(“global safety”)

(L7) After each printed job a job of some other user is eventually printed  
(“global liveness”)

A little bit infinite? Thomas Schwentick
Contents

Introduction
Data Model

Automata

Register Automata

Pebble Automata
Class Memory Automata
Alternating Register Automata

Logic

Other Models

Conclusion

A little bit infinite? Thomas Schwentick tu
Register Automata (1/4)

- **A natural idea:**
  Equip finite automata with registers that can store data values

  ➞ **Register Automata**

- (“Finite Memory Automata” in [Kaminski, Francez 90], but w/o labels)

**Example**

- Example automaton for (L6): **Between two successive print jobs of the same user some other user’s job has to be printed**

- Stated differently:
  **No two successive** \(s\)-positions **carry the same data value**

- Solution: store the data value of the previous \(s\)-position in register 1 and check that it does not occur at the next \(s\)-position

\[
\begin{array}{cccccccccccccc}
  r & r & s & r & r & t & r & s & t & s & r & t & s & t & s & t & s & t \\
  2 & 5 & 5 & 3 & 8 & 5 & 5 & 2 & 2 & 8 & 4 & 8 & 3 & 3 & 4 & 4 & 5 & 5 \\
\end{array}
\]

\[
\begin{array}{cc}
  R_1 \\
  R_2 \\
\end{array}
\]
<table>
<thead>
<tr>
<th>Theorem 1 [Kaminski, Francez 90]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Non-emptiness for register automata is decidable</td>
</tr>
<tr>
<td>(b) Testing $L(A_1) \subseteq L(A_2)$ is decidable as long as $A_2$ has $\leq 2$ registers</td>
</tr>
</tbody>
</table>

**Proof idea**

| (a) Crux: if there is a string in $L(A)$, then there is one with $\leq |Q| + 1$ different data values |

- There is a subtle difference between register automata models:
  - (1) [Demri, Lazić 06]: data values can occur in more than register
  - (2) [Kaminski, Francez 90]: they cannot
- Model (1) can simulate a Turing machine with $n$ cells and alphabet size $k$ with $n + k$ registers
  - $\Rightarrow$ Non-Emptiness is PSPACE-complete
- If a model (2) $k$-register 1RA accepts any word it accepts a word of the same length with $\leq k$ data values
  - $\Rightarrow$ Non-Emptiness is NP-complete

<table>
<thead>
<tr>
<th>Theorem 2 [Kaminski, Francez 90]</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Universality, i.e., testing whether a register automaton accepts every data string is undecidable</td>
</tr>
</tbody>
</table>
• Register automata can test global regular properties
  – That’s simple: just ignore the data values

## Theorem 3

• No register automaton can test (L4):
  “A print job of a user has to be printed before the next one can be requested”

## Proof idea

• Assume some 3-register automaton \( A \) tests (L4)
• Consider the following input:

\[
\begin{array}{cccccccc}
    r & r & r & r & r & R_1 & 4 \\
    1 & 2 & 3 & 4 & 1 & R_2 & 2 \\

\end{array}
\]

• \( A \) cannot detect that process 1 has a pending print job

\[\Rightarrow\] Easy to generalize for arbitrary number of registers
Register Automata (4/4)

- Summary of properties of register automata:

<table>
<thead>
<tr>
<th></th>
<th>RegisterA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expressiveness</strong></td>
<td>(L2),(L6),(L7)</td>
</tr>
<tr>
<td><strong>Decidability</strong></td>
<td></td>
</tr>
<tr>
<td>Non-emptiness</td>
<td>✓</td>
</tr>
<tr>
<td>Containment</td>
<td>–</td>
</tr>
<tr>
<td><strong>Efficiency</strong></td>
<td></td>
</tr>
<tr>
<td>Data complexity word problem</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Closure properties</strong></td>
<td></td>
</tr>
<tr>
<td>Union</td>
<td>✓</td>
</tr>
<tr>
<td>Intersection</td>
<td>✓</td>
</tr>
<tr>
<td>Complement</td>
<td>–</td>
</tr>
<tr>
<td><strong>Robustness</strong></td>
<td></td>
</tr>
</tbody>
</table>

- Variants of the basic RA model:
  - 1-way and 2-way
  - Deterministic and non-deterministic
  - Alternating
    [Neven et al. 01, Demri Lazić 06]
  - Look-ahead automata [Zeitlin 06]
  - “Unification based” [Tal 99]

A little bit infinite? Thomas Schwentick
Contents

Introduction
Data Model
Automata
Register Automata
Pebble Automata
Class Memory Automata
Alternating Register Automata
Logic
Other Models
Conclusion

A little bit infinite? Thomas Schwentick
Pebble automata (1/3)

- A different approach: instead of registers use pebbles (pointers/heads)
- Restrict movement and placement of pebbles:
  - Pebbles are numbered $1, 2, \ldots, k$
  - Only pebble with highest number $i$ can be moved or lifted
  - Only pebble with number $i + 1$ can be placed

Example automaton for (L6):

Between two successive print jobs of the same user some other user's job has to be printed

Again stated differently: no two successive $s$-positions carry the same data value

Solution: for each $s$-position check that the previous $s$-position has a different data value
Pebble automata are a fairly powerful model:
- E.g., they can express all example properties (L1) – (L7)
- They can even express all properties that can be described by first-order logic
- Unfortunately: first-order logic on data strings is undecidable (see below)

Non-emptiness of pebble automata is undecidable

On the other hand the model is quite robust:
- one-way and two-way, deterministic and non-deterministic pebble automata are equally expressive

<table>
<thead>
<tr>
<th></th>
<th>RegisterA</th>
<th>PebbleA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expressiveness</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(L2), (L6), (L7)</td>
<td>(L1) – (L7)</td>
</tr>
<tr>
<td><strong>Decidability</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-emptiness</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>Containment</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Efficiency</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data complexity word pr.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Closure properties</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Union</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intersection</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Complement</td>
<td>–</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Robustness</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>✓</td>
</tr>
</tbody>
</table>
Pebble Automata (3/3)

(from Neven/Sch./Vian...)

A little bit infinite? Thomas Schwentick
A little bit infinite? Thomas Schwentick

Contents

Introduction
Data Model
Automata
  Register Automata
  Pebble Automata
  Class Memory Automata
    Alternating Register Automata
Logic
Other Models
Conclusion
Class Memory Automata (1/5)

- **Intermediate state of affairs:**
  - **Register Automata:**
    - Decidable Non-emptiness: 😊
    - Not expressive enough: 😞
  - **Pebble Automata:**
    - Very expressive: 😊
    - Undecidable Non-emptiness: 😞

- **New approach:**
  - Combine a global automaton with one automaton per class
  - More precisely:
    - Transitions depend on
      - the current input symbol (from the finite set of labels)
      - the current state
      - the state assumed last time in the class of the current input data value
  - The automaton accepts if
    - the last state is in an accepting set $F_g$
    - and for each class, the last state is in a set $F_l$

→ **Class Memory Automata**
  [Bojańczyk et al. 06, Björklund, Sch 07]
Class Memory Automata (2/5)

Example

- Class memory automaton for the set of data strings
  - with global pattern \((r^*sr^*t)^*\),
  - with local pattern \((rst)^*\) (for each class),
  - where at most one (singular) process prints more than once

- States are of the form \([p, q]\), where
  - \(p\) remembers whether the singular process already has appeared: \(n, y\)
  - \(q\) is just the last symbol, (dotted if from the singular process)

- At the end, where symbols are:
  - the last state should be of the form \([t, i]\) or \([\ddot{t}, i]\)
  - each class should have a last state of the form \([t, i]\) or \([\ddot{t}, i]\)
Class Memory Automata (3/5)

- Class memory automata can express all properties (L1) – (L7)
- Later on we will see a precise characterization of their expressive power in terms of logic

### Theorem 4

(a) Non-emptiness for class memory automata is decidable
(b) \( \text{RegA} \subsetneq \text{ClassMA} \)

- The complexity of Non-Emptiness for class memory automata is open
- But there is little doubt that it is extremely bad:
  - Equivalent to Petri Net Reachability
  - Not even known to be primitive recursive

### Proof idea for (a) [Bojańczyk et al. 06a]

- In a nutshell:
  - “Simulate” a class memory automaton \( \mathcal{A} \) by a (non-data) **Multicounter Automaton**:
    - String automaton \( \mathcal{A}' \) with several counters
    - \( \mathcal{A}' \) has one counter \( C_q \) per state \( q \) of \( \mathcal{A} \)
    - \( C_q \) counts the number of classes in state \( q \)
    - Zero tests are only needed at the end of the computation: \( C_p = 0 \), for \( p \not\in F_l \)
  - Non-emptiness for multi-counter automata is decidable

- And:
  \[
  L(\mathcal{A}) \neq \emptyset \iff L(\mathcal{A}') \neq \emptyset
  \]

A little bit infinite? Thomas Schwentick
Proof sketch for (b) [Björklund, S 07]

- **Strictness:** RAs can not express (L1)
- Isn’t $\text{RegA} \subseteq \text{ClassMA}$ obvious?
- Not entirely, consider (L6): **No two successive prints by the same process**
  - The register automaton for (L6) only needs one state plus a sink state.
  - How shall a ClassMA seeing $s_d$ know what happened since $s_d$ occurred last time?
- **Idea:** $A$ “colors” positions by $+,$ $+, -,$ $-,$ $-,$ $-,$ $-$ such that:
  - If an $s$-position has $+$ the next $s$-position has $-$ (and $-$ → $+$)
  - If an $s$-position has $+$ the next $s$-position in the same class has $+$

Proof sketch for (b) (cont.)

- **Of course:** if such a coloring exists, (L6) holds: the next $s$-position is never the next $s$-position in the same class.

- **If (L6) holds** such a coloring can be constructed by applying the following rules:
  1. If no other rule applies: assign $+$ to the rightmost $s$ without upper color.
  2. Whenever $+$ is assigned to an $s$-position assign $-$ to its left $s$-neighbour and $+$ to the left $s$-neighbour in its class.
  3. Whenever $+$ is assigned to an $s$-position assign $+$ to its right $s$-neighbour.

- **General proof of (b):** similar coloring trick.
### Class Memory Automata (5/5)

<table>
<thead>
<tr>
<th></th>
<th>RegisterA</th>
<th>PebbleA</th>
<th>ClassMA</th>
<th>DClassMA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expressiveness</strong></td>
<td>(L2),(L6),(L7)</td>
<td>(L1)–(L7)</td>
<td>(L1)–(L7)</td>
<td>(L1)–(L5),(L7)</td>
</tr>
<tr>
<td><strong>Decidability</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-emptiness</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Containment</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Efficiency</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data complexity word pr.</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Closure properties</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Union</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>Intersection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Complement</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Robustness</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
</tbody>
</table>
Inclusion structure of Automata Models

A little bit infinite? Thomas Schwentick
Contents

Introduction
Data Model
Automata
  Register Automata
  Pebble Automata
  Class Memory Automata
  Alternating Register Automata
Logic
Other Models
Conclusion

A little bit infinite? Thomas Schwentick  

Folie 34
Alternating Register Automata (1/2)

- How to turn register automata into a reasonably strong, robust and decidable model?
  - 1N-RA are pretty weak
  - 2D-RA are undecidable

- [Demri, Lazić 06]:
  - Alternating one-way register automata with one register: ARA₁

<table>
<thead>
<tr>
<th>Theorem 5 [Demri, Lazić 06]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Non-emptiness (and Containment) of ARA₁ on strings is decidable but not primitive recursive</td>
</tr>
<tr>
<td>(b) Non-emptiness of ARA₁ on ω-strings is undecidable (even with Muller acceptance)</td>
</tr>
</tbody>
</table>

- Safety ARA₁ reject only in the finite (and their complement languages are closed under adding suffixes)

<table>
<thead>
<tr>
<th>Theorem 6 [Lazić 06]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Non-emptiness of safety ARA₁ on ω-strings is EXPSPACE-complete</td>
</tr>
<tr>
<td>(b) Containment of safety ARA₁ on ω-strings is decidable but not primitive recursive</td>
</tr>
</tbody>
</table>

- ARA₁ can express all properties (L1)-(L7)
- ARA₁ can not remember two data values at a time
## Alternating Register Automata (2/2)

<table>
<thead>
<tr>
<th></th>
<th>RegisterA</th>
<th>PebbleA</th>
<th>ClassMA</th>
<th>DClassMA</th>
<th>ARA₁</th>
<th>Safe ARA₁</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expressiveness</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(L2),(L6),(L7)</td>
<td>(L1)–(L7)</td>
<td>(L1)–(L7)</td>
<td>(L1)–(L5),(L7)</td>
<td>(L1)–(L7)</td>
<td>(L1),(L4),(L6)</td>
</tr>
<tr>
<td><strong>Decidability</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-emptiness</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Containment</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Efficiency</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data complexity word pr.</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Closure properties</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Union</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intersection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Complement</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td><strong>Robustness</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
</tbody>
</table>

A little bit infinite? Thomas Schwentick | 

Folie 36
Contents

Introduction
Data Model
Automata

Logic
▷ Two-Variable Logics
  Temporal Logics
Other Models
Conclusion

A little bit infinite? Thomas Schwentick
**Logics for Data Strings/Trees**

- **Automata** offer an algorithmic framework
- **Logics** offer a framework for declarative specifications
- **We will consider:**
  - Restrictions of classical first-order logic
  - Extensions of temporal logics

### Logical language...

<table>
<thead>
<tr>
<th>Logical operator</th>
<th><strong>... for strings</strong></th>
<th><strong>... for trees</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>( a(x) )</td>
<td>Letter at position ( x ) is ( a \in \Sigma )</td>
<td>( a(x) ) Label of node ( x ) is ( a \in \Sigma )</td>
</tr>
<tr>
<td>(+1)</td>
<td>successor relation on positions</td>
<td>( E \rightarrow ) horizontal neighbor (&quot;next sibling&quot;)</td>
</tr>
<tr>
<td>(&lt;)</td>
<td>order relation on positions</td>
<td>( E \downarrow ) parent-child</td>
</tr>
<tr>
<td>( E \Rightarrow ) transitive closure of ( E \rightarrow )</td>
<td>( E \downarrow ) transitive closure of ( E \downarrow )</td>
<td></td>
</tr>
<tr>
<td>( \sim )</td>
<td>( x \sim y ) if positions ( x ) and ( y ) have the same ( \text{\textit{D}} )-value</td>
<td>( \sim ) ( x \sim y ) if nodes ( x ) and ( y ) have the same ( \text{\textit{D}} )-value</td>
</tr>
<tr>
<td>( \pm 1 )</td>
<td>next position in the same class</td>
<td></td>
</tr>
</tbody>
</table>
**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 7 [Bojańczyk et al. 06a]**

- 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 bit more detail**

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

- Each occurrence of a \(u_i\) is prefixed by \(i\):
  E.g., if \(u_1 = aba\) and \(u_2 = bb\) then \(121\) is encoded by \(1aba2bb1aba\)

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

  ➔ Important subformula:

  \[
  x \sim y \rightarrow \exists z \left(x + 1 = z \land \exists x \ y + 1 = x \land z \sim x\right)
  \]

  "if \(x\) and \(y\) are equivalent then their right neighbors are also equivalent"
Two Variables on Data Strings: A Useful Restriction?

- A classical approach: Restriction to 2 variables
- Does this restriction give us anything useful?
  1. We do not have free choice...
  2. lot of useful properties can be expressed with only two variables

Examples

(L1) No two $a$-positions do have the same data value
\[ \forall x \forall y (x \sim y \land a(x) \land a(y)) \rightarrow x = y \]

(L2) There are two $a$-positions with the same data value
\[ \exists x \exists y x \sim y \land a(x) \land a(y) \land x \neq y \]

(L3) For each $a$-position there is a $b$-position with the same data value
\[ \forall x \exists y a(x) \rightarrow (b(y) \land x \sim y) \]

(L4) A print job of a user has to be printed before the next one can be requested
\[ \forall x \forall y y = x \pm 1 \rightarrow [(r(x) \rightarrow s(s)) \land (s(x) \rightarrow t(y))] \]

(L5) Each print request of a user is eventually followed by a print
\[ \forall x \exists y r(x) \rightarrow (s(y) \land x < y \land x \sim y) \]

(L6) Between two successive print jobs of the same user some other user’s job has to be printed
\[ \text{not expressible} \]

(L7) After each printed job a job of some other user is eventually printed
\[ \forall x \exists y r(x) \rightarrow (s(y) \land x < y \land x \not\sim y) \]
Example

- $\varphi_a$:
  - $\forall x \forall y (x \sim y \land a(x) \land a(y)) \rightarrow x = y$
  - 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: $\psi_{b,a}$

$\varphi = \varphi_a \land \varphi_b \land \psi_{a,b} \land \psi_{b,a}$ implies:
  - 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

The string projection of an $\text{FO}^2$-definable data language need not be context-free
More example properties

- Let $\alpha$ and $\beta$ denote unary quantifier-free formulas ("types")
- $\text{FO}^2$ can express
  - data-blind properties, i.e., properties not using $\sim$
  - 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 $\beta$:
    \[ \theta = \forall x \exists y \left( \alpha(x) \rightarrow (\beta(y) \land x \sim y) \right) \]
  - If a position is in a different class than its successor it has type $\alpha$:
    \[ \theta = \forall x \forall y (\neg (x \sim y) \land x + 1 = y) \rightarrow \alpha(x) \]

- That’s basically all!

Theorem 8 [Bojańczyk et al. 06a]

Satisfiability of $\text{FO}^2(\sim, <, +1, \neq 1)$ on data strings is decidable
Proof Sketch for Theorem 8 (1/2)

Scott and intermediate normal form

- We transform two-variable formulas into satisfiability equivalent formulas of **existential monadic second-order logic**
- “Scott normal form”: $\exists R_1, \ldots, R_k \ \forall x \ \forall y \ \chi \ \land \ \bigwedge_i \forall x \exists y \ \chi_i$
- Intermediate normal form:
  $$\exists R_1 \cdots R_m \ \theta_1 \land \cdots \land \theta_n$$
- $\theta_i$: 
  
  1. $\forall x \forall y \ (\delta(x, y) \geq 2 \land \alpha(x) \land \beta(y) \land \begin{array}{c} x \sim y \ x \nless y \end{array}) \rightarrow \begin{array}{c} x < y \\ x > y \ \\ x \sim y \ x \nless y \end{array}$
  2. $\forall x \exists y \ \alpha(x) \rightarrow (\beta(y) \land \begin{array}{c} x + 1 < y \\ x + 1 = y \\ x = y \\ x = y + 1 \\ x > y + 1 \end{array} \land \begin{array}{c} x \sim y \\ x \nless y \end{array})$
  3. $\forall x \forall y \ \theta$ **(\theta quantifier-free, DNF, no \sim)**

- Both steps are straightforward
Data normal form & Class Memory Automata

• **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) Each class contains at most one $\alpha$
    (c) In each class, every $\alpha$ occurs before every $\beta$
    (d) Each class with an $\alpha$ also has a $\beta$
    (e) If $x$ is in a different class than its successor has type $\alpha$

• **Final Step:**
  - Each $\theta_i$ can be recognized by a Class Memory Automaton
  - Existential monadic quantification corresponds to nondeterminism in CMAs
  - CMAs are closed under union and intersection
  - Formulas in data normal form can be effectively translated into Class Memory Automata

• Decidability of $\text{FO}^2(\sim, <, +1, \pm 1)$ follows from decidability of Non-emptiness for Class Memory Automata

• Corollary: $\text{ClassMA} \equiv \text{EMSO}^2(\sim, <, +1, \pm 1)$
**FO^2 on Data Strings: Complexity**

- Complexitywise, Satisfiability of $\text{FO}^2(\sim, <, +1)$ is basically equivalent to Non-Emptiness of multicounter automata
  
  $\rightarrow$ Unknown complexity

- **Restrictions:**
  - $\text{FO}^2(\sim, <)$: complete for $\text{NEXPTIME}$ [David 04]
  - $\text{FO}^2(\sim, +1)$: in $3\text{NEXPTIME}$ [Bojańczyk et al. 06b]

- **Extensions:**
  - $+2, +3, \ldots$: same results
  - $\omega$-strings: same results
  - Linear order on data values: undecidable
**Theorem 9** [Bojańczyk et al. 06b]

For any vector addition tree automaton \( A \), a formula \( \varphi_A \in \text{FO}^2(\sim, <, +1) \) can be computed such that:

\[
L(A) \neq \emptyset \text{ iff } \varphi_A \text{ has a model}
\]

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

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

**Theorem 10** [Bojańczyk et al. 06b]

Satisfiability of \( \text{FO}^2(\sim, +1) \) on data trees is decidable

- The intermediate steps of the proof are similar as for data strings
- But additional techniques needed:
  - Model normalization by cut-and-paste arguments
  → Canonical “small” models that can be recognized by simpler tree automata
- **Complexity:**
  - Upper bound: \( 3\text{-NEXPTIME} \)
  - Lower bound: \( \text{NEXPTIME} \)
- On trees of bounded depth: \( \text{FO}^2 \) with all axes decidable [Björklund, Bojańczyk 07]
Consequences for XML Reasoning

• **We already know:**
  – Unary key and inclusion constraints can be expressed in $\text{FO}^2(\sim, +1, <)$

• **Furthermore:**
  – Regular tree languages can be captured by $\text{EMSO}^2(+1)$
  – The core of XPath without data values corresponds exactly to $\text{FO}^2(+1, <)$ [Marx, de Rijke 05]
  – A simple data-aware fragment of XPath (without transitive axes) can be expressed in $\text{FO}^2(\sim, +1)$

⇒ **Query Containment for “simple data-aware XPath” relative to Schemas with integrity constraints is decidable**

• More results on reasoning about XML integrity constraints:
  [Arenas et al. 05]
Contents

Introduction
Data Model
Automata
Logic
  Two-Variable Logics
▷ Temporal Logics
Other Models
Conclusion

A little bit infinite? Thomas Schwentick tu.
Temporal Logics and the Freeze Quantifier

- **FO**$^2$ is natural to consider from an **XML** point of view.
- From a **verification** point of view it is natural to add data handling capabilities to **temporal logics**.

→ Another natural idea:
  - “Use registers in LTL formulas”
    [Demri, Lazić 06]
- More precisely, add the following two constructs to LTL (or another logic):
  - Unary “quantifiers” $\downarrow_i$
    (where $i$ is a natural number)
  - Atomic formulas $\uparrow_i$
- **Informal semantics:**
  - $\downarrow_i$ stores the current data value in register $i$
  - $\uparrow_i$ is true if the current data value equals the value in register $i$

- **Syntax of LTL with Freeze:**

$$\phi ::= \top \mid a \mid \uparrow_i \mid \phi \land \phi \mid \neg \phi \mid X\phi \mid F\phi \mid G\phi \mid \phi \lor \phi \mid \phi \land \phi \mid \phi \lor \phi \mid \downarrow_i \phi$$

- **Examples:**
  - (L5) each print request by a process is followed by a print for that user:
    $$G (r \rightarrow \downarrow_1 X F (\uparrow_1 \land s))$$
  - (L6) Between two successive print jobs of the same user, some other user’s job has to be processed:
    $$G \neg (r \land \downarrow_1 X (\neg (s \land \uparrow_1) \lor (s \land \neg \uparrow_1)))$$
LTL with Freeze

Theorem 11 [Demri, Lazić 06]

(a) Finite Satisfiability for LTL with Freeze is
   (1) undecidable in general
   (2) decidable but not primitive recursive if only 1 register is used

(b) Infinite Satisfiability for LTL with Freeze is
   • undecidable even with only 1 register

Proof idea

• More than 1 register:
  – Non-Emptiness of Minsky Counter Automata is reducible to
    Satisfiability of LTL with Freeze
  ➞ Undecidability

• 1 register:
  – Satisfiability for LTL with Freeze with 1 register is basically
    computationally equivalent to Non-Emptiness of Incrementing
    Counter Automata:
    • Automata with counters and zero tests,
    • but: counters can always be incremented non-deterministically
  – Non-Emptiness of Incrementing Counter Automata is
    • decidable but not primitive recursive for finite strings
    • undecidable for finite strings
LTL with Freeze vs. $\mathbf{FO^2}$

- LTL with Freeze cannot express:
  - (L3) for each $a$-position there is a $b$-position with the same data value
- More generally: it cannot talk about the past
- $\mathbf{FO^2}$ cannot express:
  - (L6) Between two successive print jobs of the same user some other user’s job has to be printed
- More generally: it cannot talk about “betweenness” with respect to data values

$\Rightarrow$ LTL with Freeze and $\mathbf{FO^2}$ are incomparable
### LTL with Freeze: Extensions and Restrictions

<table>
<thead>
<tr>
<th><strong>LTL with Freeze and past modalities:</strong></th>
<th><strong>Constraint LTL:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>[Demri, Lazić 06]</td>
<td>[Demri et al. 06]</td>
</tr>
<tr>
<td>– $X^{-1}$, $G^{-1}$, $F^{-1}$, $U^{-1}$</td>
<td>– More than 1 data value per position: “freeze variables”</td>
</tr>
<tr>
<td>– Can express all $\mathbf{FO}^2$ properties</td>
<td>→ Undecidable</td>
</tr>
<tr>
<td>– But: Satisfiability undecidable</td>
<td></td>
</tr>
<tr>
<td>– A certain fragment exactly corresponds to $\mathbf{FO}^2$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Safety LTL:</strong></th>
<th><strong>Constraint LTL$\Diamond$:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>[Lazić 07]</td>
<td>[Demri et al. 07]</td>
</tr>
<tr>
<td>– <strong>Safety properties:</strong> failure is determined by a finite bad prefix</td>
<td>– Future and past modalities</td>
</tr>
<tr>
<td>– Safety LTL allows $F$ and $U$ only under an odd number of nested negations</td>
<td>– Restricted use of data values, only two kinds of data value comparisons:</td>
</tr>
<tr>
<td>– Satisfiability for Safety LTL with one register is complete for EXPSPACE</td>
<td></td>
</tr>
</tbody>
</table>

- $x = X^k y$: variable $x$ at current position equals variable $y$ at current position $+k$
- $x = \Diamond y$: the current $x$ equals some future $y$

→ Finitary and Infinitary Satisfiability are decidable
## Automata and Logics

<table>
<thead>
<tr>
<th></th>
<th>RegisterA</th>
<th>PebbleA</th>
<th>ClassMA</th>
<th>$\mathbf{FO^2}$</th>
<th>LTL &amp; Freeze</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expressiveness</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(L2),(L6),(L7)</td>
<td>(L1)–(L7)</td>
<td>(L1)–(L7)</td>
<td>(L1),(L5),(L7)</td>
<td>(L1),(L2),(L4)–(L7)</td>
</tr>
<tr>
<td><strong>Decidability</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-emptiness</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Containment</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Efficiency</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data complexity word pr.</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Closure properties</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Union</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intersection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Complement</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Robustness</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

A little bit infinite? Thomas Schwentick
A little bit infinite? Thomas Schwentick
Some Related Work on Data Strings

[Boyer et al. 03] Extension of register automata based on monoids
  • Still can only remember a bounded number of data values
  ➞ Cannot express (L1), (L3)–(L5)

[Francez, Kaminski 03] Myhill-Nerode Theorem for data strings

[Kaminski, Tan 04] Regular expressions
  • ...corresponding to unification-based register automata

[Zeitlin 06] Look-ahead register automata
  • ... can guess data values
  ➞ Closed under reversal
  • Equivalent characterizations by
    – Regular expressions (stronger than the above)
    – Grammars

[Cheng, Kaminski 98] Register pushdown automata
  • Decidable Non-emptiness

LTL on top of first-order logic
  • [Spielmann 00]: Verification of relational transducers
  • [Abdulla et al. 04]: ...even on top of MSO
  • [Deutsch et al. 04]: Verification of web services
  • In all cases: restricted comparison of data values of different states
Some Related Work on Data Trees

[Kaminski, Tan 06] Register automata for trees

[Jurdziński, Lazić 07]

- Alternation-free modal $\mu$-calculus
  - Basically identical results as for LTL with Freeze
  - In particular:
    - Computationally equivalent to Incrementing Tree Counter Automata
    - Safety fragment decidable
- Alternating Automata
- XPath satisfiability
Contents

Introduction
Data Model
Automata
Logic
Other Models

▶ Conclusion
Conclusion

- **Data strings and data trees constitute a very active research area with (potential) applications in fields like Semistructured Data and Automated Verification**

- **Data strings:**
  - Attracted most attention so far
  - No obvious analogon of regular languages (so far)
  - But “logic → automaton → analysis” possible to some extent
  - Applicability in Verification has yet to be explored:
    - Data string approach is orthogonal to Reachability-based approaches
    - Its ability to talk about data values is limited (e.g., no arithmetic)
    - Is it really useful?
    - ...for other areas? (program analysis, communicating systems,...)

- **Data trees:**
  - Clearly a good model for XML data
  - Can offer a basis for data-aware static analysis
  - Needs more work

- **In both cases we need:**
  - Models with better complexity
  - Models with richer data access
Open Problems

Technical Questions:

- Precise complexity of Satisfiability of $\text{FO}^2(\sim, +1)$ on data strings
- Precise complexity of Satisfiability of $\text{FO}^2(\sim, +1)$ on data trees
- Is Satisfiability of $\text{FO}^2(\sim, <, +1)$ on data trees decidable?
- Upper complexity bounds for Satisfiability of $\text{FO}^2(\sim, <, +1, \pm 1)$ on data strings

To be explored:

- Is there a generic class of regular data (string/tree) languages?
- Find models with better complexities
- Study the trade-off between more expressive data access and complexity/decidability
- Find larger decidable fragments of data-aware XPath
Main References (for this Talk)

[Björklund, Schwentick 07] Björklund, Schwentick: On notions of regularity on words with data, FCT 2007

[Bojańczyk et al. 06a] Bojańczyk, Muscholl, Schwentick, Segoufin, David: Two-variable logic on words with data, LICS 2006

[Bojańczyk et al. 06b] Bojańczyk, David, Muscholl, Schwentick, Segoufin: Two-variable logic on data trees and XML reasoning, PODS 2006

[Demri, Lazić 06] Demri, Lazić: LTL wit freeze quantifier and register automata, LICS 2006; ACM ToCL 08

[Demri et al. 06] Demri, D'Souza, Nowak: On the freeze quantifier in Constraint LTL: decidability and complexity logic of repeating values

[Demri et al. 07] Demri, D'Souza, Gascon: A decidable temporal logic of repeating values


[Lazić 06] Lazić: Safely freezing LTL, FSTTCS 2006

[Neven et al. 01] Neven, Schwentick, Vianu: Finite state machines for strings over infinite alphabets, ACM ToCL 04 (and MFCS 01 with different title)

Surveys:

- Segoufin: Automata and logics for words and trees over an infinite alphabet, CSL 2006
- Segoufin: Static analysis of XML processing with data values, SIGMOD Record 2007
[Abdulla et al. 04] Abdulla, Jonsson, Nilsson, d’Orso, Mayank: Regular model checking for LTL(MSO), CAV 2004

[Abdulla et al. 07] Abdulla, Delzanno, Rezine: Parameterized Verification of infinite-state processes with global conditions, CAV 2007


[Bouajjani et al. 00] Bouajjani, Jonsson, Nilsson, Touili: Regular model checking, CAV 00

[Boyer et al. 03] Bouyer, Petit, Thérien: An algebraic approach to data languages and timed languages, Inf. Comp. 2003


[David 04] David: Mote et données infinis, 2004
More References (2/2)

[Deutsch et al. 04] Deutsch, Sui, Vianu: Specification and verification of data-driven web applications, PODS 04, JCSS 06

[Francez, Kaminski 03] Francez, Kaminski: An algebraic characterization of deterministic regular languages over infinite alphabets, TCS 2003

[Henzinger 90] Henzinger: Half-order modal logic: how to prove real-time properties, PODS 90


[Marx, de Rijke 05] Marx, de Rijke: Semantic Characterizations of Navigational XPath, SIGMOD record 05


[Zeitlin 06] Zeitlin: Look-ahead finite-memory automata, 2006