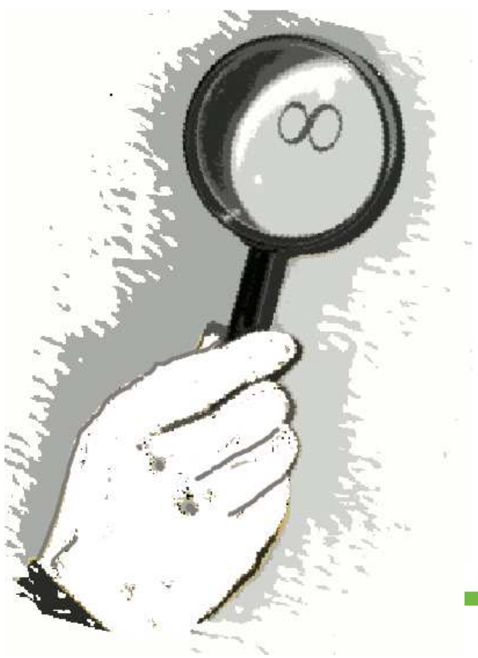
A little bit infinite?

Adding data to finitely labelled structures

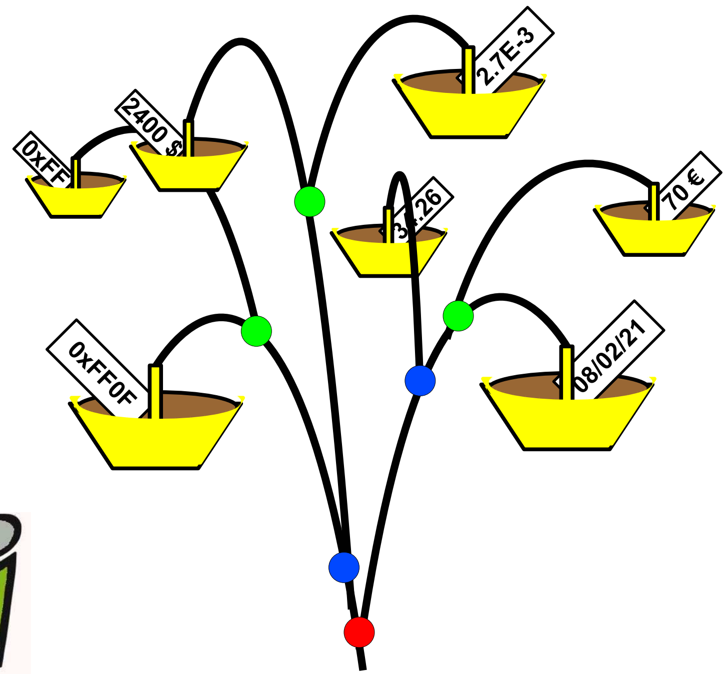


Thomas Schwentick

Grantown-on-Spey

July 2008

technische universität dortmund



Contents

Contents

Relational Databases

Composers from Southwest

Composers		
Name	Birth	Death
Ravel	Ciboure	Paris
Tournemire	Bordeaux	Arcachon

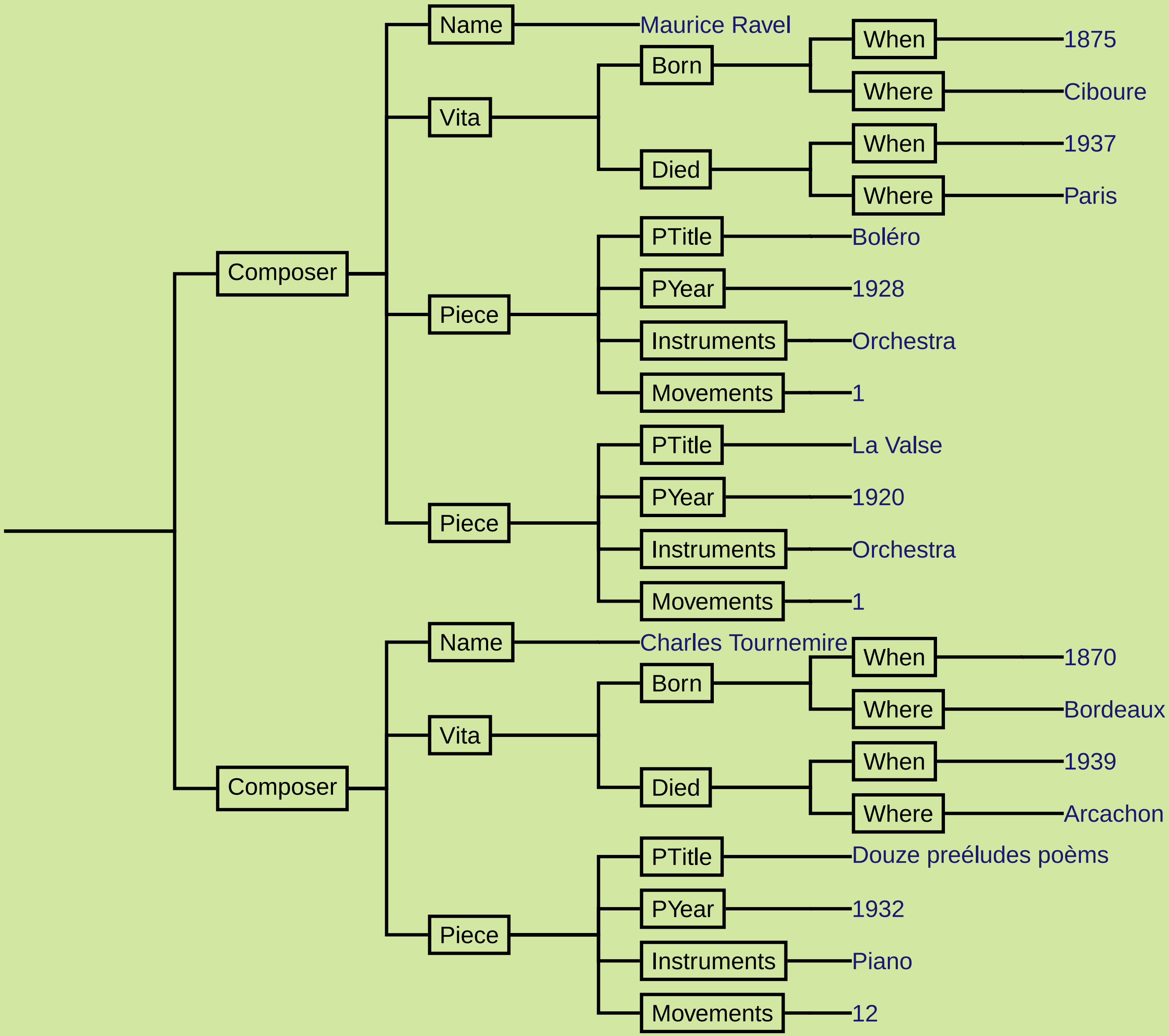
Pieces				
Name	Comp	Year	Instr	Movem
Boléro	Ravel	1928	Orch.	1
Douze Préludes	Tournemire	1932	Piano	12
La Valse	Ravel	1920	Orch.	1

```
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**:
  $$Q(x_1, x_2) \equiv \exists x_3, \ldots, x_5, y_1 \ldots, y_3 \quad \text{Pieces}(x_1, x_2, x_3, x_4, x_5) \wedge \text{Composers}(y_1, y_2, y_3) \wedge y_1 = x_2 \wedge y_3 = \text{“Bordeaux”}$$
- Integrity Constraints rely on **structure** and **equality of data items**:
  $$\forall x_1, \ldots, x_5, y_1, \ldots, y_5 \; (x_1 = y_1 \wedge x_2 = y_2) \rightarrow (x_3 = y_3 \wedge x_4 = y_5 \wedge x_5 = y_5)$$

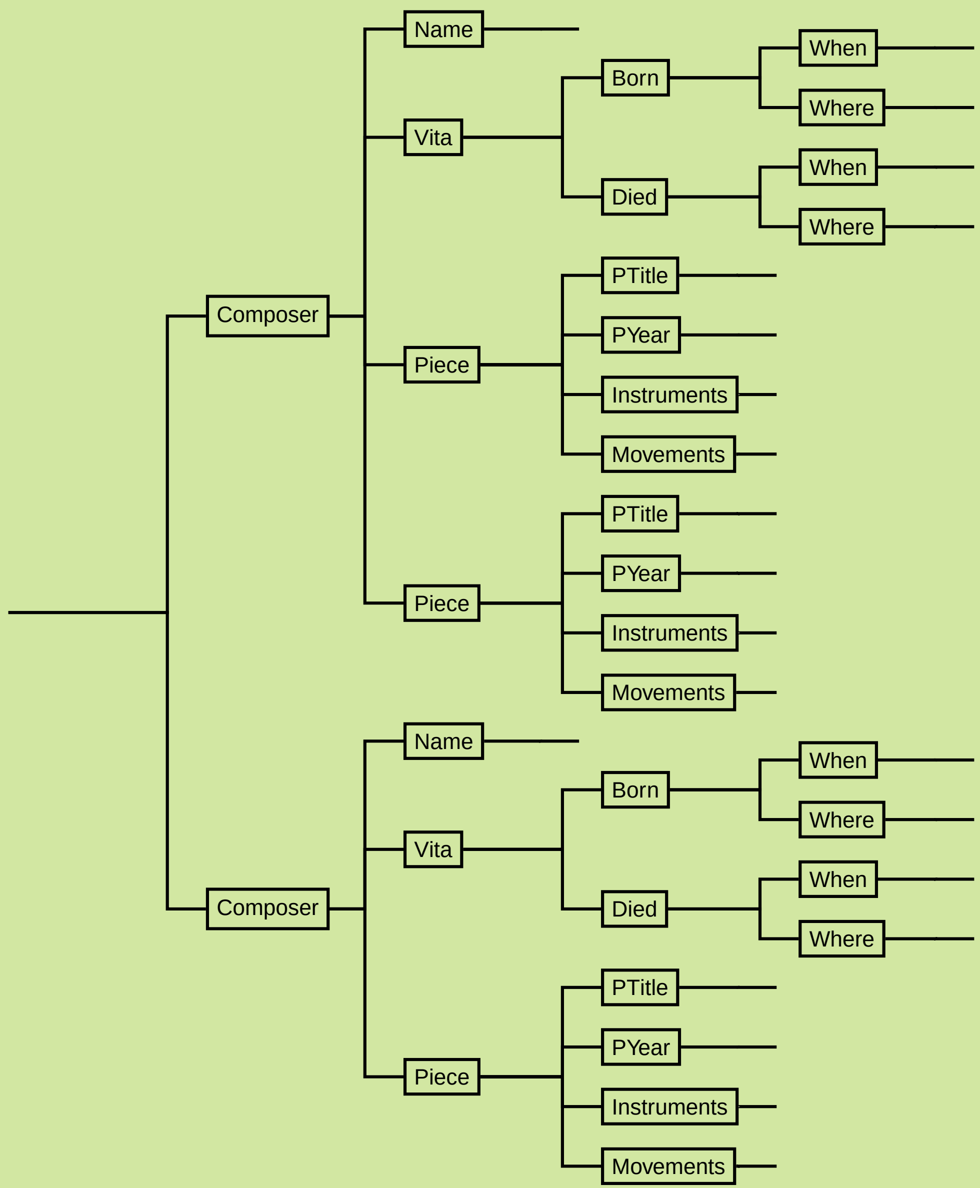
XML (1/4)

Example Tree



XML (2/4)

Example



- **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**

XML (3/4)

- **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]

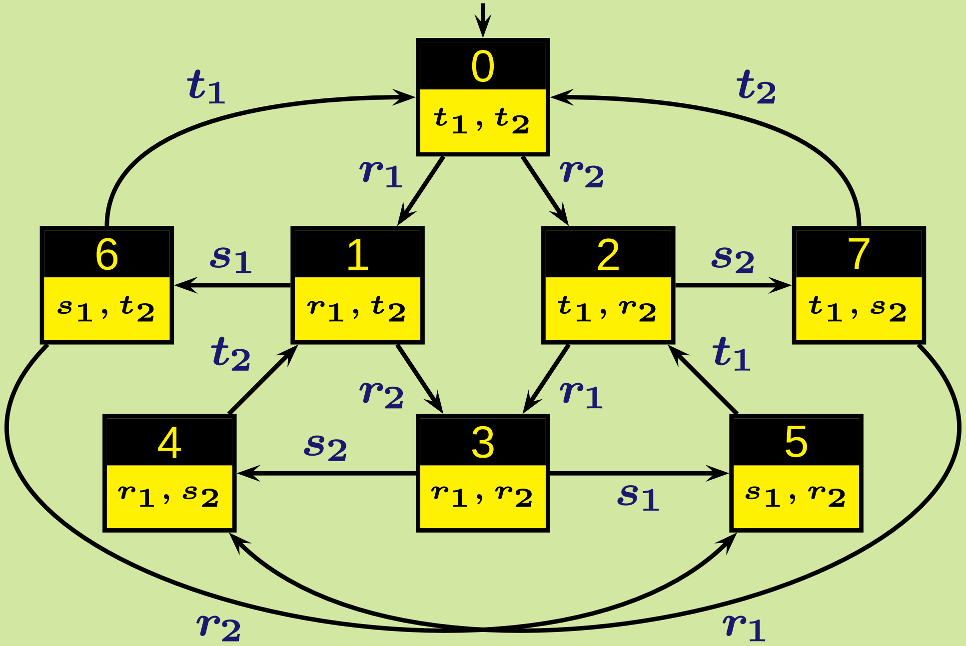
XML (4/4)

- 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 $\longrightarrow$ 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

A Toy Example from Verification

A printer and two processes



- A printer and two processes
- Possible actions:
  - r_i: User i submits print **r**equest
  - s_i: Printing of request of i **s**tarts
  - t_i: Print job for user i **t**erminates

A printer and two processes (cont.)

- 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
  ($\longrightarrow$ "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: φ
  - 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

Contents

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	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
  - Σ 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"?**

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

$$\begin{array}{cccccccccccccccccc} r & \boxed{r} & \boxed{s} & r & r & \boxed{t} & \boxed{r} & s & t & s & r & t & s & t & s & t & \boxed{s} & \boxed{t} \\ 2 & \boxed{5} & \boxed{5} & 3 & 8 & \boxed{5} & \boxed{5} & 2 & 2 & 8 & 4 & 8 & 3 & 3 & 4 & 4 & \boxed{5} & \boxed{5} \end{array}$$

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”)**

➞ (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”)**

Contents

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

$$\begin{array}{cccccccccccccccccc} r & r & s & r & r & t & r & s & t & s & r & t & s & t & s & t & s & t \\ 2 & 5 & 5 & 3 & 8 & 5 & 5 & 2 & 2 & 8 & 4 & 8 & 3 & 3 & 4 & 4 & 5 & 5 \end{array}$$

R_1	
R_2	

Register Automata (2/4)

Theorem 1 [Kaminski, Francez 90]

(a) Non-emptiness for register automata is decidable

(b) Testing $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ is decidable as long as $\mathcal{A}_2$ has ≤ 2 registers

Proof idea

(a) Crux: if there is a string in $L(\mathcal{A})$, 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
  ➞ 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
  ➞ Non-Emptiness is **NP**-complete

Theorem 2 [Kaminski, Francez 90]

- Universality, i.e., testing whether a register automaton accepts every data string is undecidable

Register Automata (3/4)

- 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 $\mathcal{A}$ tests (L4)
- Consider the following input:

$$\begin{array}{ccccc} r & r & r & r & r \\ 1 & 2 & 3 & 4 & 1 \end{array}$$

R_1	4
R_2	2
R_3	3

- $\mathcal{A}$ 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:

	RegisterA
Expressiveness	
	(L2),(L6),(L7)
Decidability	
Non-emptiness	✓
Containment	–
Efficiency	
Data complexity word problem	✓
Closure properties	
Union	✓
Intersection	✓
Complement	–
Robustness	
	-

- **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]

Contents

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

➞ Pebble automata

Example

$$\begin{array}{cccccccccccccccccc} r & r & s & r & r & t & r & s & t & s & r & t & s & t & s & t & s & t \\ 2 & 5 & 5 & 3 & 8 & 5 & 5 & 2 & 2 & 8 & 4 & 8 & 3 & 3 & 4 & 4 & 5 & 5 \end{array}$$

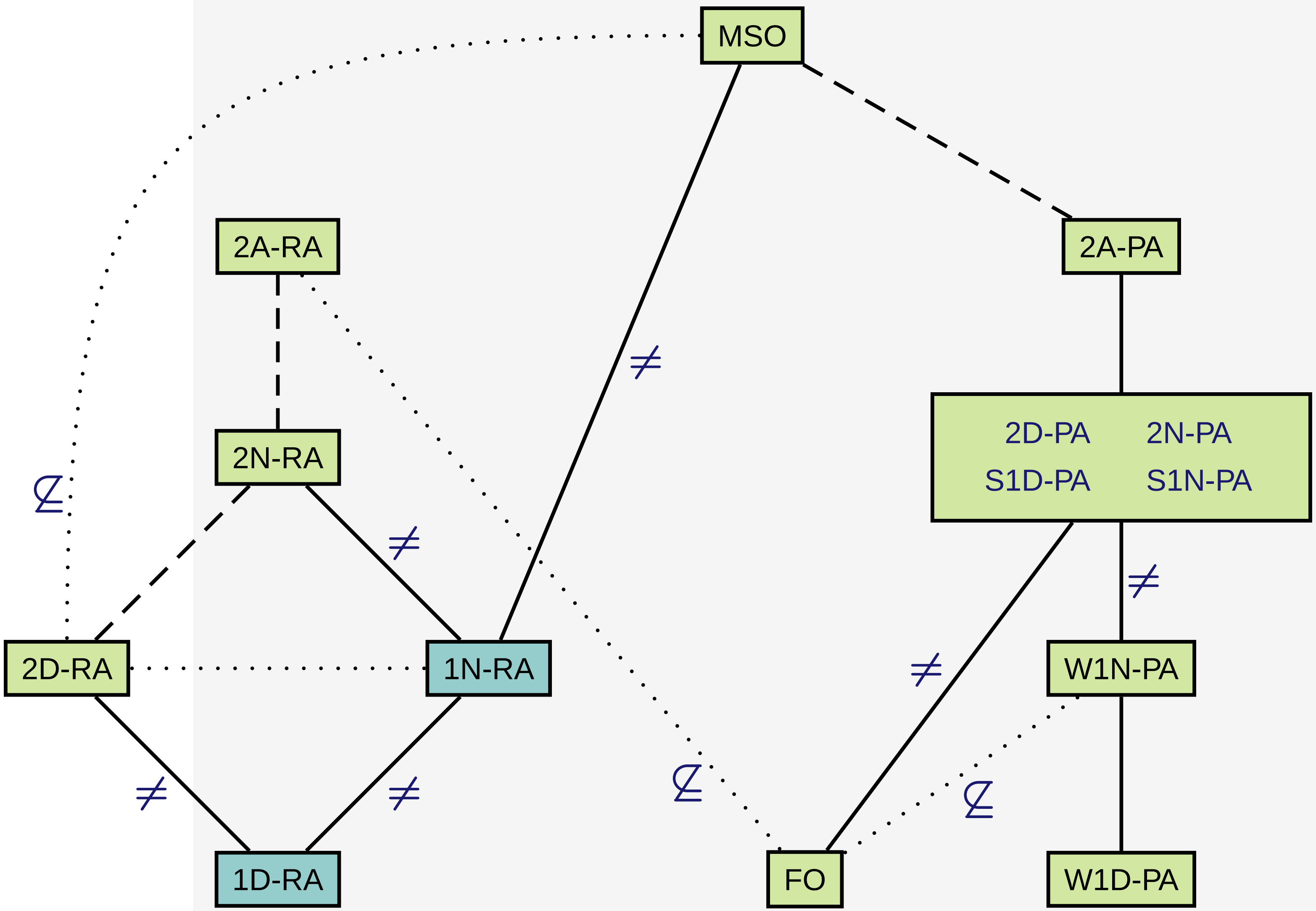
- Example automaton for (L6): **Between two successive print jobs of the same user some other user's job has to be printed**
- Again stated differently: **no two successive s-positions carry the same data value**
- **Solution:** for each s-position check that the previous s-position has a different data value

Pebble Automata (2/3)

- 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

	RegisterA	PebbleA
Expressiveness		
	(L2),(L6),(L7)	(L1)–(L7)
Decidability		
Non-emptiness	✓	–
Containment	–	–
Efficiency		
Data complexity word pr.	✓	✓
Closure properties		
Union	✓	✓
Intersection	✓	✓
Complement	–	✓
Robustness		
	–	✓

Pebble Automata (3/3)



(from [Neven/Sch./Vian

Contents

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

$$\begin{array}{cccccccccccccccccc} r & r & s & r & r & t & r & s & t & s & t & r & s & t & s & t & s & t \\ 2 & 5 & 5 & 3 & 8 & 5 & 4 & 2 & 2 & 8 & 8 & 8 & 3 & 3 & 4 & 4 & 8 & 8 \end{array}$$

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline \perp & n & n & n & n & y & y & y & y & y & y & y & y & y & y & y & y & y & y \\ \hline \perp & r & r & s & r & \dot{r} & t & r & s & t & \dot{s} & \dot{t} & \dot{r} & s & t & s & t & \dot{s} & \dot{t} \\ \hline \end{array}$$

- States are of the form $\begin{array}{|c|}\hline p \\ \hline q \\ \hline\end{array}$, 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,
  - the last state should be of the form $\begin{array}{|c|}\hline \\ \hline t \\ \hline\end{array}$ or $\begin{array}{|c|}\hline \\ \hline \dot{t} \\ \hline\end{array}$ and
  - each class should have a last state of the form $\begin{array}{|c|}\hline \\ \hline t \\ \hline\end{array}$ or $\begin{array}{|c|}\hline \\ \hline \dot{t} \\ \hline\end{array}$

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) RegA $\subsetneq$ 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 $p \notin F_l$
  - Non-emptiness for multi-counter automata is decidable

    [Mayr 81]
  - And:

$$L(\mathcal{A}) \neq \emptyset \iff L(\mathcal{A}') \neq \emptyset$$

Class Memory Automata (4/5)

Proof sketch for (b) [Björklund, S 07]

- Strictness: RAs can not express (L1)
- Isn't RegA $\subseteq$ 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 $\genfrac{}{}{0pt}{}{s}{d}$ know what happened since $\genfrac{}{}{0pt}{}{s}{d}$ occurred last time?
- **Idea:** $\mathcal{A}$ "colors" positions by $\begin{array}{|c|}\hline + \\ \hline + \\ \hline\end{array}$, $\begin{array}{|c|}\hline + \\ \hline - \\ \hline\end{array}$, $\begin{array}{|c|}\hline - \\ \hline + \\ \hline\end{array}$, $\begin{array}{|c|}\hline - \\ \hline - \\ \hline\end{array}$ such that:
  - If an s-position has $\begin{array}{|c|}\hline \\ \hline + \\ \hline\end{array}$ the next s-position has $\begin{array}{|c|}\hline - \\ \hline \\ \hline\end{array}$ (and $\begin{array}{|c|}\hline \\ \hline - \\ \hline\end{array} \longrightarrow \begin{array}{|c|}\hline + \\ \hline \\ \hline\end{array}$)
  - If an s-position has $\begin{array}{|c|}\hline \\ \hline + \\ \hline\end{array}$ the next s-position **in the same class** has $\begin{array}{|c|}\hline + \\ \hline \\ \hline\end{array}$

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

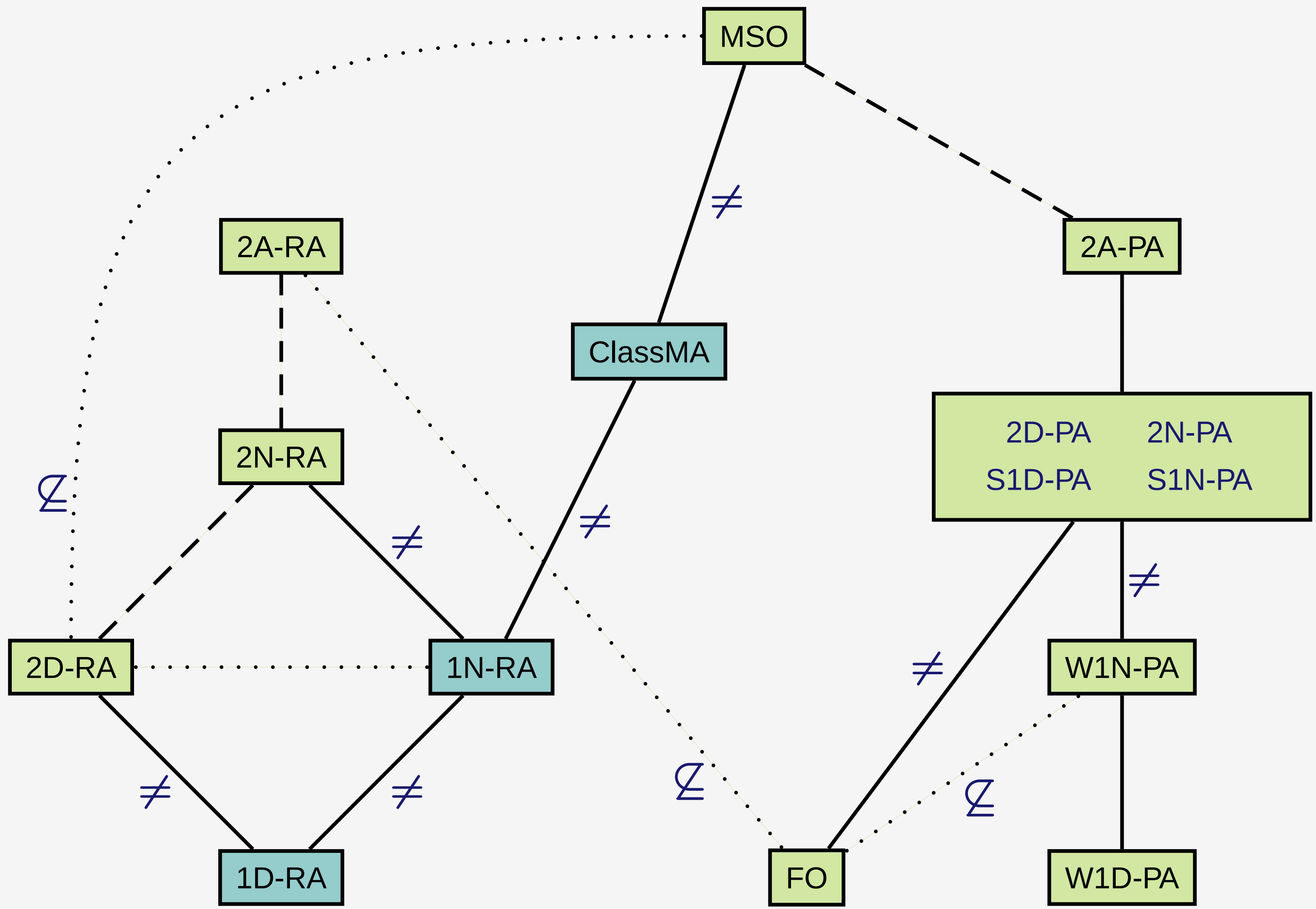
s	s	s	s	s	s	s	s	s
2	3	2	5	3	2	5	2	3
+	+	−	+	+	−	−	+	+
−	+	−	−	+	+	−	−	

- **If (L6) holds such a coloring can be constructed** by applying the following rules:
  (1) If no other rule applies: assign $\begin{array}{|c|}\hline + \\ \hline \\ \hline\end{array}$ to the rightmost s without upper color
  (2) Whenever $\begin{array}{|c|}\hline + \\ \hline \\ \hline\end{array}$ is assigned to an s-position assign $\begin{array}{|c|}\hline \\ \hline - \\ \hline\end{array}$ to its left s-neighbour and $\begin{array}{|c|}\hline \\ \hline + \\ \hline\end{array}$ to the left s-neighbour in its class
  (3) Whenever $\begin{array}{|c|}\hline \\ \hline + \\ \hline\end{array}$ is assigned to an s-position assign $\begin{array}{|c|}\hline - \\ \hline \\ \hline\end{array}$ to its right s-neighbour
- General proof of (b): similar coloring trick

Class Memory Automata (5/5)

	RegisterA	PebbleA	ClassMA	DClassMA
Expressiveness				
	(L2),(L6),(L7)	(L1)–(L7)	(L1)–(L7)	(L1)–(L5),(L7)
Decidability				
Non-emptiness	✓	–	✓	✓
Containment	–	–	–	–
Efficiency				
Data complexity word pr.	✓	✓	–	✓
Closure properties				
Union	✓	✓	✓	–
Intersection	✓	✓	✓	✓
Complement	–	✓	–	–
Robustness				
	-	✓	✓	–

Inclusion structure of Automata Models



Contents

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**: ARA_1

Theorem 5 [Demri, Lazić 06]

(a) Non-emptiness (and Containment) of ARA_1 on strings is decidable but not primitive recursive

(b) Non-emptiness of ARA_1 on ω-strings is undecidable (even with Muller acceptance)

- ARA_1 can express all properties (L1)-(L7)
- ARA_1 can not remember two data values at a time

- **Safety ARA_1** reject only in the finite (and their complement languages are closed under adding suffixes)

Theorem 6 [Lazić 06]

(a) Non-emptiness of safety ARA_1 on ω-strings is **EXPSPACE**-complete

(b) Containment of safety ARA_1 on ω-strings is decidable but not primitive recursive

Alternating Register Automata (2/2)

	RegisterA	PebbleA	ClassMA	DClassMA	ARA_1	Safe ARA_1
Expressiveness						
	(L2),(L6),(L7)	(L1)–(L7)	(L1)–(L7)	(L1)–(L5),(L7)	(L1)–(L7)	(L1),(L4),(L6)
Decidability						
Non-emptiness	✓	–	✓	✓	✓	✓
Containment	–	–	–	–	✓	✓
Efficiency						
Data complexity word pr.	✓	✓	–	✓	✓	✓
Closure properties						
Union	✓	✓	✓	–	✓	✓
Intersection	✓	✓	✓	✓	✓	✓
Complement	–	✓	–	–	✓	–
Robustness						
	-	✓	✓	–	✓	–

Contents

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...			
... for strings		**... for trees**	
$a(x)$	Letter at position x is $a \in \Sigma$	$a(x)$	Label of node x is $a \in \Sigma$
$+1$	successor relation on positions	$E_{\rightarrow}$	horizontal neighbor ("next sibling")
		$E_{\downarrow}$	parent-child
$<$	order relation on positions	$E_{\Rightarrow}$	transitive closure of $E_{\rightarrow}$
		$E_{\Downarrow}$	transitive closure of $E_{\downarrow}$
$\sim$	$x \sim y$ if positions x and y have the same D-value	$\sim$	$x \sim y$ if nodes x and y have the same D-value
$\underset{\sim}{+}1$	next position in the same class		

- Of course: $\sim$ **is an equivalence relation**
- 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?

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 $u_{i_1} \cdots u_{i_n} = v_{i_1} \cdots u_{i_n}$?

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:

$$x \sim y \rightarrow \exists z\, (x + 1 = z \wedge \exists x\, y + 1 = x \wedge z \sim x)$$

"if x and y are equivalent then their right neighbors are also equivalent"

Two Variables 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
$$\forall x\ \forall y\ (x \sim y \wedge a(x) \wedge a(y)) \rightarrow x = y$$

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

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

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

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

(L6) Between two successive print jobs of the same user some other user's job has to be printed **not expressible**

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

On the expressive power of $\mathbf{FO}^2$ on data strings (1/2)

Example

- φ_a:
  - $\forall x\, \forall y\, (x \sim y \land a(x) \land a(y)) \to x = y$
  - all a's are in different classes
- Similarly: φ_b
- $\psi_{a,b}$:
  - $\psi_{a,b} = \forall x\, \exists y\, \big(a(x) \to (b(y) \land x \sim y)\big)$
  - 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 $\mathbf{FO}^2$-definable data language need not be context-free

On the expressive power of $\mathbf{FO}^2$ on data strings (2/2)

More example properties

- Let α and β denote unary quantifier-free formulas ("types")
- $\mathbf{FO}^2$ can express
  - data-blind properties, i.e., properties not using $\sim$
  - Each class contains at most one occurrence of α:
    $$\theta = \forall x \forall y \left((\alpha(x) \wedge \alpha(y) \wedge x \sim y) \rightarrow x = y\right)$$
  - In each class, every α occurs before every β:
    $$\theta = \forall x \forall y \left((\alpha(x) \wedge \beta(y) \wedge x \sim y) \rightarrow x < y\right)$$
  - Each class with an α also has a β:
    $$\theta = \forall x \exists y \left(\alpha(x) \rightarrow (\beta(y) \wedge x \sim y)\right)$$
  - If a position is in a different class than its successor it has type α:
    $$\theta = \forall x \forall y (\neg(x \sim y) \wedge x + 1 = y) \rightarrow \alpha(x)$$
  - That's basically all!

Theorem 8 [Bojańczyk et al. 06a]

Satisfiability of $\mathbf{FO}^2(\sim, <, +1, \pm 1)$ 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"**: $\exists R_1, \ldots, R_k \, \forall x \, \forall y \, \chi \;\wedge\; \bigwedge_i \forall x \exists y \, \chi_i$
- **Intermediate normal form**:

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

- θ_i:

(1) $\forall x \forall y \quad (\delta(x,y) \geq 2 \wedge \alpha(x) \wedge \beta(y) \wedge \boxed{\begin{array}{c} x \sim y \\ x \not\sim y \end{array}}) \rightarrow \boxed{\begin{array}{c} x < y \\ x > y \\ \mathit{ff} \end{array}}$

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

(3) $\forall x \forall y \quad \theta$ $\qquad$ (θ quantifier-free, DNF, no $\sim$)

- Both steps are straightforward

Proof Sketch for Theorem 8 (2/2)

Data normal form & Class Memory Automata

- **Data normal form**:
  - Disjunction of formulas $\exists R_1 \cdots R_n\ \theta_1 \wedge \cdots \wedge \theta_n$
  - θ_i:
    (a) data-blind
    (b) Each class contains at most one α
    (c) In each class, every α occurs before every β
    (d) Each class with an α also has a β
    (e) If x is in a different class than its successor has type α
- **Final Step:**
  - **Each θ_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 $\mathbf{FO}^2(\sim,<,+1,\pm 1)$ follows from decidability of Non-emptiness for Class Memory Automata

- Corollary: ClassMA $\equiv \mathbf{EMSO}^2(\sim,<,+1,\pm 1)$

$\mathbf{FO^2}$ on Data Strings: Complexity

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

➞ Unknown complexity

- **Restrictions:**
  - $\mathbf{FO^2(\sim, <)}$: complete for **NEXPTIME** [David 04]
  - $\mathbf{FO^2(\sim, +1)}$: in **3NEXPTIME** [Bojańczyk et al. 06b]
- **Extensions:**
  - $\mathbf{+2}$, $\mathbf{+3}$,...: same results
  - ω-strings: same results
  - Linear order on data values: undecidable

Two-Variable Logic on Data Trees

Theorem 9 [Bojańczyk et al. 06b]

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

$L(A) \neq \emptyset$ iff φ_A 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 $\mathbf{FO}^2(\sim, +1)$

Theorem 10 [Bojańczyk et al. 06b]

Satisfiability of $\mathbf{FO}^2(\sim, +1)$ 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-NEXPTIME**
  - Lower bound: **NEXPTIME**
- On trees of bounded depth: $\mathbf{FO}^2$ with all axes decidable [Björklund, Bojańczyk 07]

Consequences for XML Reasoning

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

➨ **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

Temporal Logics and the Freeze Quantifier

- $\mathbf{FO^2}$ 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:**

$$\varphi ::= \top \mid a \mid \uparrow_i \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \downarrow_i\varphi$$

- **Examples:**
  - (L5) each print request by a process is followed by a print for that user:

$$\mathbf{G}(r \rightarrow \downarrow_1 \mathbf{XF}(\uparrow_1 \wedge s))$$

  - (L6) Between two successive print jobs of the same user, some other user's job has to be processed:

$$\mathbf{G}\neg(r\wedge \downarrow_1 \mathbf{X}(\neg(s\wedge \uparrow_1)\,\mathbf{U}\,(s \wedge \neg \uparrow_1)))$$

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

➨ LTL with Freeze and $\mathbf{FO^2}$ are incomparable

LTL with Freeze: Extensions and Restrictions

- **LTL with Freeze and past modalities:** [Demri, Lazić 06]
  - $\mathbf{X}^{-1}$, $\mathbf{G}^{-1}$, $\mathbf{F}^{-1}$, $\mathbf{U}^{-1}$
  - Can express all $\mathbf{FO^2}$ properties
  - But: Satisfiability undecidable
  - A certain fragment exactly corresponds to $\mathbf{FO^2}$
- **Safety LTL**: [Lazić 07]
  - **Safety properties:** failure is determined by a finite bad prefix
  - Safety LTL allows $\mathbf{F}$ and $\mathbf{U}$ only under an odd number of nested negations
  - **Satisfiability for Safety LTL with one register is complete for EXPSPACE**

- **Constraint LTL:** [Demri et al. 06]
  - More than 1 data value per position: "freeze variables"
  - → Undecidable
- **Constraint LTL$^\diamond$:** [Demri et al. 07]
  - Future and past modalities
  - Restricted use of data values, only two kinds of data value comparisons:
    - $x = X^k y$: 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

	RegisterA	PebbleA	ClassMA	$\mathbf{FO^2}$	LTL & Freeze
Expressiveness					
	(L2),(L6),(L7)	(L1)–(L7)	(L1)–(L7)	(L1)–(L5),(L7)	(L1),(L2),(L4)–(L7)
Decidability					
Non-emptiness	✓	–	✓	✓	✓
Containment	–	–	–	✓	✓
Efficiency					
Data complexity word pr.	✓	✓	–	✓	✓
Closure properties					
Union	✓	✓	✓	✓	✓
Intersection	✓	✓	✓	✓	✓
Complement	–	✓	–	✓	✓
Robustness					
	-	✓	✓	?	?

Contents

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 μ-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

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 $\longrightarrow$ automaton $\longrightarrow$ 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 arithmic)

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

[Kaminski, Francez 90] Kaminski, Francez: Finite memory automata, FOCS 90 and TCS 94

[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

More References (1/2)

[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

[Alur, Dill 90] Alur, Dill: A theory of timed automata, ICALP 90, TCS 94

[Arenas et al. 05] Arenas, Fan, Libkin: Consistency of XML specifications, Inconsisteny Tolerance 2005

[Arenas, Libkin 05] Arenas, Libkin: XML Data Exchange: Consistency and Query Answering, PODS 2005

[Autebert et al. 80] Autebert, Beauquier, Boasson: Langages sur des alphabets infinis, Discrete Appl. Math. 1980

[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

[Cheng, Kaminski 98] Cheng, Kaminski: Context-free languages over infinite alphabets, Acta Inf. 1998

[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

[Jurdzinski, Lazić 07] Jurdzinski, Lazić: Alternating Automata on Data Trees and XPath Satisfiability (extended version of LICS 2007 paper)

[Kaminski, Tan 06] Kaminski, Tan: Tree automata over infinite alphabets, ICIAA 2006

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

[Mayr 81] Mayr: An algorithm for the general Petri net reachability problem, STOC 81, SIAM J Comp 1984

[Otto 85] Otto: Classes of regular and context-free languages over countably infinite alphabets, Discrete Appl. Math. 1985

[Spielmann 90] Spielmann: Verification of relational transducers for electronic commerce, PODS 2000, JCSS 2003

[Tal 99] Tal: Decidability of inclusion for unification based automata, 1999

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