Walking on Data Words

Amaldev Manuel, Anca Muscholl, Gabriele Puppis
LaBRI / CNRS

CSR 2013

Introduction

Origins and applications of data languages

- **Databases**: to model specifications and queries on XML documents that contain both symbols from a finite alphabet and data values from a potentially infinite alphabet
- **Verification**: to model properties of programs parametrized by variables ranging over an infinite domain
- **Formal languages**: to understand how the classical theory of regular languages over finite alphabets can be extended to languages over infinite alphabets.

Introduction

What precisely is a data language?

A **data language** is a set of finite words over an infinite alphabet $\Sigma \times \mathbb{D}$ of pairs of letters and data values.

To make life easier, one usually enforces the following restriction:

Languages are invariant under permuations of data values

$$(\text{e.g.} \quad \binom{a}{1}\binom{b}{2}\binom{a}{2}\binom{b}{1}\binom{a}{3} \in L \quad \text{iff} \quad \binom{a}{5}\binom{b}{3}\binom{a}{3}\binom{b}{5}\binom{a}{7} \in L \quad)$$

focus on properties concerning equalities of data values

Introduction

What precisely is a data language?

A **data language** is a set of finite words over an
infinite alphabet $\Sigma \times \mathrm{D}$ of pairs of letters and data values.

To make life easier, one usually enforces the following restriction:

Languages are invariant under permuations of data values

$$(\text{e.g.} \quad \binom{a}{1}\binom{b}{2}\binom{a}{2}\binom{b}{1}\binom{a}{3} \in \mathrm{L} \quad \text{iff} \quad \binom{a}{5}\binom{b}{3}\binom{a}{3}\binom{b}{5}\binom{a}{7} \in \mathrm{L} \quad)$$

focus on properties concerning equalities of data values

An example of data language

$$\mathrm{L} \;=\; \left\{ w \in \mathrm{D}^* \;\middle|\; w \textit{ contains at least } 2 \textit{ distinct values} \right\}$$

$= \{$  $, \ldots \}$

Introduction

What precisely is a data language?

A **data language** is a set of finite words over an <u>infinite alphabet $\Sigma \times \mathrm{D}$ of pairs of letters and data values</u>.

To make life easier, one usually enforces the following restriction:

Languages are invariant under permuations of data values

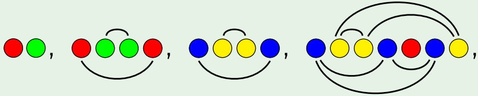
$$\text{(e.g.} \quad \binom{a}{1}\binom{b}{2}\binom{a}{2}\binom{b}{1}\binom{a}{3} \in \mathrm{L} \quad \text{iff} \quad \binom{a}{5}\binom{b}{3}\binom{a}{3}\binom{b}{5}\binom{a}{7} \in \mathrm{L} \quad \text{)}$$

focus on properties concerning <u>equalities of data values</u>

An example of data language

$$\mathrm{L} \;=\; \left\{ w \in \mathrm{D}^* \;\middle|\; w \textit{ contains at least } 2 \textit{ distinct values} \right\}$$

$= \{$



$, \ldots \}$

Introduction

Like in the classical case, automata on data words can be:

- **deterministic** / **non-deterministic** / **alternating**
- **one-way** / **two-way**

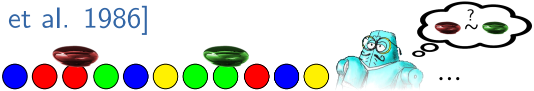
Introduction

Like in the classical case, automata on data words can be:

- **deterministic** / **non-deterministic** / **alternating**
- **one-way** / **two-way**

...moreover, to handle data values, one can equip automata with:

- **Pebbles** [Chang et al. 1986]



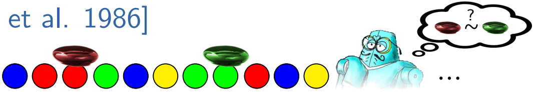
Introduction

Like in the classical case, automata on data words can be:

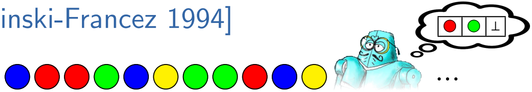
- **deterministic** / **non-deterministic** / **alternating**
- **one-way** / **two-way**

…moreover, to handle data values, one can equip automata with:

- **Pebbles** [Chang et al. 1986]



- **Registers** [Kaminski-Francez 1994]



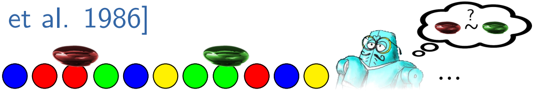
Introduction

Like in the classical case, automata on data words can be:

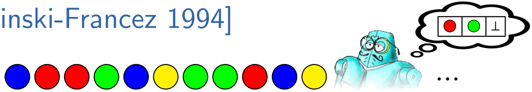
- **deterministic** / **non-deterministic** / **alternating**
- **one-way** / **two-way**

...moreover, to handle data values, one can equip automata with:

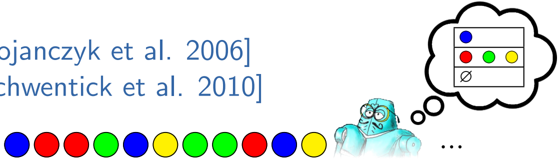
- **Pebbles** [Chang et al. 1986]



- **Registers** [Kaminski-Francez 1994]



- **Hash tables** [Bojanczyk et al. 2006]
  [Schwentick et al. 2010]



Introduction

Unlike classical automata:

- different models have often different expressive power
- difficult to get equivalence between automata and logics
- there is a tradeoff between
  expressiveness of models,
  robustness of class of recognized languages, and
  decidability of paradigmatic problems

Introduction

Unlike classical automata:

- different models have often different expressive power
- difficult to get equivalence between automata and logics
- there is a tradeoff between
  expressiveness of models,
  robustness of class of recognized languages, and
  decidability of paradigmatic problems

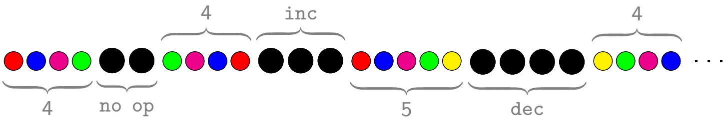
Examples for automata with registers

- One-way deterministic $\Rightarrow$ quite weak (no reversals)
- One-way non-deterministic $\Rightarrow$ no complementation
- One-way alternating $\Rightarrow$ emptiness undecidable
- Two-way deterministic $\Rightarrow$ emptiness undecidable

Introduction

☞ A word on undecidability of emptiness...

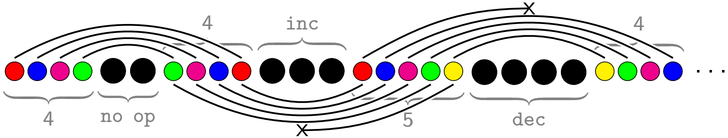
...it often arises from the possibility of encoding counter machines:



Introduction

☞ A word on undecidability of emptiness...

...it often arises from the possibility of encoding counter machines:



In some sense, data words are problematic because they can be seen as graphs with potentially unbounded grid minors.

Outline

In this talk:

Data Walking Automata

Closures and other properties

Decision problems

Data Walking Automata

We think of a data word as a special graph:

- vertices are positions in the word:

- 4 types of edges:

Data Walking Automata

We think of a data word as a special graph:

- vertices are positions in the word:

- 4 types of edges: **global successor**

Data Walking Automata

We think of a data word as a special graph:

- vertices are positions in the word:

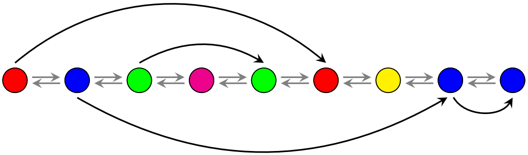
- 4 types of edges: **global successor**

  global predecessor

Data Walking Automata

We think of a data word as a special graph:

- vertices are positions in the word:

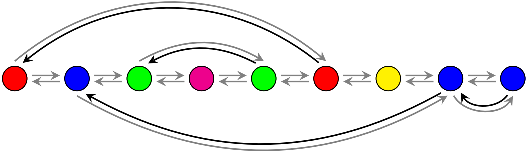


- 4 types of edges: **global successor** **class successor** **global predecessor**

Data Walking Automata

We think of a data word as a special graph:

- vertices are positions in the word:

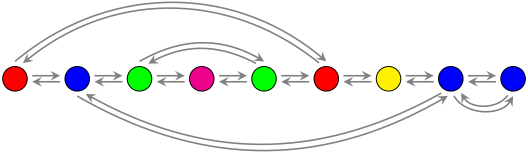


- 4 types of edges: **global successor** **class successor** **global predecessor** **class predecessor**

Data Walking Automata

We think of a data word as a special graph:

- vertices are positions in the word:



- 4 types of edges: **global successor** **class successor** **global predecessor** **class predecessor**

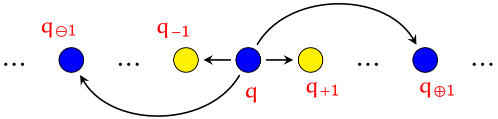
☞ Vertices are implicitly labeled with **local types** specifying whether successors/precedessors exist and whether they coincide.

Data Walking Automata

2 natural notions of automata on data words seen as graphs:

1. **Tiling Automata** [Thomas 1997]
   a.k.a. **Data Automata** [Bojanczyk et al. 2006]
   or **Class Memory Automata** [Schwentick et al. 2010]

   runs = labelings satisfying constraints on neighborhoods

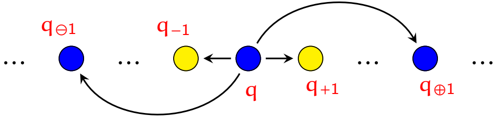


Data Walking Automata

2 natural notions of automata on data words seen as graphs:

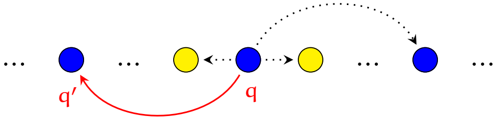
1. **Tiling Automata** [Thomas 1997]
   a.k.a. **Data Automata** [Bojanczyk et al. 2006]
   or **Class Memory Automata** [Schwentick et al. 2010]

   runs = labelings satisfying constraints on neighborhoods



2. **Walking Automata** [Aho, Ullman 1971]

   runs = deterministic / non-deterministic traversals



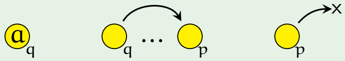
Data Walking Automata

Example 1

$\mathrm{L} = \{$ *words where all* a *are followed by* b *in the <u>same</u> class* $\}$

- A **Tiling Automaton** marks the longest b-suffix in each class:

  uses states p, q, avoids tiles



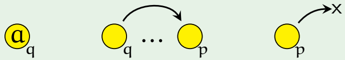
Data Walking Automata

Example 1

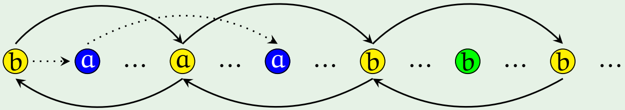
$\mathrm{L} = \{$ *words where all* a *are followed by* b *in the* *same* *class* $\}$

- A **Tiling Automaton** marks the longest b-suffix in each class:

uses states p, q, avoids tiles



- A **Walking Automaton** scans each class verifying $\{a, b\}^*\{b\}^+$:

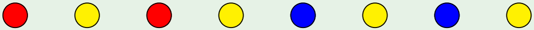


Data Walking Automata

Example 2

$$\mathtt{L} = \left\{ \textit{words where all } \mathtt{a} \textit{ are followed by } \mathtt{b} \textit{ in a } \underline{\textit{different}} \textit{ class} \right\}$$

To accept a data word, e.g.



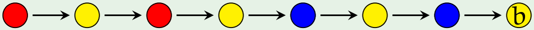
a Data Walking Automata should perform the following steps:

Data Walking Automata

Example 2

$\mathrm{L} = \Big\{$ *words where all* $\mathfrak{a}$ *are followed by* $\mathfrak{b}$ *in a <u>different</u> class* $\Big\}$

To accept a data word, e.g.



a Data Walking Automata should perform the following steps:

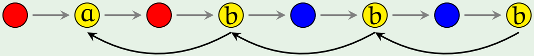
1. go to last position and check it is $\mathfrak{b}$

Data Walking Automata

Example 2

$\mathrm{L} = \left\{ \textit{words where all } \mathfrak{a} \textit{ are followed by } \mathfrak{b} \textit{ in a } \underline{\textit{different}} \textit{ class} \right\}$

To accept a data word, e.g.



a Data Walking Automata should perform the following steps:

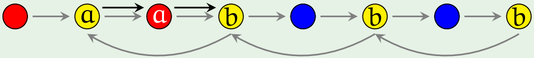
1. go to last position and check it is $\mathfrak{b}$
2. go to the last $\mathfrak{a}$ in the same class (accept if there is none)

Data Walking Automata

Example 2

$L = \{$ *words where all* a *are followed by* b *in a <u>different</u> class* $\}$

To accept a data word, e.g.



a Data Walking Automata should perform the following steps:

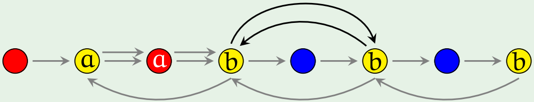
1. go to last position and check it is b
2. go to the last a in the same class (accept if there is none)
3. move to next b using global successor

Data Walking Automata

Example 2

$\mathrm{L} = \left\{ \textit{words where all } \mathtt{a} \textit{ are followed by } \mathtt{b} \textit{ in a } \underline{\textit{different}} \textit{ class} \right\}$

To accept a data word, e.g.



a Data Walking Automata should perform the following steps:

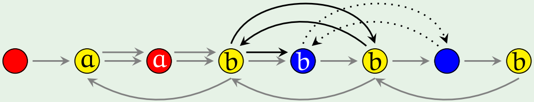
1. go to last position and check it is $\mathtt{b}$
2. go to the last $\mathtt{a}$ in the same class (accept if there is none)
3. move to next $\mathtt{b}$ using global successor
4. if class successor is $\mathtt{b}$...

Data Walking Automata

Example 2

$\mathrm{L} = \{$ *words where all* $\mathfrak{a}$ *are followed by* $\mathfrak{b}$ *in a* *<u>different</u> class* $\}$

To accept a data word, e.g.



a Data Walking Automata should perform the following steps:

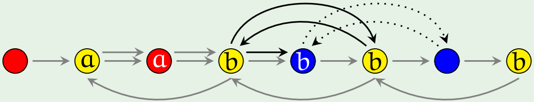
1. go to last position and check it is $\mathfrak{b}$
2. go to the last $\mathfrak{a}$ in the same class (accept if there is none)
3. move to next $\mathfrak{b}$ using global successor
4. if class successor is $\mathfrak{b}$... then repeat step 3

Data Walking Automata

Example 2

$\mathrm{L} = \{$ *words where all* $\mathtt{a}$ *are followed by* $\mathtt{b}$ *in a <u>different</u> class* $\}$

To accept a data word, e.g.



a Data Walking Automata should perform the following steps:

1. go to last position and check it is $\mathtt{b}$
2. go to the last $\mathtt{a}$ in the same class (accept if there is none)
3. move to next $\mathtt{b}$ using global successor
4. if class successor is $\mathtt{b}$... then repeat step 3
5. accept iff reached position is not the last.

Closures and other properties

Proposition

Non-deterministic Data Walking Automata are effectively closed under **union** and **intersection**.

Deterministic Data Walking Automata are effectively closed under **union**, **intersection**, and **complementation**.

Closures and other properties

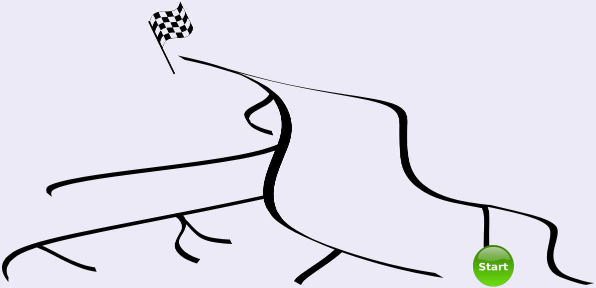
Proposition

Non-deterministic Data Walking Automata are effectively closed under **union** and **intersection**.

Deterministic Data Walking Automata are effectively closed under **union**, **intersection**, and **complementation**.

Proof idea (deterministic case)

Use **Sipser's technique** to avoid infinite loops:



Closures and other properties

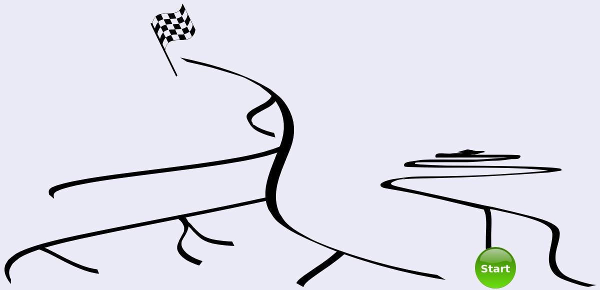
Proposition

Non-deterministic Data Walking Automata are effectively closed under **union** and **intersection**.

Deterministic Data Walking Automata are effectively closed under **union**, **intersection**, and **complementation**.

Proof idea (deterministic case)

Use **Sipser's technique** to avoid infinite loops:



Closures and other properties

Theorem

Deterministic		Non-deterministic		Tiling
Data Walking	$\subsetneq$	Data Walking	$\not\subseteq$	Automata
Automata		Automata		on data words

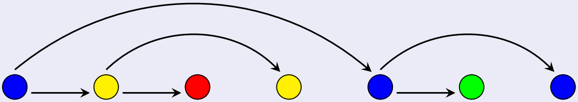
Closures and other properties

Theorem

Deterministic Data Walking Automata	$\subsetneq$	Non-deterministic Data Walking Automata	$\not\subseteq$	Tiling Automata on data words

Proof idea of separations

Reduce to analogous results [Colcombet and Bojanczyk 2006,2008] for Tree Walking Automata, by **encoding trees** with data words:



- root = first position
- child 1 = class successor
- child 2 = global successor (only if class successor is defined!)

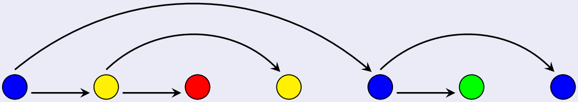
Closures and other properties

Theorem

Deterministic Data Walking Automata	$\subsetneq$	Non-deterministic Data Walking Automata	$\not\subseteq$	Tiling Automata on data words

Proof idea of separations

Reduce to analogous results [Colcombet and Bojanczyk 2006,2008] for Tree Walking Automata, by **encoding trees** with data words:



- root = first position
- child 1 = class successor
- child 2 = global successor (only if class successor is defined!)

a Deterministic Data Walking Automaton can check whether a data word is a valid tree encoding.

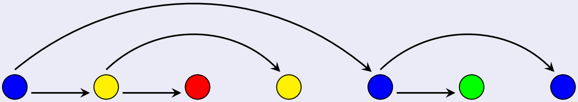
Closures and other properties

Theorem

Deterministic Data Walking Automata	$\subsetneq$	Non-deterministic Data Walking Automata	$\subseteq$ $\nsubseteq$	Tiling Automata on data words

Proof idea of separations

Reduce to analogous results [Colcombet and Bojanczyk 2006,2008] for Tree Walking Automata, by **encoding trees** with data words:



- root = first position
- child 1 = class successor
- child 2 = global successor (only if class successor is defined!)

a Deterministic Data Walking Automaton can check whether a data word is a valid tree encoding.

Decision problems

- A Deterministic Data Walking Automaton can recognize the language of all correct tilings on data words and hence the set of all runs of a Tiling Automaton

Decision problems

☛ A Deterministic Data Walking Automaton can recognize the language of all correct tilings on data words and hence the set of all runs of a Tiling Automaton

Corollary

Tiling Automata recognize **projections** of languages recognized by Deterministic Data Walking Automata.

Corollary

Emptiness of Deterministic Data Walking Automata is as hard as emptiness of Tiling Automata and **Petri Nets reachability**.

Decision problems

Theorem

Non-deterministic Data Walking Automata	$\subseteq$	Tiling Automata on data words

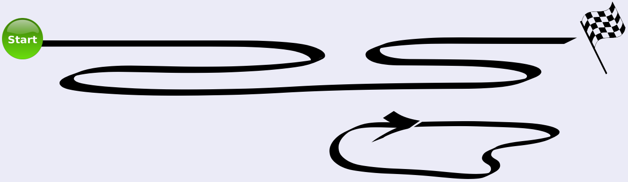
Decision problems

Theorem

Non-deterministic Data Walking Automata $\subseteq$ Tiling Automata on data words

Proof idea

W.l.o.g. acceptance witnessed by a graph with no accessible loops



☞ Every internal vertex (i.e. position + state) has **in-degree = out-degree = 1** $\Rightarrow$ Special form of tiling

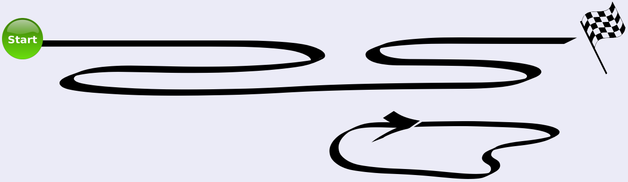
Decision problems

Theorem

Non-deterministic Data Walking Automata $\subseteq$ Tiling Automata on data words

Proof idea

W.l.o.g. acceptance witnessed by a graph with no accessible loops



Every internal vertex (i.e. position + state) has
in-degree = out-degree = 1 $\Rightarrow$ Special form of tiling

Corollary

Emptiness is decidable for Non-det. Data Walking Automata.

Decision problems

Theorem

Complements of Non-det. Data Walking Automata	$\subseteq$	Tiling Automata on data words

Decision problems

Theorem

Complements of Non-det. Data Walking Automata	$\subseteq$	Tiling Automata on data words

Proof idea

Consider any set of vertices (positions + state) that

1. contains the initial configuration
2. is closed under all possible transitions.

☞ The automaton rejects the input data word iff the final configuration is not in the above set

☞ The above set is a special form of tiling

Decision problems

Theorem

Complements of Non-det. Data Walking Automata	$\subseteq$	Tiling Automata on data words

Proof idea

Consider any set of vertices (positions + state) that

1. contains the initial configuration
2. is closed under all possible transitions.

☞ The automaton rejects the input data word iff the final configuration is not in the above set

☞ The above set is a special form of tiling

Corollary

Universality is decidable for Non-det. Data Walking Automata.

Decision problems

☞ Tiling Automata are closed under unions and intersections.

Corollary

Containment and, more generally, any boolean combination of Non-deterministic Data Walking Automata is decidable.

The general picture