

Logics and automata over infinite alphabets

Anca Muscholl

LIAFA, Univ. Paris 7 & CNRS

Joint work with:

Mikołaj Bojańczyk (Warsaw, Paris), Claire David (Paris),
Thomas Schwentick (Dortmund) and Luc Segoufin (Paris)

Plan

- Infinite alphabets: motivation and applications
- Data languages
- Logics and automata with data
- Two-variable case
- Open problems

Data

Automata-based techniques in program verification and manipulation of XML data mostly use abstraction:

- program verification: **abstraction** of values to **finite** range domains or **finitary** representations (eg. semi-linear sets, well quasi-orders, time regions/zones)
- semi-structured data (XML): documents viewed as ranked/unranked trees with labels from **finite** domain (no attributes, references etc)

Example: Parametrized verification

- Check that an (asynchronous) product of n identical processes that communicate in a given topology (e.g. token-based in array or ring) satisfies a property for **any n** .
- Property refers to process actions indexed by **process-id** → **infinite alphabet**
- Parametrized verification is **undecidable** (Apt/Kozen'86).
- **Decidability** obtained using network invariants, finite models, regular model-checking.

Parametrized verification: decidability

→ (Emerson/Namjoshi '95)

$\forall\forall$ -CTL* properties about single processes or pairs of processes in token-passing ring topology:

$$\forall i, j, i \neq j. AG \neg(\text{crit}(i) \wedge \text{crit}(j))$$

→ Properties: $\forall i. P(i), \forall i, j, i \neq j. P(i, j),$

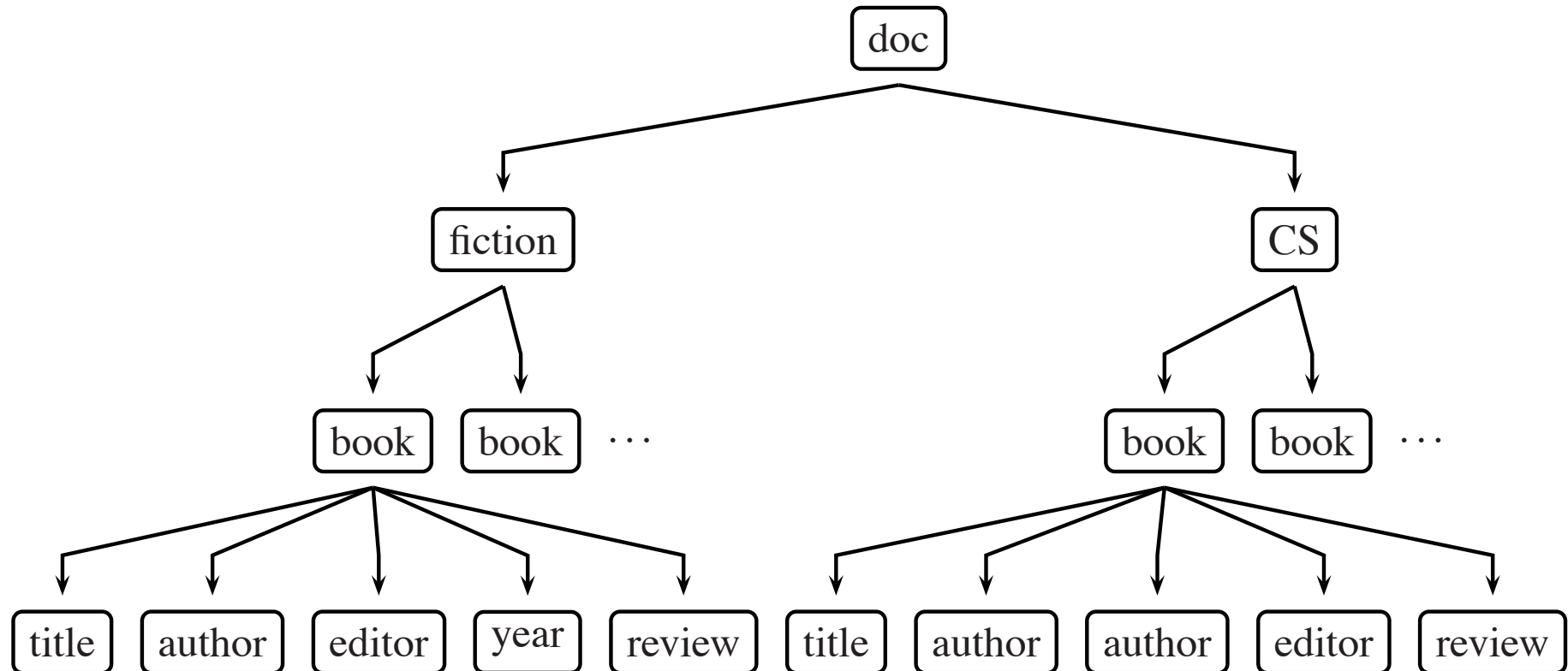
$\forall i, j, i \neq j. P(i, i + 1, j)$ with P in CTL*.

→ Show that such properties have **constant cutoffs**, using symmetry.

Parametrized verification: decidability

- (Abdulla, Jonsson et al. '04)
LTL(MSO) over 2-dim. matrix (time t , process i):
 $\forall i. G F(a(i) \vee \neg \text{enabled}(a(i)))$
- LTL reasons about time, MSO about space (processes).
- Show that **regular model-checking** applies (i.e., one-step transitions expressed by transducers; study transducers for which iteration is computable).

Example: semi-structured data (XML)



XML documents:

- hierarchically structured through **tags**
- linear representation of unranked, ordered **tree**
- tree shape defined through **type** (DTD): vertical and horizontal (regular) restrictions on tags

XML documents:

- hierarchically structured through **tags**
- linear representation of unranked, ordered **tree**
- tree shape defined through **type** (DTD): vertical and horizontal (regular) restrictions on tags

... and beyond:

- **arbitrary data** such as text (leaf nodes), attribute values, references etc.

Queries:

- selection of all tree nodes satisfying a property
- query expressed by e.g. XQuery, XPath, FO, MSO, automata, ...

Queries:

- selection of all tree nodes satisfying a property
- query expressed by e.g. XQuery, XPath, FO, MSO, automata, ...

Consistency:

Given a regular tree language R and a set of constraints \mathcal{C} on values, check the existence of some $t \in R$ that satisfies \mathcal{C} .

Theory for infinite alphabets

→ Look for a nice theory of **regular-like** word and tree languages over **infinite alphabets**.

Nice means:

- **Decidability** of emptiness and satisfiability.
- **Good** complexities.
- **Equivalence** between automata, logics (MSO, temporal,...), algebra and expressions.

Theory for infinite alphabets

→ Look for a nice theory of **regular-like** word and tree languages over **infinite alphabets**.

Nice means:

- **Decidability** of emptiness and satisfiability.
- **Good** complexities.
- **Equivalence** between automata, logics (MSO, temporal,...), algebra and expressions.

Operations on infinite alphabet: only **equality comparisons**

Plan

- Infinite alphabets: motivation and applications
- **Data languages**
- Logics and automata with data
- Two-variable case
- Open problems

Data languages

Data words:

$(\Sigma \times D)$ -labeled finite words, with
 Σ finite and D infinite alphabet (values)

Data word language $L \subseteq \Sigma_{\sim}^*$

Data trees:

same for unranked, ordered trees

Only equality tests on values \rightarrow positions (nodes) in
data words (trees) are partitioned into classes.

Reasoning with data (1)

→ XML: **satisfiability** of logics (resp. automata emptiness)

Query: Books that have been reedited:

$$\exists y. (x.\text{isbn} = y.\text{isbn} \wedge x.\text{year} \neq y.\text{year})$$

Unary keys: Attribute A has distinct values:

$$\forall x, y. (x.A = y.A \rightarrow x = y)$$

Navigation: From node x we can access nodes y_1, y_2 via paths of type $p_1, p_2 \in R$ such that $y_1.A = y_2.B$.

Reasoning with data (2)

→ Verification: **model-checking/satisfiability**

Given a parametric model S_n (array, ring,...) and a property $P(n)$, is there some n such that S_n violates $P(n)$?

→ Model S_n gives for each process i a "finite-state" description R_i that may refer to states of other processes:
e.g. process $i - 1$ is in state q .

→ Associate each process with a **distinct value** and consider the set of runs over pairs (state,process-value).

$$(a_1^1, 1)(a_2^1, 2) \cdots (a_n^1, n) \# (a_1^2, 1)(a_2^2, 2) \cdots (a_n^2, n) \# \cdots$$

Plan

- Infinite alphabets: motivation and applications
- Data languages
- **Logics and automata with data**
- Two-variable case
- Open problems

Automata over data words (1)

k-register automata (k-RA) [Kaminski/Francez '94, Bouyer/Petit/Thérien '03, Neven/Schwentick/Vianu '04]

→ upon reading $(a, \nu) \in \Sigma \times D$:

- can check in which registers value ν occurs
- can store value ν into a register

→ can be 1- or 2-way, deterministic (D), non-deterministic (N) or alternating (A)

Automata over data words (1)

Projection of $L \subseteq \Sigma_{\sim}^*$:

$$\text{Proj}(L) = \{a_1 \cdots a_n \in \Sigma^* \mid \exists (a_1, \nu_1) \cdots (a_n, \nu_n) \in L\}$$

Let L be recognized by a 1-way k -RA (D/N). Then $\text{Proj}(L)$ is a regular language. From a k -RA \mathcal{A} that accepts L one can construct an automaton of size $|\mathcal{A}|2^{O(k^2)}$ that accepts $\text{Proj}(L)$.

Automata over data words (1)

Projection of $L \subseteq \Sigma_{\sim}^*$:

$$\text{Proj}(L) = \{a_1 \cdots a_n \in \Sigma^* \mid \exists (a_1, \nu_1) \cdots (a_n, \nu_n) \in L\}$$

Let L be recognized by a 1-way k -RA (D/N). Then $\text{Proj}(L)$ is a regular language. From a k -RA \mathcal{A} that accepts L one can construct an automaton of size $|\mathcal{A}|2^{O(k^2)}$ that accepts $\text{Proj}(L)$.

proof: matrix $\{=, \neq\}^{k \times k}$ says which registers are equal;
guess (in)equalities on-the-fly

Automata over data words (1)

Projection of $L \subseteq \Sigma_{\sim}^*$:

$$\text{Proj}(L) = \{a_1 \cdots a_n \in \Sigma^* \mid \exists (a_1, \nu_1) \cdots (a_n, \nu_n) \in L\}$$

Let L be recognized by a 1-way k -RA (D/N). Then $\text{Proj}(L)$ is a regular language. From a k -RA \mathcal{A} that accepts L one can construct an automaton of size $|\mathcal{A}|2^{O(k^2)}$ that accepts $\text{Proj}(L)$.

Rem.

Projections of alternating 1-way RA (1-ARA) need not be regular. 1-ARA can test that all values are distinct.

Automata over data words (1)

Projection of $L \subseteq \Sigma_{\sim}^*$:

$$\text{Proj}(L) = \{a_1 \cdots a_n \in \Sigma^* \mid \exists (a_1, \nu_1) \cdots (a_n, \nu_n) \in L\}$$

Let L be recognized by a 1-way k -RA (D/N). Then $\text{Proj}(L)$ is a regular language. From a k -RA \mathcal{A} that accepts L one can construct an automaton of size $|\mathcal{A}|2^{O(k^2)}$ that accepts $\text{Proj}(L)$.

Rem. (Bouyer et al.)

Emptiness of 1-way k -RA (D/N) is decidable in PSPACE.

Automata over data words (2)

k-pebble automata (k-PA) [Neven/Schwentick/Vianu '04]:

nested pebbles

→ upon reading $(a, \nu) \in \Sigma \times D$:

- can check which pebbles mark positions with value ν
- can check which pebbles are under head position
- can lift highest pebble (head returns at previous pebble) and place new pebble

→ can be 1- or 2-way, deterministic (D), non-deterministic (N) or alternating (A)

Logic with data

→ $\text{FO}(<, +1, \sim, \oplus 1)$:

- $a(x)$, with $a \in \Sigma$
- order $<$, successor $+1$,
- binary relations "same value" \sim , class successor $\oplus 1$

→ $\text{EMSO}(<, +1, \sim, \oplus 1)$:

$\text{FO}(<, +1, \sim, \oplus 1)$ + existential set quantification

Logic with data

→ $\text{FO}(<, +1, \sim, \oplus 1)$:

- $a(x)$, with $a \in \Sigma$
- order $<$, successor $+1$,
- binary relations "same value" \sim , class successor $\oplus 1$

→ $\text{EMSO}(<, +1, \sim, \oplus 1)$:

$\text{FO}(<, +1, \sim, \oplus 1)$ + existential set quantification

→ Models: **Data words and trees.**

(Trees: vertical/horizontal successors.)

Example 1

Data language L with **non-regular** $\text{Proj}(L)$:

→ Each class contains precisely one a followed by one b , or one b only.

$$\forall x \forall y. ((x < y \wedge x \sim y) \rightarrow (a(x) \wedge b(y)))$$

$$\wedge \forall x \exists y. (b(x) \vee (x \sim y \wedge x \neq y))$$

$$\wedge \forall x \forall y. ((a(x) \wedge b(y)) \rightarrow x < y)$$

$$\text{Proj}(L) = \{a^n b^m \mid m \geq n \geq 0\}$$

Example 1

Data language L with **non-regular** $\text{Proj}(L)$:

→ Each class contains precisely one a followed by one b , or one b only.

$$\forall x \forall y. ((x < y \wedge x \sim y) \rightarrow (a(x) \wedge b(y)))$$

$$\wedge \forall x \exists y. (b(x) \vee (x \sim y \wedge x \neq y))$$

$$\wedge \forall x \forall y. ((a(x) \wedge b(y)) \rightarrow x < y)$$

$$\text{Proj}(L) = \{a^n b^m \mid m \geq n \geq 0\}$$

L is accepted by a 1-way 1-ARA.

Example 2

Sequences over $\{0, 1\}$ with **same subsequence** of 0-values and 1-values:

- All 0's have different values; same for 1's.
- Bijection between 0-values and 1-values.
- For each pair of 0-positions $x < y$ and every 1-position z with $x \sim z$ there exists a 1-position $z < z'$ with $y \sim z'$.

Example 2

Sequences over $\{0, 1\}$ with **same subsequence** of 0-values and 1-values:

- All 0's have different values; same for 1's.
- Bijection between 0-values and 1-values.
- For each pair of 0-positions $x < y$ and every 1-position z with $x \sim z$ there exists a 1-position $z < z'$ with $y \sim z'$.

Needs 3 variables. Is accepted by 2-way 2-RA or 2-PA.

Undecidability (1)

$\text{FO}^k(<, +1, \sim, \oplus 1)$: k variables only

$\text{EMSO}^k(<, +1, \sim, \oplus 1)$: existential set quantification + FO^2

Undecidability (1)

$\text{FO}^k(<, +1, \sim, \oplus 1)$: k variables only

$\text{EMSO}^k(<, +1, \sim, \oplus 1)$: existential set quantification + FO^2

Satisfiability of $\text{FO}^3(<, \sim)$ formulas and emptiness of 2-way 2-RA (on data words) is undecidable.

Undecidability (1)

$\text{FO}^k(<, +1, \sim, \oplus 1)$: k variables only

Satisfiability of $\text{FO}^3(\sim, <)$ formulas and emptiness of 2-way 2-RA is undecidable.

proof: Post Correspondence encoding:

$$u_{i_1} \overline{v_{i_1}} u_{i_2} \overline{v_{i_2}}, \dots, u_{i_n} \overline{v_{i_n}} \in \text{Reg} \cap \text{EqualSequences}$$

Undecidability (2)

Emptiness of 2-way 1-RA is undecidable.

proof: Encoding of 2-counter (Minsky) machine

- Configuration $(p, m, n) \in Q \times \mathbb{N} \times \mathbb{N}$ encoded as data word:

$$\cdots \# p (a, \nu_1) \cdots (a, \nu_m) (b, \nu'_1) \cdots (b, \nu'_n) \# \cdots$$

with distinct values ν_i, ν'_j

Undecidability (2)

Emptiness of 2-way 1-RA (on data words) is undecidable.

proof:

- Transition $\langle p, \text{Inc}_1, q \rangle$:

⋮

$p (a, \nu_1) \cdots (a, \nu_m)(b, \nu'_1) \cdots (b, \nu'_n)$

$q (a, \nu_{\pi(1)}) \cdots (a, \nu_{\pi(m)})(a, \nu_{\text{ins}})(b, \nu'_{\sigma(1)}) \cdots (b, \nu'_{\sigma(n)})$

⋮

Rem. Needs bidirectionality.

Automata versus Logics (1)

[Neven/Schwentick/Vianu '04]: mostly uncomparability results, better behavior for pebble automata (PA) than register automata (RA)

- 2-way RA are not captured by MSO
- FO is not captured by 2-way alternating RA (ARA)
- 2-way APA are captured by MSO
- FO is captured by 2-way strong DPA
- emptiness undecidable for weak 1-way PA

Automata versus Logics (2)

[Demri/Lazic '06]: RA versus LTL + freeze

LTL_n + freeze: n registers + new formulas $\downarrow_k \varphi$, $\uparrow_k \varphi$

$w, i \models \downarrow_k \varphi$: store value of position i into register k and satisfy φ

$w, i \models \uparrow_k \varphi$: check that value of position i is equal to register k and satisfy φ

Automata versus Logics (2)

[Demri/Lazic '06]: RA versus LTL + freeze

LTL_n + freeze: n registers + new formulas $\downarrow_k \varphi$, $\uparrow_k \varphi$

$w, i \models \downarrow_k \varphi$: store value of position i into register k and satisfy φ

$w, i \models \uparrow_k \varphi$: check that value of position i is equal to register k and satisfy φ

- $LTL_n(X, X^{-1}, U, U^{-1})$ is included in 2-way n -ARA.
- Over **finite** data words, satisfiability of $LTL_1(X, F)$ and non-emptiness of 1-way **1**-ARA are non primitive-recursive. Over **infinite** words, the above problems are undecidable.

Automata versus Logics (3)

proof idea (emptiness of 1-way 1-ARA):

Gainy 2-counter machine: emptiness is non primitive-recursive (Schnoebelen '02)

- Same encoding of configurations as for 2-way 1-RA
- With 1-way moves (plus alternation), an automaton can only check that the data values of configuration C_i are contained in those of configuration C_{i+1} (but not equal)

Automata versus Logics (3)

proof idea (emptiness of 1-way 1-ARA):

Gainy 2-counter machine: emptiness is non primitive-recursive (Schnoebelen '02)

- Transition $\langle p, \text{Dec}_1, q \rangle$:

⋮

$p (a, \nu_1)(a, \nu_2) \cdots (a, \nu_m)(b, \nu'_1) \cdots (b, \nu'_n)$

$q (a, \nu_{\pi(2)})(a, \tau_1)(a, \tau_2) \cdots (a, \tau_3)(a, \nu_{\pi(m)})(a, \tau_4)$

$(b, \nu'_{\sigma(1)})(b, \tau_5)(b, \tau_6) \cdots (b, \nu'_{\sigma(n)})(b, \tau_{42})$

⋮

Plan

- Infinite alphabets: motivation and applications
- Data languages
- Logics and automata with data
- **Two-variable case**
- Open problems

Two-variable case

Why considering two-variables (first-order) logic?

Two-variable case

Why considering two-variables (first-order) logic?

- More hope for decidability. Rich structure over words.

Two-variable case

Why considering **two-variables** (first-order) logic?

- More hope for decidability. Rich structure over words.
- Navigation formalism **Core XPath** (without attributes) corresponds to $FO^2(<, +1)$ over trees [Gottlob et al. '02, Marx '05]. With one attribute it includes $FO^2(<, +1, \sim)$.

Two-variables

- FO^2 (over graphs) has finite model property
(Mortimer'75), **NEXPTIME-complete** (Grädel, Otto '99)

Two-variables

- FO^2 (over graphs) has finite model property (Mortimer'75), **NEXPTIME-complete** (Grädel, Otto '99)
- Over **words**: $\text{FO}^2(<)$ is equivalent to
 - unary LTL and $\Sigma^2 \cap \Pi^2$ (Etessami, Vardi, Wilke '02)
 - variety DA (Thérien, Wilke '98)
 - 2-way partially-ordered DFA (Schwentick, Thérien, Vollmer '01)and is **NEXPTIME-complete**.

2-variable properties

Essentially regular + counting properties :

- Same number of a 's and b 's (and c 's).

2-variable properties

Essentially regular + counting properties :

- Same number of a 's and b 's (and c 's).
- The first a in the **second** class containing an a :

$$\dots (a, \nu_1) \cdots (a, \nu_1) \cdots (a, \nu_1) \cdots (a, \nu_2) \dots$$

2-variable properties

Essentially regular + counting properties :

- Same number of a 's and b 's (and c 's).
- The first a in the **second** class containing an a :

$$\dots (a, \nu_1) \cdots (a, \nu_1) \cdots (a, \nu_1) \cdots (a, \nu_2) \dots$$

- Every second position is in a **new** class.

$$\exists X_0 \exists X_1 (\cdots \wedge \forall x (x \in X_1 \rightarrow (\forall y (y < x \rightarrow x \not\sim y))))$$

2-variable properties

- (Björklund/Schwentick '06)

1-way k -RA are included in $\text{EMSO}^2(<, +1, \sim, \oplus 1)$

- "Every class contains an a -position x s.t. the class successor $x \oplus 1$ is an a , too."

Is **not** expressible in $\text{FO}^2(<, +1, \sim)$ and **not** recognized by a 1-way RA. Is expressible in $\text{FO}^2(<, +1, \sim, \oplus 1)$.

2-variable data logic over words

Satisfiability of $\text{EMSO}^2(<, +1, \sim, \oplus 1)$ formulas over data words is decidable.

Complexity:

2-NEXPTIME reduction from $\text{EMSO}^2(<, +1, \sim, \oplus 1)$ formulas to emptiness of multicounter automata (Petri nets).

PTIME reduction from emptiness of multicounter automata to $\text{FO}^2(<, +1, \sim)$ (and to $\text{FO}^2(+1, \sim, \oplus 1)$).

Satisfiability proof: words

Multicounter automata (Petri nets):

finite automata + positive counters

- no test for zero (except at the end)
- accept with final state + all counters zero

Emptiness of multicounter automata is decidable (Mayr, Kosaraju '84). Complexity is not known to be elementary.

Satisfiability proof: words

projection $\text{Proj}(L)$ of data language $L \subseteq \Sigma_{\sim}^*$:
projection onto Σ -component

Satisfiability proof: words

projection Proj(L) of data language $L \subseteq \Sigma_{\sim}^*$:
projection onto Σ -component

Projections of EMSO²($<, +1, \sim, \oplus 1$) definable languages
are recognized by multcounter automata.

Satisfiability proof: words

projection $\text{Proj}(L)$ of data language $L \subseteq \Sigma_{\sim}^*$:
projection onto Σ -component

Projections of $\text{EMSO}^2(<, +1, \sim, \oplus 1)$ definable languages are recognized by multicounter automata.

Idea: $\text{EMSO}^2(<, +1, \sim, \oplus 1)$ formulas express (modulo recoloring) 1) **regular** properties of $\text{Proj}(L)$ and 2) **regular** properties for each class (subsequence of \sim -equivalent positions).

Example: $\text{Proj}(L) = \{a^n b^n \mid n \geq 0\}$

L : each class contains precisely one a and one b (to its right) \Rightarrow each class equals ab

Example: $\text{Proj}(L) = \{a^n b^n \mid n \geq 0\}$

L : each class contains precisely one a and one b (to its right) \Rightarrow each class equals ab

Equivalent:

$$\text{Proj}(L) = \text{Shuffle}(\{ab\}) \cap a^*b^*$$

Shuffle of words w_1, \dots, w_n :

set of words w that can be colored with n colors s.t. the subsequence of w colored by k is w_k .

Shuffle

example: $aaabbb, aababb \in \text{Shuffle}(\{ab\})$

Shuffle

example: $aaabbb, aababb \in \text{Shuffle}(\{ab\})$

$\text{Shuffle}(L)$ = set of shuffles of words in L

For any regular language L , the language $\text{Shuffle}(L)$ is recognized by a multcounter automaton (Gischer '81).

Shuffle

example: $aaabbb, aababb \in \text{Shuffle}(\{ab\})$

$\text{Shuffle}(L) = \text{set of shuffles of words in } L$

For any regular language L , the language $\text{Shuffle}(L)$ is recognized by a multcounter automaton (Gischer '81).

Conversely: the set of accepting runs of a multcounter automaton can be expressed as $\text{Shuffle}(L) \cap R$.

EMSO²($<$, $+1$, \sim , $\oplus 1$) and automata

Data automaton $(\mathcal{A}, \mathcal{B})$

- **Base** automaton \mathcal{A} : non-deterministic (letter-to-letter) transducer
- **Class** automaton \mathcal{B} : finite-state automaton

EMSO²($<, +1, \sim, \oplus 1$) and automata

Data automaton $(\mathcal{A}, \mathcal{B})$

- **Base** automaton \mathcal{A} : non-deterministic (letter-to-letter) transducer
- **Class** automaton \mathcal{B} : finite-state automaton

Accepts a data word w by:

- **Base** automaton outputs a word x (over finite alphabet)
- Check for each \sim -class that the subword of x corresponding to the class is accepted by the **class** automaton.

EMSO²(+1, ~): better complexity

Projections of EMSO²(+1, ~) definable languages are recognized by linear constraint automata (LCA).

LCA: finite automata with acceptance defined by linear constraints on the number of transitions

EMSO²(+1, ~): better complexity

Projections of EMSO²(+1, ~) definable languages are recognized by linear constraint automata (LCA).

LCA: finite automata with acceptance defined by linear constraints on the number of transitions

- class automaton only checks that some letters occur exactly once/never
- accepting run of LCA yields assignment of values left-to-right

EMSO²(+1, ~): better complexity

Projections of EMSO²(+1, ~) definable languages are recognized by linear constraint automata (LCA).

LCA: finite automata with acceptance defined by linear constraints on the number of transitions

→ formula yields a 2exp-sized LCA

Emptiness of linear constraint automata is NP-complete.

[Seidl/Schwentick/M./Habermehl '04]

EMSO²($<$, $+1$, \sim) on unranked trees

Emptiness of **tree multicounter automata** can be reduced to satisfiability of FO²($<$, $+1$, \sim) on binary trees.

→ **tree multicounter automata**: bottom-up tree automata with counters; transitions sum up the values of the children (+ increment/decrement).

→ Emptiness of tree multicounter automata is an **open problem** [de Groote et al., MELL].

Rem. Converse reduction is open.

EMSO²($<$, $+1$, \sim) on unranked trees

EMSO²(\sim , $+1$) on unranked (finite) trees is decidable.

EMSO²(\sim , $+1$) on unranked trees captures a fragment of Core XPath + data that restricts data comparisons (+ convenient encoding of attributes in the data tree)

Cannot do: from every node x we can reach y , then z (via regular paths), s.t. $x \sim z$

EMSO²($<$, $+1$, \sim) on unranked trees

EMSO²($+1$, \sim) on unranked (finite) trees is decidable

Idea:

- Satisfiability of EMSO²($+1$, \sim) formulas is reduced to non-emptiness of **puzzle automaton**.
- Non-empty puzzle automata have **small** models.
- Small models are recognized by **linear constraint** tree automata (= bottom-up unranked TA with linear constraints over transitions frequencies).
- Emptiness of linear constraint tree automata is decidable.

Proof ideas: trees

Puzzle automaton $\mathcal{A}, (D_k, S_k)_{k=1}^m$: aka normal form

- \mathcal{A} is a bottom-up letter-to-letter transducer with finite output alphabet Γ
- $D_k, S_k \subseteq \Gamma$ are alphabets

Proof ideas: trees

Puzzle automaton $\mathcal{A}, (D_k, S_k)_{k=1}^m$: aka normal form

- \mathcal{A} is a bottom-up letter-to-letter transducer with finite output alphabet Γ
- $D_k, S_k \subseteq \Gamma$ are alphabets

A **data tree** $t \in \mathcal{T}(\Sigma)_{\sim}$ is accepted if:

- \mathcal{A} has an accepting run on the marked Σ -projection of t with output $t' \in \mathcal{T}(\Gamma)$
- every class of t (t') satisfies some pair (D_k, S_k) : each letter in D_k occurs exactly once and there are no other occurrences than $D_k \cup S_k$

Proof ideas: trees

- **Small models** of puzzle automata: few large connected zones with same data value + few nodes with many marked children
- **Linear constraint automata**: can recognize the projections of small models by counting that letters from each D_k appear equally often.

Unranked trees: related work

- Core XPath + data is **undecidable** (Geerts/Fan '05)
- Restricted fragment of Core XPath + data is **decidable** (Benedikt/Fan/Geerts '05): no horizontal navigation + either no negation or no vertical order
- Consistency problem for unary/foreign keys wrt. DTDs is **decidable** (Arenas/Fan/Libkin '05): included into $\text{EMSO}^2(+1, \sim)$

Open problems and conclusion

- words: "nice" automata equivalent to $\text{EMSO}^2(<, +1, \sim, \oplus 1)$?
- trees: satisfiability of $\text{FO}^2(<, +1, \sim)$?
- trees: reduction to tree multicounter automata?
- applications to the verification of parametrized systems?
extensions of $\text{EMSO}^2(<, +1, \sim, \oplus 1)$?

Open problems and conclusion

- words: "nice" automata equivalent to $\text{EMSO}^2(<, +1, \sim, \oplus 1)$?
- trees: satisfiability of $\text{EMSO}^2(<, +1, \sim)$?
- trees: reduction to tree multicounter automata?
- applications to the verification of parametrized systems?
extensions of $\text{EMSO}^2(<, +1, \sim, \oplus 1)$?

Thank you for your attention!