Counter Automata and Classical Logics for Data Words

Amal Dev Manuel

amal@imsc.res.in

Institute of Mathematical Sciences,
Taramani, Chennai, India.

January 31, 2012

Data Words

Definition (Data Words)

A data word $w = (a_1, d_1) \ldots (a_n, d_n)$, $a_i \in \Sigma$, $d_i \in \Delta$ where,

- Σ is a finite alphabet.
- Δ is an (recursive) infinite set .

Definition (Data Language)

A data language $L \subseteq (\Sigma \times \Delta)^*$.

Example

$L_{\exists n}$	All w in which at least n distinct data values occur.
$L_{<n}$	All w in which every data value occurs at most n times.
$L_{a^*b^*}$	All w whose string projections are in the set a^*b^*.
L_a	All w under the label a are different.
$L_{a \to b}$	All w occurring under a occurs under b as well.
L_{dd}	There is a d in w which occurs in consecutive positions.

Regularity for Data Languages

$$\text{Regularity — Confluence of} \begin{cases} \text{Robustness,} \\ \text{Low complexity decision problems,} \\ \text{Alternate characterizations,} \\ \text{Nice closure properties.} \end{cases}$$

Question. What constitutes the class of regular data languages?
Approach. Try to extend regular word "devices" to data words.

"devices" – Regular expressions, Linear grammars, Monadic second order logic, Finite state automata.

Extensions of finite state automata

Memory-structures

- stack
- push-down
- hash-table
- registers
- counters

Register automata

Finite state automata + registers storing data values

Definition ([KF94])

A k-**Register automaton** $A = (Q, \Sigma, \Delta, k, q_0, F)$, where

- Q is a finite set of states
- $q_0 \subseteq Q$ is the initial state
- $F \subseteq Q$ is the set of final states
- k is the number of registers
- $\Delta \subseteq (Q \times \Sigma \times [k] \times Q) \cup (Q \times \Sigma \times Q \times [k])$

For $p, q \in Q$, $a \in \Sigma$, $i \in [k]$, transitions of the form (p, a, i, q) are called `read` transitions and transitions of the form (p, a, q, i) are called `write` transitions.

Register automaton – example

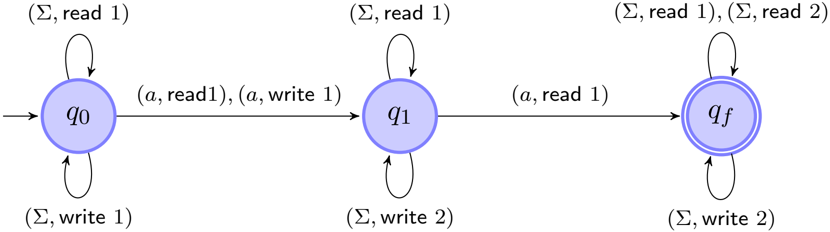


Figure: Register automaton accepting the language $\overline{L_a}$.

Register automaton – example

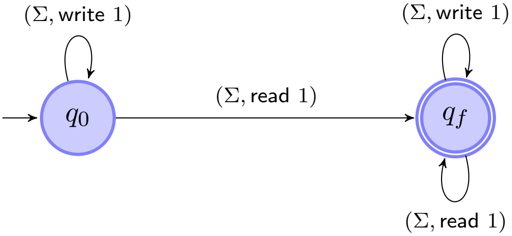


Figure: 1-Register automaton accepting the language L_{dd}

Register automaton – properties

Fact

Register automata are closed under union, intersection, length-preserving morphisms.

Not closed under complementation (L_a is not accepted by any register automaton.)

Register automaton – properties

Fact

Register automata are closed under union, intersection, length-preserving morphisms.

Not closed under complementation (L_a is not accepted by any register automaton.)

Lemma

If a k-register automaton A accepts any word at all, then it accepts a word containing at most $k+1$ distinct data values.

Register automaton – properties

Fact

Register automata are closed under union, intersection, length-preserving morphisms.

Not closed under complementation (L_a is not accepted by any register automaton.)

Lemma

If a k-register automaton A accepts any word at all, then it accepts a word containing at most $k+1$ distinct data values.

Theorem ([KF94])

Emptiness checking of register automata is decidable (NP-c).

Data automaton

Definition

A **data automaton** is a tuple $A = (B, C)$ where

- B is a finite state transducer with input alphabet Σ and output alphabet Σ'.
- C is a finite state auotmaton with alphabet Σ'.

A has an (accepting) run on w if

- B has an (accepting) run on w defining a unique output word w'.
- C has an (accepting) run on each class of w'.

Data automaton – example

Example (The language L_a)

- The transducer B is a copy machine, copies every letter to the output
- The automaton C accepts the language $\overline{\Sigma^* a \Sigma^* a \Sigma^*}$.

Data automaton – example

Example (The language L_{dd})

Choose the intermediate alphabet to be $\{0, 1\}$.

- B chooses two consecutive positions and label them by '1', all other positions are labelled 0.
- The automaton C accepts the language $0^*10^*10^* + 0^*$.

Data automaton – properties

Theorem ([KF94, BS10])

Register automata are strictly less powerful than Data automata in terms of expressiveness.

Theorem ([BMS$^+$06, BS10])

The emptiness problem for Data automata is decidable (not known to be elementary).

Counters for data words

Setup : Finite state automata + $|\Gamma|$-many counters.

- A counter for each data value.
- All counters are initially zero.
- Whenever the automaton encounters a pair (a, d)
  - The counter for d is checked against a constraint,
  - Counter is incremented or reset.

Class counting automata

Definition

A **class counting automaton**, abbreviated as CCA, is a tuple $\mathrm{CCA} = (Q, \Sigma, \Delta, I, F)$, where

- Q is a finite set of states,
- $I \subseteq Q$ is the set of initial states,
- $F \subseteq Q$ is the set of final states,
- $\Delta \subseteq_{fin} (Q \times \Sigma \times C \times \mathsf{Inst} \times \mathbb{N} \times Q)$, $\mathsf{Inst} = \{\mathsf{inc}, \mathsf{reset}\}$, C is the set of all univariate inequalities over $\mathbb{N}$.

Class counting automata – run

- A **configuration** of A is a pair (q, h), where $q \in Q$ and $h : \Gamma \to \mathbb{N}$.
- An initial configuration of A is (q_0, h_0), $q_0 \in I$ and $\forall d \in \Gamma, h_0(d) = 0$.

Given a data word $w = (a_1, d_1), \dots (a_n, d_n)$, a **run** of A on w is a sequence $\gamma = (q_0, h_0)(q_1, h_1) \dots (q_n, h_n)$ such that (q_0, h_0) is an initial configuration and for each $1 \leq i \leq n$ there exists a transition $t_i = (q, a, c, \pi, m, q') \in \Delta$ such that $q = q_i$, $q' = q_{i+1}$, $a = a_{i+1}$ and:

- $h_i(d_{i+1}) \models c$.
- h_{i+1} is given by:

$$h_{i+1} = \begin{cases} h_i \oplus (d_{i+1}, m') & \text{if} \quad \pi = \mathsf{inc}, m' = h_i(d_{i+1}) + m \\ h_i \oplus (d_{i+1}, m) & \text{if} \quad \pi = \mathsf{reset} \end{cases}$$

CCA – example

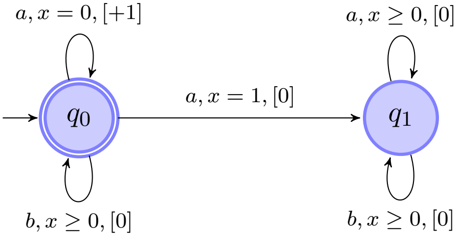


Figure: CCA accepting the language L_a

CCA – example

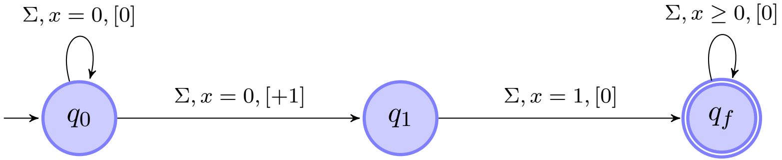


Figure: CCA accepting the language L_{dd}.

CCA – properties

Fact

CCA-recognizable data languages are closed under union and intersection but not under complementation.

CCA – properties

Fact

CCA-recognizable data languages are closed under union and intersection but not under complementation.

Theorem

The non-emptiness problem for CCA is EXPSPACE*-complete.*

CCA – extensions and subclasses

- Many bag CCA is equivalent to one bag CCA.
- CCA + context check contains register automata.
- CCA with counter acceptance conditions is equivalent to Data automata.
- CCA with presburger constraints is still in EXPSPACE.
- Two-way-ness and alternation leads to undecidability.

Logic for data words

A data word can be naturally represented as a first-order structure $w = ([n], \Sigma, <, +1, \sim)$.

Example

The word *ababab* is encoded as the structure,

$$([6], P_a = \{1,3,5\}, P_b = \{2,4,6\}, <, +1)\,.$$

Example

The data word $(a, d_2)(b, d_1)(a, d_1)(b, d_2)(a, d_3)(b, d_2)$ is encoded as the structure,

$$([6], P_a = \{1,3,5\}, P_b = \{2,4,6\}, <, +1, \sim = \{\{1,4,6\}, \{2,3\}, \{5\}\})\,.$$

First-order logic over data words

The set of first order (abbreviated as FO) formulas over the vocabulary τ is given by the following syntax;

$$\varphi ::= x = y \mid R(x_1, \ldots, x_n) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists x\, \varphi$$

Theorem ([BMS+06])

(finite) satisfiability of FO *is undecidable over data words. Undecidability prevails even for three variable fragment.*

First-order logic over data words

The set of first order (abbreviated as FO) formulas over the vocabulary τ is given by the following syntax;

$$\varphi ::= x = y \mid R(x_1, \ldots, x_n) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists x\, \varphi$$

Theorem ([BMS+06])

(finite) satisfiability of FO *is undecidable over data words. Undecidability prevails even for three variable fragment.*

Theorem ([BMS+06])

(finite) satisfiability of FO^2 *is decidable over data words.*

Two-variable logic – examples

Example

The following $\mathrm{FO}^2(\Sigma, <, +1)$ formula describes that the model (in this case a word) contains three 'a's.

$$\varphi_1 = \exists x \left(P_a(x) \wedge \exists y \left(x < y \wedge P_a(y) \wedge \exists x \left(y < x \wedge P_a(x) \right)\right)\right).$$

Example

The formula below states that each class contains an 'a' if it contains a 'b' and vice versa.

$$\varphi_2 = \forall x \left(\left(P_a(x) \rightarrow \exists y \; \left(P_b(y) \wedge x \sim y \right)\right) \wedge \left(P_b(x) \rightarrow \exists y \; \left(P_a(y) \wedge x \sim y \right)\right.\right.$$

Ordered data words

Let $\leq_\Gamma$ be a linear order on Γ.
Data values d_i and d_j on positions i and j can have any of the following relationships: $d_i = d_j$ or $d_i <_\Gamma d_j$ or $d_i >_\Gamma d_j$.
This relationship can be expressed by a total preorder on positions given by,

$$i \leq_p j \Leftrightarrow d_i <_\Gamma d_j \text{ or } d_i = d_j.$$

Hence an ordered data word can be represented logically as a first order structure $w = ([n], \Sigma, \leq_l, +1_l, \leq_p, +1_p)$; where $\leq_l$ denotes the linear order on positions and $\leq_p$ denotes the total preorder on positions induced by the order on the data values.

Two-variable logic on ordered data words

Theorem ([BMS$^+$06, MZ11])

Two variable logic on ordered data words is undecidable. More precisely FO^2 *is undecidable on the vocabularies* $(\Sigma, <, +1, +1_p)$ *and* $(\Sigma, <, +1, \leq_p)$.

To retrieve decidability one has to drop either $<$ or $+1$.

Theorem ([SZ10])

Finsat of $\mathrm{FO}^2(\Sigma, <_{l_1}, <_{p_2}, +1_{p_2})$ *is decidable in* EXPSPACE.

Theorem ([Man10])

Finsat of $\mathrm{FO}^2(\Sigma, +1_{l_1}, +1_{l_2})$ *is decidable in* 2-NEXPTIME.

Undecidability

Theorem ([Man10])

The finite satisfiability problems for the following logics are undecidable.

(a) $\mathrm{FO}^2(\Sigma, \leq_{l_1}, +1_{l_1}, \leq_{l_2}, +1_{l_2})$

(b) $\mathrm{FO}^3(\Sigma, +1_{l_1}, +1_{l_2})$

(c) $\mathrm{FO}^2(\Sigma, +1_{l_1}, +2_{l_1}, +3_{l_1}, +1_{l_2}, +2_{l_2})$

Two-variable logic on ordered data words

Theorem ([MZ11])

Finite satisfiability of $\mathrm{FO}^2(\Sigma, +1_{l_1}, <_{p_2}, +1_{p_2})$ *is decidable when classes of* $<_{p_2}$ *are of size at most* k*.*

For the proof, the notion of data automata are generalized so that they accept ordered data words. A translation from the above logic to these automata is established and finally the non-emptiness of these automata are shown to be decidable by reduction to reachability problem in vector addition systems. Since it is definable in FO^2 that $<_{p_2}$ is a linear order,

Corollary

Finite satisfiability of $\mathrm{FO}^2(\Sigma, +1_{l_1}, <_{l_2}, +1_{l_2})$ *is decidable (not known to be elementary).*

This corollary completes the classification of FO over two linear orders.

Undecidability in 2-SS

Theorem ([Man10])

The finite satisfiability problems for the following logics are undecidable.

(a) $\text{FO}^2(\Sigma, \leq_{l_1}, +1_{l_1}, \leq_{l_2}, +1_{l_2})$

(b) $\text{FO}^3(\Sigma, +1_{l_1}, +1_{l_2})$

(c) $\text{FO}^2(\Sigma, +1_{l_1}, +2_{l_1}, +3_{l_1}, +1_{l_2}, +2_{l_2})$

Proof.

Reduction from PCP.
$I = \{(u_i, v_i) \mid i \in [n], u_i, v_i \in \Sigma^{\leq 2}\}$ over the alphabet $\Sigma = \{l_1, l_2, \dots l_k\}$.
We encode the PCP solution as structures in the above vocabularies, in the following way. Let $\Sigma' = \{l'_1, l'_2, \dots l'_k\}$ and $\hat{\Sigma} = \Sigma \cup \Sigma'$.

□

Proof contd.

Given a word $w = a_1 a_2 \ldots a_n$ in Σ^*, we denote by w' the word $a'_1 a'_2 \ldots a'_n$ in Σ'^*.
A solution of I is a structure $\mathcal{A} = (A, \hat{\Sigma}, +1_{l_1}, +1_{l_2})$ over $\hat{\Sigma}$ such that,

(1) The word $(A, \hat{\Sigma}, +1_{l_1})$ is in the language $(u_1 v'_1 + u_2 v'_2 \ldots + u_n v'_n)^+$. This language is expressible in FO^2 $(\hat{\Sigma}, +1_{l_1})$, let us call it φ_1.

(2) The word $(A, \hat{\Sigma}, +1_{l_2})$ is in the language $(l_1 l'_1 + l_2 l'_2 \ldots + l_k l'_k)^+$. This language is expressible in FO^2 $(\hat{\Sigma}, +1_{l_2})$ by the formulas (call them φ_2),

Proof contd.

Enforcing the matching,

- $$\varphi_{3a} \equiv \forall xy \ \left((\Sigma(x) \wedge \Sigma(y) \wedge x \leq_{l_1} y \rightarrow x \leq_{l_2} y) \right.$$
  $$\left. \wedge \left(\Sigma'(x) \wedge \Sigma'(y) \wedge x \leq_{l_1} y \rightarrow x \leq_{l_2} y \right) \right)$$
- $$\varphi_{3b} \equiv \forall xyz \Big(\big(\Sigma(x) \wedge \Sigma(y) \wedge \Sigma'(z) \wedge S(x,y) \wedge x + 1_{l_2} z \big) \rightarrow z + 1_{l_2} y \Big)$$
  $$\wedge \forall xyz \Big(\big(\Sigma'(x) \wedge \Sigma'(y) \wedge \Sigma(z) \wedge S(x,y) \wedge x + 1_{l_2} z \big) \rightarrow z + 1_{l_2} y \Big)$$
- $$\varphi_{3c} \equiv \forall xy \ \left((\Sigma(x) \wedge \Sigma(y) \wedge S(x,y)) \rightarrow x + 2_{l_2} y \right)$$
  $$\wedge \forall xy \ \Big(\big(\Sigma'(x) \wedge \Sigma'(y) \wedge S(x,y) \big) \rightarrow x + 2_{l_2} y \Big)$$

Logic	**Complexity** (lower/upper)	**Comments**
One linear order		
$FO^2(+1_l)$	NEXPTIME-complete	[EVW02]
$FO^2(\leq_l)$	NEXPTIME-complete	[EVW02]
$FO^2(+1_l, \leq_l)$	NEXPTIME-complete	[EVW02]
One total preorder		
$FO^2(+1_p)$	NEXPTIME-complete	
$FO^2(\leq_p)$	NEXPTIME-complete	
$FO^2(+1_p, \leq_p)$	EXPSPACE-complete	[SZ11]
Two linear orders		
$FO^2(+1_{l_1}; +1_{l_2})$	NEXPTIME/2-NEXPTIME	[Man10]
$FO^2(+1_{l_1}; \leq_{l_2})$	NEXPTIME/EXPSPACE	[SZ11]
$FO^2(+1_{l_1}, \leq_{l_1}; +1_{l_2})$	VASS-REACHABILITY/Decidable [MZ11]	
$FO^2(+1_{l_1}, \leq_{l_1}; \leq_{l_2})$	NXPTIME/EXPSPACE	[SZ11]
$FO^2(+1_{l_1}, \leq_{l_1}; +1_{l_2}, \leq_{l_2})$	Undecidable	[MZ11]

Figure: Summary of results on finite satisfiability of FO^2 with successor and order relations. Cases that are symmetric and where undecidability is implied are omitted.

Logic	**Complexity** (lower/upper)	**Comments**
Two total preorders		
$FO^2(+1_{p_1}, +1_{p_2})$	Undecidable	[MZ11]
$FO^2(+1_{p_1}; \leq_{p_2})$	Undecidable	[MZ11]
$FO^2(\leq_{p_1}; \leq_{p_2})$	Undecidable	[SZ10]
One linear order and one total preorder		
$FO^2(+1_{l_1}; +1_{p_2})$	?	
$FO^2(+1_{l_1}, \leq_{l_1}; +1_{p_2})$	Undecidable	[MZ11]
$FO^2(+1_{l_1}, \leq_{l_1}; \leq_{p_2})$	Undecidable	[BMS+06]
$FO^2(+1_{l_1}; +1_{p_2}, \leq_{p_2})$	?	
$FO^2(\leq_{l_1}; +1_{p_2}, \leq_{p_2})$	EXPSPACE-complete	[SZ11]
Many orders		
$FO^2(\leq_{l_1}, \leq_{l_2}, \leq_{p_3})$	Undecidable	[SZ10]
$FO^2(\leq_{l_1}, \ldots, \leq_{l_3})$	Undecidable	[Kie11]
$FO^2(+1_{l_1}, \ldots, +1_{l_k})$	?	

Figure: Summary of results on finite satisfiability of FO^2 with successor and order relations. Cases that are symmetric and where undecidability is implied are omitted.

2-successor structures

- Marking alphabet $\Gamma = \{-1, 0, 1\}$.

Definition (Marked String Projections of $\mathfrak{A}$)

$$\mathrm{msp}_{\prec_1}(\mathfrak{A}) = \left(A, (P_a)_{a\in\Sigma}, (M_i)_{i\in\Gamma}, \prec_1\right)$$
$$\mathrm{msp}_{\prec_2}(\mathfrak{A}) = \left(A, (P_a)_{a\in\Sigma}, (M_i)_{i\in\Gamma}, \prec_2\right)$$

- msp's are words over the alphabet $\Sigma \times \Gamma$.

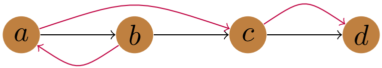
Lemma

Let $x \prec_1 y$. The marking $M_{\prec_2}(y)$ can be computed from $M_{\prec_1}(x)$ and $M_{\prec_1}(y)$.

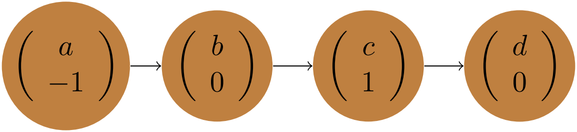
Proof.

Construct a table. □

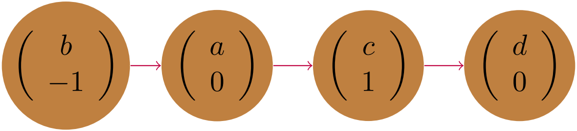
Example



$$\mathrm{msp}_{\prec_1}(\mathfrak{A})$$



$$\mathrm{msp}_{\prec_2}(\mathfrak{A})$$



Automata on 2-SS

Definition (2-SS Automaton)

A 2-SS automaton $\mathcal{T}$ is a tuple $(\mathcal{B}, \Sigma_o, \mathcal{C})$ where,

- $\mathcal{B}$ Finite state transducer with input alphabet $\Sigma \times \Gamma$ and output alphabet Σ_o,
- Σ_o Intermediate alphabet,
- $\mathcal{C}$ Finite state recognizer with input alphabet Σ_o.

Definition (Run of $\mathcal{T}$)

A run $\rho_{\mathcal{T}}$ of the 2-SS automaton $\mathcal{T}$ is of the form $\rho_{\mathcal{T}} = (\rho_{\mathcal{B}}, \rho_{\mathcal{C}})$,

- $\rho_{\mathcal{B}}$ is a run of $\mathcal{B}$ on $\mathrm{msp}_{\prec_1}(\mathfrak{A})$ outputting $\left(A, (P_a)_{a \in \Sigma_o}, \prec_1\right)$ over Σ_o,
- $\rho_{\mathcal{C}}$ is a run of $\mathcal{C}$ on $\left(A, (P_a)_{a \in \Sigma_o}, \prec_2\right)$.

The run is *accepting* if both $\rho_{\mathcal{B}}$ and $\rho_{\mathcal{C}}$ are accepting.
$\mathcal{L}(\mathcal{T}) = \{\mathfrak{A} \mid \mathcal{T} \text{ has an accepting run on } \mathfrak{A}\}$.

Example Languages

Example

$\mathcal{L}_1 = \{\mathfrak{A} = \left(A, (P_a)_{a\in\Sigma}, \prec_1, \prec_2\right) \mid \prec_1 = \prec_2\}$
Check the markings.

Example

$\mathcal{L}_2 = \{\mathfrak{A} \mid \mathrm{sp}_{\prec_1}(\mathfrak{A}) \in a^* \cdot b^* \cdot c^*, \mathrm{sp}_{\prec_2}(\mathfrak{A}) \in (a \cdot b \cdot c)^*\}$
The transducer $\mathcal{B}$ projects the marked string to Σ and checks if it belongs to $a^* \cdot b^* \cdot c^*$. The automaton $\mathcal{C}$ checks if its input is in $(a \cdot b \cdot c)^*$.

Example

$\mathcal{L}_3 = \{\mathfrak{A} \mid \forall xy\,(a(x) \wedge b(y) \to x \prec_1 y \vee x \prec_2 y)\}$
Use transduction!.

Lemma

1. Given a regular language $\mathcal{L} \subseteq \Sigma^*$, there is a 2-SS *automaton accepting all* 2-SS *whose projections to $\prec_1$ is in $\mathcal{L}$.*

2. Similarly, there is a 2-SS *automaton accepting all* 2-SS *whose projections to $\prec_2$ is in $\mathcal{L}$.*

Proof.

1. The transducer $\mathcal{B}$ checks if the projection to $\prec_1$ (ignoring the markings) is in $\mathcal{L}$ and $\mathcal{C}$ accepts Σ_o^*.

2. The transducer $\mathcal{B}$ simply copies the string (ignoring the markings) and $\mathcal{C}$ accepts if its input is in $\mathcal{L}$. □

Lemma

Languages recognized by 2-SS *automata are closed under* union, intersection *and* renaming.

Proof.

Closure under union and intersection is obtained from usual product construction (using a composed output alphabet).

Closure under renaming is achieved using the non-determinism of the transducer. □

$\mathcal{L}_m$ is the set of all 2-SS $\mathfrak{A} = (A, \lambda, \prec_1, \prec_2)$ such that,

- $\mathrm{sp}_{\prec_1}(\mathfrak{A}) \in \diamondsuit \cdot a^+ \cdot \clubsuit \cdot \heartsuit \cdot b^+ \cdot \spadesuit$,
- $\mathrm{sp}_{\prec_2}(\mathfrak{A}) \in \diamondsuit \cdot \heartsuit \cdot (a \cdot b)^+ \cdot \clubsuit \cdot \spadesuit$,
- $\exists x, y \in A, \lambda(x) = \lambda(y)$ such that $x \prec_1^+ y$ and $y \prec_2^+ x$.

$\mathcal{L}_m$ is accepted by a 2-SS automaton. But $\overline{\mathcal{L}_m}$ is not accepted by any 2-SS automaton.

Proof.

Pumping and Crosswiring. □

Lemma

The class of languages accepted by 2-SS *automata are not closed under* complementation.

Theorem

Given a 2-SS automaton $\mathcal{T}$, there is a formula $\varphi_{\mathcal{T}} \in \mathrm{EMSO}^2(\Sigma, \prec_1, \prec_2)$ such that $\mathcal{L}(\mathcal{T}) = \mathcal{L}(\varphi_{\mathcal{T}})$.

Proof.

Let $\Sigma_o = \{l_1, \ldots, l_n\}$. The formula $\varphi_{\mathcal{T}}$ states that there is a run of $\mathcal{T}$ on $\mathfrak{A}$ in the following way,

$$\varphi_{\mathcal{T}} = \exists P_{l_1} P_{l_2} \ldots P_{l_n} \left(\varphi_{\text{part}}\left(P_{l_1}, \ldots, P_{l_n}\right) \wedge \varphi_{\mathcal{B}} \wedge \varphi_{\mathcal{C}}\right)$$

- $\varphi_{\text{part}}\left(P_{l_1}, \ldots, P_{l_n}\right)$ says that the predicates $P_{l_1}, \ldots, P_{l_n}$ form a partition of the set of all positions.
- $\varphi_{\mathcal{B}}$ is the encoding of $\mathcal{B}$ in $\mathrm{EMSO}^2(\Sigma, P_{l_1}, \ldots, P_{l_n}, \prec_1)$.
- $\varphi_{\mathcal{C}}$ is the encoding of $\mathcal{C}$ in $\mathrm{EMSO}^2(P_{l_1}, \ldots, P_{l_n}, \prec_2)$.

$P_{l_1}, \ldots, P_{l_n}$ are free in $\varphi_{\mathcal{B}}$ and $\varphi_{\mathcal{C}}$.

□

Logic to Automata

Translation to Scott Form

$$\varphi \Leftrightarrow \exists R_1 \dots R_n \left(\forall x \forall y \ \chi \wedge \bigwedge_i \forall x \exists y \ \psi_i \right)$$

The predicates R_i are unary, and χ and ψ_i are quantifier-free formulas in $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2)$.

2-SS are closed under renaming and intersection.

Hence it suffices to construct a 2-SS automaton for each of the formulas $\forall x \forall y \ \chi$ and $\forall x \exists y \ \psi_i$.

Lemma

Given an $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2)$ *formula of the form* $\varphi = \forall x \forall y\ \chi$ *where* χ *is quantifier free, an equivalent* 2-SS *automaton of doubly exponential size can be constructed.*

Proof.

φ can be reduced to a conjunction of exponentially many formulas in one the following forms,

1. True, False, A formula over one successor relation,
2. $\forall xy\,(\alpha(x) \wedge \beta(y) \wedge x \neq y \wedge x \prec_1 y \rightarrow \delta_2(x,y))$,
3. $\forall xy\,(\alpha(x) \wedge \beta(y) \wedge x \neq y \wedge x \prec_2 y \rightarrow \delta_1(x,y))$,
4. $\forall xy\,\big(\alpha(x) \wedge \beta(y) \wedge x \neq y \rightarrow \delta_1^+(x,y) \vee \delta_2^+(x,y)\big)$,

where

α, β : types, δ_i : disjunction over O_i, δ_i^+ : disjunction over O_i^+.
$O_i^+ = \{x \prec_i y, y \prec_i x\}$, $O_i = \{x \prec_i y, x \not\prec_i y, y \prec_i x, y \not\prec_i x\}$.
Each of these formulas can be translated to a 2-SS automaton. □

Lemma

For each $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2)$ *formula of the form* $\varphi = \forall x \exists y\ \psi$ *where* ψ *is quantifier free, an equivalent* 2-SS *automaton of doubly exponential size can be constructed.*

Proof.

φ can be reduced to a conjunction of exponentially many formulas in one the following forms,

1. A formula over one successor relation,
2. $\forall x \exists y \left(\alpha(x) \rightarrow \beta(y) \wedge x \neq y \wedge \delta_1^+(x,y) \wedge \delta_2(x,y)\right)$,
3. $\forall x \exists y \left(\alpha(x) \rightarrow \beta(y) \wedge x \neq y \wedge \delta_2^+(x,y) \wedge \delta_1(x,y)\right)$,
4. $\forall x \exists y \left(\alpha(x) \rightarrow \beta(y) \wedge x \neq y \wedge \delta_1^-(x,y) \wedge \delta_2^-(x,y)\right)$.

where

α, β : types, $\delta_i \in O_i$, $\delta_i^+ \in O_i^+$, δ_i^- : conjunction over O_i^-.

$O_i^- = \{x \not\prec_i y, y \not\prec_i x\}$,

Each of these formulas can be translated to a 2-SS automaton.

□

Lemma

Given an $\mathrm{EMSO}^2(\Sigma, \prec_1, \prec_2)$ *formula* φ*, there exists a* 2-SS *automaton* $\mathcal{T}_\varphi$ *such that* $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{T}_\varphi)$.

Theorem

$\mathcal{L}$ *is definable in* $\mathrm{EMSO}^2(\Sigma, \prec_1, \prec_2)$ *if and only if* $\mathcal{L}$ *is recognized by a* 2-SS *automaton.*

Decidability of 2-SS Automata

Proof Idea

Given a 2-SS automaton $\mathcal{T} = (\mathcal{B}, \mathcal{C})$, $\mathcal{L}(\mathcal{T})$ is non-empty if there is a marked word w such that,

- w is accepted by $\mathcal{B}$
- a permutation of output of $\mathcal{B}$ on w, 'consistent' with the marking of w, is accepted by $\mathcal{C}$.

- Let $w = ([n], \lambda, \prec)$ be a marked word of length n. We denote the projection of w to Σ by $w \downarrow \Sigma$.
- Given a permutation $\pi : [n] \to [n]$, $\pi(w)$ is defined as the word
$$\left([n], \pi^{-1} \circ \lambda, \prec\right).$$
- π defines a successor relation $\prec_\pi = \pi^{-1}(1) \ldots \pi^{-1}(n)$ on the positions.

We say that the permutation π is *consistent* with the marking if w is the marked string projection of the 2-SS $\mathfrak{A} = ([n], \lambda, \prec, \prec_\pi)$ to the order $\prec$.

Definition (mperm(w))

Given $w \in \Sigma^*$, by mperm(w), we denote the set of all the marked words w' such that there is a permutation π consistent with the marking such that $\pi(w \downarrow \Sigma) = w$.
Given $\mathcal{L} \subseteq \Sigma^*$, we define $\text{mperm}(\mathcal{L}) = \cup_{w' \in \mathcal{L}} \text{mperm}(w')$.

Definition (Presburger word automaton, Habermahl–Muscholl–Schwentick–Seidl, 04)

A Presburger word automaton $\mathcal{P}$ is a tuple $(\mathcal{A}, \mathcal{S})$,

- $\mathcal{A}$ is a finite state automaton with states $Q = \{q_1, \ldots q_n\}$,
- $\mathcal{S}$ is a semi-linear set in $\mathbb{N}^n$.

A word w is accepted by the automaton $\mathcal{P}$ if,

- there is an accepting run ρ of $\mathcal{A}$ on w,
- $(|\rho|_{q_0}, \ldots, |\rho|_{q_n}) \in \mathcal{S}$.

Example

The following languages are recognizable by a Presburger word automaton.

$\{a^n b^n c^n \mid n \in \mathbb{N}\}$, $\{w \in \Sigma^* \mid 5 \cdot |w|_a = 2 \cdot |w|_b - 3 \cdot |w|_c\}$,

Permutation($\mathcal{L}$) if $\mathcal{L}$ is context-free.

Lemma

If $\mathcal{L}$ is regular then mperm($\mathcal{L}$) *is accepted by a Presburger automaton.*

Proof.

- Given an automaton $\mathcal{C}$ for $\mathcal{L}$, the Presburger automaton $\mathcal{P}$ checks non-deterministically if there is a run of $\mathcal{C}$ on some consistent permutation of w.
- To achieve this, the automaton $\mathcal{P}$ assigns a transition $\delta = (p, a_i, q) \in \Delta$ to each position i of the marked word.
- We can define a flow f where each transition δ of $\mathcal{C}$ is labelled by the number of times it is associated with a position.
- Finally, we can write linear constraints which checks that,
    1. f is locally consistent,
    2. the subgraph induced by the states with a non-zero flow is connected,
    3. f is consistent with the marking.
- The resulting automata $\mathcal{P}$ is poly-sized in terms of the size

Theorem

Emptiness checking of a 2-SS *automaton* $\mathcal{T} = (\mathcal{B}, \mathcal{C})$ *is in* NP.

Proof.

- Construct a Presburger automaton $\mathcal{P}$ with linear constraints which accepts $\mathrm{mperm}(\mathcal{L}(\mathcal{C}))$.
- Take the intersection of the transducer $\mathcal{B}$ and $\mathcal{P}$ in such a way that the output of $\mathcal{B}$ is supplied as the input of $\mathcal{P}$.
- Finally we check the emptiness of the resulting automaton which is in NP.

□

Theorem

Emptiness checking of a Presburger automaton is polynomial time reducible to the emptiness checking of a 2-SS *automaton.*

Theorem

Finite Satisfiability problem of $\mathrm{FO}^2(\Sigma, \prec_1, \prec_2)$ *is in* *2*-NEXPTIME.

Proof.

Given φ, construct $\mathcal{T}_\varphi$, check if $\mathcal{L}(\mathcal{T}_\varphi)$ is non-empty. □

Over words, FINSAT of both $\mathrm{FO}^2(\Sigma)$ and $\mathrm{FO}^2(\prec)$ with one unary predicate are NEXPTIME-hard [Etessami, 02].

Thank You!

Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David.
Two-variable logic on words with data.
In *LICS*, pages 7–16. IEEE Computer Society, 2006.

Henrik Björklund and Thomas Schwentick.
On notions of regularity for data languages.
Theoretical Computer Science, 411(4-5):702–715, 2010.

Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke.
First-order logic with two variables and unary temporal logic.
Information and Computation, 179(2):279 – 295, 2002.

Michael Kaminski and Nissim Francez.
Finite-memory automata.
Theoretical Computer Science, 134(2):329–363, 1994.

Emanuel Kieronski.
Decidability issues for two-variable logics with several linear orders.

In *CSL*, volume 12 of *LIPIcs*, pages 337–351, 2011.

Amaldev Manuel.
Two orders and two variables.
In *MFCS*, volume 6281 of *LNCS*, pages 513–524, 2010.

Amaldev Manuel and Thomas Zeume.
Two variable logic with a linear successor and a preorder.
Under preparation, 2011.

Thomas Schwentick and Thomas Zeume.
Two-variable logic with two order relations.
In *CSL*, volume 6247 of *LNCS*, pages 499–513, 2010.

Thomas Schwentick and Thomas Zeume.
Two-variable logic with two order relations.
To appear, 2011.