Logics for Word Transductions with Synthesis

DELTA meeting, IRIF

Luc Dartois

Université Libre de Bruxelles

March 26, 2018

Joint work with Emmanuel Filiot (ULB) and Nathan Lhote (ULB/LABRI)

Verification setting

System	**Specification**
Machine	Logic formulas
$\mathcal{M}$	φ

Model-Checking: $\forall w$ accepted by M, does $w \models \varphi$?

Verification setting

System	**Specification**
Machine	Logic formulas
$\mathcal{M}$	φ

Model-Checking: $\forall w$ accepted by M, does $w \models \varphi$?
Synthesis: Construct M such that M models φ ?

Verification setting

System		**Specification**
Automata	$\Leftrightarrow$	$MSO[<]$
$\mathcal{A}$	(Büchi60)	φ

Model-Checking: $\forall w$ accepted by M, does $w \models \varphi$?

Synthesis: Construct M such that M models φ ?

Verification of Reactive systems

Executions of a **reactive system** can be seen as sequences of Actions (the input) and Reactions (the output). In the case of **reactive** systems, we get intertwined sequences:

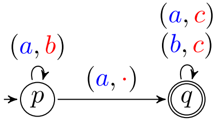
$$a_1 r_1 a_2 r_2 \dots a_n r_n$$

$\rightarrow$ Can be seen as a word over *Actions* $\times$ *Reactions* and use automata methods.

Verification of Reactive systems

Reactive Systems

Automata $\mathcal{A}$ (Mealy Machine)



$\models$

Specification

Logic formulas φ (MSO, LTL)

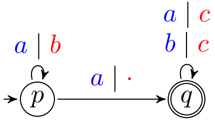
$\exists x\ (a, \cdot)(x) \wedge \forall y\ (\cdot, c)(y) \to x < y$

$(\neg(\cdot, c))U(a, \cdot)$

Verification of Reactive systems

Reactive Systems

Automata $\mathcal{A}$ (Mealy Machine)



$\models$

Specification

Logic formulas φ (MSO, LTL)

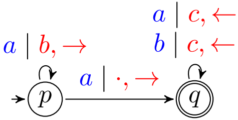
$\exists x\ (a, \cdot)(x) \wedge \forall y\ (\cdot, c)(y) \rightarrow x < y$

$(\neg(\cdot, c))U(a, \cdot)$

Verification of Reactive systems

Reactive Systems

Automata $\mathcal{A}$ (Mealy Machine)



$\models$

Specification

Logic formulas φ

??

→ What about non reactive systems, i.e. systems without sequential link between input and output (relations $R \subseteq \Sigma^* \times \Gamma^*$)?

Aim of the talk

Define a logic for transformation of finite words (i.e. $R \subseteq \Sigma^* \times \Gamma^*$) that

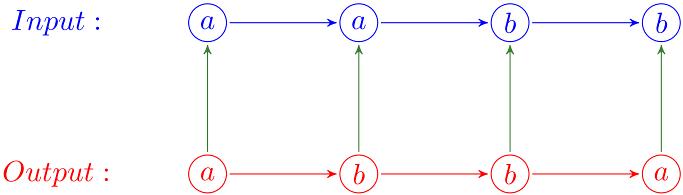
- Can express specific or generic relations
  - Express functions: reversing the input: $\{(u_1 \dots u_n, u_n \dots u_1) \mid u \in A^*\}$,
  - Nondeterminism: Every input letter appears in the output exactly once (*shuffle*): $\{(u_1 \dots u_n, u_{\pi(1)} \dots u_{\pi(n)} \mid \pi \text{ a permutation}\}$

Aim of the talk

Define a logic for transformation of finite words (i.e. $R \subseteq \Sigma^* \times \Gamma^*$) that

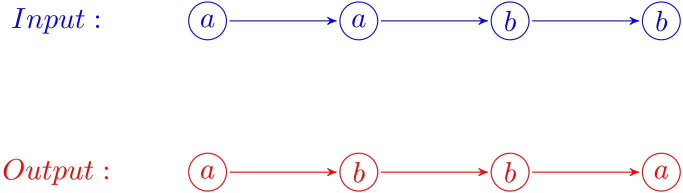
- Can express specific or generic relations
  - Express functions: reversing the input: $\{(u_1 \dots u_n, u_n \dots u_1) \mid u \in A^*\}$,
  - Nondeterminism: Every input letter appears in the output exactly once (*shuffle*): $\{(u_1 \dots u_n, u_{\pi(1)} \dots u_{\pi(n)} \mid \pi \text{ a permutation}\}$
- Has good decidable verification properties:
  - Model-checking:
    Does a deterministic 2-way transducer T satisfies a formula φ ?
  - Synthesis:
    Construct a deterministic 2-way transducer T that satisfies φ.

Non reactive systems



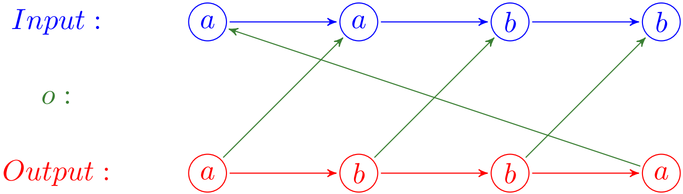
Input/*Output* relation in the case of reactive systems.

Non reactive systems



→ How to relate input with output ?

Non reactive systems: Origin semantics [Bojanczyk14]

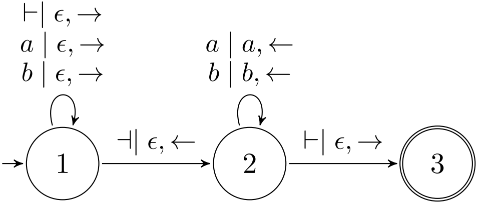


Origin graphs: A total function o from Output positions to Input positions, as well as $<_i$ and $<_o$ order relations.

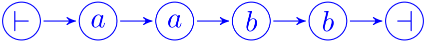
The origin of an output position is the input position from which it originates.

Origin semantics of a transducer

A transducer realising the reverse function $w \to rev(w)$.



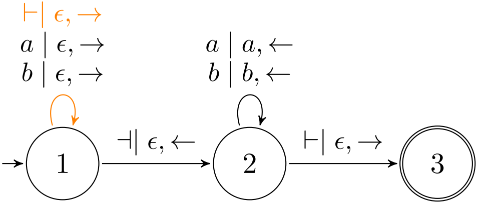
Input :



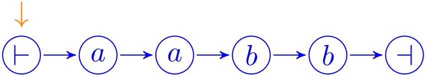
Output :

Origin semantics of a transducer

A transducer realising the reverse function $w \to rev(w)$.



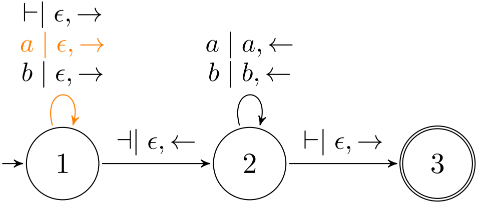
Input :



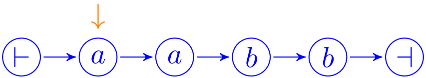
Output :

Origin semantics of a transducer

A transducer realising the reverse function $w \to rev(w)$.



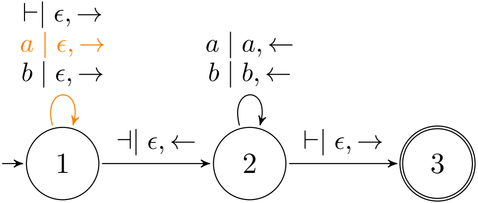
Input :



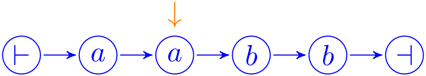
Output :

Origin semantics of a transducer

A transducer realising the reverse function $w \to rev(w)$.



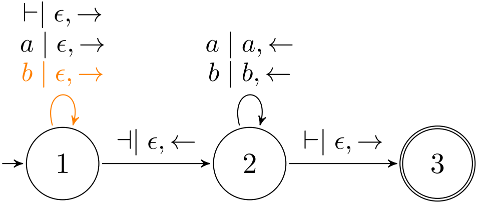
Input :



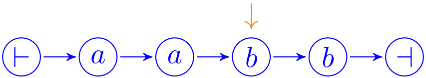
Output :

Origin semantics of a transducer

A transducer realising the reverse function $w \rightarrow rev(w)$.



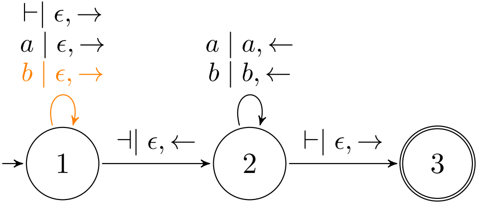
Input :



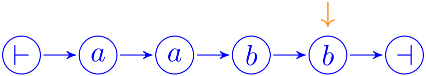
Output :

Origin semantics of a transducer

A transducer realising the reverse function $w \rightarrow rev(w)$.



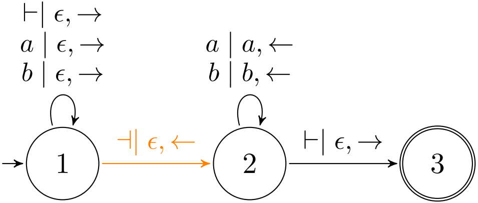
Input :



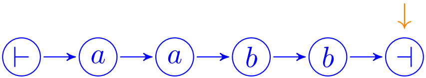
Output :

Origin semantics of a transducer

A transducer realising the reverse function $w \rightarrow rev(w)$.



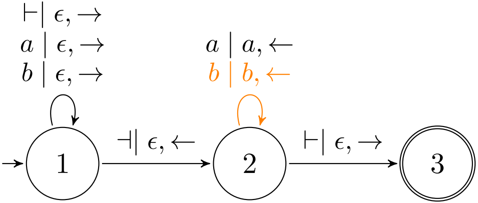
Input :

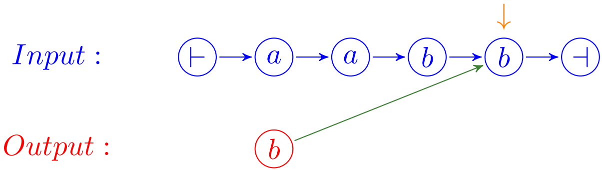


Output :

Origin semantics of a transducer

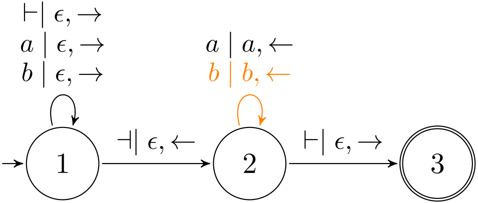
A transducer realising the reverse function $w \to rev(w)$.

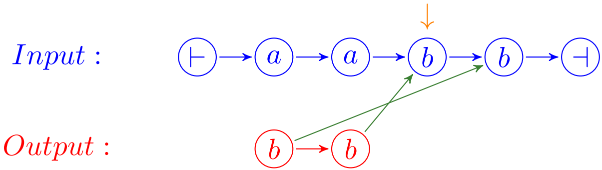




Origin semantics of a transducer

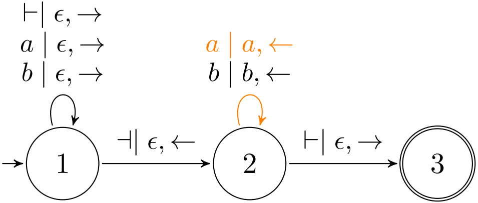
A transducer realising the reverse function $w \to rev(w)$.

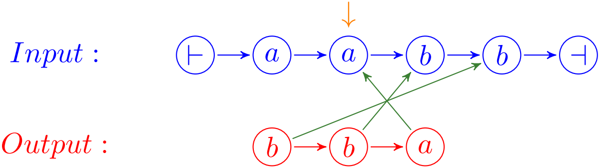




Origin semantics of a transducer

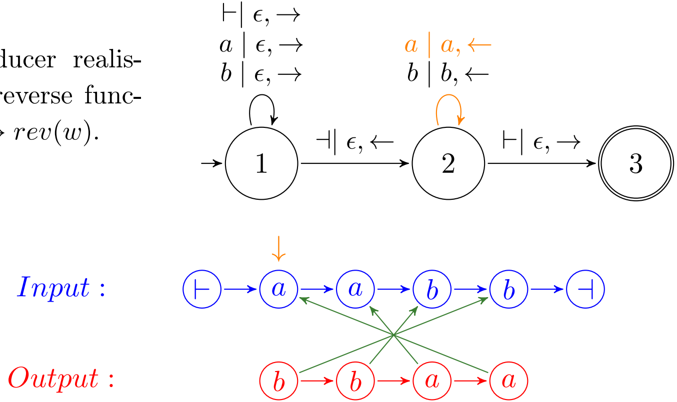
A transducer realising the reverse function $w \to rev(w)$.





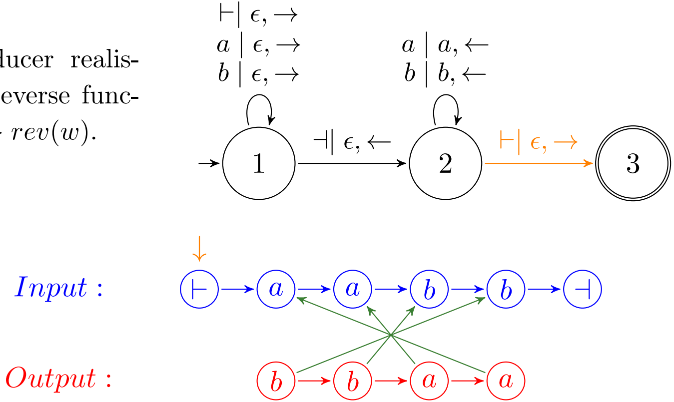
Origin semantics of a transducer

A transducer realising the reverse function $w \to rev(w)$.

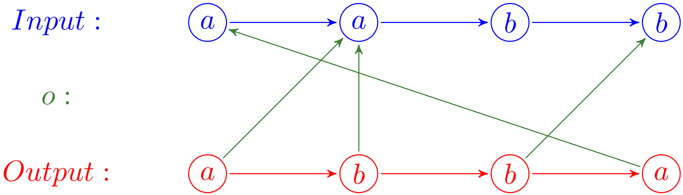


Origin semantics of a transducer

A transducer realising the reverse function $w \to rev(w)$.



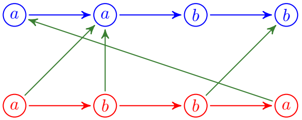
Origin semantics of a transducer



Origin graphs as models for traces of executions.

The logic $MSO[<_i, <_o, \mathsf{o}]$

$MSO[<_i, <_o, \mathsf{o}]$



Examples:

- Order-preserving: $\phi_{op} = \forall xy\ x <_o y \leftrightarrow \mathsf{o}(x) \leq_i \mathsf{o}(y)$
- Injection: $\phi_{inj} = \forall x \neq_o y\ \mathsf{o}(x) \neq_i \mathsf{o}(y)$
- Surjection: $\phi_{surj} = \forall x\ x \leq_i x \to (\exists y\ \mathsf{o}(y) =_i x)$
- Reactive systems executions are modelled by the bijective and order-preserving origin graphs.
- Shuffle: $\phi_{shu} = \phi_{inj} \wedge \phi_{surj} \wedge (\forall xy\ x =_i \mathsf{o}(y) \to \bigwedge_{a \in A} a(x) \leftrightarrow a(y))$

Decidability of origin logics

Problem:

Satisfiability of $MSO[<_i, <_o, \mathsf{o}]$ is undecidable.

Decidability of origin logics

Problem:

Satisfiability of $MSO[<_i, <_o, \mathsf{o}]$ is undecidable.

$\rightarrow$ In fact, even $FO^2[<_i, <_o, S_o, \mathsf{o}]$ is undecidable.

Decidability of origin logics

Problem:

Satisfiability of $MSO[<_i, <_o, \mathsf{o}]$ is undecidable.

$\rightarrow$ In fact, even $FO^2[<_i, <_o, S_o, \mathsf{o}]$ is undecidable.

Logic FO^2

First-Order logic with two reusable variables:

$$\exists x\ a(x) \wedge \exists y \big(b(y) \wedge x < y \wedge \exists x (c(x) \wedge y < x)\big)$$

describes the language $\Sigma^* a \Sigma^* b \Sigma^* c \Sigma^*$.

Undecidability of Satisfiability

Proving undecidability of $FO^2[<_i, <_o, S_o, \mathsf{o}]$:
Encoding PCP instances: given $(u_i, v_i)_{i \leq n}$, does there exists $i_1 \ldots i_k$ such that $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$.

$$
\begin{aligned}
u_1 &= ab \\
v_1 &= \epsilon \\
u_2 &= b \\
v_2 &= abb
\end{aligned}
$$

Undecidability of Satisfiability

Proving undecidability of $FO^2[<_i, <_o, S_o, \mathsf{o}]$:
Encoding PCP instances: given $(u_i, v_i)_{i \leq n}$, does there exists $i_1 \dots i_k$ such that $u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}$.

$$u_1 = ab$$
$$v_1 = \epsilon$$
$$u_2 = b$$
$$v_2 = abb$$

1 2

Undecidability of Satisfiability

Proving undecidability of $FO^2[<_i, <_o, S_o, \mathsf{o}]$:
Encoding PCP instances: given $(u_i, v_i)_{i \leq n}$, does there exists $i_1 \dots i_k$ such that $u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}$.

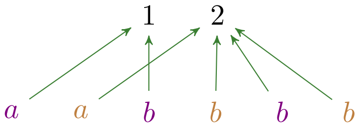
$$u_1 = ab$$
$$v_1 = \epsilon$$
$$u_2 = b$$
$$v_2 = abb$$

		1	2		
a	a	b	b	b	b

Undecidability of Satisfiability

Proving undecidability of $FO^2[<_i, <_o, S_o, \mathsf{o}]$:
Encoding PCP instances: given $(u_i, v_i)_{i \leq n}$, does there exists $i_1 \ldots i_k$ such that $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$.

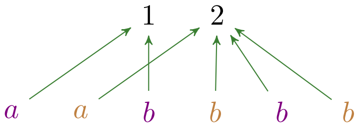
$$u_1 = ab$$
$$v_1 = \epsilon$$
$$u_2 = b$$
$$v_2 = abb$$



Undecidability of Satisfiability

Proving undecidability of $FO^2[<_i, <_o, S_o, \mathsf{o}]$:
Encoding PCP instances: given $(u_i, v_i)_{i \leq n}$, does there exists $i_1 \dots i_k$ such that $u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}$.

$$\begin{aligned} u_1 &= ab \\ v_1 &= \epsilon \\ u_2 &= b \\ v_2 &= abb \end{aligned}$$

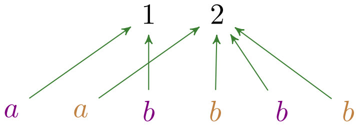


PCP instance $(u_i, v_i)_{i \leq n}$ encoded as
$\forall^{in} x \bigwedge_i i(x) \rightarrow u_i(x) \wedge v_i(x)$

Undecidability of Satisfiability

Proving undecidability of $FO^2[<_i, <_o, S_o, \mathsf{o}]$:
Encoding PCP instances: given $(u_i, v_i)_{i \leq n}$, does there exists $i_1 \dots i_k$ such that $u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}$.

$$u_1 = ab$$
$$v_1 = \epsilon$$
$$u_2 = b$$
$$v_2 = abb$$



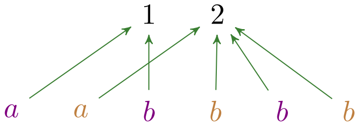
PCP instance $(u_i, v_i)_{i \leq n}$ encoded as
$\forall^{in} x \bigwedge_i i(x) \rightarrow u_i(x) \wedge v_i(x)$
$u_1(x)$: the production of x belongs to $\Sigma^* a \Sigma^* b \Sigma^*$

$$\exists y\ o(y) = x \wedge a(y) \wedge \exists x \big(o(x) = o(y) \wedge y <_o x \wedge b(x)\big)$$

Undecidability of Satisfiability

Proving undecidability of $FO^2[<_i, <_o, S_o, \mathsf{o}]$:
Encoding PCP instances: given $(u_i, v_i)_{i \leq n}$, does there exists $i_1 \ldots i_k$ such that $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$.

$$u_1 = ab$$
$$v_1 = \epsilon$$
$$u_2 = b$$
$$v_2 = abb$$



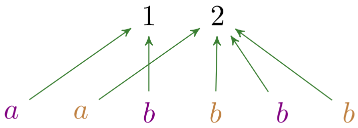
PCP instance $(u_i, v_i)_{i \leq n}$ encoded as
$\forall^{in} x \bigwedge_i i(x) \to u_i(x) \wedge v_i(x)$
$u_1(x)$: the production of x belongs to $\Sigma^* a \Sigma^* b \Sigma^* \cap \Sigma^{\leq 2}$

$$\neg(\exists y\, o(y) = x \wedge \exists x \left(o(x) = o(y) \wedge y <_o x \wedge \exists y(o(y) = o(x) \wedge y < x)\right)$$

Undecidability of Satisfiability

Proving undecidability of $FO^2[<_i, <_o, S_o, \mathsf{o}]$:
Encoding PCP instances: given $(u_i, v_i)_{i \leq n}$, does there exists $i_1 \dots i_k$ such that $u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}$.

$$u_1 = ab$$
$$v_1 = \epsilon$$
$$u_2 = b$$
$$v_2 = abb$$

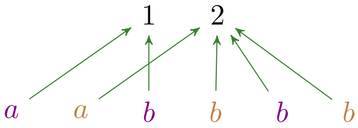


PCP instance $(u_i, v_i)_{i \leq n}$ encoded as
$\forall^{in} x \bigwedge_i i(x) \rightarrow u_i(x) \wedge v_i(x)$
$\wedge \phi_{op} \wedge \phi_{op}$

Undecidability of Satisfiability

Proving undecidability of $FO^2[<_i, <_o, S_o, \mathsf{o}]$:
Encoding PCP instances: given $(u_i, v_i)_{i \leq n}$, does there exists $i_1 \ldots i_k$ such that $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$.

$$u_1 = ab$$
$$v_1 = \epsilon$$
$$u_2 = b$$
$$v_2 = abb$$



PCP instance $(u_i, v_i)_{i \leq n}$ encoded as
$\forall^{in} x \bigwedge_i i(x) \rightarrow u_i(x) \wedge v_i(x)$
$\wedge \phi_{op} \wedge \phi_{op}$
$\wedge \forall^{out} x \ \bigwedge_{a \in \Sigma} a(x) \rightarrow \exists y (S_o(x, y) \wedge a(y))$
(the output word belongs to $\left(\bigcup_{a \in \Sigma} aa\right)^*$)

The logic L_T

Let $MSO_{bin}[<_i]$ be the set of formulas $\varphi(x, y)$ with $\varphi \in MSO[<_i]$.

Example:

for $\varphi(x, y) = x < y$,

$$\phi_{op} = \forall xy \; x <_o y \leftrightarrow \varphi(\mathsf{o}(x), \mathsf{o}(y))$$

The logic L_T

Let $MSO_{bin}[<_i]$ be the set of formulas $\varphi(x, y)$ with $\varphi \in MSO[<_i]$.

Example:

for $\varphi(x, y) = x < y$,

$$\phi_{op} = \forall xy \; x <_o y \leftrightarrow \varphi(\mathsf{o}(x), \mathsf{o}(y))$$

We set $L_T = FO^2[<_o, \mathsf{o}, MSO_{bin}[<_i]]$.

The logic L_T

Let $MSO_{bin}[<_i]$ be the set of formulas $\varphi(x, y)$ with $\varphi \in MSO[<_i]$.

Example:

for $\varphi(x, y) = x < y$,

$$\phi_{op} = \forall xy\ x <_o y \leftrightarrow \varphi(\mathsf{o}(x), \mathsf{o}(y))$$

We set $L_T = FO^2[<_o, \mathsf{o}, MSO_{bin}[<_i]]$.

Expressivity

The logic L_T can express:

- Highly non deterministic relations such as the *shuffle* relation,
- Any function definable by a 2-way transducer.

Decidability of L_T

$$L_T = FO^2[<_o, \mathsf{o}, MSO_{bin}[<_i]]$$

Theorem:

The satisfiability problem is decidable for the logic L_T.

Decidability of L_T

$$L_T = FO^2[<_o, \mathsf{o}, MSO_{bin}[<_i]]$$

Theorem:

The satisfiability problem is decidable for the logic L_T.

Corollary

Model-checking of a 2-way transducer T against a formula φ of L_T is decidable.

Proof: Transform T into a formula ϕ_T and check satisfiability of $\phi_T \wedge \neg\varphi \in L_T$

Decidability of L_T

$L_T = FO^2[<_o, \mathsf{o}, MSO_{bin}[<_i]]$

Theorem:

The satisfiability problem is decidable for the logic L_T.

Corollary

Model-checking of a 2-way transducer T against a formula φ of L_T is decidable.

Proof: Transform T into a formula ϕ_T and check satisfiability of $\phi_T \wedge \neg\varphi \in L_T$

[Bojanczyk et al.17]: Model-checking against $MSO[<_i, <_o, o]$ is decidable

A note on complexity

Non elementary Complexity

Due to the $MSO_{bin}[<_i]$ formulas, satisfiability is non elementary.

A note on complexity

Non elementary Complexity

Due to the $MSO_{bin}[<_i]$ formulas, satisfiability is non elementary.

ExpSpace-c complexity

If the $MSO_{bin}[<]$ formulas are given as query automata, then satisfiability becomes *ExpSpace*-complete.

$\rightarrow$ Query automata A: accepts words w with letters marked by x and y that satisfy $\varphi(x, y)$.

Synthesis of L_T

Regular Synthesis problem:

Given a formula φ of L_T, construct a deterministic 2-way transducer T such that:

- $dom(\varphi) = dom(T)$,
- for all $u \in dom(T)$, $(u, T(u)) \models \varphi$.

Theorem

From any formula φ of L_T, we can construct a deterministic 2-way transducer that realises it.

Synthesis of L_T

Regular Synthesis problem:

Given a formula φ of L_T, construct a deterministic 2-way transducer T such that:

- $dom(\varphi) = dom(T)$,
- for all $u \in dom(T)$, $(u, T(u)) \models \varphi$.

Theorem

From any formula φ of L_T, we can construct a deterministic 2-way transducer that realises it.

$\rightarrow$ Functional L_T characterises the class of regular functions.

Data words

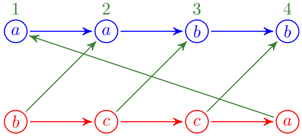
Definition

Here, *Data words* are words over an infinite alphabet (ex: $\mathbb{N}$ or $\Gamma \times \mathbb{N}$.)
ex: $(b, 2)(c, 3)(c, 4)(a, 1)$.

$\rightarrow$ used to model words over unbounded sets such as processus identifiers.
$\rightarrow$ Structure on the infinite alphabet can be added (ex: order $<$ on $\mathbb{N}$).

Casting Origin graphs to words

An origin graph can be equivalently seen as a word over alphabet $\Gamma \times \{1, \ldots, n\} \times \Sigma$ where $n = |Input|$.

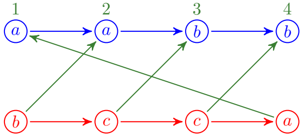


is represented by the word

$$(b, 2, a)(c, 3, b)(c, 4, b)(a, 1, a)$$

Casting Origin graphs to words

An origin graph can be equivalently seen as a word over alphabet $\Gamma \times \{1, \ldots, n\} \times \Sigma$ where $n = |Input|$.



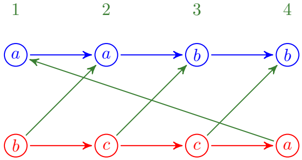
is represented by the word

$$(b, 2, a)(c, 3, b)(c, 4, b)(a, 1, a)$$

But $(a, 1, a)(a, 1, b)$ represents no origin graph !

Typed data words

A *typed data word* is a pair (w, τ) where $w \in (\Gamma \times \mathbb{N})^*$ and $\tau : \mathbb{N} \to \Sigma$.



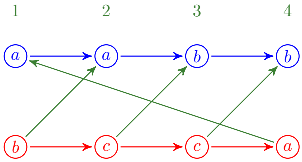
$w : \quad (b, 2)(c, 3)(c, 4)(a, 1)$

$\tau : \quad 1 \to \text{a} \qquad 2 \to \text{a}$

$\qquad 3 \to \text{b} \qquad 4 \to \text{b}$

Typed data words

A *typed data word* is a pair (w, τ) where $w \in (\Gamma \times \mathbb{N})^*$ and $\tau : \mathbb{N} \to \Sigma$.



$w: \quad (b,2)(c,3)(c,4)(a,1)$

$\tau: \quad 1 \to a \qquad 2 \to a$

$\qquad 3 \to b \qquad 4 \to b$

$MSO[<_i, <_o, \mathsf{o}]$ is equivalent to $MSO[\preccurlyeq, <, d]$.
The order-preserving formula is equivalent to having data appear in increasing order:

$$\forall xy\ x <_o y \leftrightarrow \mathsf{o}(x) \leq_i \mathsf{o}(y) \quad \Leftrightarrow \quad \forall xy\ x < y \leftrightarrow d(x) \preccurlyeq d(y)$$

New results on data words

- The logic L_T can be seen as a logic $L_D = FO^2[<, MSO[\preccurlyeq]]$ on data words
  $\rightarrow$ Quantifications in $MSO[\preccurlyeq]$ on data values.
- Results on L_T apply to L_D.
- Satisfiability of L_D extends known results of Satisfiability of $FO^2[<, <, S]$ [Schwentick,Zeume13].

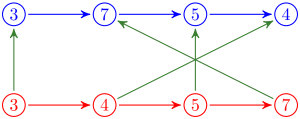
Wrapping up

In the end, the logic L_T has good properties:

- Expressive,
- Decidable model-checking and synthesis,
- Results robusts under extensions
    - $\exists X_1 \ldots X_n L_T$,
    - To capture all rational relations.

Future works

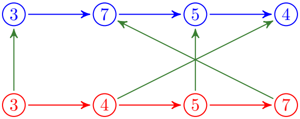
- Extensions to higher models (ongoing work: graph to word relations),
- Synthesis to other target implementations
  - Church synthesis,
  - Highly expressive automata model with non decidable emptiness but good evaluation complexity,
- Application to other problems. Ex: Algorithm synthesis.



Expressing an array sorting algorithm using FO^2 on data and origin.

Future works

- Extensions to higher models (ongoing work: graph to word relations),
- Synthesis to other target implementations
  - Church synthesis,
  - Highly expressive automata model with non decidable emptiness but good evaluation complexity,
- Application to other problems. Ex: Algorithm synthesis.



Expressing an array sorting algorithm using FO^2 on data and origin.

Thanks !