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 Machine $\mathcal{M}$

Specification Logic formulas $\varphi$

Model-Checking: $\forall w$ accepted by $\mathcal{M}$, does $w \models \varphi$?
## Verification setting

<table>
<thead>
<tr>
<th>System Machine ( M )</th>
<th>Specification Logic formulas ( \varphi )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model-Checking:</strong> ( \forall w ) accepted by ( M ), does ( w \models \varphi )?</td>
<td></td>
</tr>
<tr>
<td><strong>Synthesis:</strong> Construct ( M ) such that ( M ) models ( \varphi )?</td>
<td></td>
</tr>
</tbody>
</table>
Verification setting

<table>
<thead>
<tr>
<th>System</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automata $\mathcal{A}$ (Büchi60)</td>
<td>$MSO[&lt;] \varphi$</td>
</tr>
</tbody>
</table>

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

**Synthesis:** Construct $M$ such that $M$ models $\varphi$ ?
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:

\[ a_1r_1a_2r_2 \ldots a_nr_n \]

→ Can be seen as a word over **Actions** × **Reactions** and use automata methods.
Verification of Reactive systems

Reactive Systems
Automata $A$ (Mealy Machine)

$p \xrightarrow{(a, \cdot)} q$

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

$\neg (\cdot, c) U (a, \cdot)$
Verification of Reactive systems

Reactive Systems
Automata \( A \) (Mealy Machine)

\[
\begin{align*}
    & a \mid b \quad a \mid c \\
    & b \mid c \\
    & a \mid \cdot \\
    & p \rightarrow a \rightarrow q
\end{align*}
\]

Specification
Logic formulas \( \varphi \) (MSO, LTL)

\[
\exists x \ (a, \cdot)(x) \land \forall y \ (\cdot, c)(y) \rightarrow x < y
\]

\[
(\neg(\cdot, c))U(a, \cdot)
\]
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:
  \[
  \{ (u_1 \ldots u_n, u_n \ldots u_1) \mid u \in A^* \},
  \]

- Nondeterminism: Every input letter appears in the output exactly once (shuffle):
  \[
  \{ (u_1 \ldots u_n, u_{\pi(1)} \ldots u_{\pi(n)} \mid \pi \text{ a permutation} \} \]
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:
    $$\{(u_1 \ldots u_n, u_n \ldots u_1) \mid u \in A^*\},$$
  - Nondeterminism: Every input letter appears in the output exactly once (shuffle):
    $$\{(u_1 \ldots u_n, u_{\pi(1)} \ldots u_{\pi(n)} \mid \pi \text{ a permutation}\}$$
- Has good decidable verification properties:
  - Model-checking: Does a deterministic 2-way transducer $T$ satisfies a formula $\varphi$?
  - Synthesis: Construct a deterministic 2-way transducer $T$ that satisfies $\varphi$. 
Non reactive systems

\[\text{Input} : \quad a \quad \rightarrow \quad a \quad \rightarrow \quad b \quad \rightarrow \quad b\]

\[\text{Output} : \quad a \quad \rightarrow \quad b \quad \rightarrow \quad b \quad \rightarrow \quad a\]

\text{Input/Output relation in the case of reactive systems.}
Non reactive systems

Input:

Output:

→ 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 $w \rightarrow \text{rev}(w)$.

**Input**:

$\vdash \quad a \quad a \quad b \quad b \quad \vdash$

**Output**:
Origin semantics of a transducer

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

Input: $\operatorname{Input} : \quad \overline{\quad} \rightarrow a \rightarrow a \rightarrow b \rightarrow b \rightarrow \overline{\quad}$

Output: $\overline{\quad}$

\[
\begin{align*}
\text{Input: } & a | \epsilon, \rightarrow & \text{Output: } & a | \epsilon, \rightarrow \\
b | \epsilon, \rightarrow & & \quad \quad & b | \epsilon, \rightarrow \\
\end{align*}
\]
Origin semantics of a transducer

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

Input: \[ \dashv \rightarrow a \rightarrow a \rightarrow b \rightarrow b \rightarrow \dashv \]

Output:
Origin semantics of a transducer

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

\[
\begin{align*}
\models & \epsilon, \rightarrow \\
\models a & \epsilon, \rightarrow \\
\models b & \epsilon, \rightarrow \\
\models \epsilon, \leftarrow & a \\
\models \epsilon, \leftarrow & b \\
\models \epsilon, \rightarrow & a \\
\models \epsilon, \leftarrow & b \\
\models \epsilon, \rightarrow & \end{align*}
\]

Input: $\vdash a \rightarrow a \rightarrow b \rightarrow b \rightarrow \rightarrow$

Output:
Origin semantics of a transducer

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

Input: \[ \vdash | \epsilon, \rightarrow \quad a | \epsilon, \rightarrow \quad b | \epsilon, \rightarrow \]

Output: \[ \vdash | \epsilon, \leftarrow \quad a | a, \leftarrow \quad b | b, \leftarrow \]
A transducer realising the reverse function $w \rightarrow \text{rev}(w)$.

\[\begin{align*}
&\mbox{Input:} & &\iff & \epsilon, \rightarrow & &\mbox{Output:} & &\iff & \epsilon, \rightarrow \\
& a & \iff & \epsilon, \rightarrow & & a & \iff & a, \leftarrow & & a & \iff & \epsilon, \rightarrow \\
& b & \iff & \epsilon, \rightarrow & & b & \iff & b, \leftarrow & & b & \iff & \epsilon, \rightarrow \\
\end{align*}\]
Origin semantics of a transducer

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

\[
\begin{align*}
1 & \xrightarrow{\epsilon} 2 \\
2 & \xrightarrow{\epsilon} 3 \\
1 & \xrightarrow{a} 2 \\
2 & \xrightarrow{a} 3 \\
1 & \xrightarrow{b} 2 \\
2 & \xrightarrow{b} 3 \\
\end{align*}
\]

Input: \[\overline{\epsilon} \rightarrow a \rightarrow a \rightarrow b \rightarrow b \rightarrow \overline{\epsilon}\]

Output:
Origin semantics of a transducer

A transducer realising the reverse function \( w \rightarrow \text{rev}(w) \).

Input: \( \downarrow \quad a \rightarrow a \rightarrow b \rightarrow b \rightarrow \downarrow \)

Output: \( \uparrow b \)
Origin semantics of a transducer

A transducer realising the reverse function \( w \rightarrow \text{rev}(w) \).

\[ \begin{align*}
\vdash & \quad \epsilon, \rightarrow \\
\epsilon, \rightarrow & \quad \rightarrow \\
\epsilon, \rightarrow & \quad a, \rightarrow \\
b, \rightarrow & \quad b, \rightarrow \\
\epsilon, \leftarrow & \quad \vdash \\
\epsilon, \leftarrow & \quad \leftarrow \\
\epsilon, \leftarrow & \quad a, \leftarrow \\
b, \leftarrow & \quad b, \leftarrow
\end{align*} \]

**Input:**  
\[ \begin{array}{c}
\vdash \\
a \\
a \\
b \\
b \\
\vdash
\end{array} \]

**Output:**  
\[ \begin{array}{c}
b \\
b
\end{array} \]
Origin semantics of a transducer

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

Input: $\parallel a \rightarrow a \rightarrow b \rightarrow b \rightarrow \parallel$

Output: $b \rightarrow b \rightarrow a$
Origin semantics of a transducer

A transducer realising the reverse function \( w \rightarrow \text{rev}(w) \).

\[
\begin{align*}
\top &| \epsilon, \rightarrow \\
\alpha &| \epsilon, \rightarrow \\
\beta &| \epsilon, \rightarrow \\
\alpha &| \alpha, \leftarrow \\
\beta &| \beta, \leftarrow
\end{align*}
\]

Input:

\[
\begin{align*}
\top &\rightarrow \alpha \\
\alpha &\rightarrow \alpha \\
\alpha &\rightarrow \beta \\
\beta &\rightarrow \beta \\
\beta &\rightarrow \top
\end{align*}
\]

Output:

\[
\begin{align*}
\beta &\rightarrow \beta \\
\beta &\rightarrow \alpha \\
\alpha &\rightarrow \alpha
\end{align*}
\]
Origin semantics of a transducer

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

\[
\begin{align*}
\text{Input:} & \quad \vdash | \epsilon, \rightarrow \\
& \quad a | \epsilon, \rightarrow \\
& \quad b | \epsilon, \rightarrow \\
\text{Output:} & \quad b \rightarrow b \rightarrow a \rightarrow a
\end{align*}
\]
Origin semantics of a transducer

*Input*: \[a \rightarrow a \rightarrow b \rightarrow b\]

*Output*: \[a \rightarrow b \rightarrow b \rightarrow a\]

*Origin graphs* as models for traces of executions.
The logic $MSO[<_i,<_o,\circ]$
Decidability of origin logics

Problem:

Satisfiability of $MSO[<_i, <_o, o]$ is undecidable.
Problem:

Satisfiability of $MSO[<_i,<_o,o]$ is undecidable.

→ In fact, even $FO^2[<_i,<_o,S_o,o]$ is undecidable.
Decidability of origin logics

Problem:

Satisfiability of $MSO[<_{i}, <_{o}, 0]$ is undecidable.

→ In fact, even $FO^{2}[<_{i}, <_{o}, S_{o}, 0]$ is undecidable.

Logic $FO^{2}$

First-Order logic with two reusable variables:

$$\exists x \ a(x) \land \exists y (b(y) \land x < y \land \exists x (c(x) \land y < x))$$

describes the language $\Sigma^{*}a\Sigma^{*}b\Sigma^{*}c\Sigma^{*}$. 
Undecidability of Satisfiability

Proving undecidability of $FO^2[<_i,<_o,S_o,o]$:

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

\[
\begin{align*}
u_1 &= ab \\
v_1 &= \epsilon \\
u_2 &= b \\
v_2 &= abb
\end{align*}
\]
Undecidability of Satisfiability

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

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

$1 \ 2$
$a \ a \ b \ b \ b \ b \ b$
Undecidability of Satisfiability

Proving undecidability of $FO^2[<_i, <_o, S_o, o]$:

Encoding PCP instances: given $(u_i, v_i)_{i \leq n}$, does there exist $i_1 \ldots i_k$ 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 $FO^2[<_i, <_o, S_o, o]$: 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

$$\forall in x \land_i i(x) \rightarrow u_i(x) \land v_i(x)$$
Undecidability of Satisfiability

Proving undecidability of $FO^2[<_i,<_o,S_o,o]$: 

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

$\forall \in \in x \bigwedge_i \ i(x) \rightarrow u_i(x) \wedge v_i(x)$

$u_1(x)$: the production of $x$ belongs to $\Sigma^* a \Sigma^* b \Sigma^*$

$\exists y \ o(y) = x \wedge a(y) \wedge \exists x \ (o(x) = o(y) \wedge y <_o x \wedge b(x))$
Undecidability of Satisfiability

Proving undecidability of $FO^2[<_i,<_o,S_o,o]$:

Encoding PCP instances: given $(u_i, v_i)_{i \leq n}$, does there exist $i_1 \ldots i_k$ such that $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$.

- $u_1 = ab$
- $v_1 = \varepsilon$
- $u_2 = b$
- $v_2 = abb$

PCP instance $(u_i, v_i)_{i \leq n}$ encoded as

$\forall^\infty x \land \land_i i(x) \rightarrow u_i(x) \land v_i(x)$

$u_1(x)$: the production of $x$ belongs to $\Sigma^*a\Sigma^*b\Sigma^* \cap \Sigma^2$

$\neg (\exists y o(y) = x \land \exists x (o(x) = o(y) \land y <_o x \land \exists y (o(y) = o(x) \land y < x))$
Undecidability of Satisfiability

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

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

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 $FO^2[<_i,<_o,S_o,o]$:

Encoding PCP instances: given $(u_i,v_i)_{i \leq n}$, does there exists $i_1 \ldots i_k$ such that $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$.

$$
egin{align*}
    u_1 &= ab \\
    v_1 &= \epsilon \\
    u_2 &= b \\
    v_2 &= abb
\end{align*}
$$

PCP instance $(u_i,v_i)_{i \leq n}$ encoded as

$$
\forall^{in} x \bigwedge_i i(x) \rightarrow u_i(x) \wedge v_i(x)
\wedge_\phi_{op} \wedge \phi_{op} \\
\bigwedge^{out} x \bigwedge_{a \in \Sigma} a(x) \rightarrow \exists y (S_o(x,y) \wedge a(y))
$$

(the output word belongs to $(\bigcup_{a \in \Sigma} aa)^*$)
The logic $L_T$

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

Example:

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

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

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

Example:

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

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

We set $L_T = FO^2[\prec_o, o, MSO_{bin}[\prec_i]]$. 
The logic $L_T$

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

Example:

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

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

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

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 $L_T$

$L_T = FO^2[<_o, o, MSObin[<_i]]$

Theorem:
The satisfiability problem is decidable for the logic $L_T$. 

[Bojanczyk et al.17]: Model-checking against $MSObin[<_i]$ is decidable.
Decidability of $L_T$

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

**Theorem:**
The satisfiability problem is decidable for the logic $L_T$.

**Corollary**
Model-checking of a 2-way transducer $T$ against a formula $\varphi$ of $L_T$ is decidable.

**Proof:** Transform $T$ into a formula $\phi_T$ and check satisfiability of $\phi_T \land \neg \varphi \in L_T$
Decidability of $L_T$

\[ L_T = FO^2[<o, o, MSO_{bin}[<i]] \]

**Theorem:**

The satisfiability problem is decidable for the logic $L_T$.

**Corollary**

Model-checking of a 2-way transducer $T$ against a formula $\varphi$ of $L_T$ is decidable.

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

[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.

→ 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 $\varphi$ of $L_T$, construct a deterministic 2-way transducer $T$ such that:

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

Theorem
From any formula $\varphi$ of $L_T$, we can construct a deterministic 2-way transducer that realises it.
Synthesis of $L_T$

Regular Synthesis problem:

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

1. $\text{dom}(\varphi) = \text{dom}(T)$,
2. for all $u \in \text{dom}(T)$, $(u, T(u)) \models \varphi$.

Theorem

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

→ 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: $(b, 2)(c, 3)(c, 4)(a, 1)$.

→ used to model words over unbounded sets such as processus identifiers.

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

\[
\begin{align*}
1 & \quad 2 & \quad 3 & \quad 4 \\
\text{a} & \quad \text{a} & \quad \text{b} & \quad \text{b} \\
\text{b} & \quad \text{c} & \quad \text{c} & \quad \text{a}
\end{align*}
\]

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

\[
\begin{align*}
\tau : & \quad 1 \rightarrow a & 2 \rightarrow a \\
& \quad 3 \rightarrow b & 4 \rightarrow b
\end{align*}
\]
A **typed data word** is a pair \((w, \tau)\) where \(w \in (\Gamma \times \mathbb{N})^*\) and \(\tau : \mathbb{N} \to \Sigma\).

\[
\begin{align*}
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\end{array} & \begin{array}{cccc}
\xrightarrow{a} & \xrightarrow{a} & \xrightarrow{b} & \xrightarrow{b} \\
\xleftarrow{b} & \xleftarrow{c} & \xleftarrow{c} & \xleftarrow{a} \\
\end{array}
\end{align*}
\]

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

\[
\begin{align*}
\tau : & \\
1 & \rightarrow \ a & 2 & \rightarrow \ a \\
3 & \rightarrow \ b & 4 & \rightarrow \ b
\end{align*}
\]

\(MSO[<_i, <_o, \circ] \) is equivalent to \(MSO[\preceq, <, \delta]\).

The order-preserving formula is equivalent to having data appear in increasing order:

\[
\forall xy \ x <_o y \iff \circ(x) \leq_i \circ(y) \iff \forall xy \ x < y \iff \delta(x) \preceq \delta(y)
\]
New results on data words

- The logic $L_T$ can be seen as a logic $L_D = FO^2[<, MSO[\preceq]]$ on data words.
  - Quantifications in $MSO[\preceq]$ 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].
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!