On Some Transducer Synthesis Problems

Emmanuel Filiot

Université libre de Bruxelles & FNRS

IRIF, June 2021

A basic zero-sum infinite game

Adam picks symbols in Σ

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

a

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

aa

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

aab

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

$aabb$

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

$aabbc$

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

$aabbca$

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

$aabbcaa$

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

$aabbcaaa$

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

$aabbcaaab$

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

$aabbcaaabb$

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

$aabbcaaabba$

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

$aabbcaaabbaa$

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

$$aabbcaaabbaac$$

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

$$aabbcaaabbaacb\dots$$

A basic zero-sum infinite game

Adam picks symbols in Σ

Eve picks symbols in Γ

$$u = aabbcaaabbaacb\dots$$

- Eve wins if $u \in W \subseteq (\Sigma\Gamma)^\omega$

Example: mimic Adam until first c

$$W = (aa + bb)^* c (\Gamma\Sigma)^\omega$$

A basic zero-sum infinite game

Adam picks symbols in Σ Eve picks symbols in Γ

$$u = aabbcaaabbaacb\ldots$$

- Eve wins if $u \in W \subseteq (\Sigma\Gamma)^\omega$

Example: mimic Adam until first c

$$W = (aa + bb)^* c(\Gamma\Sigma)^\omega$$

- Formally, she wins if she has a strategy $\lambda : \Sigma^+ \to \Gamma$ s.t.

$$\forall \sigma_1\sigma_2 \ldots \in \Sigma^\omega,\ \sigma_1\beta_1\sigma_2\beta_2 \cdots \in W$$

where $\beta_i = \lambda(\sigma_1 \ldots \sigma_i)$

A basic zero-sum infinite game

Adam picks symbols in Σ Eve picks symbols in Γ

$$u = aabbcaaabbaacb\dots$$

- Eve wins if $u \in W \subseteq (\Sigma\Gamma)^\omega$

Example: mimic Adam until first c

$$W = (aa + bb)^* c(\Gamma\Sigma)^\omega$$

- Formally, she wins if she has a strategy $\lambda : \Sigma^+ \to \Gamma$ s.t.

$$\forall \sigma_1\sigma_2 \ldots \in \Sigma^\omega,\ \sigma_1\beta_1\sigma_2\beta_2 \cdots \in W$$

Example: Eve *always* mimics Adam

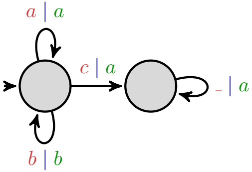
$$\lambda(u\sigma) = \sigma$$

Church Synthesis

- W is ω-regular (MSO, automata, ...)
- W is the *specification*, λ the *implementation*

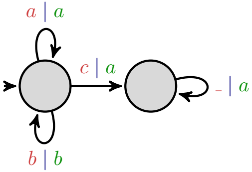
Church Synthesis

- W is ω-regular (MSO, automata, ...)
- W is the *specification*, λ the *implementation*
- finite memory always suffices (Mealy machines)



Church Synthesis

- W is ω-regular (MSO, automata, ...)
- W is the *specification*, λ the *implementation*
- finite memory always suffices (Mealy machines)

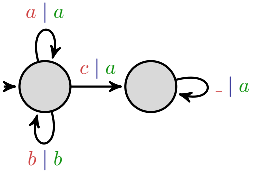


Complexity Results

- decidable problem [Büchi-Landweber 69] (via parity games)
- ExpTime-c from automata
- 2ExpTime-c for LTL specifications [Pnueli/Rosner 89]

Church Synthesis

- W is ω-regular (MSO, automata, ...)
- W is the *specification*, λ the *implementation*
- finite memory always suffices (Mealy machines)



Complexity Results

- decidable problem [Büchi-Landweber 69] (via parity games)
- ExpTime-c from automata
- 2ExpTime-c for LTL specifications [Pnueli/Rosner 89]

- reactive synthesis competition SYNTCOMP since 2014

synchronous specification / implementation

Church synthesis as a uniformisation problem

Definition (Uniformiser)

Given $R \subseteq I \times O$, a uniformiser of R is a function $f : I \to O$ such that

1. for all $i \in dom(f)$, $(i, f(i)) \in R$,

Church synthesis as a uniformisation problem

Definition (Uniformiser)

Given $R \subseteq I \times O$, a uniformiser of R is a function $f : I \to O$ such that

1. for all $i \in dom(f)$, $(i, f(i)) \in R$,
2. $dom(f) = dom(R)$

Church synthesis as a uniformisation problem

Definition (Uniformiser)

Given $R \subseteq I \times O$, a uniformiser of R is a function $f : I \to O$ such that

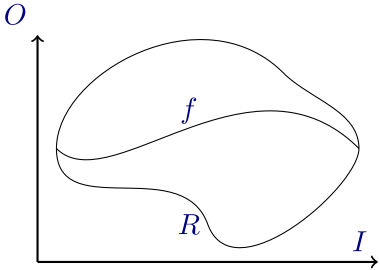
1. for all $i \in dom(f)$, $(i, f(i)) \in R$,
2. $dom(f) = dom(R)$

Church synthesis as a uniformisation problem

Definition (Uniformiser)

Given $R \subseteq I \times O$, a uniformiser of R is a function $f : I \to O$ such that

1. for all $i \in dom(f)$, $(i, f(i)) \in R$,
2. $dom(f) = dom(R)$



Church synthesis as a uniformisation problem

Definition (Uniformiser)

Given $R \subseteq I \times O$, a uniformiser of R is a function $f : I \to O$ such that

1. for all $i \in dom(f)$, $(i, f(i)) \in R$,
2. $dom(f) = dom(R)$

Definition ($(\mathcal{S}, \mathcal{I})$-uniformisation problem)

Given a relation $R \in \mathcal{S}$, does there exists a uniformiser $f \in \mathcal{I}$?

Church synthesis as a uniformisation problem

- $I = \Sigma^\omega$, $O = \Gamma^\omega$
- any $W \subseteq (\Sigma\Gamma)^\omega$ defines a relation $R_W \subseteq \Sigma^\omega \times \Gamma^\omega$:

$$R_W = \{(\sigma_1\sigma_2\ldots, \beta_1\beta_2\ldots) \mid \sigma_1\beta_1\sigma_2\beta_2\cdots \in W\}$$

AUT : class of ω-automatic relations.

Church synthesis as a uniformisation problem

- $I = \Sigma^\omega$, $O = \Gamma^\omega$
- any $W \subseteq (\Sigma\Gamma)^\omega$ defines a relation $R_W \subseteq \Sigma^\omega \times \Gamma^\omega$:

$$R_W = \{(\sigma_1\sigma_2 \ldots, \beta_1\beta_2 \ldots) \mid \sigma_1\beta_1\sigma_2\beta_2 \cdots \in W\}$$

AUT : class of ω-automatic relations.

- any strategy $\lambda : \Sigma^* \to \Gamma$ defines a "strategic" function

$$f_\lambda : \Sigma^\omega \to \Gamma^\omega$$

STR : class of strategic functions.

Church synthesis as a uniformisation problem

- $I = \Sigma^\omega$, $O = \Gamma^\omega$
- any $W \subseteq (\Sigma\Gamma)^\omega$ defines a relation $R_W \subseteq \Sigma^\omega \times \Gamma^\omega$:

$$R_W = \{(\sigma_1\sigma_2\ldots, \beta_1\beta_2\ldots) \mid \sigma_1\beta_1\sigma_2\beta_2\cdots \in W\}$$

AUT : class of ω-automatic relations.

- any strategy $\lambda : \Sigma^* \to \Gamma$ defines a "strategic" function

$$f_\lambda : \Sigma^\omega \to \Gamma^\omega$$

STR : class of strategic functions.

Church synthesis as a uniformisation problem

- $I = \Sigma^\omega$, $O = \Gamma^\omega$
- any $W \subseteq (\Sigma\Gamma)^\omega$ defines a relation $R_W \subseteq \Sigma^\omega \times \Gamma^\omega$:

$$R_W = \{(\sigma_1\sigma_2\dots, \beta_1\beta_2\dots) \mid \sigma_1\beta_1\sigma_2\beta_2\dots \in W\}$$

AUT : class of ω-automatic relations.

- any strategy $\lambda : \Sigma^* \to \Gamma$ defines a "strategic" function

$$f_\lambda : \Sigma^\omega \to \Gamma^\omega$$

STR : class of strategic functions.

Proposition

R_W is uniformizable by f_λ iff λ is winning for Eve in the Church game with objective W.

Church synthesis as a uniformisation problem

- $I = \Sigma^\omega$, $O = \Gamma^\omega$
- any $W \subseteq (\Sigma\Gamma)^\omega$ defines a relation $R_W \subseteq \Sigma^\omega \times \Gamma^\omega$:

$$R_W = \{(\sigma_1\sigma_2\ldots, \beta_1\beta_2\ldots) \mid \sigma_1\beta_1\sigma_2\beta_2\cdots \in W\}$$

AUT : class of ω-automatic relations.

- any strategy $\lambda : \Sigma^* \to \Gamma$ defines a "strategic" function

$$f_\lambda : \Sigma^\omega \to \Gamma^\omega$$

STR : class of strategic functions.

Proposition

R_W is uniformizable by f_λ iff λ is winning for Eve in the Church game with objective W.

Reformulation of Büchi-Landweber's theorem

- The (AUT, STR)-uniformisation problem is decidable,
- and same as $(AUT, MEALY)$-uniformisation problem.

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W: Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W: Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W: Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{ccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W: Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

$\perp$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W: Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

$$\begin{array}{c} \perp \\ ? \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W**:** Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

$$\begin{array}{cc} \perp & \perp \\ ? & \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W**:** Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

$$\begin{array}{cc} \perp & \perp \\ ? & ? \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\quad\Gamma = \{a, b\}$

Spec W: Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

$$\begin{array}{ccc} \perp & \perp & a \\ ? & ? & \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W: Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

$$\begin{array}{ccc} \perp & \perp & a \\ a & a & a \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W: Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

$$\begin{array}{cccc} \perp & \perp & a & \perp \\ a & a & a & \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W: Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some ”streaming” algorithm

$$\begin{array}{cccc} \perp & \perp & a & \perp \\ a & a & a & ? \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\quad \Gamma = \{a, b\}$

Spec W**:** Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

$$\begin{array}{ccccc} \perp & \perp & a & \perp & \perp \\ a & a & a & ? & \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W**:** Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

$$\begin{array}{ccccc} \perp & \perp & a & \perp & \perp \\ a & a & a & ? & ? \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W: Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \ldots & \perp & b & \ldots \\ a & a & a & b & b & b & \ldots & b & b & \ldots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some ”streaming” algorithm

$$\begin{array}{cccccc} \perp & \perp & a & \perp & \perp & \perp \\ a & a & a & ? & ? & \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W**:** Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \ldots & \perp & b & \ldots \\ a & a & a & b & b & b & \ldots & b & b & \ldots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

$$\begin{array}{cccccc} \perp & \perp & a & \perp & \perp & \perp \\ a & a & a & ? & ? & ? \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W**:** Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

$$\begin{array}{cccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp \\ a & a & a & ? & ? & ? & & \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W: Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

$$\begin{array}{cccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp \\ a & a & a & ? & ? & ? & \dots & ? \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\bot, a, b\}$ $\Gamma = \{a, b\}$

Spec W: Replace each symbol by the following non $\bot$ symbol

$$\begin{array}{cccccccccc} \bot & \bot & a & \bot & \bot & \bot & \dots & \bot & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\bot a)^* aa + (\bot b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

$$\begin{array}{ccccccccc} \bot & \bot & a & \bot & \bot & \bot & \dots & \bot & b \\ a & a & a & ? & ? & ? & \dots & ? & \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\Gamma = \{a, b\}$

Spec W: Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

$$\begin{array}{ccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b \\ a & a & a & b & b & b & \dots & b & b \end{array}$$

Wait and see: beyond Mealy machines

$\Sigma = \{\perp, a, b\}$ $\quad\quad$ $\Gamma = \{a, b\}$

Spec W: Replace each symbol by the following non $\perp$ symbol

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

- $W \in AUT$

$$W = ((\perp a)^* aa + (\perp b)^* bb)^\omega$$

- unrealisable with only finite memory
- but realisable by some "streaming" algorithm

$$\begin{array}{cccccccccc} \perp & \perp & a & \perp & \perp & \perp & \dots & \perp & b & \dots \\ a & a & a & b & b & b & \dots & b & b & \dots \end{array}$$

Talk Outline

1. beyond Mealy implementations
2. beyond automatic specifications

Beyond Mealy Implementations

Beyond Mealy Implementations

- based on works together with Vrunda Dave, Nathan Lhote, Krishna S. and Sarah Winter.

Beyond Mealy Implementations

- based on works together with Vrunda Dave, Nathan Lhote, Krishna S. and Sarah Winter.
- **Hypothesis**: specifications are automatic relations

Motivating question

Question

If a spec is not Mealy-realizable, is it still realizable in some larger class of implementations ?

Motivating question

Question

If a spec is not Mealy-realizable, is it still realizable in some larger class of implementations ?

Theorem (Uniformization property)

Any automatic relation can be uniformized by an automatic function.

Motivating question

Question

If a spec is not Mealy-realizable, is it still realizable in some larger class of implementations ?

Theorem (Uniformization property)

Any automatic relation can be uniformized by an automatic function.

- f automatic if $\{\sigma_1\beta_1\sigma_2\beta_2\cdots \mid f(\sigma_1\sigma_2\ldots) = \beta_1\beta_2\ldots\}$ is ω-regular.
- in other words, any automatic spec is realizable by some automatic function
- result due to Siefkes 75 and Choffrut/Grigorieff 99, some proof in Carayol/Löding 2014 too

Proof of the theorem (for finite words)

Let $A = (Q, q_0, F, \Delta)$ be a DFA for R.

1. order the transitions $<_\Delta$, extend to runs $<_{\Delta^*}$ lexicographically

Proof of the theorem (for finite words)

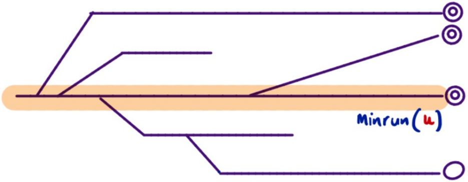
Let $A = (Q, q_0, F, \Delta)$ be a DFA for R.

1. order the transitions $<_\Delta$, extend to runs $<_{\Delta^*}$ lexicographically
2. for $u \in \Sigma^*$,
   $\mathrm{minrun}(u) = min_{<_{\Delta^*}}\{r \mid r \text{ accepting and input}(r) = u\}$

Proof of the theorem (for finite words)

Let $A = (Q, q_0, F, \Delta)$ be a DFA for R.

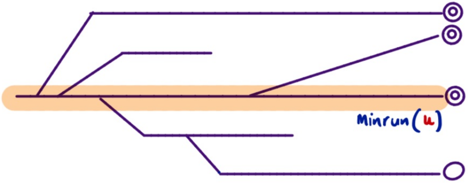
1. order the transitions $<_\Delta$, extend to runs $<_{\Delta^*}$ lexicographically
2. for $u \in \Sigma^*$,
   $\text{minrun}(u) = min_{<_{\Delta^*}}\{r \mid r \text{ accepting and input}(r) = u\}$



Proof of the theorem (for finite words)

Let $A = (Q, q_0, F, \Delta)$ be a DFA for R.

1. order the transitions $<_\Delta$, extend to runs $<_{\Delta^*}$ lexicographically
2. for $u \in \Sigma^*$,
   $\text{minrun}(u) = min_{<_{\Delta^*}}\{r \mid r \text{ accepting and input}(r) = u\}$

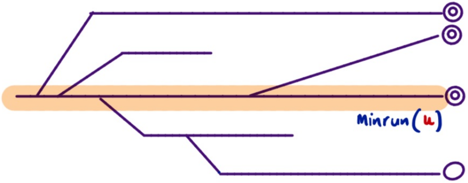


Lemma $L_{\min} = \{\text{minrun}(u) \mid u \in \Sigma^*\}$ is regular.

Proof of the theorem (for finite words)

Let $A = (Q, q_0, F, \Delta)$ be a DFA for R.

1. order the transitions $<_\Delta$, extend to runs $<_{\Delta^*}$ lexicographically
2. for $u \in \Sigma^*$,
   $\text{minrun}(u) = min_{<_{\Delta^*}}\{r \mid r \text{ accepting and input}(r) = u\}$



Lemma $L_{\min} = \{\text{minrun}(u) \mid u \in \Sigma^*\}$ is regular.

3. let $A_{\min}$ be a DFA accepting $L_{\min}$ and use it as a filter
4. the product $A \otimes A_{\min}$ defines an automatic function. $\square$

Computability over infinite words

- the latter theorem is not very useful for synthesis
- because some automatic functions are **not** computable

Computability over infinite words

- the latter theorem is not very useful for synthesis
- because some automatic functions are **not** computable

Example

$\Sigma = \{a, b\}$,$\Gamma = \{a, b\}$.

$$f(u) = \begin{cases} a^\omega & \text{if } |u|_a = \infty \\ b^\omega & \text{otherwise.} \end{cases}$$

Computability over infinite words

- the latter theorem is not very useful for synthesis
- because some automatic functions are **not** computable

Example

$\Sigma = \{a, b\}, \Gamma = \{a, b\}$.

$$f(u) = \begin{cases} a^\omega & \text{if } |u|_a = \infty \\ b^\omega & \text{otherwise.} \end{cases}$$

- f is automatic: $((\Sigma a)^* aa)^\omega + (\Sigma b)^* (bb)^\omega$
- f is not computable:
  no algorithm computes longer and longer output prefixes
  while reading longer and longer input prefixes

Turing-computability over infinite words

Consider a deterministic Turing machine M with 3 tapes:

- a one-way read-only input tape
- a two-way working tape
- a one-way write-only output tape

Turing-computability over infinite words

Consider a deterministic Turing machine M with 3 tapes:

- a one-way read-only input tape
- a two-way working tape
- a one-way write-only output tape

$M(u, k)$:output written after reading $u_1 u_2 \ldots u_k$

Turing-computability over infinite words

Consider a deterministic Turing machine M with 3 tapes:

- a one-way read-only input tape
- a two-way working tape
- a one-way write-only output tape

$M(u,k)$:output written after reading $u_1 u_2 \dots u_k$

Definition

M computes f if for all $u \in \mathrm{dom}(f)$, there exists $j_1 < j_2 < \dots$:

$$M(\alpha, j_1) \prec M(\alpha, j_2) \prec \cdots \preceq f(u)$$

Turing-computability over infinite words

Consider a deterministic Turing machine M with 3 tapes:

- a one-way read-only input tape
- a two-way working tape
- a one-way write-only output tape

$M(u, k)$:output written after reading $u_1 u_2 \dots u_k$

Definition

M computes f if for all $u \in \mathrm{dom}(f)$, there exists $j_1 < j_2 < \dots$:

$$M(\alpha, j_1) \prec M(\alpha, j_2) \prec \dots \preceq f(u)$$

The following function $f_\perp$ is computable

$$\begin{array}{ccccccccccc} & \perp & \perp & a & \perp & \dots & \perp & b & \dots & \in \Box\Diamond(a \vee b) \\ \mapsto & a & a & a & b & \dots & b & b & \dots & \end{array}$$

Turing-computability over infinite words

Consider a deterministic Turing machine M with 3 tapes:

- a one-way read-only input tape
- a two-way working tape
- a one-way write-only output tape

$M(u,k)$:output written after reading $u_1 u_2 \dots u_k$

Definition

M computes f if for all $u \in \mathrm{dom}(f)$, there exists $j_1 < j_2 < \dots$:

$$M(\alpha, j_1) \prec M(\alpha, j_2) \prec \dots \preceq f(u)$$

The following function $f_\perp$ is computable

$$\begin{array}{ccccccccccc} & \perp & \perp & a & \perp & \dots & \perp & b & \dots & \in \Box\Diamond(a \vee b) \\ \mapsto & a & a & a & b & \dots & b & b & \dots & \end{array}$$

but $f_\perp \cup (\Diamond\Box\perp \mapsto \perp^\omega)$ is not.

Some characterization and some decidability result

- any computable function is continuous for the Cantor distance $d(u,v) = 0$ if $u = v$, and $2^{-|lcp(u,v)|}$ otherwise
- the converse is not true, but true in some cases:

Theorem (Dave,F.,Krishna,Lhote,19)

Let f be a function preserving regular languages under inverse image.

f is computable iff f is continuous.

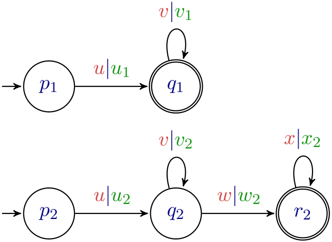
Theorem

Continuity (and so computability) is decidable in NLogSpace for automatic functions.

Proved for a large class (regular functions), already known for rational functions by Prieur,01.

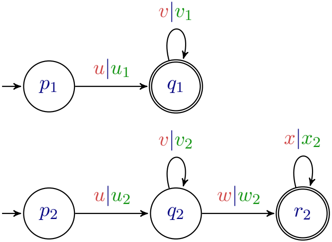
Idea of the algorithm

1. Check the following forbidden pattern where $u_1 \neq u_2$:



Idea of the algorithm

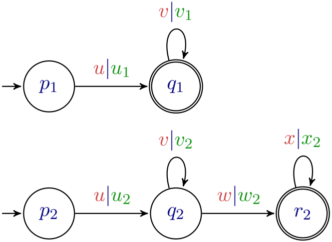
1. Check the following forbidden pattern where $u_1 \neq u_2$:



$$\lim_{n\to\infty} f(uv^n wx^\omega) \quad \neq \quad f(\lim_{n\to\infty} uv^n wx^\omega)$$

Idea of the algorithm

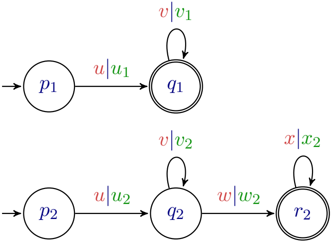
1. Check the following forbidden pattern where $u_1 \neq u_2$:



$$\lim_{n\to\infty} f(uv^n wx^\omega) \quad \neq \quad f(\lim_{n\to\infty} uv^n wx^\omega)$$

Idea of the algorithm

1. Check the following forbidden pattern where $u_1 \neq u_2$:

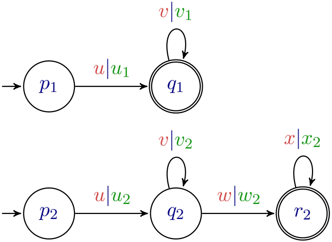


$$\lim_{n\to\infty} f(uv^n wx^\omega) \quad \neq \quad f(\lim_{n\to\infty} uv^n wx^\omega)$$

2. encode it in the pattern logic of [F., Mazzocchi, Raskin, 20] which has decidable model-checking problem

Idea of the algorithm

1. Check the following forbidden pattern where $u_1 \neq u_2$:



$$\lim_{n\to\infty} f(uv^n wx^\omega) \quad \neq \quad f(\lim_{n\to\infty} uv^n wx^\omega)$$

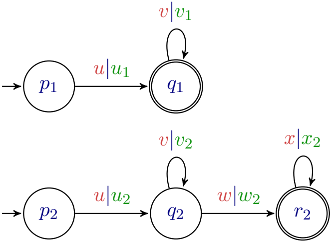
$$\parallel$$

$$\lim_{n\to\infty} u_2 v_2^n w_2 x_2^\omega$$

2. encode it in the pattern logic of [F., Mazzocchi, Raskin, 20] which has decidable model-checking problem

Idea of the algorithm

1. Check the following forbidden pattern where $u_1 \neq u_2$:

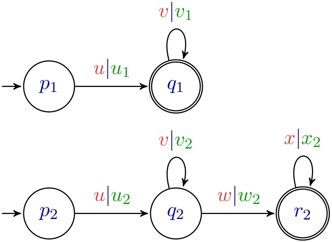


$$\lim_{n\to\infty} f(uv^n wx^\omega) \quad \neq \quad f(\lim_{n\to\infty} uv^n wx^\omega)$$
$$\shortparallel$$
$$\lim_{n\to\infty} u_2 v_2^n w_2 x_2^\omega$$
$$\shortparallel$$
$$u_2 v_2^\omega$$

2. encode it in the pattern logic of [F., Mazzocchi, Raskin, 20] which has decidable model-checking problem

Idea of the algorithm

1. Check the following forbidden pattern where $u_1 \neq u_2$:



$$\lim_{n\to\infty} f(uv^n wx^\omega) \neq f(\lim_{n\to\infty} uv^n wx^\omega)$$

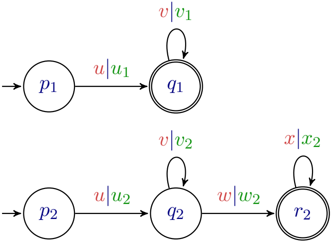
$$\lim_{n\to\infty} f(uv^n wx^\omega) = \lim_{n\to\infty} u_2 v_2^n w_2 x_2^\omega = u_2 v_2^\omega$$

$$f(\lim_{n\to\infty} uv^n wx^\omega) = f(uv^\omega)$$

2. encode it in the pattern logic of [F., Mazzocchi, Raskin, 20] which has decidable model-checking problem

Idea of the algorithm

1. Check the following forbidden pattern where $u_1 \neq u_2$:



$$\begin{array}{ccc}
\lim_{n\to\infty} f(uv^n w x^\omega) & \neq & f(\lim_{n\to\infty} uv^n w x^\omega \\
\| & & \| \\
\lim_{n\to\infty} u_2 v_2^n w_2 x_2^\omega & & f(uv^\omega) \\
\| & & \| \\
u_2 v_2^\omega & \neq & u_1 v_1^\omega
\end{array}$$

2. encode it in the pattern logic of [F., Mazzocchi, Raskin, 20] which has decidable model-checking problem

Back to synthesis

Problem

Input: an automatic specification $R \subseteq \Sigma^\omega \times \Gamma^\omega$

Output: Is R uniformisable by some computable function f ?

$f \subseteq R$ and $\mathrm{dom}(f) = \mathrm{dom}(R)$

Back to synthesis

Problem

Input: an automatic specification $R \subseteq \Sigma^\omega \times \Gamma^\omega$
Output: Is R uniformisable by some computable function f ?

$f \subseteq R$ and $\mathrm{dom}(f) = \mathrm{dom}(R)$

Results

- 2EXPTime when R has **total domain** and is given as a DPA Holtmann/Kaiser/Thomas'10

Back to synthesis

Problem

Input: an automatic specification $R \subseteq \Sigma^\omega \times \Gamma^\omega$
Output: Is R uniformisable by some computable function f ?

$f \subseteq R$ and $\mathrm{dom}(f) = \mathrm{dom}(R)$

Results

- 2EXPTIME when R has **total domain** and is given as a DPA Holtmann/Kaiser/Thomas'10
- EXPTIME-c Klein/Zimmermann'14

Back to synthesis

Problem

Input: an automatic specification $R \subseteq \Sigma^\omega \times \Gamma^\omega$
Output: Is R uniformisable by some computable function f ?

$f \subseteq R$ and $\mathrm{dom}(f) = \mathrm{dom}(R)$

Results

- 2EXPTIME when R has **total domain** and is given as a DPA Holtmann/Kaiser/Thomas'10
- EXPTIME-c Klein/Zimmermann'14
- EXPTIME-c even when R has **partial domain** F., Winter,21

Partial vs Total Domain

- classical formulation of Church synthesis

$$\exists\lambda_{Eve} : \Sigma^* \to \Gamma \cdot \forall\sigma_1\sigma_2\dots \in \Sigma^\omega \cdot (\sigma_1\sigma_2\dots, \lambda(\sigma_1)\lambda(\sigma_1\sigma_2\dots)) \in R$$

[1] also called good-enough synthesis in Almagor,Kupferman,20

Partial vs Total Domain

- classical formulation of Church synthesis

$$\exists\lambda_{Eve} : \Sigma^* \to \Gamma \cdot \forall\sigma_1\sigma_2\dots \in \Sigma^\omega \cdot (\sigma_1\sigma_2\dots, \lambda(\sigma_1)\lambda(\sigma_1\sigma_2\dots)) \in R$$

- consequently, if $dom(R) \neq \Sigma^\omega$, R is unrealizable

[1] also called good-enough synthesis in Almagor,Kupferman,20

Partial vs Total Domain

- classical formulation of Church synthesis

$$\exists \lambda_{Eve} : \Sigma^* \to \Gamma \cdot \forall \sigma_1 \sigma_2 \dots \in \Sigma^\omega \cdot (\sigma_1 \sigma_2 \dots, \lambda(\sigma_1)\lambda(\sigma_1\sigma_2 \dots)) \in R$$

- consequently, if $dom(R) \neq \Sigma^\omega$, R is unrealizable
- uniformization = synthesis under assumption[1]
- only asks that implementation and spec have the same domain ($dom(R) = dom(f)$)

$$\exists \lambda_{Eve} : \Sigma^* \to \Gamma \cdot \forall \sigma_1 \sigma_2 \dots \in \mathrm{dom}(R) \cdot (\sigma_1 \sigma_2 \dots, \lambda(\sigma_1)\lambda(\sigma_1\sigma_2 \dots)) \in R$$

[1] also called good-enough synthesis in Almagor,Kupferman,20

Partial vs Total Domain

- classical formulation of Church synthesis

$$\exists\lambda_{Eve} : \Sigma^* \to \Gamma \cdot \forall\sigma_1\sigma_2 \dots \in \Sigma^\omega \cdot (\sigma_1\sigma_2 \dots, \lambda(\sigma_1)\lambda(\sigma_1\sigma_2 \dots)) \in R$$

- consequently, if $dom(R) \neq \Sigma^\omega$, R is unrealizable
- uniformization = synthesis under assumption[1]
- only asks that implementation and spec have the same domain ($dom(R) = dom(f)$)

$$\exists\lambda_{Eve} : \Sigma^* \to \Gamma \cdot \forall\sigma_1\sigma_2 \dots \in \mathrm{dom}(R) \cdot (\sigma_1\sigma_2 \dots, \lambda(\sigma_1)\lambda(\sigma_1\sigma_2 \dots)) \in R$$

From partial to total domain

- $R \mapsto R_{tot} = R \cup \overline{\mathrm{dom}(R)} \times \Gamma^\omega$
- R is realizable under assumption $\mathrm{dom}(R)$ iff R_{tot} is realizable (in the classical sense)
- R is automatic iff R_{tot} is automatic

[1] also called good-enough synthesis in Almagor,Kupferman,20

Partial vs Total Domain

- the latter reduction fails if one asks realizability by a computable function instead of a strategy function
- it does not preserve the existence of a computable realizer

Partial vs Total Domain

- the latter reduction fails if one asks realizability by a computable function instead of a strategy function
- it does not preserve the existence of a computable realizer

Counter-example

- $R = f_\perp$:

$$\begin{array}{ccccccccccc} & \perp & \perp & a & \perp & \dots & \perp & b & \dots & \in \Box\Diamond(a \vee b) \\ \mapsto & a & a & a & b & \dots & b & b & \dots & \end{array}$$

- R is uniformizable by a computable function ($f_\perp$ **is** computable)
- R_{tot} is not

Transducer synthesis (total case)

Theorem (Holtmann/Kaiser/Thomas'10, Klein/Zimmermann'14)

*Let $R \in AUT$ with **total** domain.*
If R is uniformisable by a computable function, it is uniformisable by a deterministic (one-way) transducer.

Transducer synthesis (total case)

Theorem (Holtmann/Kaiser/Thomas'10, Klein/Zimmermann'14)

*Let $R \in AUT$ with **total** domain.*
If R is uniformisable by a computable function, it is uniformisable by a deterministic (one-way) transducer.

Example

- R_1: behaves as $f_\perp$ as long as there is no two consecutive $\perp$, otherwise output $\perp$ forever.

$$\perp \quad a \quad \perp \quad \perp \quad b \quad \dots \qquad \mapsto \quad a \quad a \quad \perp \quad \perp \quad \perp \quad \perp^\omega$$

Transducer synthesis (total case)

Theorem (Holtmann/Kaiser/Thomas'10, Klein/Zimmermann'14)

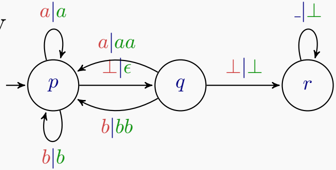
*Let $R \in AUT$ with **total** domain.*
If R is uniformisable by a computable function, it is uniformisable by a deterministic (one-way) transducer.

Example

- R_1: behaves as $f_\perp$ as long as there is no two consecutive $\perp$, otherwise output $\perp$ forever.

$$\perp \quad a \quad \perp \quad \perp \quad b \quad \ldots \quad \mapsto \quad a \quad a \quad \perp \quad \perp \quad \perp \quad \perp^\omega$$

- R_1 is a function of total domain
- it is computed by



Transducer synthesis (partial case)

- total setting: Eve only needs to wait a constant number of steps
- partial setting: she may need to wait arbitrarily long

Transducer synthesis (partial case)

- total setting: Eve only needs to wait a constant number of steps
- partial setting: she may need to wait arbitrarily long

Theorem (F., Winter)

*Let $R \in AUT$ with **partial** domain.*
*If R is uniformisable by a computable function, it is uniformisable by a deterministic **two**-way transducer.*

Two-wayness is necessary:

Uniformisation by computable functions: proof idea

- reduction to a turn-based two-player parity game where
- Adam plays input letters
- Eve plays output letters or a waiting action

Uniformisation by computable functions: proof idea

- reduction to a turn-based two-player parity game where
- Adam plays input letters
- Eve plays output letters or a waiting action

Problem

Uniformisation by computable functions: proof idea

- reduction to a turn-based two-player parity game where
- Adam plays input letters
- Eve plays output letters or a waiting action

Problem

- Eve might need to wait an arbitrary amount of time

Uniformisation by computable functions: proof idea

- reduction to a turn-based two-player parity game where
- Adam plays input letters
- Eve plays output letters or a waiting action

Problem

- Eve might need to wait an arbitrary amount of time
- We want a finite game arena, cannot store Adam's input

Uniformisation by computable functions: proof idea

- reduction to a turn-based two-player parity game where
- Adam plays input letters
- Eve plays output letters or a waiting action

Problem

- Eve might need to wait an arbitrary amount of time
- We want a finite game arena, cannot store Adam's input

Solution

Uniformisation by computable functions: proof idea

- reduction to a turn-based two-player parity game where
- Adam plays input letters
- Eve plays output letters or a waiting action

Problem

- Eve might need to wait an arbitrary amount of time
- We want a finite game arena, cannot store Adam's input

Solution

- store state transformations of the specification automaton induced by Adam's inputs
- monitor membership to the domain
- Eve picks a state transformation instead of concrete outputs
- winning condition: if Adam's input is in the domain, then Eve infinitely often picks an accepting state transformation
- convert this into a parity condition

Summary of Part 1

- synthesis of computable functions from automatic specifications given by DPA is ExpTime-c
- without assuming that inputs comes from the domain: deterministic one-way transducer suffice
- with that assumption: two-way transducer are necessary and sufficient.

Summary of Part 1

- synthesis of computable functions from automatic specifications given by DPA is ExpTime-c
- without assuming that inputs comes from the domain: deterministic one-way transducer suffice
- with that assumption: two-way transducer are necessary and sufficient.

Can we go beyond automatic specifications ?

Beyond Automatic Specifications

Automata model for asynchronous spec / impl

- Non-automatic spec: alphabet $\Sigma = \{0, +, -\}$, $\Gamma = \{0, \#\}$.

$$R \ : \ +0^{n_1} - 0^{n_2} + 0^{n_3} \ldots \mapsto \#0^{n_1+1}\#0^{n_2-1}\#0^{n_3+1} \ldots$$

Automata model for asynchronous spec / impl

- Non-automatic spec: alphabet $\Sigma = \{0, +, -\}$, $\Gamma = \{0, \#\}$.

  $$R \ : \ +0^{n_1} - 0^{n_2} + 0^{n_3} \ldots \mapsto \#0^{n_1+1}\#0^{n_2-1}\#0^{n_3+1} \ldots$$
- The language $\{i_1 o_1 \cdots \mid (i_1 \ldots, o_1 \ldots) \in R\}$ is not regular.

Automata model for asynchronous spec / impl

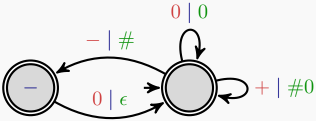
- Non-automatic spec: alphabet $\Sigma = \{0, +, -\}$, $\Gamma = \{0, \#\}$.

  $$R : +0^{n_1} - 0^{n_2} + 0^{n_3} \ldots \mapsto \#0^{n_1+1}\#0^{n_2-1}\#0^{n_3+1} \ldots$$

- The language $\{i_1 o_1 \cdots \mid (i_1 \ldots, o_1 \ldots) \in R\}$ is not regular.

Transducers

- (input) deterministic, define sequential functions (SEQ):



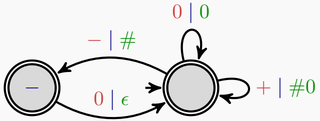
Automata model for asynchronous spec / impl

- Non-automatic spec: alphabet $\Sigma = \{0, +, -\}$, $\Gamma = \{0, \#\}$.

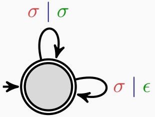
  $R : +0^{n_1} - 0^{n_2} + 0^{n_3} \ldots \mapsto \#0^{n_1+1}\#0^{n_2-1}\#0^{n_3+1} \ldots$
- The language $\{i_1 o_1 \cdots \mid (i_1 \ldots, o_1 \ldots) \in R\}$ is not regular.

Transducers

- (input) deterministic, define sequential functions (SEQ):



- non-deterministic, define rational relations (RAT):



Undecidability Result

Theorem (Löding,Carayol, 14)

The uniformization problem of rational relations by sequential functions is undecidable.

Proof Overview

Post-correspondence problem (PCP)

Given $(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n) \in \{0,1\}^* \times \{0,1\}^*$, find indices $i_1, \ldots, i_k$ such that

$$u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}.$$

Proof Overview

$\boxed{\text{PCP:} \exists i_1, \ldots, i_k \cdot u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}?}$

Proof Overview

$\boxed{\text{PCP:}\exists i_1, \ldots, i_k \cdot u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}?}$

Reduction: $\Sigma = \{1, \ldots, n, \#, A, B\}$, $\Gamma = \{0, 1, \#, A, B\}$.

Construct R such that:

$$i_1 \ldots i_k \# \alpha \mapsto \begin{cases} u_{i_1} \ldots u_{i_k} \#^\omega & \text{if } |u|_a = \infty \\ v\#^\omega,\ v \in \{0,1\}^* \setminus \{v_{i_1} \ldots v_{i_k}\} & \text{otherwise.} \end{cases}$$

Proof Overview

PCP: $\exists i_1, \ldots, i_k \cdot u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$?

Reduction: $\Sigma = \{1, \ldots, n, \#, A, B\}$, $\Gamma = \{0, 1, \#, A, B\}$.

Construct R such that:

$$i_1 \ldots i_k \# \alpha \mapsto \begin{cases} u_{i_1} \ldots u_{i_k} \#^\omega & \text{if } |u|_a = \infty \\ v\#^\omega,\ v \in \{0,1\}^* \setminus \{v_{i_1} \ldots v_{i_k}\} & \text{otherwise.} \end{cases}$$

Correctness

- no PCP solution $\Rightarrow$ always output $u_{i_1} \ldots u_{i_j} \#^\omega$

Proof Overview

PCP:$\exists i_1, \ldots, i_k \cdot u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$?

Reduction: $\Sigma = \{1, \ldots, n, \#, A, B\}$, $\Gamma = \{0, 1, \#, A, B\}$.

Construct R such that:

$$i_1 \ldots i_k \# \alpha \mapsto \begin{cases} u_{i_1} \ldots u_{i_k} \#^\omega & \text{if } |u|_a = \infty \\ v\#^\omega,\ v \in \{0,1\}^* \setminus \{v_{i_1} \ldots v_{i_k}\} & \text{otherwise.} \end{cases}$$

Correctness

- no PCP solution $\Rightarrow$ always output $u_{i_1} \ldots u_{i_j} \#^\omega$
- $i_1, \ldots, i_k$ is a PCP solution: reading $i_1 \ldots i_k \#$, any uniformiser can decide what to output w/o reading the infinite suffix

Asynchronous specifications: hopeless ?

- what about computable functions ?

[2] $\exists k \forall u \in \Sigma^\omega |\{v \mid (u,v) \in R\}| \leq k$

Asynchronous specifications: hopeless ?

- what about computable functions ? still undecidable

[2] $\exists k \forall u \in \Sigma^\omega |\{v \mid (u, v) \in R\}| \leq k$

Asynchronous specifications: hopeless ?

- what about computable functions ? still undecidable
- what about finite words and sequential functions ?

[2] $\exists k \forall u \in \Sigma^\omega |\{v \mid (u,v) \in R\}| \leq k$

Asynchronous specifications: hopeless ?

- what about computable functions ? still undecidable
- what about finite words and sequential functions ? still undecidable

[2] $\exists k \forall u \in \Sigma^\omega |\{v \mid (u,v) \in R\}| \leq k$

Asynchronous specifications: hopeless ?

- what about computable functions ? still undecidable
- what about finite words and sequential functions ? still undecidable
- decidability recovered for finite-valued rational relations of finite words[F.,Jecker,Löding,Winter,16] [2]
- and for stronger inclusion notions $f \subseteq R$ [F.,Jecker,Löding,Winter,16]

[2] $\exists k \forall u \in \Sigma^\omega |\{v \mid (u,v) \in R\}| \leq k$

A logic for asynchronous specifications

- so far, specifications are given by automata or transducers
- what about a logic ?

A logic for asynchronous specifications

- so far, specifications are given by automata or transducers
- what about a logic ?
- for automatic specifications: LTL, MSO, well-known

Goal

Define a logic which is

- expressive enough to capture a large class of asynchronous specifications
- has decidable satisfiability problem
- have decidable model-checking problem for known transducer models
- have some decidable synthesis problems

A logic for asynchronous specifications

- so far, specifications are given by automata or transducers
- what about a logic ?
- for automatic specifications: LTL, MSO, well-known

Goal

Define a logic which is

- expressive enough to capture a large class of asynchronous specifications
- has decidable satisfiability problem
- have decidable model-checking problem for known transducer models
- have some decidable synthesis problems

We introduce a logic to define finite word relations.

Motivation: Model-checking and Synthesis

- model-checking automata: $A \models \varphi$ if $L_A \subseteq L_\varphi$

Motivation: Model-checking and Synthesis

- model-checking automata: $A \models \varphi$ if $L_A \subseteq L_\varphi$
- transducers define relations in $\Sigma^* \times \Gamma^*$
- likewise, properties φ of transducers are relations

$$T \models \varphi \text{ if } R_T \subseteq R_\varphi$$

Motivation: Model-checking and Synthesis

- model-checking automata: $A \models \varphi$ if $L_A \subseteq L_\varphi$
- transducers define relations in $\Sigma^* \times \Gamma^*$
- likewise, properties φ of transducers are relations

$$T \models \varphi \text{ if } R_T \subseteq R_\varphi$$

Some Examples

True $\Sigma^* \times \Gamma^*$

Motivation: Model-checking and Synthesis

- model-checking automata: $A \models \varphi$ if $L_A \subseteq L_\varphi$
- transducers define relations in $\Sigma^* \times \Gamma^*$
- likewise, properties φ of transducers are relations

$$T \models \varphi \text{ if } R_T \subseteq R_\varphi$$

Some Examples

True	$\Sigma^* \times \Gamma^*$
There is an a in the output	$\Sigma^* \times \Gamma^* a \Gamma^*$

Motivation: Model-checking and Synthesis

- model-checking automata: $A \models \varphi$ if $L_A \subseteq L_\varphi$
- transducers define relations in $\Sigma^* \times \Gamma^*$
- likewise, properties φ of transducers are relations

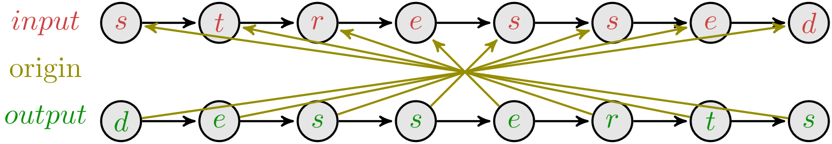
$$T \models \varphi \text{ if } R_T \subseteq R_\varphi$$

Some Examples

True	$\Sigma^* \times \Gamma^*$
There is an a in the output	$\Sigma^* \times \Gamma^* a \Gamma^*$
Every task is scheduled exactly once	$\{(t_1 \dots t_n, t_{\pi(1)} \dots t_{\pi(n)}) \mid \pi$ is a permutation $\}$

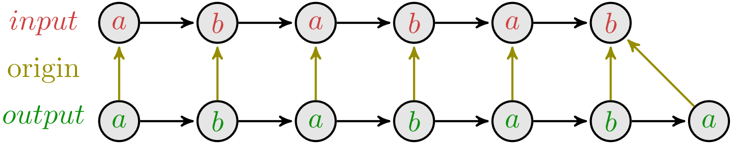
MSO over Origin Graphs (MSO$_o$)

- see pairs (u, v) as structures with origin information (origin graphs)



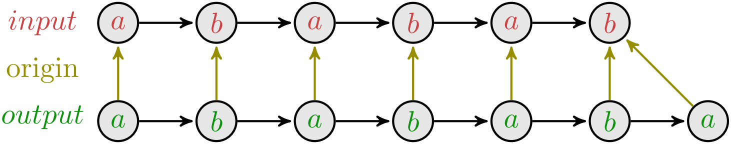
MSO over Origin Graphs (MSO$_o$)

- see pairs (u, v) as structures with origin information (origin graphs)



MSO over Origin Graphs (MSO$_o$)

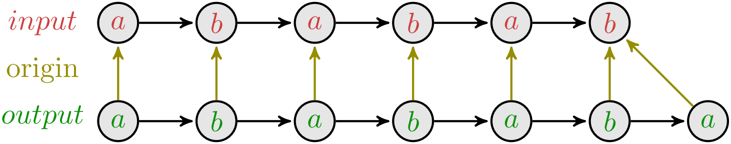
- see pairs (u, v) as structures with origin information (origin graphs)



- MSO$_o$:= MSO$[\leq_{in}, \leq_{out}, o]$

MSO over Origin Graphs (MSO_o)

- see pairs (u, v) as structures with origin information (origin graphs)



- $\text{MSO}_o := \text{MSO}[\leq_{in}, \leq_{out}, o]$
- a formula φ defines a set of origin graphs $\{g \mid g \models \varphi\}$
- and hence a relation (by projecting away the origin)

Examples

- True

$$\top$$

Examples

- True

$$\top$$

- There is an a in the output

$$\exists x \; x \leq_{out} x \wedge a(x)$$

Examples

- True

$$\top$$

- There is an a in the output

$$\exists^{out} x \; a(x)$$

Examples

- True

$$\top$$

- There is an a in the output

$$\exists^{out} x \ a(x)$$

- Every task is scheduled exactly once

$$\phi_s =_{def} bijection(o) \wedge \bigwedge_{\sigma \in \Sigma} \forall^{out} x \ \sigma(x) \rightarrow \sigma(o(x))$$

Examples

- True

$$\top$$

- There is an a in the output

$$\exists^{out} x\ a(x)$$

- Every task is scheduled exactly once

$$\phi_s =_{def} bijection(o) \wedge \bigwedge_{\sigma \in \Sigma} \forall^{out} x\ \sigma(x) \rightarrow \sigma(o(x))$$

- Mirror

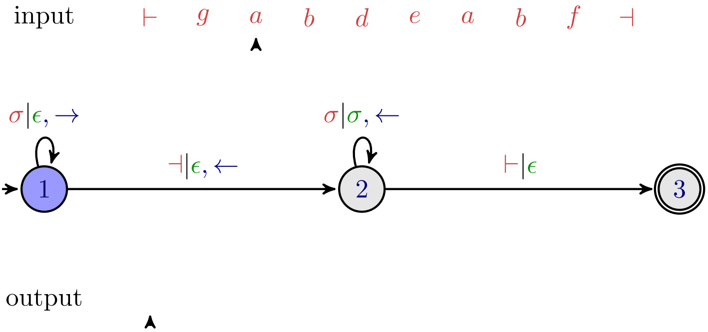
$$\phi_s \wedge \forall^{out} x, y\ x \leq_{out} y \rightarrow o(y) \leq_{in} o(z)$$

Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$

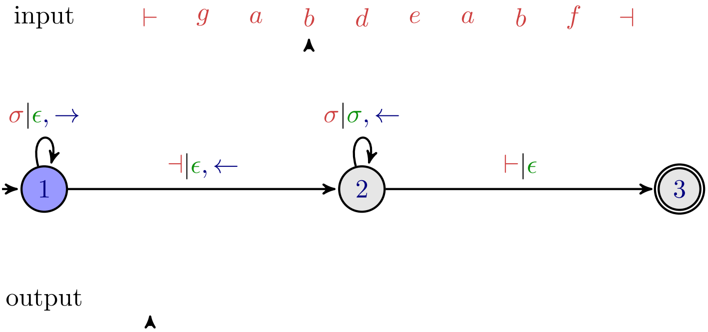
Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



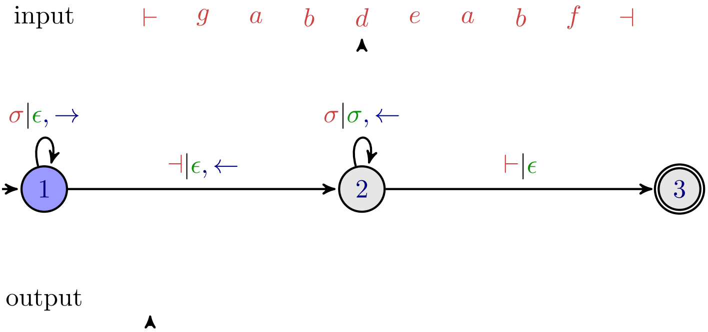
Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



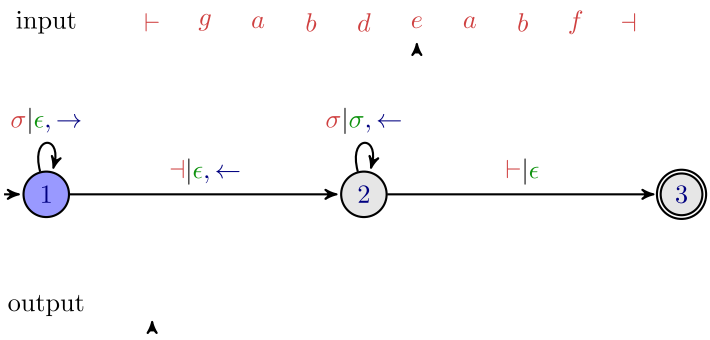
Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



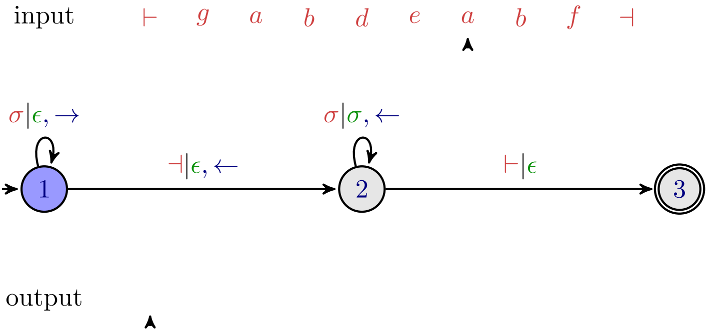
Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



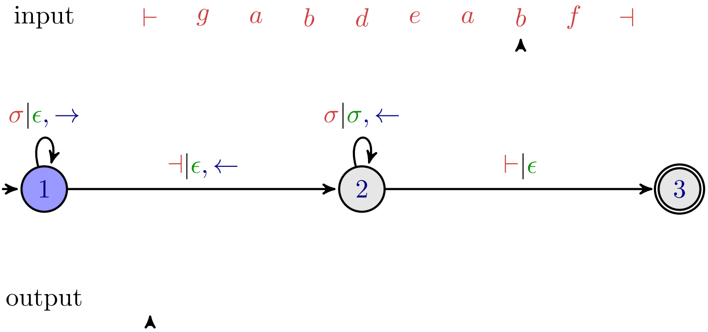
Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



Origin information

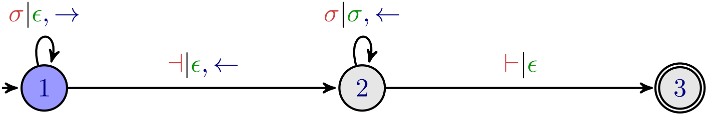
Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$

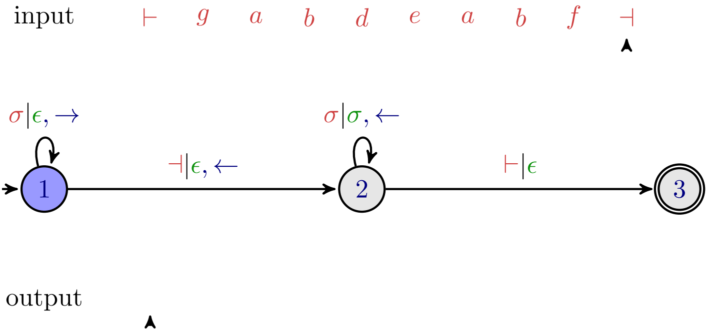
input $\vdash$ g a b d e a b f $\dashv$



output

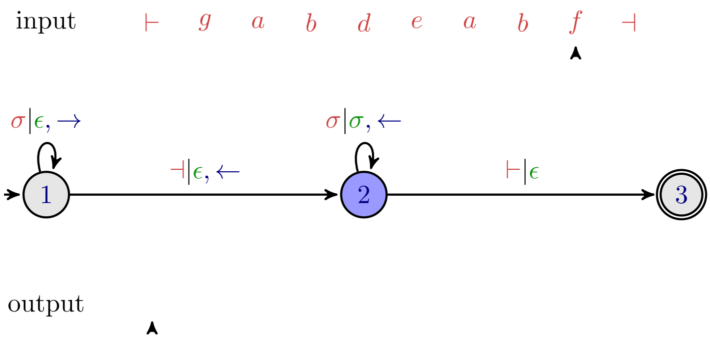
Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



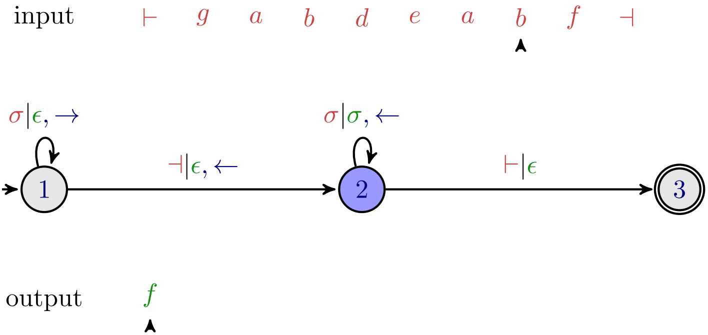
Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



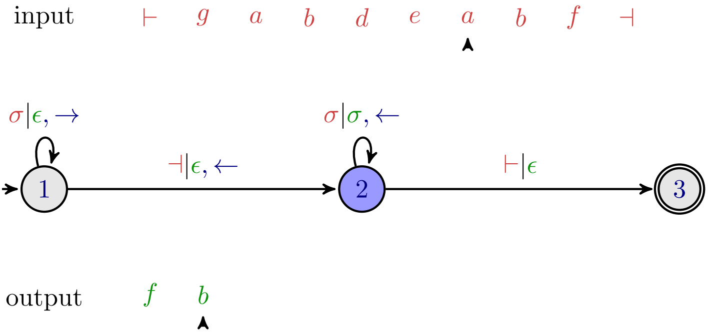
Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



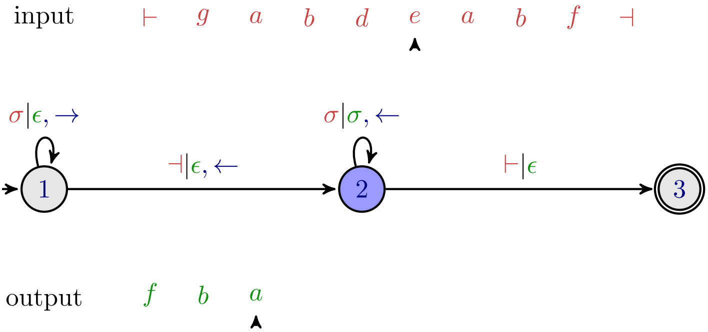
Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



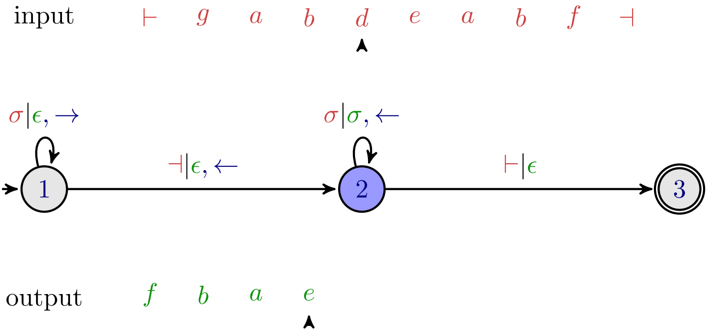
Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



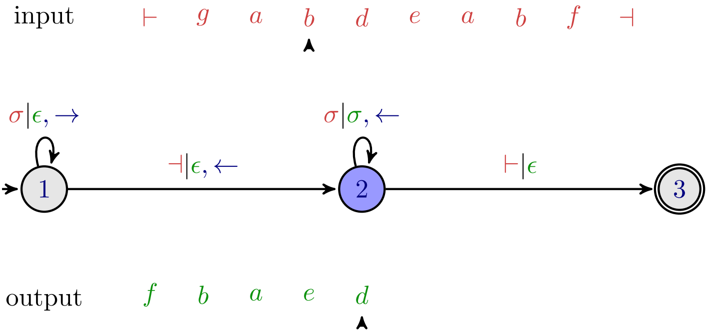
Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



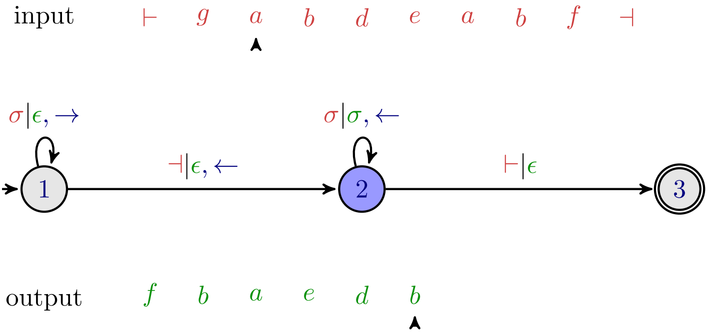
Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



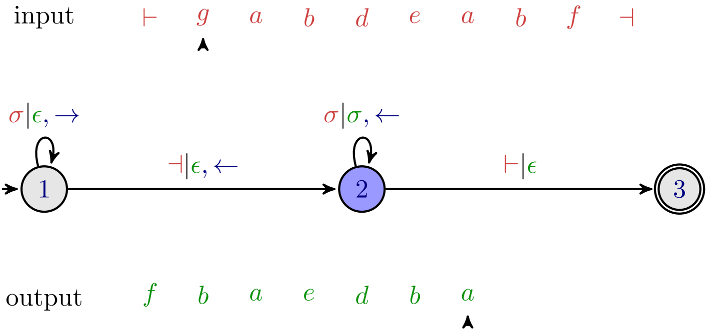
Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



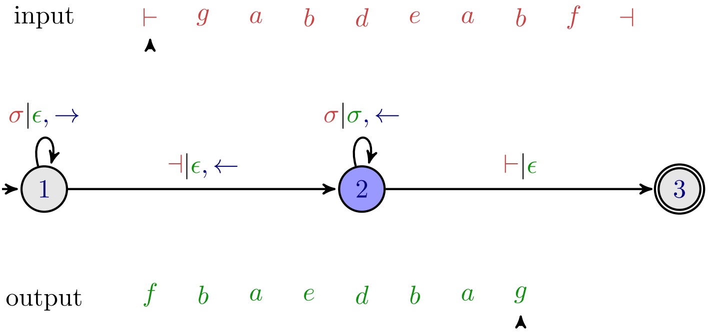
Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



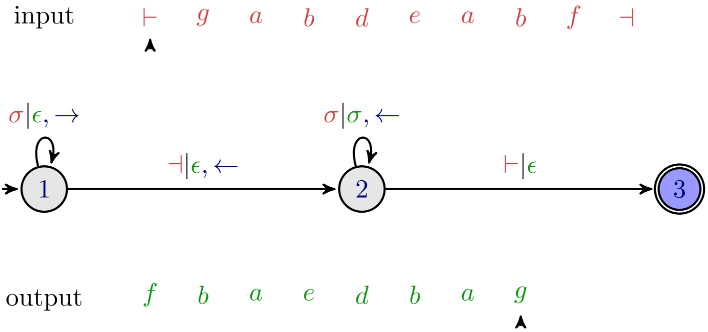
Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$



Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$

Other examples of two-way transductions

- doubling function $aabbc \mapsto aabbcaabbc$
- local mirror: $ab\#baa\#bcc \mapsto ba\#aab\#ccb$

Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$

Other examples of two-way transductions

- doubling function $aabbc \mapsto aabbcaabbc$
- local mirror: $ab\#baa\#bcc \mapsto ba\#aab\#ccb$

The class of functions defined by det. 2-way transducers

- closed under composition Chytil, Jákl, 77
- decidable equivalence problem Gurari 82

Origin information

Mikolaj Bojańczyk's Observation (14) Most transducer models T define a set of origin graphs $ographs(T)$

Other examples of two-way transductions

- doubling function $aabbc \mapsto aabbcaabbc$
- local mirror: $ab\#baa\#bcc \mapsto ba\#aab\#ccb$

The class of functions defined by det. 2-way transducers

- closed under composition Chytil, Jákl, 77
- decidable equivalence problem Gurari 82

Many Characterisations

- reversible 2-way transducers Dartois, Fournier, Jecker, Lhote, 17
- deterministic 1-way transducers with registers Alur, Cerny, 09
- MSO-transductions on strings Engelfriet,Hoogeboom,01
- regular combinators (Alur, Freilich, Raghothaman, 14) (Dave, Gastin,

Results

Results

Theorem (Bojanczyk,Daviaud,Guillon,Penelle,17)

The model-checking problem $ographs(T) \models \varphi$ *for* T *deterministic 2-way trans and* $\varphi \in MSO_o$ *is decidable.*

Results

Theorem (Bojanczyk,Daviaud,Guillon,Penelle,17)

The model-checking problem $ographs(T) \models \varphi$ *for* T *deterministic 2-way trans and* $\varphi \in MSO_o$ *is decidable.*

Theorem

The satisfiability problem ($\exists G \cdot G \models \varphi$?) *is undecidable.*

Results

Theorem (Bojanczyk,Daviaud,Guillon,Penelle,17)

The model-checking problem $ographs(T) \models \varphi$ for T deterministic 2-way trans and $\varphi \in MSO_o$ is decidable.

Theorem

The satisfiability problem ($\exists G \cdot G \models \varphi$?) is undecidable.

Contribution Dartois,F.,Lhote,18

A fragment $\mathcal{F} = FO_2[\leq_{out}, MSO_{bin}[\leq_{in}]]$such that:

1. any $\mathcal{F}$-defined relation is uniformisable by a det. 2-way transducer
2. satisfiability is decidable
3. it is expressive (captures regular functions and more)
4. implies decidability of an expressive logic for data words: $FO_2[\leq_{pos}, MSO_{bin}[\leq_{data}]]$

Summary

Summary

beyond Mealy machines

- synthesis of computable functions from automatic spec is decidable
- if Adam can input anything: one-way transducer suffice
- if Adam can play only in the domain of the spec: two-way transducers are necessary

Summary

beyond Mealy machines

- synthesis of computable functions from automatic spec is decidable
- if Adam can input anything: one-way transducer suffice
- if Adam can play only in the domain of the spec: two-way transducers are necessary
- what about synthesis of functions computable with bounded-memory ?

Summary

beyond Mealy machines

- synthesis of computable functions from automatic spec is decidable
- if Adam can input anything: one-way transducer suffice
- if Adam can play only in the domain of the spec: two-way transducers are necessary
- what about synthesis of functions computable with bounded-memory ?

beyond automatic specifications

- decidability of synthesis is lost (for asynchronous spec defined by non-deterministic transducers)
- recovered in some cases (finite-valued relations, ...)
- over finite words: a logic to specify binary relations

Summary

beyond Mealy machines

- synthesis of computable functions from automatic spec is decidable
- if Adam can input anything: one-way transducer suffice
- if Adam can play only in the domain of the spec: two-way transducers are necessary
- what about synthesis of functions computable with bounded-memory ?

beyond automatic specifications

- decidability of synthesis is lost (for asynchronous spec defined by non-deterministic transducers)
- recovered in some cases (finite-valued relations, ...)
- over finite words: a logic to specify binary relations

Conclusion

- many generalizations of the classical reactive synthesis problem: quantitative specifications, pushdown, multiplayers, rational synthesis, data words ...
- this work generalizes the class of implementations by relaxing the reactivity requirement
- relevant when a spec is not realizable in a reactive manner
- this relaxation could be studied in other synthesis settings
- for instance: synthesis of computable functions from synchronous specifications given as weighted automata ?

Conclusion

- many generalizations of the classical reactive synthesis problem: quantitative specifications, pushdown, multiplayers, rational synthesis, data words ...
- this work generalizes the class of implementations by relaxing the reactivity requirement
- relevant when a spec is not realizable in a reactive manner
- this relaxation could be studied in other synthesis settings
- for instance: synthesis of computable functions from synchronous specifications given as weighted automata ?
- other interesting question: synthesis under assumptions (regular, quantitative, etc.)