Synthesizing Computable Functions from Synchronous Specifications

Sarah Winter

Université libre de Bruxelles, Belgium

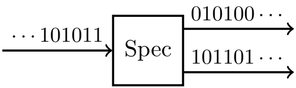
January 6, 2021
YR-OWLS, online

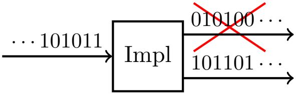
Reactive Synthesis of Non-terminating Systems

$$\textbf{Specification} \overset{\text{synthesize}}{\dashrightarrow} \textbf{Implementation}$$

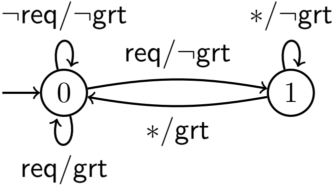
one input is in relation
with several outputs

algorithm that selects
a unique output for each input



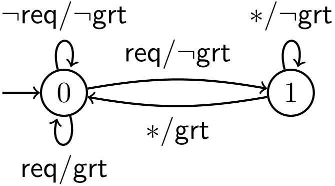


Church Synthesis

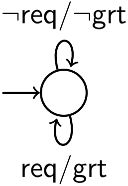


Synchronous specifications
(**synchronous relations**)
e.g, given by
synchronous transducers with
parity acceptance

Church Synthesis

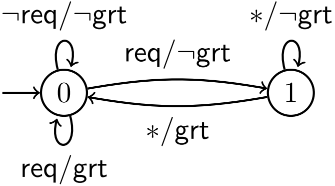


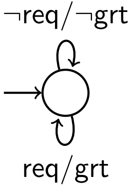
Synchronous specifications
(**synchronous relations**)
e.g, given by
synchronous transducers with
parity acceptance



Synchronous implementations
given by
Mealy machines

Church Synthesis





Synchronous specifications
(**synchronous relations**)
e.g, given by
synchronous transducers with
parity acceptance

Synchronous implementations
given by
Mealy machines

Theorem (Büchi/Landweber'69). It is decidable whether a synchronous specification is implementable by a Mealy machine.

More Relaxed Implementations

Goal Decide whether a synchronous specification is implementable (by an algorithm/a program/a deterministic Turing machine).

More Relaxed Implementations

Goal Decide whether a synchronous specification is implementable (by an algorithm/a program/a deterministic Turing machine).

Example.

- Specification: contains pairs of the form

$$(a_1 a_2 a_3 \cdots, a_3 \cdots) \in \{a, b\}^\omega \times \{a, b\}^\omega$$

More Relaxed Implementations

Goal Decide whether a synchronous specification is implementable (by an algorithm/a program/a deterministic Turing machine).

Example.

- Specification: contains pairs of the form

$$(a_1 a_2 a_3 \cdots, a_3 \cdots) \in \{a, b\}^\omega \times \{a, b\}^\omega$$

- no implementation by a Mealy machine exists,

More Relaxed Implementations

Goal Decide whether a synchronous specification is implementable (by an algorithm/a program/a deterministic Turing machine).

Example.

- Specification: contains pairs of the form

$$(a_1 a_2 a_3 \cdots, a_3 \cdots) \in \{a,b\}^\omega \times \{a,b\}^\omega$$

- no implementation by a Mealy machine exists,
- can be implemented, every deterministic machine has to wait until it sees the third input letter

More Relaxed Implementations

Example.

- Specification: contains pairs of the form

$$(uA\alpha, A^{|u|}\beta) \quad (uB\alpha, B^{|u|}\beta),$$

where $u \in \{a,b\}^*, \alpha, \beta \in \{a,b\}^\omega$, A, B are special letters

More Relaxed Implementations

Example.

- Specification: contains pairs of the form

$$(uA\alpha, A^{|u|}\beta) \quad (uB\alpha, B^{|u|}\beta),$$

  where $u \in \{a,b\}^*, \alpha, \beta \in \{a,b\}^\omega$, A, B are special letters
- can be implemented, but, every deterministic machine has to wait arbitrary long to output something valid

More Relaxed Implementations

Example.

- Specification: contains pairs of the form

$$(uA\alpha, A^{|u|}\beta) \quad (uB\alpha, B^{|u|}\beta),$$

  where $u \in \{a,b\}^*, \alpha, \beta \in \{a,b\}^\omega$, A, B are special letters
- can be implemented, but, every deterministic machine has to wait arbitrary long to output something valid
- e.g., implemented by a deterministic machine that computes the function

$$uA\alpha \mapsto A^{|u|}\alpha \quad uB\alpha \mapsto B^{|u|}\alpha$$

Computability

What does it mean to be **implementable** for a relation?

Computability

What does it mean to be **implementable** for a relation?

- There is a computable function f with the same domain as the relation R such that $(\alpha, f(\alpha)) \in R$ for all $\alpha \in \mathrm{dom}(R)$.

Computability

What does it mean to be **implementable** for a relation?

- There is a computable function f with the same domain as the relation R such that $(\alpha, f(\alpha)) \in R$ for all $\alpha \in \mathrm{dom}(R)$.

A function $f \colon \Sigma^\omega \rightharpoonup \Gamma^\omega$ is **computable** if there exists a deterministic Turing machine that

- outputs longer and longer prefixes of an acceptable output
- while it reads longer and longer prefixes of the input.

Computability

Consider a deterministic Turing machine M with

Computability

Consider a deterministic Turing machine M with

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

Computability

Consider a deterministic Turing machine M with

- three tapes
  - a one-way read-only input tape
  - a two-way working tape
  - a one-way write-only output tape
- $M(\alpha, k)$ denotes the output written after reading the first k letters of the input sequence α

Computability

Consider a deterministic Turing machine M with

- three tapes
  - a one-way read-only input tape
  - a two-way working tape
  - a one-way write-only output tape
- $M(\alpha, k)$ denotes the output written after reading the first k letters of the input sequence α

M **computes** f if for all $\alpha \in \mathrm{dom}(f)$:

- $\forall k$: $M(\alpha, k)$ is a prefix of $f(\alpha)$, and
- $\forall i \; \exists j$: $|M(\alpha, j)| \geq i$

Computability and Continuity

Computability and Continuity

A function $f\colon \Sigma^\omega \rightharpoonup \Gamma^\omega$ is **continuous** at $\alpha \in \mathrm{dom}(f)$ if

- $\forall i\ \exists j\ \forall \beta \in \mathrm{dom}(f)$: $|\alpha \wedge \beta| \geq j$ implies $|f(\alpha) \wedge f(\beta)| \geq i$.

Computability and Continuity

A function $f\colon \Sigma^\omega \rightharpoonup \Gamma^\omega$ is **continuous** at $\alpha \in \mathrm{dom}(f)$ if

- $\forall i\ \exists j\ \forall \beta \in \mathrm{dom}(f)$: $|\alpha \wedge \beta| \geq j$ implies $|f(\alpha) \wedge f(\beta)| \geq i$.

f is **continuous** if it is continuous at every $\alpha \in \mathrm{dom}(f)$.

Computability and Continuity

A function $f\colon \Sigma^\omega \rightharpoonup \Gamma^\omega$ is **continuous** at $\alpha \in \mathrm{dom}(f)$ if

- $\forall i\ \exists j\ \forall \beta \in \mathrm{dom}(f)$: $|\alpha \wedge \beta| \geq j$ implies $|f(\alpha) \wedge f(\beta)| \geq i$.

f is **continuous** if it is continuous at every $\alpha \in \mathrm{dom}(f)$.

Examples.

- $f_1\colon uA\alpha \mapsto A^{|u|}\alpha \quad uB\alpha \mapsto B^{|u|}\alpha$,
  for all $u \in \{a,b\}^*, \alpha \in \{a,b\}^\omega$ is continuous

Computability and Continuity

A function $f\colon \Sigma^\omega \rightharpoonup \Gamma^\omega$ is **continuous** at $\alpha \in \mathrm{dom}(f)$ if

- $\forall i\ \exists j\ \forall \beta \in \mathrm{dom}(f)$: $|\alpha \wedge \beta| \geq j$ implies $|f(\alpha) \wedge f(\beta)| \geq i$.

f is **continuous** if it is continuous at every $\alpha \in \mathrm{dom}(f)$.

Examples.

- $f_1\colon uA\alpha \mapsto A^{|u|}\alpha \quad uB\alpha \mapsto B^{|u|}\alpha$,
  for all $u \in \{a,b\}^*, \alpha \in \{a,b\}^\omega$ is continuous
- $f_2\colon \alpha \mapsto \begin{cases} a^\omega & \text{if } \alpha \text{ contains } \infty \text{ many } a \\ b^\omega & \text{otherwise} \end{cases}$
  for all $\alpha \in \{a,b\}^\omega$ is not continuous

Computability and Continuity

A function $f\colon \Sigma^\omega \rightharpoonup \Gamma^\omega$ is **continuous** at $\alpha \in \mathrm{dom}(f)$ if

- $\forall i\ \exists j\ \forall \beta \in \mathrm{dom}(f)$: $|\alpha \wedge \beta| \geq j$ implies $|f(\alpha) \wedge f(\beta)| \geq i$.

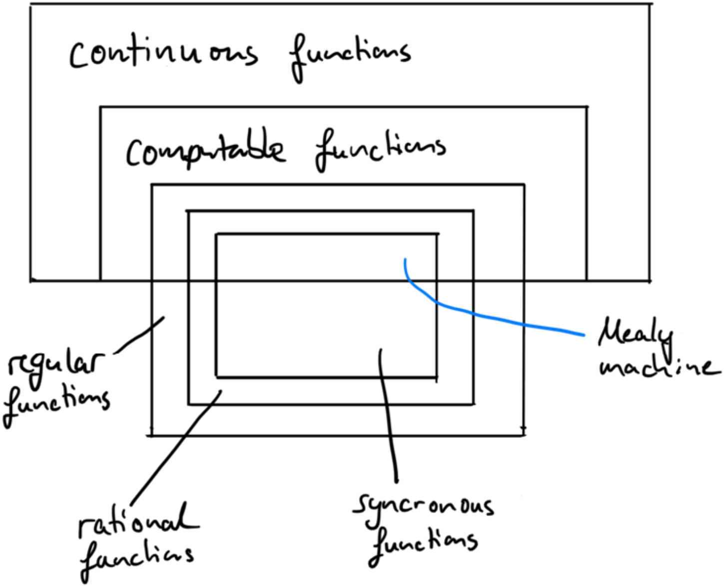
f is **continuous** if it is continuous at every $\alpha \in \mathrm{dom}(f)$.

Examples.

- $f_1\colon uA\alpha \mapsto A^{|u|}\alpha \quad uB\alpha \mapsto B^{|u|}\alpha$,
  for all $u \in \{a,b\}^*, \alpha \in \{a,b\}^\omega$ is continuous
- $f_2\colon \alpha \mapsto \begin{cases} a^\omega & \text{if } \alpha \text{ contains } \infty \text{ many } a \\ b^\omega & \text{otherwise} \end{cases}$
  for all $\alpha \in \{a,b\}^\omega$ is not continuous

- If $f\colon \Sigma^\omega \rightharpoonup \Gamma^\omega$ is computable, then it is continuous,
- the converse does not hold.

Computability and Continuity



Total vs. Partial Domain

Total vs. Partial Domain

- In synthesis, often a total specification domain is assumed, else the synthesis task fails by design

Total vs. Partial Domain

- In synthesis, often a total specification domain is assumed, else the synthesis task fails by design
- Here: We allow partial domain

Total vs. Partial Domain

- In synthesis, often a total specification domain is assumed, else the synthesis task fails by design
- Here: We allow partial domain

Example.

- Specification: contains pairs of the form

$$(uA\alpha, A^{|u|}\beta) \quad (uB\alpha, B^{|u|}\beta),$$

  where $u \in \{a,b\}^*, \alpha, \beta \in \{a,b\}^\omega$, A, B are special letters
- has partial domain $\{a,b\}^*\{A,B\}\{a,b\}^\omega$
- e.g., implemented by a deterministic machine that computes the function $uA\alpha \mapsto A^{|u|}\alpha \quad uB\alpha \mapsto B^{|u|}\alpha$

Total vs. Partial Domain

- In synthesis, often a total specification domain is assumed, else the synthesis task fails by design
- Here: We allow partial domain

Example.

- Specification: contains pairs of the form

$$(uA\alpha, A^{|u|}\beta) \quad (uB\alpha, B^{|u|}\beta),$$

  where $u \in \{a,b\}^*, \alpha, \beta \in \{a,b\}^\omega$, A, B are special letters
- has partial domain $\{a,b\}^*\{A,B\}\{a,b\}^\omega$
- e.g., implemented by a deterministic machine that computes the function $uA\alpha \mapsto A^{|u|}\alpha \quad uB\alpha \mapsto B^{|u|}\alpha$

- There is no way to complete the domain and remain implementable!

Results for Total Domain

Results for Total Domain

Theorem (Holtmann/Kaiser/Thomas'10). It is decidable in 2EXPTIME whether a continuous function can be synthesized from a given synchronous relation with **total domain**.

Results for Total Domain

Theorem (Holtmann/Kaiser/Thomas'10). It is decidable in 2EXPTime whether a continuous function can be synthesized from a given synchronous relation with **total domain**.

Theorem (Klein/Zimmermann'14). It is EXPTime-complete to decide whether a continuous function can be synthesized from a given synchronous relation with **total domain**.

Results for Total Domain

Theorem (Holtmann/Kaiser/Thomas'10). It is decidable in 2EXPTIME whether a continuous function can be synthesized from a given synchronous relation with **total domain**.

Theorem (Klein/Zimmermann'14). It is EXPTIME-complete to decide whether a continuous function can be synthesized from a given synchronous relation with **total domain**.

Is the function computable?

Implementations for Total Domain

Implementations for Total Domain

Theorem (Holtmann/Kaiser/Thomas'10). Such a synthesized function is computable by a sequential transducer.

Implementations for Total Domain

Theorem (Holtmann/Kaiser/Thomas'10)**.** Such a synthesized function is computable by a sequential transducer.

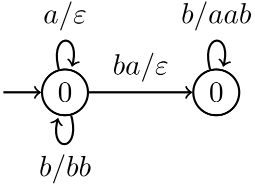
A transducer is **sequential** if its underlying input automaton is a DFA.

Implementations for Total Domain

Theorem (Holtmann/Kaiser/Thomas'10). Such a synthesized function is computable by a sequential transducer.

A transducer is **sequential** if its underlying input automaton is a DFA.

Example.



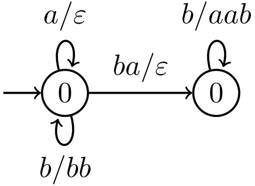
(asynchronous) transducer

Implementations for Total Domain

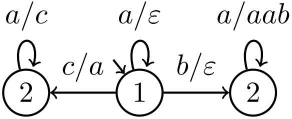
Theorem (Holtmann/Kaiser/Thomas'10). Such a synthesized function is computable by a sequential transducer.

A transducer is **sequential** if its underlying input automaton is a DFA.

Example.



(asynchronous) transducer



sequential transducer

Results for Partial Domain

Results for Partial Domain

Theorem (Filiot/W.). It is EXPTIME-complete to decide whether a continuous function can be synthesized from a given synchronous relation with **partial domain**. Such a synthesized function is computable.

Proof Idea

Proof Idea

Game view

- Adam plays input letters

Proof Idea

Game view

- Adam plays input letters
- Eve plays output letters

Proof Idea

Game view

- Adam plays input letters
- Eve plays output letters
- If the input sequence is in the specification domain, input + output sequence must be in relation wrt the specification

Proof Idea

Game view

- Adam plays input letters
- Eve plays output letters
- If the input sequence is in the specification domain, input + output sequence must be in relation wrt the specification

Problem

Proof Idea

Game view

- Adam plays input letters
- Eve plays output letters
- If the input sequence is in the specification domain, input + output sequence must be in relation wrt the specification

Problem

- Eve might need an unbounded lookahead on Adams moves

Proof Idea

Game view

- Adam plays input letters
- Eve plays output letters
- If the input sequence is in the specification domain, input + output sequence must be in relation wrt the specification

Problem

- Eve might need an unbounded lookahead on Adams moves
- We want a finite game arena, cannot store the lookahead explicitly

Proof Idea

Game view

- Adam plays input letters
- Eve plays output letters
- If the input sequence is in the specification domain, input + output sequence must be in relation wrt the specification

Problem

- Eve might need an unbounded lookahead on Adams moves
- We want a finite game arena, cannot store the lookahead explicitly

Solution

Proof Idea

Game view

- Adam plays input letters
- Eve plays output letters
- If the input sequence is in the specification domain, input + output sequence must be in relation wrt the specification

Problem

- Eve might need an unbounded lookahead on Adams moves
- We want a finite game arena, cannot store the lookahead explicitly

Solution

- Instead of an explicit lookahead, store a finite abstraction

Proof Idea

Proof Idea

Given a finite input word $u \in \Sigma^*$, its **profile** P_u stores all inducible state transformations wrt the specification automaton.

Proof Idea

Given a finite input word $u \in \Sigma^*$, its **profile** P_u stores all inducible state transformations wrt the specification automaton.

Game Idea

Proof Idea

Given a finite input word $u \in \Sigma^*$, its **profile** P_u stores all inducible state transformations wrt the specification automaton.

Game Idea

- Adam plays input letters, building lookahead profiles

Proof Idea

Given a finite input word $u \in \Sigma^*$, its **profile** P_u stores all inducible state transformations wrt the specification automaton.

Game Idea

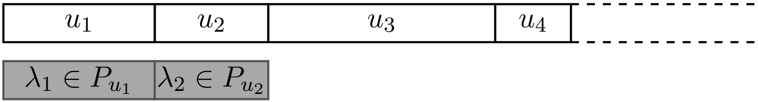
- Adam plays input letters, building lookahead profiles
- Eve can delay her her move, or chose a state transformation from a lookahead profile (instead of playing output letters)

Proof Idea

Given a finite input word $u \in \Sigma^*$, its **profile** P_u stores all inducible state transformations wrt the specification automaton.

Game Idea

- Adam plays input letters, building lookahead profiles
- Eve can delay her her move, or chose a state transformation from a lookahead profile (instead of playing output letters)

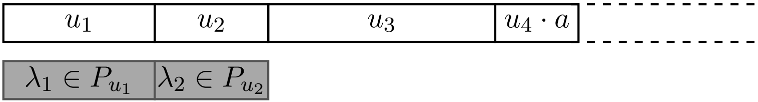


Proof Idea

Given a finite input word $u \in \Sigma^*$, its **profile** P_u stores all inducible state transformations wrt the specification automaton.

Game Idea

- Adam plays input letters, building lookahead profiles
- Eve can delay her her move, or chose a state transformation from a lookahead profile (instead of playing output letters)

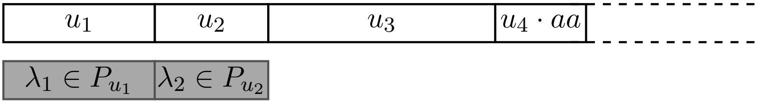


Proof Idea

Given a finite input word $u \in \Sigma^*$, its **profile** P_u stores all inducible state transformations wrt the specification automaton.

Game Idea

- Adam plays input letters, building lookahead profiles
- Eve can delay her her move, or chose a state transformation from a lookahead profile (instead of playing output letters)

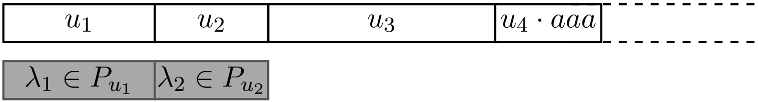


Proof Idea

Given a finite input word $u \in \Sigma^*$, its **profile** P_u stores all inducible state transformations wrt the specification automaton.

Game Idea

- Adam plays input letters, building lookahead profiles
- Eve can delay her her move, or chose a state transformation from a lookahead profile (instead of playing output letters)

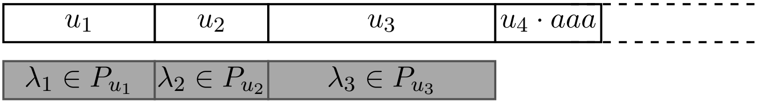


Proof Idea

Given a finite input word $u \in \Sigma^*$, its **profile** P_u stores all inducible state transformations wrt the specification automaton.

Game Idea

- Adam plays input letters, building lookahead profiles
- Eve can delay her her move, or chose a state transformation from a lookahead profile (instead of playing output letters)

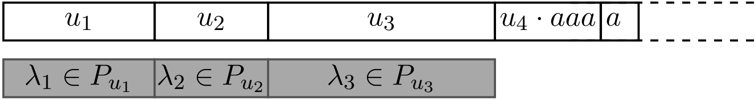


Proof Idea

Given a finite input word $u \in \Sigma^*$, its **profile** P_u stores all inducible state transformations wrt the specification automaton.

Game Idea

- Adam plays input letters, building lookahead profiles
- Eve can delay her her move, or chose a state transformation from a lookahead profile (instead of playing output letters)



Proof Idea

Given a finite input word $u \in \Sigma^*$, its **profile** P_u stores all inducible state transformations wrt the specification automaton.

Game Idea

- Adam plays input letters, building lookahead profiles
- Eve can delay her her move, or chose a state transformation from a lookahead profile (instead of playing output letters)

u_1	u_2	u_3	$u_4 \cdot aaa$	$a \cdots$
$\lambda_1 \in P_{u_1}$	$\lambda_2 \in P_{u_2}$	$\lambda_3 \in P_{u_3}$	$\lambda_4 \in \cdots$	

Proof Idea

Given a finite input word $u \in \Sigma^*$, its **profile** P_u stores all inducible state transformations wrt the specification automaton.

Game Idea

- Adam plays input letters, building lookahead profiles
- Eve can delay her her move, or chose a state transformation from a lookahead profile (instead of playing output letters)

u_1	u_2	u_3	$u_4 \cdot aaa$	$a \cdots$
$\lambda_1 \in P_{u_1}$	$\lambda_2 \in P_{u_2}$	$\lambda_3 \in P_{u_3}$	$\lambda_4 \in \cdots$	

Winning condition If Adam plays a valid input sequence,

Proof Idea

Given a finite input word $u \in \Sigma^*$, its **profile** P_u stores all inducible state transformations wrt the specification automaton.

Game Idea

- Adam plays input letters, building lookahead profiles
- Eve can delay her her move, or chose a state transformation from a lookahead profile (instead of playing output letters)

u_1	u_2	u_3	$u_4 \cdot aaa$	$a \cdots$
$\lambda_1 \in P_{u_1}$	$\lambda_2 \in P_{u_2}$	$\lambda_3 \in P_{u_3}$	$\lambda_4 \in \cdots$	

Winning condition If Adam plays a valid input sequence,

- Eves makes a move infinitely often,
- her moves describe an accepting run wrt the specification.

Implementations for Partial Domain

Implementations for Partial Domain

Theorem (Filiot/W.). If a synchronous relation with partial domain is implementable, then it can be implemented by a deterministic two-way transducer.

Implementations for Partial Domain

Theorem (Filiot/W.). If a synchronous relation with partial domain is implementable, then it can be implemented by a deterministic two-way transducer.

Example.

- Specification: contains pairs of the form

$$(uA\alpha, A^{|u|}\beta) \quad (uB\alpha, B^{|u|}\beta),$$

  where $u \in \{a,b\}^*, \alpha, \beta \in \{a,b\}^\omega$, A, B are special letters
- e.g., implemented by a deterministic two-way transducer that computes $uA\alpha \mapsto A^{|u|}\alpha \quad uB\alpha \mapsto B^{|u|}\alpha$

Implementations for Partial Domain

Theorem (Filiot/W.). If a synchronous relation with partial domain is implementable, then it can be implemented by a deterministic two-way transducer.

Example.

- Specification: contains pairs of the form

$$(uA\alpha, A^{|u|}\beta) \quad (uB\alpha, B^{|u|}\beta),$$

  where $u \in \{a,b\}^*, \alpha, \beta \in \{a,b\}^\omega$, A, B are special letters
- e.g., implemented by a deterministic two-way transducer that computes $uA\alpha \mapsto A^{|u|}\alpha \quad uB\alpha \mapsto B^{|u|}\alpha$
  - transducer goes right until A resp. B is read, no output

Implementations for Partial Domain

Theorem (Filiot/W.). If a synchronous relation with partial domain is implementable, then it can be implemented by a deterministic two-way transducer.

Example.

- Specification: contains pairs of the form

$$(uA\alpha, A^{|u|}\beta) \quad (uB\alpha, B^{|u|}\beta),$$

  where $u \in \{a,b\}^*, \alpha, \beta \in \{a,b\}^\omega$, A, B are special letters
- e.g., implemented by a deterministic two-way transducer that computes $uA\alpha \mapsto A^{|u|}\alpha \quad uB\alpha \mapsto B^{|u|}\alpha$
  - transducer goes right until A resp. B is read, no output
  - goes back left to the beginning, no output

Implementations for Partial Domain

Theorem (Filiot/W.). If a synchronous relation with partial domain is implementable, then it can be implemented by a deterministic two-way transducer.

Example.

- Specification: contains pairs of the form

  $$(uA\alpha, A^{|u|}\beta) \quad (uB\alpha, B^{|u|}\beta),$$

  where $u \in \{a,b\}^*, \alpha, \beta \in \{a,b\}^\omega$, A, B are special letters
- e.g., implemented by a deterministic two-way transducer that computes $uA\alpha \mapsto A^{|u|}\alpha \quad uB\alpha \mapsto B^{|u|}\alpha$
  - transducer goes right until A resp. B is read, no output
  - goes back left to the beginning, no output
  - goes right, outputs A resp. B for every letter until A resp. B is read,

Implementations for Partial Domain

Theorem (Filiot/W.). If a synchronous relation with partial domain is implementable, then it can be implemented by a deterministic two-way transducer.

Example.

- Specification: contains pairs of the form

$$(uA\alpha, A^{|u|}\beta) \quad (uB\alpha, B^{|u|}\beta),$$

  where $u \in \{a,b\}^*, \alpha, \beta \in \{a,b\}^\omega$, A, B are special letters
- e.g., implemented by a deterministic two-way transducer that computes $uA\alpha \mapsto A^{|u|}\alpha \quad uB\alpha \mapsto B^{|u|}\alpha$
  - transducer goes right until A resp. B is read, no output
  - goes back left to the beginning, no output
  - goes right, outputs A resp. B for every letter until A resp. B is read,
  - goes right and copies the input

Total vs. Partial Domain Implementations

Total vs. Partial Domain Implementations

Total domain

- Sequential transducers with bounded lookahead suffice

Total vs. Partial Domain Implementations

Total domain

- Sequential transducers with bounded lookahead suffice
- Intuitive reason for bounded lookahead

Total vs. Partial Domain Implementations

Total domain

- Sequential transducers with bounded lookahead suffice
- Intuitive reason for bounded lookahead
  - If an arbitrary long lookahead is needed to determine the next output,

Total vs. Partial Domain Implementations

Total domain

- Sequential transducers with bounded lookahead suffice
- Intuitive reason for bounded lookahead
  - If an arbitrary long lookahead is needed to determine the next output,
  - then a deterministic machine may wait forever to output something valid.

Total vs. Partial Domain Implementations

Total domain

- Sequential transducers with bounded lookahead suffice
- Intuitive reason for bounded lookahead
  - If an arbitrary long lookahead is needed to determine the next output,
  - then a deterministic machine may wait forever to output something valid.
  - Result: a finite output sequence, but the infinite input sequence is valid ↯

Total vs. Partial Domain Implementations

Total domain

- Sequential transducers with bounded lookahead suffice
- Intuitive reason for bounded lookahead
  - If an arbitrary long lookahead is needed to determine the next output,
  - then a deterministic machine may wait forever to output something valid.
  - Result: a finite output sequence, but the infinite input sequence is valid $\lightning$

Partial domain

- Deterministic two-way transducers suffice, sequential transducers do not

Total vs. Partial Domain Implementations

Total domain

- Sequential transducers with bounded lookahead suffice
- Intuitive reason for bounded lookahead
  - If an arbitrary long lookahead is needed to determine the next output,
  - then a deterministic machine may wait forever to output something valid.
  - Result: a finite output sequence, but the infinite input sequence is valid $\lightning$

Partial domain

- Deterministic two-way transducers suffice, sequential transducers do not
- Unbounded lookahead may be necessary

Summary

Summary

Impl / Spec	Mealy machine	computable
synchronous w/ total domain	EXPTime-c[1]	EXPTime-c[2]
synchronous w/ partial domain	EXPTime-c[1]	**EXPTime-c**[2]

[1] Starting from a specification given by a non-deterministic automaton
[2] Starting from a specification given by a deterministic automaton

Summary

Impl / Spec	Mealy machine	computable
synchronous w/ total domain	EXPTIME-c[1]	EXPTIME-c[2]
synchronous w/ partial domain	EXPTIME-c[1]	**EXPTime-c**[2]

[1] Starting from a specification given by a non-deterministic automaton
[2] Starting from a specification given by a deterministic automaton

- Implementations for total domain
  - sequential transducers suffice
  - bounded lookahead suffices

Summary

Spec \ Impl	Mealy machine	computable
synchronous w/ total domain	EXPTIME-c[1]	EXPTIME-c[2]
synchronous w/ partial domain	EXPTIME-c[1]	**EXPTime-c**[2]

[1] Starting from a specification given by a non-deterministic automaton
[2] Starting from a specification given by a deterministic automaton

- Implementations for total domain
  - sequential transducers suffice
  - bounded lookahead suffices
- Implementations for partial domain
  - deterministic two-way transducers suffice
  - unbounded lookahead may be necessary

Going Beyond Synchronous Specifications

Going Beyond Synchronous Specifications

- It is decidable whether a synchronous specification can be implemented.

Going Beyond Synchronous Specifications

- It is decidable whether a synchronous specification can be implemented.
- What about more powerful specifications?

Going Beyond Synchronous Specifications

- It is decidable whether a synchronous specification can be implemented.
- What about more powerful specifications?

Theorem (Filiot/W.). It is undecidable whether a given rational relation can be implemented.

Going Beyond Synchronous Specifications

- It is decidable whether a synchronous specification can be implemented.
- What about more powerful specifications?

Theorem (Filiot/W.). It is undecidable whether a given rational relation can be implemented.

Theorem (Filiot/W.). It is undecidable whether a continuous, computable, resp., sequential function can be synthesized from a given rational relation.

Going Beyond Synchronous Specifications

- It is decidable whether a synchronous specification can be implemented.
- What about more powerful specifications?

Theorem (Filiot/W.). It is undecidable whether a given rational relation can be implemented.

Theorem (Filiot/W.). It is undecidable whether a continuous, computable, resp., sequential function can be synthesized from a given rational relation.

- Finite word setting: Undecidable whether a sequential function can be synthesized. (Carayol/Löding'14)

Undecidability Proof (similar to finite word setting)

Reduction from Post's Correspondence Problem

Undecidability Proof (similar to finite word setting)

Reduction from Post's Correspondence Problem

- A PCP instance $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$.

Undecidability Proof (similar to finite word setting)

Reduction from Post's Correspondence Problem

- A PCP instance $u_1, \dots, u_n$ and $v_1, \dots, v_n$.
- Rational relation with domain $\{1,\dots,n\}^*\{a,b\}^\omega$ and pairs

$$i_1 \cdots i_m \alpha \begin{cases} \mapsto u_{i_1} \cdots u_{i_m} \beta & \text{if } \alpha \text{ contains } \infty \text{ many } a \\ \not\mapsto v_{i_1} \cdots v_{i_m} \beta & \text{otherwise} \end{cases}$$

with $i_1 \cdots i_m \in \{1,\dots,n\}^*$ and $\alpha, \beta \in \{a,b\}^\omega$.

Undecidability Proof (similar to finite word setting)

Reduction from Post's Correspondence Problem

- A PCP instance $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$.
- Rational relation with domain $\{1, \ldots, n\}^*\{a, b\}^\omega$ and pairs

$$i_1 \cdots i_m \alpha \begin{cases} \mapsto u_{i_1} \cdots u_{i_m} \beta & \text{if } \alpha \text{ contains } \infty \text{ many } a \\ \not\mapsto v_{i_1} \cdots v_{i_m} \beta & \text{otherwise} \end{cases}$$

with $i_1 \cdots i_m \in \{1, \ldots, n\}^*$ and $\alpha, \beta \in \{a, b\}^\omega$.

PCP instance has no solution

- $i_1 \cdots i_m \alpha \mapsto u_{i_1} \cdots u_{i_m} \alpha$ is an implementation
- always $u_{i_1} \cdots u_{i_m} \neq v_{i_1} \cdots v_{i_m}$

Undecidability Proof (similar to finite word setting)

Reduction from Post's Correspondence Problem

- A PCP instance $u_1, \dots, u_n$ and $v_1, \dots, v_n$.
- Rational relation with domain $\{1, \dots, n\}^*\{a, b\}^\omega$ and pairs

$$i_1 \cdots i_m \alpha \begin{cases} \mapsto u_{i_1} \cdots u_{i_m} \beta & \text{if } \alpha \text{ contains } \infty \text{ many } a \\ \not\mapsto v_{i_1} \cdots v_{i_m} \beta & \text{otherwise} \end{cases}$$

with $i_1 \cdots i_m \in \{1, \dots, n\}^*$ and $\alpha, \beta \in \{a, b\}^\omega$.

PCP instance has no solution

- $i_1 \cdots i_m \alpha \mapsto u_{i_1} \cdots u_{i_m} \alpha$ is an implementation
- always $u_{i_1} \cdots u_{i_m} \neq v_{i_1} \cdots v_{i_m}$

PCP instance has a solution

- no implementation exists
- never known whether the input sequence has ∞ many a

Work in Progress: Deterministic Rational Relations

Work in Progress: Deterministic Rational Relations

Class between synchronous and rational relations.

Work in Progress: Deterministic Rational Relations

Class between synchronous and rational relations.

Recognized by special kind of transducers

- state set is partitioned into input and output states

Work in Progress: Deterministic Rational Relations

Class between synchronous and rational relations.

Recognized by special kind of transducers

- state set is partitioned into input and output states
- transition function: $Q_{\mathsf{i}} \times \Sigma \to Q \quad \cup \quad Q_{\mathsf{o}} \times \Gamma \to Q$

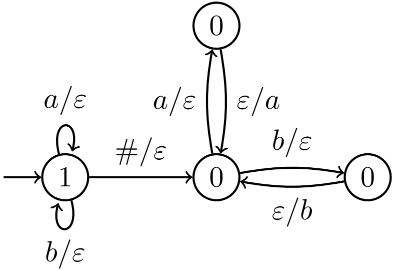
Work in Progress: Deterministic Rational Relations

Class between synchronous and rational relations.

Recognized by special kind of transducers

- state set is partitioned into input and output states
- transition function: $Q_{\mathsf{i}} \times \Sigma \to Q \quad \cup \quad Q_{\mathsf{o}} \times \Gamma \to Q$

Example.



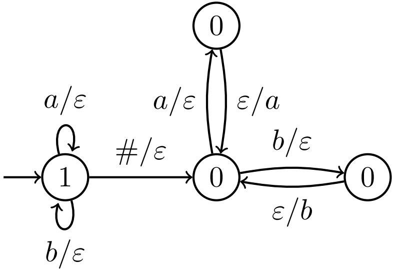
Work in Progress: Deterministic Rational Relations

Class between synchronous and rational relations.

Recognized by special kind of transducers

- state set is partitioned into input and output states
- transition function: $Q_{\mathsf{i}} \times \Sigma \to Q \quad \cup \quad Q_{\mathsf{o}} \times \Gamma \to Q$

Example.



- recognizes $f\colon u\#\alpha \mapsto \alpha, \quad u \in \{a,b\}^*, \alpha \in \{a,b\}^\omega$
- f is not synchronous

Work in Progress: Deterministic Rational Relations

Work in Progress: Deterministic Rational Relations

Almost Sure Theorem. It is decidable whether a continuous function can be synthesized from a given deterministic rational relation.

Work in Progress: Deterministic Rational Relations

Almost Sure Theorem. It is decidable whether a continuous function can be synthesized from a given deterministic rational relation.

Almost Sure Theorem. Such a synthesized function is computable by a deterministic two-way transducer.

Open question

Open question

Is it decidable whether a synchronous relation with **partial domain** is implementable using only finite memory?

Open question

Is it decidable whether a synchronous relation with **partial domain** is implementable using only finite memory?

Example.

- Specification: $(a^*b\cdots, b\cdots) \quad (a^*c\cdots, c\cdots)$

Open question

Is it decidable whether a synchronous relation with **partial domain** is implementable using only finite memory?

Example.

- Specification: $(a^*b\cdots, b\cdots)$ $(a^*c\cdots, c\cdots)$
- Specification is implementable, e.g., by a finite-memory machine (sequential transducer) that computes the function

$$a^*b\cdots \mapsto b^\omega \quad a^*c\cdots \mapsto c^\omega$$

Summary

Spec \ Impl	Mealy machine	sequential transducer	computable
synchronous w/ total domain	EXPTIME-c[1]	EXPTIME-c[2]	EXPTIME-c[2]
synchronous w/ partial domain	EXPTIME-c[1]	open	EXPTIME-c[2]
det. rational	open	open	EXPTIME-c
rational	undecidable	undecidable	undecidable

[1] non-deterministic specification [2] deterministic specification