

Computability of Data Word Functions Defined by Transducers

Léo Exibard^{1,2}

Pierre-Alain Reynier¹

Emmanuel Filiot²

Thursday, June 25th, 2020

¹Laboratoire d'Informatique et des Systèmes
Aix-Marseille Université
France

²Méthodes Formelles et Vérification
Université libre de Bruxelles
Belgium

→ Derive an implementation from the specification of a behaviour

Example (Nondeterministic Finite Automata)

An NFA A *specifies* a language, or equivalently a program that takes as input a word w and outputs 0 or 1.

Nondeterminism does not exist in practice \Rightarrow how to implement such program?

→ Derive an implementation from the specification of a behaviour

Example (Nondeterministic Finite Automata)

An NFA A specifies a language, or equivalently a program that takes as input a word w and outputs 0 or 1.

Nondeterminism does not exist in practice \Rightarrow how to implement such program?

- Enumerate all possible runs of A over w and output 1 as soon as an accepting run is found (0 otherwise).

→ Derive an implementation from the specification of a behaviour

Example (Nondeterministic Finite Automata)

An NFA A specifies a language, or equivalently a program that takes as input a word w and outputs 0 or 1.

Nondeterminism does not exist in practice \Rightarrow how to implement such program?

- Enumerate all possible runs of A over w and output 1 as soon as an accepting run is found (0 otherwise).
- There can be (exponentially) many runs \Rightarrow we can do better

→ Derive an implementation from the specification of a behaviour

Example (Nondeterministic Finite Automata)

An NFA A specifies a language, or equivalently a program that takes as input a word w and outputs 0 or 1.

Nondeterminism does not exist in practice \Rightarrow how to implement such program?

- Enumerate all possible runs of A over w and output 1 as soon as an accepting run is found (0 otherwise).
- There can be (exponentially) many runs \Rightarrow we can do better
- NFA can always be determinised \Rightarrow an equivalent DFA is a program which implements A and is guaranteed to take only a finite amount of memory.

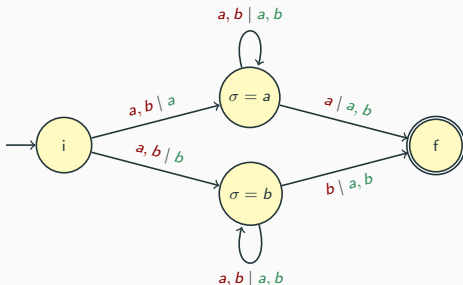
Synthesis

→ Derive an implementation from the specification of a behaviour

Example (Nondeterministic Finite Transducers)

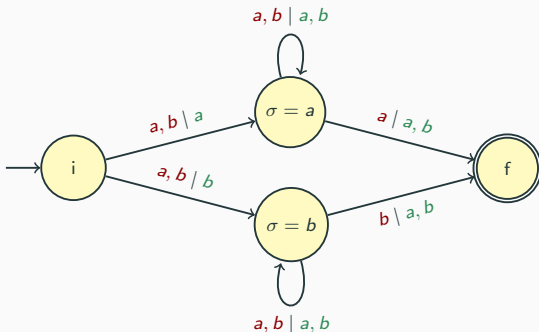
A transducer is an automaton with outputs. To every input word, it associates a set of acceptable outputs

⇒ An implementation **chooses** an acceptable output for each input.



A transducer recognising $S = \{(u\sigma, \sigma w) \mid \sigma \in \Sigma, u, w \in \Sigma^*, |u| = |w|\}$

Synthesis

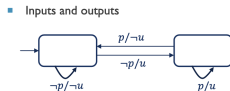


- The above specification can for instance be implemented by a program that computes $f : u\sigma \mapsto \sigma u$
- It cannot be implemented by any deterministic transducer
- Nor by any *synchronous* program, which outputs a letter as soon as it reads a letter

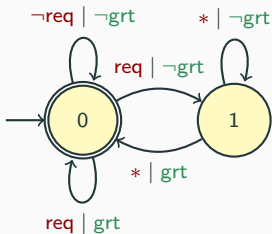
Synthesis: Nondeterministic ω -Transducers

→ Generalisation of transducers to infinite words, with a parity acceptance condition.

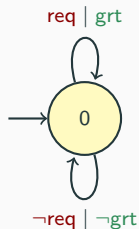
- **Non-deterministic sequential transducers**



Infinite words do not exist in practice: we are specifying the behaviour of a non-terminating program *in the limit*.



An ω -transducer specifying that every request must eventually be granted



A program which immediately grants any request, represented as a deterministic transducer

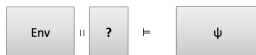
What does it mean to be computable for non-terminating behaviours?

In the classical reactive synthesis setting

- An implementation is a *synchronous* program, i.e. a strategy in the parity game induced by the transducer.
- As parity games are positionally determined, we can restrict to finite-memory *synchronous* programs, also known as deterministic transducers.

Reactive system synthesis as solving a game

☞ support the design process with **automatic synthesis**



- Sys is constructed by an **algorithm**
- Sys is **correct** by construction
- Underlying theory: **2-player zero-sum games**
- Env is **adversarial** (worst-case assumption)

Winning strategy = Correct Sys

Synthesis: to sum up

Definition ($\text{Real}(\mathcal{S}, \mathcal{I})$)

- \mathcal{S} : class of specifications
- \mathcal{I} : class of implementations Given $S \in \mathcal{S}$, decide whether there exists $I \in \mathcal{I}$ such that I implements S , i.e. I and S have same domain and for all $x \in \text{dom}(S)$, $(x, I(x)) \in S$

Synthesis: to sum up

Definition ($\text{Real}(\mathcal{S}, \mathcal{I})$)

- \mathcal{S} : class of specifications
- \mathcal{I} : class of implementations Given $S \in \mathcal{S}$, decide whether there exists $I \in \mathcal{I}$ such that I implements S , i.e. I and S have same domain and for all $x \in \text{dom}(S)$, $(x, I(x)) \in S$
- $\text{REAL}(\text{NFA}, \text{TM}) = \text{REAL}(\text{NFA}, \text{DFA})$ and is always true

Synthesis: to sum up

Definition ($\text{Real}(\mathcal{S}, \mathcal{I})$)

- \mathcal{S} : class of specifications
- \mathcal{I} : class of implementations Given $S \in \mathcal{S}$, decide whether there exists $I \in \mathcal{I}$ such that I implements S , i.e. I and S have same domain and for all $x \in \text{dom}(S)$, $(x, I(x)) \in S$
- $\text{REAL}(\text{NFA}, \text{TM}) = \text{REAL}(\text{NFA}, \text{DFA})$ and is always true
- $\text{REAL}(\text{NFT}_{\text{syn}}, \text{TM})$ is always true, but not $\text{REAL}(\text{NFT}_{\text{syn}}, \text{DFT}_{\text{syn}})$ (which is decidable)

Synthesis: to sum up

Definition ($\text{Real}(\mathcal{S}, \mathcal{I})$)

- \mathcal{S} : class of specifications
- \mathcal{I} : class of implementations Given $S \in \mathcal{S}$, decide whether there exists $I \in \mathcal{I}$ such that I implements S , i.e. I and S have same domain and for all $x \in \text{dom}(S)$, $(x, I(x)) \in S$
- $\text{REAL}(\text{NFA}, \text{TM}) = \text{REAL}(\text{NFA}, \text{DFA})$ and is always true
- $\text{REAL}(\text{NFT}_{\text{syn}}, \text{TM})$ is always true, but not $\text{REAL}(\text{NFT}_{\text{syn}}, \text{DFT}_{\text{syn}})$ (which is decidable)
- $\text{REAL}(\omega\text{NT}_{\text{syn}}, \text{SP}) = \text{REAL}(\omega\text{NT}_{\text{syn}}, \omega\text{DT}_{\text{syn}})$ is decidable and equivalent with finding a winning strategy in a parity game

What does it mean to be computable for non-terminating behaviours?

Relax the synchronicity requirement

An implementation is a program which outputs longer and longer prefixes of an acceptable output as it reads longer and longer prefixes of the input.

Example (Guessing the last letter of a chunk)

Consider a specification that takes as input an ω -word of the form $u_1\sigma_1\#u_2\sigma_2\#u_3\sigma_3\dots$ and accepts any output of the form $\sigma_1w_1\#\sigma_2w_2\#\sigma_3w_3\dots$

- It cannot be implemented by a synchronous program
- It can be implemented by a program computing $f_{\#} : u\sigma \mapsto \sigma u$ on each chunk

What does it mean to be computable for non-terminating behaviours?

Relax the synchronicity requirement

An implementation is a program which outputs longer and longer prefixes of an acceptable output as it reads longer and longer prefixes of the input.

Example (Guessing if the first letter appears again)

To an input σu , associate the output σ^ω if σ occurs in u , and σu otherwise.

- Such specification is definable by a transducer which initially guesses whether σ will appear again and checks such property
- It cannot be implemented by any program

What does it mean to be computable for non-terminating behaviours?

Relax the synchronicity requirement

An implementation is a program which outputs longer and longer prefixes of an acceptable output as it reads longer and longer prefixes of the input.

Asynchronous specifications

We now consider **asynchronous** transducers: on reading a letter $\sigma \in \Sigma$, a transducer can output a word $w \in \Sigma^*$.



Theorem ([Holtmann et al., 2012])

Deciding whether a specification defined by a transducer is realisable by a computable function is undecidable.

Computability

A function $f : \Sigma^\omega \rightarrow \Sigma^\omega$ is *computable* if there exists a deterministic Turing machine M which outputs longer and longer prefixes of the output when reading longer and longer prefixes of the input

- Three tape deterministic Turing machine
 - Read-only one-way input tape
 - Two-way working tape
 - Write-only one-way output tape
- $M(x, k)$: the output written after having the k first input letters of x
- Since the output is write-only, $M(x, k)$ is nondecreasing

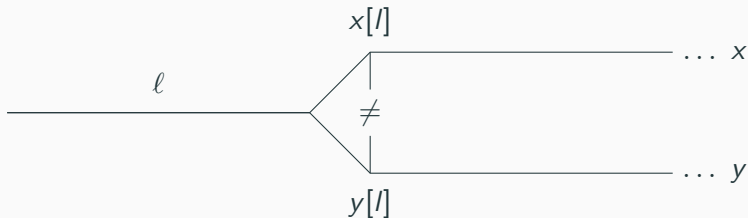
M **computes** f if

for all $x \in \text{dom}(f)$, $M(x, k)$ converges towards $f(x)$

Cantor distance

$$\text{For } u, v \in \Sigma^\omega, d(u, v) = \begin{cases} 0 & \text{if } u = v \\ 2^{-\|u \wedge v\|} & \text{otherwise} \end{cases}$$

where $u \wedge v$ denotes the longest common prefix ℓ of u and v



Continuous function

A function $f : \Sigma^\omega \rightarrow \Sigma^\omega$ is *continuous* at $x \in \text{dom}(f)$ if:

- (a) for all sequences of data words $(x_n)_{n \in \mathbb{N}}$ converging towards x , we have that $(f(x_n))_{n \in \mathbb{N}}$ converges to $f(x)$. (where for all $i \in \mathbb{N}$, $x_i \in \text{dom}(f)$), or **equivalently**:
- (b) $\forall i \geq 0, \exists j \geq 0, \forall y \in \text{dom}(f), \|x \wedge y\| \geq j \Rightarrow \|f(x) \wedge f(y)\| \geq i$.

Continuity

Continuous function

A function $f : \Sigma^\omega \rightarrow \Sigma^\omega$ is *continuous* at $x \in \text{dom}(f)$ if:

- (a) for all sequences of data words $(x_n)_{n \in \mathbb{N}}$ converging towards x , we have that $(f(x_n))_{n \in \mathbb{N}}$ converges to $f(x)$. (where for all $i \in \mathbb{N}$, $x_i \in \text{dom}(f)$), or **equivalently**:
- (b) $\forall i \geq 0, \exists j \geq 0, \forall y \in \text{dom}(f), \|x \wedge y\| \geq j \Rightarrow \|f(x) \wedge f(y)\| \geq i$.

Functionality

A specification is *functional* if any input admits at most one acceptable output.

- **Unless otherwise stated**, specifications are now functional.
- Deciding if a transducer T is functional is doable in polynomial time.

Computability and Continuity

Computability

$f : \Sigma^\omega \rightarrow \Sigma^\omega$ is computable if there exists a deterministic Turing machine which outputs longer and longer prefixes of the output when reading longer and longer prefixes of the input.

Continuity

$\forall i \geq 0, \exists j \geq 0, \forall y \in \text{dom}(f), \|x \wedge y\| \geq j \Rightarrow \|f(x) \wedge f(y)\| \geq i$

Computability and Continuity

Computability

$f : \Sigma^\omega \rightarrow \Sigma^\omega$ is computable if there exists a deterministic Turing machine which outputs longer and longer prefixes of the output when reading longer and longer prefixes of the input.

Continuity

$\forall i \geq 0, \exists j \geq 0, \forall y \in \text{dom}(f), \|x \wedge y\| \geq j \Rightarrow \|f(x) \wedge f(y)\| \geq i$

Computability \Rightarrow Continuity

If $f : \Sigma^\omega \rightarrow \Sigma^\omega$ is computable, then it is continuous.

Computability and Continuity

Computability

$f : \Sigma^\omega \rightarrow \Sigma^\omega$ is computable if there exists a deterministic Turing machine which outputs longer and longer prefixes of the output when reading longer and longer prefixes of the input.

Continuity

$\forall i \geq 0, \exists j \geq 0, \forall y \in \text{dom}(f), \|x \wedge y\| \geq j \Rightarrow \|f(x) \wedge f(y)\| \geq i$

Computability \Rightarrow Continuity

If $f : \Sigma^\omega \rightarrow \Sigma^\omega$ is computable, then it is continuous.

\rightarrow For $i \geq 0$ take $j \geq 0$ such that $\|M(x, j)\| \geq i$.

Since M is deterministic, any input y such that $\|x \wedge y\| \geq j$ satisfies $M(y, j) = M(x, j)$.

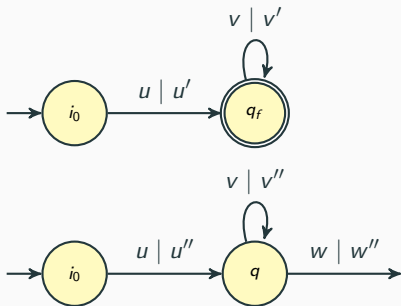
Thus, $M(x, j)$ is a common prefix of $f(x)$ and $f(y)$, so $\|f(x) \wedge f(y)\| \geq i$.

Characterising continuity with a pattern

Theorem (Excluded pattern [Dave et al., 2019])

Let T be transducer defining a function f_T .

f_T is continuous iff T does not have the following pattern:



where $mismatch(u', u'') \vee (v'' = \varepsilon \wedge mismatch(u', u''w''))$

Our Contribution: extension to the infinite alphabet case

Until now

- Behaviour specified by functional asynchronous transducers
- Computability defined with deterministic Turing machines

Extend to devices computing over (slightly) infinite sets

A register automaton with equality only
assume a countably infinite set of *atoms*, which can only be compared for equality

Definition. The syntax of a *register automaton* consists of:

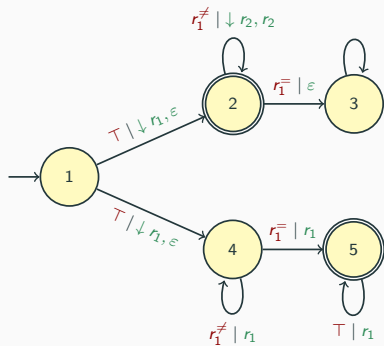
- finite set of *locations*
- subsets of *initial* and *accepting* locations
- finite set of *register names*
- a *transition relation*, which is an equivariant subset of:

$$\underbrace{\text{STATES}}_{(\text{locations} \times \underbrace{\text{register valuations}}_{\substack{\text{partial function from} \\ \text{register names to atoms}}}) \times \underbrace{\text{input letters}}_{\text{atoms}} \times (\text{locations} \times \text{register valuations})}$$

- Behaviour is defined using register transducers
- Computability is defined by allowing Turing machines to work over an infinite alphabet

Register Transducers

- \mathcal{D} is a countably infinite set whose elements can be compared for equality only
- Equip a transducer with a finite set of registers
- Recognise relations S over data words, i.e.
 $S \subseteq (\Sigma \times \mathcal{D}) \times (\Sigma \times \mathcal{D})$



A register transducer taking as input dw and outputting w if d does not appear in w , d^ω otherwise (finite labels are irrelevant and not depicted)

Indistinguishability property

As register machines only have k registers, any run over some data word w can be renamed into a run over some data word w' with at most $k + 1$ data.

Corollary

Let A be a nondeterministic register automaton with k registers. If $L(A) \neq \emptyset$, then, for any $X \subseteq \mathcal{D}$ of size $|X| \geq k + 1$

$$L(A) \cap (\Sigma \times X)^\omega \neq \emptyset.$$

Indistinguishability property

As register machines only have k registers, any run over some data word w can be renamed into a run over some data word w' with at most $k + 1$ data.

Corollary

Let A be a nondeterministic register automaton with k registers. If $L(A) \neq \emptyset$, then, for any $X \subseteq \mathcal{D}$ of size $|X| \geq k + 1$
 $L(A) \cap (\Sigma \times X)^\omega \neq \emptyset$.

Theorem (Functionality)

Deciding whether a register transducer T is functional is PSPACE-complete

Indistinguishability property

As register machines only have k registers, any run over some data word w can be renamed into a run over some data word w' with at most $k + 1$ data.

Corollary

Let A be a nondeterministic register automaton with k registers. If $L(A) \neq \emptyset$, then, for any $X \subseteq \mathcal{D}$ of size $|X| \geq k + 1$
 $L(A) \cap (\Sigma \times X)^\omega \neq \emptyset$.

Theorem (Functionality)

Deciding whether a register transducer T is functional is PSPACE-complete

→ Thanks to the indistinguishability property, we can show that T is functional if and only if it is functional over $(\Sigma \times X)^\omega$, where X is a finite subset of \mathcal{D} of size $k + 1$.

Continuity and computability

For functions defined by register transducers, computability and continuity again coincide.

Computability \Rightarrow Continuity is proved as before.

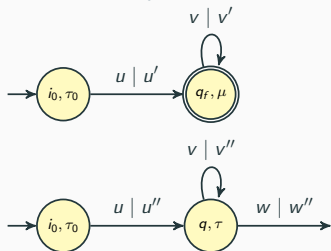
Continuity \Rightarrow Computability: requires to decide $o\sigma \preceq \hat{f}(x[:j])$

Algorithm 1: Algorithm describing the machine M_f computing f .

```
Data:  $x \in \text{dom}(f)$   
1  $o := \epsilon$  ;  
2 for  $j = 0$  to  $\infty$  do  
3   for  $(\sigma, d) \in \Sigma \times (dt(x[:j]) \cup \{d_0\})$  do  
4     if  $o.(\sigma, d) \preceq \hat{f}(x[:j])$  then // such test is decidable  
5        $o := o.(\sigma, d)$  ;  
6       output  $(\sigma, d)$  ;  
7     end  
8   end  
9 end
```

Continuity: extend the pattern characterisation

Theorem (Excluded pattern)



where:

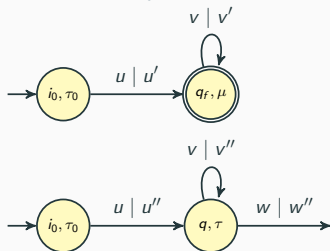
$$\text{mismatch}(u', u'') \vee$$

$$v'' = \varepsilon \wedge \text{mismatch}(u', u''w'')$$

Moreover, such pattern is present iff it is present for data words with at most $k + 1$ data.

Continuity: extend the pattern characterisation

Theorem (Excluded pattern)



where:

$$\text{mismatch}(u', u'') \vee$$

$$v'' = \varepsilon \wedge \text{mismatch}(u', u''w'')$$

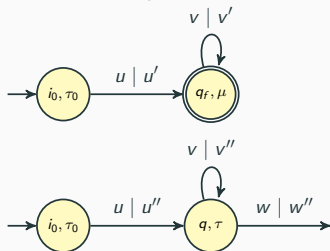
Moreover, such pattern is present iff it is present for data words with at most $k + 1$ data.

Corollary

$f_{\mathcal{T}}$ is continuous iff it is continuous over $(\Sigma \times X)^\omega$.

Continuity: extend the pattern characterisation

Theorem (Excluded pattern)



where:

$$\text{mismatch}(u', u'') \vee$$

$$v'' = \varepsilon \wedge \text{mismatch}(u', u''w'')$$

Moreover, such pattern is present iff it is present for data words with at most $k + 1$ data.

Corollary

$f_{\mathcal{T}}$ is continuous iff it is continuous over $(\Sigma \times X)^\omega$.



This yields a PSPACE algorithm to decide whether a function $f_{\mathcal{T}}$ defined by a register transducer is computable.

Conclusion

- For functions defined by register transducers, continuity and computability coincide, and are decidable
- Such class is moreover closed under composition, and decidable
- Those problems are decidable in polynomial time for a subclass of functions, namely those recognised by test-free register-transducers

Future work

- Can we allow the devices to guess a data and put it in its registers?
- Extension to the 2-way case

-  Dave, V., Filiot, E., Krishna, S. N., and Lhote, N. (2019).
Deciding the computability of regular functions over infinite words.
CoRR, abs/1906.04199.
-  Holtmann, M., Kaiser, L., and Thomas, W. (2012).
Degrees of lookahead in regular infinite games.
Logical Methods in Computer Science, 8(3).