Computability of Data Word Functions Defined by Transducers

Léo Exibard[1,2] Pierre-Alain Reynier[1] Emmanuel Filiot[2]

Thursday, June 25th, 2020

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

[2]Méthodes Formelles et Vérification
Université libre de Bruxelles
Belgium

Synthesis

- ➔ 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?

Synthesis

➔ 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).

Synthesis

➔ 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

Synthesis

➔ 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?

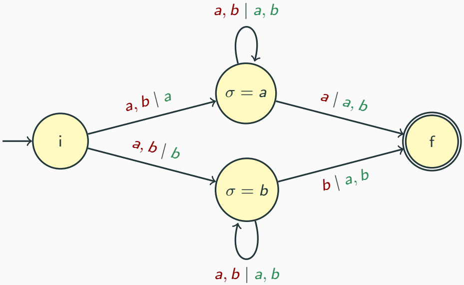
- Enumerate all possible runs of A over w and output 1 as soon as an accepting run is found (0 otherwise).
- There can be (exponentially) many runs $\Rightarrow$ we can do better
- NFA can always be determinised $\Rightarrow$ an equivalent DFA is a program which implements A and is guaranteed to take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

➔ Derive an implementation from the specification of a behaviour

Example (Nondeterministic Finite Transducers)

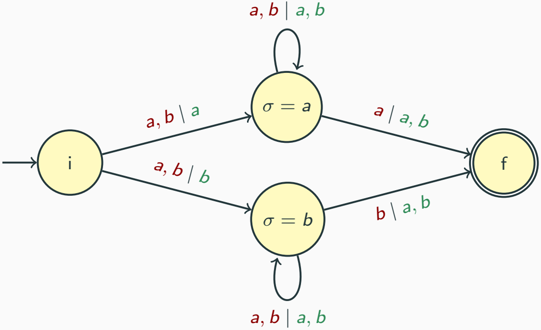
A transducer is an automaton with outputs. To every input word, it associates a set of acceptable outputs

$\Rightarrow$ An implementation chooses an acceptable output for each input.



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

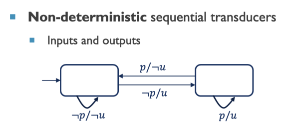
Synthesis



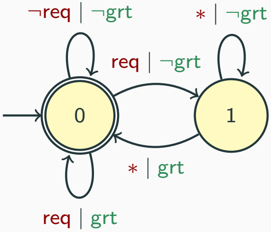
- The above specification can for instance be implemented by a program that computes $f : u\sigma \mapsto \sigma u$
- It cannot be implemented by any deterministic transducer
- Nor by any *synchronous* program, which outputs a letter as soon as it reads a letter

Synthesis: Nondeterministic ω-Transducers

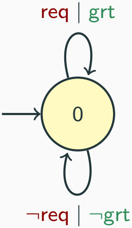
➔ Generalisation of transducers to infinite words, with a parity acceptance condition.



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

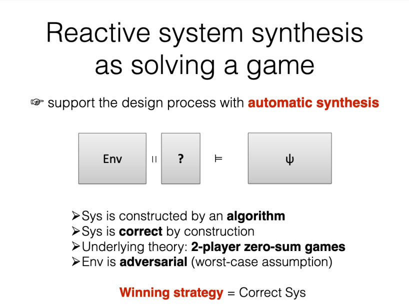


A program which immediately grants any request, represented as a deterministic transducer

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.



Synthesis: to sum up

Definition (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 (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 $x \in \text{dom}(S), (x, I(x)) \in S$

- $\textsc{Real}(\mathit{NFA}, \mathit{TM}) = \textsc{Real}(\mathit{NFA}, \mathit{DFA})$ and is always true

Synthesis: to sum up

Definition (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 $x \in \text{dom}(S), (x, I(x)) \in S$

- $\textsc{Real}(\mathit{NFA}, \mathit{TM}) = \textsc{Real}(\mathit{NFA}, \mathit{DFA})$ and is always true
- $\textsc{Real}(\mathit{NFT}_{\text{syn}}, \mathit{TM})$ is always true, but not $\textsc{Real}(\mathit{NFT}_{\text{syn}}, \mathit{DFT}_{\text{syn}})$ (which is decidable)

Synthesis: to sum up

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

- $\mathcal{S}$: 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 $x \in \mathrm{dom}(S), (x, I(x)) \in S$

- $\textsc{Real}(NFA, TM) = \textsc{Real}(NFA, DFA)$ and is always true
- $\textsc{Real}(NFT_{\mathsf{syn}}, TM)$ is always true, but not $\textsc{Real}(NFT_{\mathsf{syn}}, DFT_{\mathsf{syn}})$ (which is decidable)
- $\textsc{Real}(\omega NT_{\mathsf{syn}}, SP) = \textsc{Real}(\omega NT_{\mathsf{syn}}, \omega DT_{\mathsf{syn}})$ 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 $u_1\sigma_1\#u_2\sigma_2\#u_3\sigma_3\ldots$ and accepts any output of the form $\sigma_1 w_1\#\sigma_2 w_2\#\sigma_3 w_3\ldots$

- 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

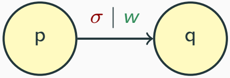
What does it mean to be computable for non-terminating behaviours?

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 $f : \Sigma^\omega \to \Sigma^\omega$ 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

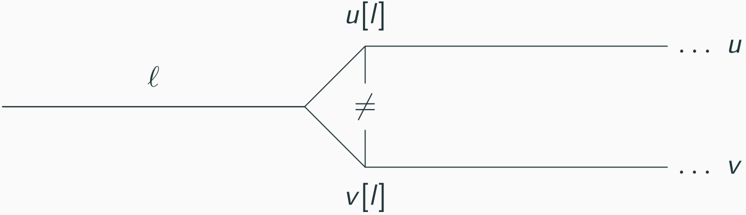
for all $x \in \mathrm{dom}(f)$, $M(x, k)$ converges towards $f(x)$

Continuity

Cantor distance

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



Continuity

Continuous function

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

(a) for all sequences of data words $(x_n)_{n\in\mathbb{N}}$ converging towards, we have that $(f(x_n))_{n\in\mathbb{N}}$ converges to $f(x)$. (where for all $i \in \mathbb{N}$, $x_i \in \mathrm{dom}(f)$), or equivalently:

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

Continuity

Continuous function

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

(a) for all sequences of data words $(x_n)_{n\in\mathbb{N}}$ converging towards, we have that $(f(x_n))_{n\in\mathbb{N}}$ converges to $f(x)$. (where for all $i \in \mathbb{N}$, $x_i \in \mathrm{dom}(f)$), or equivalently:

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

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

$f : \Sigma^\omega \to \Sigma^\omega$ 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

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

Computability and Continuity

Computability

$f : \Sigma^\omega \to \Sigma^\omega$ 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

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

Computability $\Rightarrow$ Continuity

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

Computability and Continuity

Computability

$f : \Sigma^\omega \to \Sigma^\omega$ 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

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

Computability $\Rightarrow$ Continuity

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

$\to$ 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 $\|f(x) \wedge f(y)\| \geq i$.

Computability and Continuity

Computability

$f : \Sigma^\omega \to \Sigma^\omega$ 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

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

Continuity $\Rightarrow$ Computability [Dave et al., 2019]

Let $f : \Sigma^\omega \to \Sigma^\omega$ be a function definable by a nondeterministic transducer. Then f is continuous iff it is computable.

Algorithm 1: Algorithm describing the machine M_f computing f.

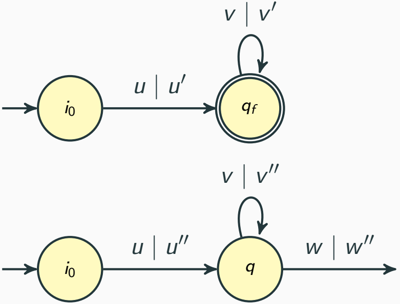
```
Data: x ∈ dom(f)
o := ε; // the output so far
for j = 0 to ∞ do
    for σ ∈ Σ do
        if o.σ ⪯ f̂(x[:j]) then // σ can be safely output
            o := o.σ;
            output σ;
        end
    end
end
```

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:



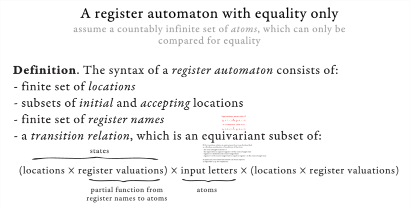
where $mismatch(u', u'') \vee (v'' = \varepsilon \wedge mismatch(u', u''w''))$

Our Contribution: extension to the infinite alphabet case

Until now

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

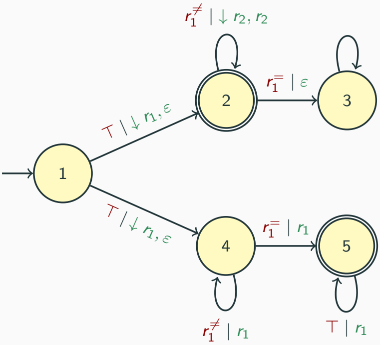
Extend to devices computing over (slightly) infinite sets



- 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 [Kaminski and Francez, 1994]

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 $L(A) \neq \varnothing$, then, for any $X \subseteq \mathcal{D}$ of size $|X| \geq k+1$ $L(A) \cap (\Sigma \times X)^\omega \neq \varnothing$.

Indistinguishability property [Kaminski and Francez, 1994]

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 $L(A) \neq \varnothing$, then, for any $X \subseteq \mathcal{D}$ of size $|X| \geq k+1$ $L(A) \cap (\Sigma \times X)^\omega \neq \varnothing$.

Theorem (Functionality)

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

Indistinguishability property [Kaminski and Francez, 1994]

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 $L(A) \neq \varnothing$, then, for any $X \subseteq \mathcal{D}$ of size $|X| \geq k+1$ $L(A) \cap (\Sigma \times X)^\omega \neq \varnothing$.

Theorem (Functionality)

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

$\rightarrow$ *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 $2k+1$.*

Continuity and computability

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

Computability ⇒ Continuity is proved as before.

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

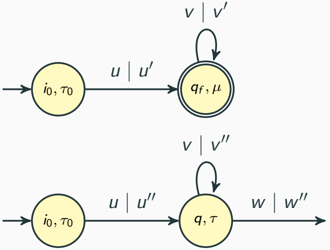
Algorithm 1: Algorithm describing the machine M_f computing f.

Data: $x \in \mathrm{dom}(f)$

```
o := ε ;
for j = 0 to ∞ do
    for (σ, d) ∈ Σ × (dt(x[:j]) ∪ {d_0}) do
        if o.(σ, d) ⪯ f̂(x[:j]) then // such test is decidable
            o := o.(σ, d);
            output (σ, d);
        end
    end
end
```

Continuity: extend the pattern characterisation

Theorem (Excluded pattern)



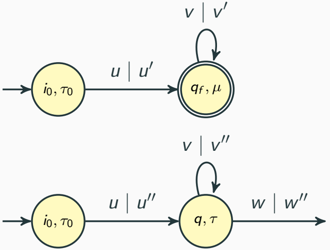
where:

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

Moreover, such pattern is present iff it is present for data words with at most $2k + 1$ data.

Continuity: extend the pattern characterisation

Theorem (Excluded pattern)



where:

$mismatch(u', u'') \vee$

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

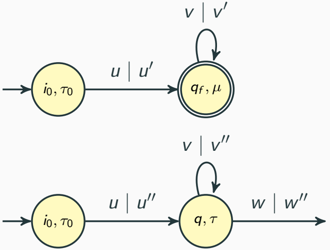
Moreover, such pattern is present iff it is present for data words with at most $2k+1$ data.

Corollary

f_T is continuous iff it is continuous over $(\Sigma \times X)^\omega$.

Continuity: extend the pattern characterisation

Theorem (Excluded pattern)



where:

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

Moreover, such pattern is present iff it is present for data words with at most $2k + 1$ data.

Corollary

f_T is continuous iff it is continuous over $(\Sigma \times X)^\omega$.

This yields a PSPACE algorithm to decide whether a function f_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

Bibliography i

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).

Kaminski, M. and Francez, N. (1994).
Finite-memory automata.
Theor. Comput. Sci., 134(2):329–363.