Computability of Data Word Functions Defined by Transducers

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

Thursday, April 1st, 2021

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

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

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?

Computability and Nondeterminism

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).

Computability and Nondeterminism

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

Example (Nondeterministic Finite Automata)

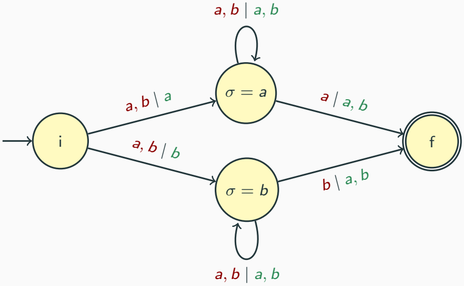
An NFA A *specifies* a language, or equivalently a program that takes as input a word w and outputs 0 or 1.

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.

Definition (Nondeterministic Finite Transducers)

A transducer is an automaton with outputs.

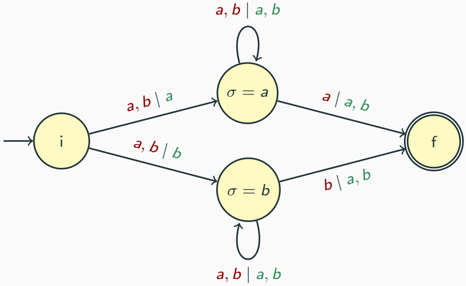


A transducer checking that the first output letter is equal to the last input letter: $S = \{(u\sigma, \sigma w) \mid \sigma \in \Sigma, u, w \in \Sigma^*, |u| = |w|\}$

Functions from Words to Words

Definition (Nondeterministic Finite Transducers)

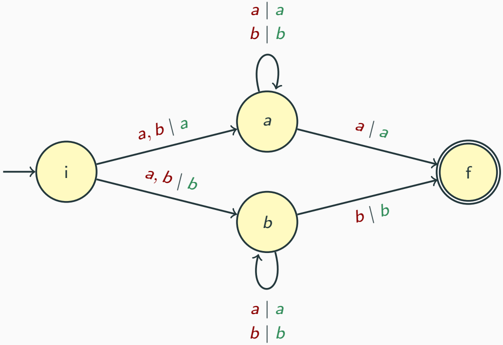
A transducer is an automaton with outputs.



A transducer checking that the first output letter is equal to the last input letter: $S = \{(u\sigma, \sigma w) \mid \sigma \in \Sigma, u, w \in \Sigma^*, |u| = |w|\}$

- Nondeterminism $\Rightarrow$ they do not always specify functions.
- Here, we focus on *functional* transducers.
- Functionality can be checked in PTIME.

Computation of Functions Defined by Transducers



A transducer replacing the first letter with the last: $f_{\text{last}} : \gamma u \sigma \mapsto \sigma u \sigma$

The above function can be computed, but:

- it cannot be implemented by a 1-way deterministic transducer
- nor by any *synchronous* program, which outputs a letter as soon as it reads a letter

The ω-word Setting

Transducers can be equipped with a parity condition to recognise functions over infinite words $f : \Sigma^\omega \to \Gamma^\omega$

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

Examples

- Iterated f_{last}: input is an infinite sequence of *chunks* $\gamma_i u_i \sigma_i$, separated by $\#$, and the program applies f_{last} on each chunk. $f_{\#\text{last}} : \gamma_1 u_1 \sigma_1 \# \gamma_2 u_2 \sigma_2 \cdots \mapsto \sigma_1 u_1 \sigma_1 \# \sigma_2 u_2 \sigma_2 \ldots$

The ω-word Setting

Transducers can be equipped with a parity condition to recognise functions over infinite words $f : \Sigma^\omega \to \Gamma^\omega$

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

Examples

- Iterated f_{last}: input is an infinite sequence of *chunks* $\gamma_i u_i \sigma_i$, separated by $\#$, and the program applies f_{last} on each chunk.
  $f_{\#\text{last}} : \gamma_1 u_1 \sigma_1 \# \gamma_2 u_2 \sigma_2 \cdots \mapsto \sigma_1 u_1 \sigma_1 \# \sigma_2 u_2 \sigma_2 \ldots$
- Detecting whether the first letter appears again.
  $f_{\text{again}} : \sigma u \mapsto \begin{cases} a^\omega \text{ if u contains } \sigma \\ b^\omega \text{ otherwise} \end{cases}$

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

In the classical reactive synthesis setting

The target implementation is a *synchronous* program, i.e. one which outputs a letter everytime it reads an input letter.

$\Rightarrow$ It corresponds to a strategy in the parity game induced by the transducer, so finite memory suffices.

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

In the classical reactive synthesis setting

The target implementation is a *synchronous* program, i.e. one which outputs a letter everytime it reads an input letter.

$\Rightarrow$ It corresponds to a strategy in the parity game induced by the transducer, so finite memory suffices.

Co-example

Iterated f_{last} is not synchronously computable, as f_{last} requires to wait for the last letter of the chunk.

$f_{\#\text{last}} : \gamma_1 u_1 \sigma_1 \# \gamma_2 u_2 \sigma_2 \cdots \mapsto \sigma_1 u_1 \sigma_1 \# \sigma_2 u_2 \sigma_2 \ldots$

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

In the classical reactive synthesis setting

The target implementation is a *synchronous* program, i.e. one which outputs a letter everytime it reads an input letter.

$\Rightarrow$ It corresponds to a strategy in the parity game induced by the transducer, so finite memory suffices.

Co-example

Iterated f_{last} is not synchronously computable, as f_{last} requires to wait for the last letter of the chunk.

$f_{\#\text{last}} : \gamma_1 u_1 \sigma_1 \# \gamma_2 u_2 \sigma_2 \cdots \mapsto \sigma_1 u_1 \sigma_1 \# \sigma_2 u_2 \sigma_2 \ldots$

In our setting

We relax the *synchronicity* requirement.

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

Iterated f_{last} is computable, as the program can wait for the end of the chunk.

$f_{\#\text{last}} : \gamma_1 u_1 \sigma_1 \# \gamma_2 u_2 \sigma_2 \cdots \mapsto \sigma_1 u_1 \sigma_1 \# \sigma_2 u_2 \sigma_2 \ldots$

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

Iterated f_{last} is computable, as the program can wait for the end of the chunk.

$f_{\#\text{last}} : \gamma_1 u_1 \sigma_1 \# \gamma_2 u_2 \sigma_2 \cdots \mapsto \sigma_1 u_1 \sigma_1 \# \sigma_2 u_2 \sigma_2 \ldots$

Co-example

f_{again} is not computable, as a program cannot know whether it will read the first letter again.

$$f_{\text{again}} : \sigma u \mapsto \begin{cases} a^\omega \text{ if u contains } \sigma \\ b^\omega \text{ otherwise} \end{cases}$$

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

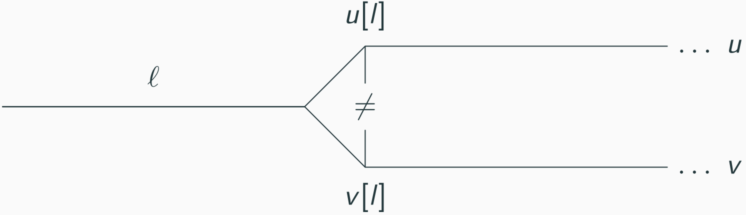
for all $x \in \mathrm{dom}(f)$, $M(x, k)$ converges towards $f(x)$

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 $x \in \text{dom}(f)$ if:

(a) for all sequences of data words $(x_n)_{n\in\mathbb{N}}$ converging to 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) $\forall i \geq 0, \exists j \geq 0, \forall y \in \text{dom}(f), \|x \wedge y\| \geq j \Rightarrow \|f(x) \wedge f(y)\| \geq i$.

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.

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.

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.

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.

Theorem ([Dave et al., 2019])

Computability of functions defined by nondeterministic transducers is decidable in PTIME.

Our Contribution: Extension to the Infinite Alphabet Case

Until now

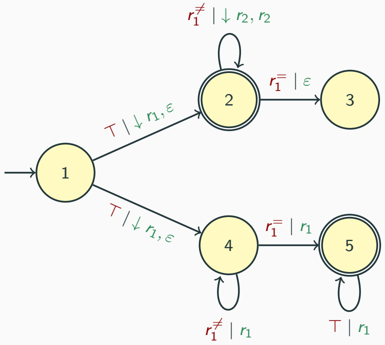
- Behaviour specified by functional asynchronous transducers
- Computability defined with deterministic Turing machines

Extend to devices computing over infinite sets

- Behaviour is specified by 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 functions over data words $f : \mathcal{D}^\omega \to \mathcal{D}^\omega$



A register transducer computing f_{again} over data words: taking as input dw and outputting w if d does not appear in w, d^ω otherwise

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 $|X| \geq k+1$ $L(A) \cap X^\omega \neq \varnothing$.

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 $|X| \geq k+1$
$L(A) \cap X^\omega \neq \varnothing$.

Theorem (Functionality)

Deciding whether a register transducer T is functional is $\textsc{PSpace}$*-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 $|X| \geq k+1$ $L(A) \cap X^\omega \neq \varnothing$.

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 X^ω, where X is a finite subset of $\mathcal{D}$ of size $2k+3$.*

Continuity and computability

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

Computability $\Rightarrow$ Continuity is proved as before.

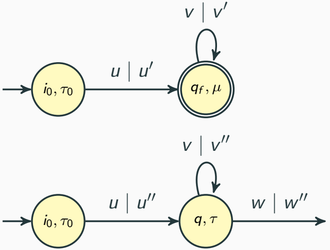
Continuity $\Rightarrow$ Computability: requires to decide $o\sigma \preceq \hat{f}(x[:j])$

Algorithm 1: Algorithm describing the machine M_f computing f.

```
Data: x ∈ 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 of [Dave et al., 2019]

Theorem (Excluded pattern)



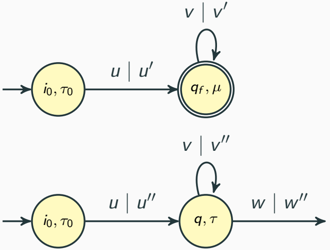
where:

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

Moreover, such pattern is present iff it is present for data words with at most $2k + 3$ data.

Continuity: Extend the Pattern of [Dave et al., 2019]

Theorem (Excluded pattern)



where:

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

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

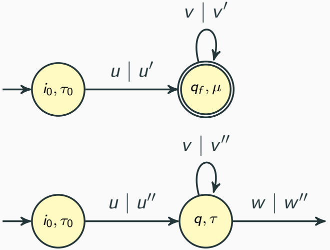
Moreover, such pattern is present iff it is present for data words with at most $2k + 3$ data.

Corollary

f_T is continuous iff it is continuous over X^ω with $|X| \geq 2k + 3$.

Continuity: Extend the Pattern of [Dave et al., 2019]

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 $2k + 3$ data.

Corollary

f_T is continuous iff it is continuous over X^ω with $|X| \geq 2k + 3$.

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

Extended Version with Nathan Lhote

The above results still hold:

- When we allow nondeterministic reassigment of data.
- Over data domain $(\mathbb{Q}, <)$, and more generally for oligomorphic data domains
- Over data domain $(\mathbb{N}, <)$

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.

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