Synthesis of Computable Regular Functions of Infinite Words

Vrunda Dave
IIT Bombay, India
vrunda@cse.iitb.ac.in

Emmanuel Filiot
Université Libre de Bruxelles, Belgium
efiliot@ulb.ac.be

Shankara Narayanan Krishna
IIT Bombay, India
krishnas@cse.iitb.ac.in

Nathan Lhote
MIMUW, University of Warsaw, Poland
nlhote@mimuw.edu.pl

Abstract

Regular functions from infinite words to infinite words can be equivalently specified by MSO-transducers, streaming ω-string transducers as well as deterministic two-way transducers with look-ahead. In their one-way restriction, the latter transducers define the class of rational functions. Even though regular functions are robustly characterised by several finite-state devices, even the subclass of rational functions may contain functions which are not computable (by a Turing machine with infinite input). This paper proposes a decision procedure for the following synthesis problem: given a regular function f (equivalently specified by one of the aforementioned transducer model), is f computable and if it is, synthesize a Turing machine computing it.

For regular functions, we show that computability is equivalent to continuity, and therefore the problem boils down to deciding continuity. We establish a generic characterisation of continuity for functions preserving regular languages under inverse image (such as regular functions). We exploit this characterisation to show the decidability of continuity (and hence computability) of rational and regular functions. For rational functions, we show that this can be done in NLogSpace (it was already known to be in PTime by Prieur). In a similar fashion, we also effectively characterise uniform continuity of regular functions, and relate it to the notion of uniform computability, which offers stronger efficiency guarantees.

2012 ACM Subject Classification Theory of computation → Transducers; Theory of computation → Automata over infinite objects; Theory of computation → Computability

Keywords and phrases transducers, infinite words, computability, continuity, synthesis

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.43

Related Version A full version of the paper is available at `https://arxiv.org/abs/1906.04199`.

Funding *Emmanuel Filiot*: This work is partially supported by the MIS project F451019F (F.R.S.-FNRS) and the EOS project Verifying Learning Artificial Intelligence Systems (F.R.S.-FNRS and FWO). Emmanuel Filiot is research associate at F.R.S.-FNRS.

1 Introduction

Let Inputs and $\mathsf{Outputs}$ be two arbitrary sets of elements called inputs and outputs respectively. A general formulation of the synthesis problem is as follows: for a given specification of a function $S\colon \mathsf{Inputs} \to 2^{\mathsf{Outputs}}$ relating any input $u \in \mathsf{dom}(S)$[1] to a set of outputs $S(u) \subseteq \mathsf{Outputs}$, decide whether there exists a (total) function $f\colon \mathsf{dom}(S) \to \mathsf{Outputs}$ such that (i) for all $u \in \mathsf{dom}(S)$, $f(u) \in S(u)$ and (ii) f satisfies some additional constraints such as being computable in some way, by some device which is effectively returned by the synthesis procedure. Assuming the axiom of choice, the relaxation of this problem without constraint (ii) always has a positive answer. However with additional requirement (ii), a function f realising S may not exist in general. In this paper, we consider the particular case where the specification S is *functional*,[2] in the sense that $S(u)$ is singleton set for all $u \in \mathsf{dom}(S)$. Even in this particular case, S may not be realisable while satisfying requirement (ii).

The latter observation on the functional case can already be made in the Church approach to synthesis [1, 18], for which $\mathsf{Inputs}, \mathsf{Outputs}$ are sets of infinite words and f is required to be implementable by a Mealy machine (a deterministic automaton which can output symbols). More precisely, an infinite word α over a finite alphabet Σ is a function $\alpha\colon \mathbb{N} \to \Sigma$ and is written as $\alpha = \alpha(0)\alpha(1)\ldots$. The set of infinite words over Σ is denoted by Σ^ω. In Church ω-regular synthesis, we have $\mathsf{Inputs} = \Sigma_i^\omega$ and $\mathsf{Outputs} = \Sigma_o^\omega$, and functions S are specified by ω-automata over $\Sigma_i.\Sigma_o$. Thus, such an automaton defines a language $L \subseteq (\Sigma_i.\Sigma_o)^\omega$ and in turn, through projection, a function S_L defined by $S_L(i_1 i_2 \ldots) = \{o_1 o_2 \cdots \mid i_1 o_1 i_2 o_2 \cdots \in L\}$. Such a specification is said to be synchronous, meaning that they alternatively read an input symbol and produce an output symbol deterministically. It is also ω-regular because it can be represented as an automaton over $\Sigma_i.\Sigma_o$. As an example, consider $\Sigma_i = \Sigma_o = \{a, b, @\}$ and the function S_{swap} defined only for all words of the form $u_1\sigma@u_2$ such that $u_1 \in \{a, b\}^*$ and $\sigma \in \{a, b\}$ by $S(u_1\sigma@u_2) = \sigma u_1@u_2$. The specification S is easily seen to be synchronous and ω-regular, but not realisable by any Mealy machine. This is because a Mealy machine is an input deterministic model and so cannot guess the last symbol before the @ symbol.

Computability of functions over infinite words. In Church synthesis, the notion of computability used for requirement (ii) is that of being computable by a Mealy machine. While this makes sense in a reactive scenario where output symbols (reactions) have to be produced immediately after input symbols are received, this computability notion is too strong in a more relaxed scenario where reactivity is not required. Instead, we propose here to investigate the synthesis problem for functional specifications over infinite words where the computability assumption (ii) for f is just being computable by some algorithm (formally a Turing machine) running on infinite inputs. In other words, our goal is to synthesize *algorithms* from specifications of functions of infinite words. There are classical computability notions for infinite objects, like infinite sequences of natural numbers, motivated by real analysis, or computation of functions of real numbers. The model of computation we consider for infinite words is a deterministic machine with 3 tapes : a read-only one-way tape holding the input, a two-way working tape with no restrictions and a write-only one-way output tape. All three tapes are infinite on the right. A function f is computable if there exists such a machine M such that, if its input tape is fed with an infinite word x in the domain of f, then M outputs longer and longer prefixes of $f(x)$ when reading longer and longer prefixes

[1] $\mathsf{dom}(S)$ is the domain of S, *i.e.* the set of inputs that have a non-empty image by S.
[2] In this case, we just write $S(u) = v$ instead of $S(u) = \{v\}$.

of x. This machine model has been defined in [25, Chap. 2]. If additionally one requires the existence of a computable function $m\colon \mathbb{N} \to \mathbb{N}$ such that for all $i \in \mathbb{N}$, for all infinite input x, M writes at least i output symbols when reading $m(i)$ input symbols of x, we obtain the notion of uniform computability. This offers promptness guarantees on the production of output symbols with respect to the number of symbols read on input.

Obviously, not all functions are computable. In this paper, we aim to solve the following synthesis problem: given a (finite) specification of a (partial) function f from infinite words to infinite words, is f computable (respectively uniformly computable)? If it is the case, then the procedure should return a Turing machine computing f (respectively uniformly computing f).

Examples. The function S_{swap} is computable. Since it is defined over all inputs containing at least one @ symbol, if a Turing machine is fed with such a word, it suffices for it to read its input until the first @ symbol is met, store in memory the symbol $\sigma \in \{a, b\}$ just before @, come back to the beginning of the tape and start producing the output infinite word $\sigma u_1 @ u_2$.

Over the alphabet $\Sigma = \{a, b\}$, consider the function f_∞ defined by $f_\infty(u) = a^\omega$ if u contains infinitely many as, and by b^ω otherwise. This simple function is not computable, as it requires to read the whole infinite input to produce even the very first output symbol. For any word $u \in \Sigma^*$, we denote by $\overline{u}$ its mirror (e.g. $\overline{abaa} = aaba$). Consider the (partial) function f_{mir} defined on $(\Sigma^*\sharp)^\omega$ by $f(u_1\sharp u_2\sharp\dots) = \overline{u_1}\sharp\overline{u_2}\sharp\dots$. It is computable by a machine that stores its input u_1 in memory until the first $\sharp$ is read, then outputs $\overline{u_1}$, and proceeds with u_2, and so on. It is however not uniformly computable. Indeed, if it were, with some $m\colon \mathbb{N} \to \mathbb{N}$, then, for inputs $u_1\sharp u_2\sharp\cdots$ such that $|u_1| > m(1)$, it is impossible to determine the first output symbol (which is the last of u_1) by reading only a prefix of length $m(1)$ of u_1.

Finally, consider the (partial) function f_{dbl} defined on $(\Sigma^*\sharp)^\omega$ by $f(u_1\sharp u_2\sharp\dots) = u_1u_1\sharp u_2u_2\sharp\dots$. Similarly as before, it is computable but also uniformly computable: to determine the ith output symbol, it suffices to read an input prefix of length at most i. Indeed, let $u_1\sharp u_2\sharp\dots\sharp u$ be a prefix of length i of the input x, then $u_1u_1\sharp u_2u_2\sharp\dots\sharp u$ is a prefix of $f(x)$ of length $\geq i$.

Computability and continuity. There are strong connections between computability and continuity: computable functions are continuous for the Cantor topology, that is where words are close to each other if they share a long common prefix. Intuitively, it is because the very definition of continuity asks that input words sharing longer and longer prefixes also share longer and longer output prefixes. It is the case of the functions f_{mir} and f_{dbl} seen before. Likewise, uniformly computable functions are uniformly continuous. The reverse direction does not hold in general: assuming an effective enumeration $M_1, M_2, \dots$ of Turing machines (on finite word inputs), the function f_{halt} defined as $f_{\mathsf{halt}}(a^\omega) = b_1b_2b_3\dots$ where $b_i \in \{0, 1\}$ is such that $b_i = 1$ iff M_i halts on input ϵ, is not computable but (uniformly) continuous (as it is defined on a single point).

Beyond synchronous functions: regular functions. functional specifications in Church ω-regular synthesis problem range over the class of *synchronous* functions as described before: they can be specified using automata over $\Sigma_i.\Sigma_o$. For example, while S_{swap} and f_∞ are synchronous, f_{mir} and f_{dbl} are not. In this paper, we intend to go much beyond this class by dropping the synchronicity assumption and consider the so-called class of regular functions. It is a well-behaved class, captured by several models such as streaming ω-string

transducers (SST), deterministic two-way Muller transducers with look around ($2\mathsf{DMT}_{\mathsf{la}}$), and also by MSO-transducers [2, Thm. 1, Prop. 1]. We propose the model of deterministic two-way transducers with a prophetic Büchi look-ahead ($2\mathsf{DFT}_{\mathsf{pla}}$) and show that they are equivalent to $2\mathsf{DMT}_{\mathsf{la}}$. This kind of transducer is defined by a *deterministic* two-way automaton without accepting states, extended with output words on the transitions, and which can consult another automaton, called the look-ahead automaton, to check whether an infinite suffix satisfies some regular property. We assume this automaton to be a prophetic Büchi automaton [7, Sec. 7], because this class is naturally suited to implement a regular look-ahead, while capturing all regular languages of infinite words. Look-ahead is necessary to capture functions such as f_∞. Two-wayness is needed to capture, for instance, functions f_{dbl} and f_{mir}.

Contributions. We call *effectively reg-preserving functions* those functions that effectively preserve regular languages by inverse image [24, 19, 23, 20]. This includes for instance rational functions, regular functions and the more general class of polyregular functions [4, 11]. We first show that for effectively reg-preserving functions, computability and continuity coincide, respectively, uniform computability and uniform continuity (Section 3, Theorem 6). To the best of our knowledge, this connection was not made before. The connection is effective, in the sense that when f is effectively reg-preserving and continuous (resp. uniformly continuous), we can effectively construct a Turing machine computing f (resp. uniformly computing f).

For rational functions (functions defined by non-deterministic one-way Büchi transducers), we show that continuity and uniform continuity are decidable in NLogSpace (Section 5, Theorem 12). Continuity and uniform continuity for rational functions were already known to be decidable in PTime, from Prieur [21, Prop. 4]. However, Prieur's proof techniques do not transfer to the two-way case. We then prove that continuity (and hence computability) is decidable for regular functions given by deterministic Büchi two-way transducers with look-ahead (Section 5, Theorem 16). Using our techniques, we also get the decidability of uniform continuity for regular functions (also Theorem 16). Our proof technique relies on a characterisation of non-continuous reg-preserving functions by the existence of pairs of sequences of words which have a nice regular structure (Section 4, Corollary 10). Based on this, we derive a decision procedure for continuity of rational functions by checking in NLogSpace a structural transducer pattern. For regular functions, we rely on a characterisation of the form of output words produced by idempotent loops in two-way transducers [3]. *Most of the proofs have been sketched and the full proofs can be found in full version [10].*

Related work. To the best of our knowledge, our results are new and the notion of continuity has not been extensively studied in the transducers literature over infinite words. The work by Prieur [21] is the closest to ours, while [8] looks at continuity of regular functions encoded by ω-automata.

Notions of continuity with respect to language varieties have been studied for rational functions of *finite words* in [5]. Our notion of uniform continuity can be linked to continuity with respect to a particular language variety which was *not* studied in [5] (namely the non-erasing variety generated by languages of the shape uA^*). A quite strong Lipschitz continuity notion, called *bounded variation* due to Choffrut (*e.g.* [9]), was shown to capture, over finite words, the sequential functions (the corresponding topology is however trivial, hence simple continuity is not very interesting in this context).

Another result connecting computability and continuity is from [6] where the authors find that some notion of computability by AC^0 circuits corresponds, over sequential functions, to continuity with respect to some language variety.

Our result on rational functions has been extended recently to rational functions of infinite words over an infinite alphabet in [12]. More precisely, continuity for functions of infinite data words defined by (one-way) transducers with registers has been shown to be decidable. The proof of [12] goes by reduction to the finite alphabet setting and uses the result presented in this paper, which is publicly available on Arxiv [10], to decide continuity.

Finally, a discussion and comparison of Church ω-synthesis with our work is given in the conclusion of this paper.

2 Languages, Automata and Transducers over ω-Words

Given a finite set Σ, we denote by Σ^* (resp. Σ^ω) the set of finite (resp. infinite) words over Σ, and by Σ^∞ the set of finite and infinite words. Let Σ^j represent the set of all words over Σ with length j. We denote by $|u| \in \mathbb{N} \cup \{\infty\}$ the length of $u \in \Sigma^\infty$ (in particular $|u| = \infty$ if $u \in \Sigma^\omega$). For a word $w = a_1a_2a_3\ldots$, $w[:j]$ denotes the prefix $a_1a_2\ldots a_j$ of w. Let $w[j]$ denote a_j, the j^{th} symbol of w and $w[j:]$ denote the suffix $a_{j+1}a_{j+2}\ldots$ of w. For a word w and $i \leq j$, $w[i:j]$ denotes the factor of w with positions from i to j, both included. For two words $u, v \in \Sigma^\infty$, $u \preceq v$ (resp. $u \prec v$) denotes that u is a prefix (resp. strict prefix) of v (in particular if $u, v \in \Sigma^\omega$, $u \preceq v$ iff $u = v$). For $u \in \Sigma^*$, let $\uparrow u$ denote the set of words $w \in \Sigma^\infty$ having u as prefix *i.e.* $u \preceq w$. Let $\mathsf{mismatch}$ be a function which takes two words, and returns a boolean value, denoted by $\mathsf{mismatch}(u, v)$ for u and v; it returns true if there exists a position $i \leq |u|, |v|$ such that $u[i] \neq v[i]$, and returns false otherwise. The longest common prefix between two words u and v is denoted by $u \wedge v$ and their distance is defined as $d(u, v) = 0$ if $u = v$, and $2^{-|u\wedge v|}$ if $u \neq v$.

A Büchi automaton is a tuple $B = (Q, \Sigma, \delta, Q_0, F)$ consisting of a finite set of states Q, a finite alphabet Σ, a set $Q_0 \subseteq Q$ of initial states, a set $F \subseteq Q$ of accepting states, and a transition relation $\delta \subseteq Q \times \Sigma \times Q$. A run ρ on a word $w = a_1a_2\ldots \in \Sigma^\omega$ starting in a state q_1 in B is an infinite sequence $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \ldots$ such that $(q_i, a_i, q_{i+1}) \in \delta$ for all $i \in \mathbb{N}$. Let $\mathsf{Inf}(\rho)$ denote the set of states visited infinitely often along ρ. The run ρ is a final run iff $\mathsf{Inf}(\rho) \cap F \neq \emptyset$. A run is *accepting* if it is final and starts from an initial state. A word $w \in \Sigma^\omega$ is accepted ($w \in L(B)$) iff it has an accepting run. A language L of ω-words is called *ω-regular* if $L = L(B)$ for some Büchi automaton B.

An automaton is co-deterministic if any two final runs on any word w are the same [7, Sec. 7.1]. Likewise, an automaton is co-complete if every word has at least one final run. A prophetic automaton $P = (Q_P, \Sigma, \delta_P, Q_0, F_P)$ is a Büchi automaton which is co-deterministic and co-complete. Equivalently, a Büchi automaton is prophetic iff each word admits a unique final run. The states of the prophetic automaton partition Σ^ω: each state q defines a set of words w such that w has a final run starting from q. For any state q, let $L(P, q)$ be the set of words having a final run starting at q. Then $\Sigma^\omega = \uplus_{q\in Q_P} L(P, q)$. It is known [7, Thm. 7.2] that prophetic automata capture ω-regular languages.

Transducers. We recall the definitions of one-way and two-way transducers over infinite words. A one-way transducer $\mathcal{A}$ is a tuple $(Q, \Sigma, \Gamma, \delta, Q_0, F)$ where Q is a finite set of states, Q_0, F respectively are sets of initial and accepting states; Σ, Γ respectively are the input and output alphabets; $\delta \subseteq (Q \times \Sigma \times Q \times \Gamma^*)$ is the transition relation. We equip $\mathcal{A}$ with a Büchi acceptance condition. A transition in δ of the form (q, a, q', γ) represents that from state q, on reading a symbol a, the transducer moves to state q', producing the output γ. Runs, final runs and accepting runs are defined exactly as in Büchi automata, with the addition that each transition produces some output $\in \Gamma^*$.

The output produced by a run ρ, denoted $\mathsf{out}(\rho)$, is obtained by concatenating the outputs generated by transitions along ρ. Let $\mathsf{dom}(\mathcal{A})$ represent the language accepted by the underlying automaton of $\mathcal{A}$, ignoring the outputs. The relation computed by $\mathcal{A}$ is defined as $[\![\mathcal{A}]\!] = \{(u, v) \in \Sigma^\omega \times \Gamma^\omega \mid u \in \mathsf{dom}(\mathcal{A}), \rho \text{ is an accepting run of } u, \mathsf{out}(\rho) = v\}$.[3] We say that $\mathcal{A}$ is functional if $[\![\mathcal{A}]\!]$ is a function. A relation (function) is *rational* iff it is recognised by a one-way (functional) transducer.

Two-way transducers extend one-way transducers and two-way finite state automata. A two-way transducer is a two-way automaton with outputs. In [2, Prop. 1], regular functions are shown to be those definable by a two-way deterministic transducer with Muller acceptance condition, along with a regular look-around ($2\mathsf{DMT}_{\mathsf{la}}$). In this paper, we propose an alternative machine model for regular functions, namely, $2\mathsf{DFT}_{\mathsf{pla}}$. A $2\mathsf{DFT}_{\mathsf{pla}}$ is a deterministic two-way automaton with outputs, along with a look-ahead given by a prophetic automaton.

Let $\Sigma_\vdash = \Sigma \uplus \{\vdash\}$. Formally, a $2\mathsf{DFT}_{\mathsf{pla}}$ is a pair $(\mathcal{T}, A)$ where $A = (Q_A, \Sigma, \delta_A, S_A, F_A)$ is a prophetic Büchi automaton and $\mathcal{T} = (Q, \Sigma, \Gamma, \delta, q_0)$ is a two-way transducer s.t. Σ and Γ are finite input and output alphabets, Q is a finite set of states, $q_0 \in Q$ is a unique initial state, $\delta \colon Q \times \Sigma_\vdash \times Q_A \to Q \times \Gamma^* \times \{-1, +1\}$ is a partial transition function. $\mathcal{T}$ has no acceptance condition: every infinite run in $\mathcal{T}$ is a final run. A two-way transducer stores its input $\vdash a_1 a_2 \dots$ on a two-way tape, and each index of the input can be read multiple times. A configuration of a two-way transducer is a tuple $(q, i) \in Q \times \mathbb{N}$ where $q \in Q$ is a state and $i \in \mathbb{N}$ is the current position on the input tape. The position is an integer representing the gap between consecutive symbols. Thus, before $\vdash$, the position is 0, between $\vdash$ and a_1, the position is 1, between a_i and a_{i+1}, the position is $i + 1$ and so on. The $2\mathsf{DFT}_{\mathsf{pla}}$ is deterministic: for every word $w = \vdash a_1 a_2 a_3 \dots \in \vdash\Sigma^\omega$, every input position $i \in \mathbb{N}$, and state $q \in Q$, there is a unique state $p \in Q_A$ such that $a_i a_{i+1} \dots \in L(A, p)$. Given $w = a_1 a_2 \dots$, from a configuration (q, i), on a transition $\delta(q, a_i, p) = (q', \gamma, d)$, $d \in \{-1, +1\}$, such that $a_i a_{i+1} \dots \in L(A, p)$, we obtain the configuration $(q', i + d)$ and the output γ is appended to the output produced so far. This transition is denoted as $(q, i) \xrightarrow{a_i, p/\gamma} (q', i + d)$. A run ρ of a $2\mathsf{DFT}_{\mathsf{pla}}$ $(\mathcal{T}, A)$ is a sequence of transitions $(q_0, i_0 = 0) \xrightarrow{a_{i_0}, p_1/\gamma_1} (q_1, i_1) \xrightarrow{a_{i_1}, p_2/\gamma_2} \cdots$. The output of ρ, denoted $\mathsf{out}(\rho)$ is then $\gamma_1 \gamma_2 \cdots$. The run ρ reads the whole word w if $\mathsf{sup}\{i_n \mid 0 \le n < |\rho|\} = \infty$. The output $[\![(\mathcal{T}, A)]\!](w)$ of a word w on run ρ is defined only when $\mathsf{sup}\{i_n \mid 0 \le n < |\rho|\} = \infty$, and equals $\mathsf{out}(\rho)$. $2\mathsf{DFT}_{\mathsf{pla}}$ are equivalent to $2\mathsf{DMT}_{\mathsf{la}}$, and capture all regular functions (see full version [10] for the proof).

▶ **Theorem 1.** *A function $f \colon \Sigma^\omega \to \Gamma^\omega$ is regular iff it is $2\mathsf{DFT}_{\mathsf{pla}}$ definable.*

▶ **Example 2.** Consider the function $g \colon \Sigma^\omega \to \Gamma^\omega$ over $\Sigma = \Gamma = \{a, b\}$ such that $g(uab^\omega) = uub^\omega$ for $u \in \Sigma^*$ and $g(b^\omega) = b^\omega$. The $2\mathsf{DFT}_{\mathsf{pla}}$ is shown in Figure 1 with the prophetic look-ahead automaton A on the right. The transitions are decorated as $\alpha, p \mid \gamma, d$ where $\alpha \in \{a, b\}$, p is a state of A, γ is the output and d is the direction. In transitions not using the look-ahead information, the decoration is simply $\alpha \mid \gamma, d$. Notice that $\mathcal{L}(A, p_1) = \vdash\Sigma^* ab^\omega$, $\mathcal{L}(A, p_2) = \vdash b^\omega$, $\mathcal{L}(A, p_3) = \Sigma^+ ab^\omega$, $\mathcal{L}(A, p_4) = ab^\omega$. Each word in $\vdash\Sigma^\omega$ has a unique final run; $\mathcal{L}(A, p_1) \cup \mathcal{L}(A, p_2) = \vdash\mathsf{dom}(g)$. The remaining states ensure that each word in $(\vdash\Sigma^\omega \setminus \vdash\mathsf{dom}(g)) \uplus \Sigma^\omega$ has a unique final run. ⌟

[3] We assume that final runs always produce infinite words, which can be enforced syntactically by a Büchi condition such that any input word produces non-empty output in a single loop execution containing Büchi accepting state.

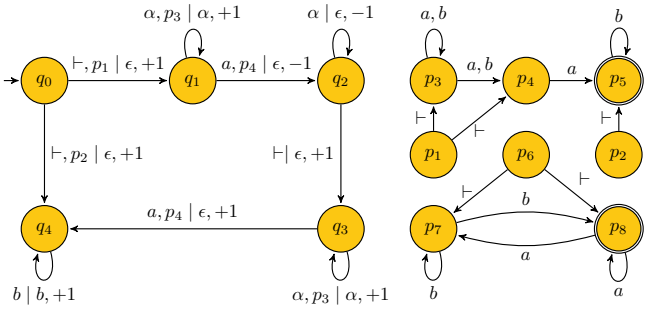

Figure 1 A $2\mathsf{DFT}_{\mathsf{pla}}$ with automaton on the right implementing the look-ahead.

We also use a look-ahead-free version of two-way transducers in some of the proofs, where we also have a Büchi acceptance condition given by a set of states F, just as for Büchi automata. The resulting model is called two-way deterministic Büchi transducer (2DBT). The definitions of configuration, run, and the semantics are done just like for $2\mathsf{DFT}_{\mathsf{pla}}$.

3 Computability versus Continuity

Computability of a function on infinite words can be described intuitively in the following way: there is an algorithm which, given access to the input word, can enumerate the letters in the output word. We also investigate a stronger notion of computability, which we call *uniform computability*. The main idea is that given some input word x and some position j one can compute the j^{th} position of the output in time that depends on j but not on x. An appealing aspect of uniform computability is that it offers a uniform bound on the number of input symbols one needs to read in order to produce the output at some fixed precision.

▶ **Definition 3** (Computability/Uniform computability). *A function $f\colon \Sigma^\omega \to \Gamma^\omega$ is* computable *if there exists a deterministic multitape Turing machine M computing it in the following sense. The machine M has a read-only one-way input tape, a two-way working tape, and a write-only one-way output tape. All tapes have a left delimiter $\vdash$ and are infinite to the right. Let $x \in \mathsf{dom}(f)$. For any $j \in \mathbb{N}$, let $M(x, j)$ denote the output produced by M till the time it moves to the right of position j, onto position $j+1$ in the input (or ϵ if this move never happens). The function f is computable by M if for all $x \in \mathsf{dom}(f)$, for all $i \geq 0$, there exists $j \geq 0$ such that $f(x)[:i] \preceq M(x, j)$.*

Moreover if there exists a computable function $m\colon \mathbb{N} \to \mathbb{N}$ (called a modulus of continuity *for M) such that for all $x \in \mathsf{dom}(f)$, for all $i \geq 0$, $f(x)[:i] \preceq M(x, m(i))$, f is called* uniformly computable.

It turns out that there is a quite strong connection between computability and continuity of functions. In particular computable functions are always continuous. This can be seen intuitively since given a deterministic Turing machine, it must behave the same on the common prefixes of two words. Hence two words with a very long common prefix must have images by the machine that have a somewhat long common prefix. This connection also transfers to uniform computability and uniform continuity. We start by formally defining continuity and uniform continuity. We interchangeably use the following two definitions [22] of continuity.

▶ **Definition 4** (Continuity/Uniform continuity)**.**

1. *A function* $f\colon \Sigma^\omega \to \Gamma^\omega$ *is continuous at* $x \in \mathsf{dom}(f)$ *if (equivalently)*
   - **(a)** *for all* $(x_n)_{n\in\mathbb{N}}$ *converging to* x*, where* $x_i \in \mathsf{dom}(f)$ *for all* $i \in \mathbb{N}$*,* $(f(x_n))_{n\in\mathbb{N}}$ *converges.*
   - **(b)** $\forall i \geq 0\ \exists j \geq 0\ \forall y \in \mathsf{dom}(f),\ |x \wedge y| \geq j \Rightarrow |f(x) \wedge f(y)| \geq i$
2. *A function is continuous if it is continuous at every* $x \in \mathsf{dom}(f)$*.*
3. *A function* $f\colon \Sigma^\omega \to \Gamma^\omega$ *is uniformly continuous if:*
   there exists $m\colon \mathbb{N} \to \mathbb{N}$*, called a* modulus of continuity *for* f *such that,*
   $\forall i \geq 0,\ \forall x, y \in \mathsf{dom}(f),\ |x \wedge y| \geq m(i) \Rightarrow |f(x) \wedge f(y)| \geq i$*.*

▶ **Example 5.** As explained in the introduction, the function f_∞ is not continuous, and, as we will see later, is thus not computable. The function f_{halt} is continuous, even uniformly continuous (it is constant) yet is obviously not computable. The function f_{mir} is computable, however is not uniformly continuous, two words can be arbitrarily close but with far away outputs: consider $a^n\sharp^\omega$ and $a^n b\sharp^\omega$. Finally, the function f_{dbl} is uniformly computable. ⌟

We now investigate the relationship between continuity and computability for functions that are effectively reg-preserving. More precisely, we say that a function $f\colon \Sigma^\omega \to \Gamma^\omega$ is *effectively reg-preserving* if there is an algorithm which, for any automaton recognizing a regular language $L \subseteq \Gamma^\omega$, produces an automaton recognizing the language $f^{-1}(L) = \{u \mid f(u) \in L\}$.

Two well-studied classes (see *e.g.* [13]) of reg-preserving functions are the rational and the regular functions, which we will study in Section 5. As announced, continuity and computability coincide for effectively reg-preserving functions:

▶ **Theorem 6.** *An effectively reg-preserving function* $f\colon \Sigma^\omega \to \Gamma^\omega$ *is computable (resp. uniformly computable) if and only if it is continuous (resp. uniformly continuous).*

Proof. $\Rightarrow$) This implication is easy and actually holds without the reg-preserving assumption. If f is computable by some machine M, then it is not difficult to see that it is continuous. Intuitively, the longer the prefix of input $x \in \mathsf{dom}(f)$ is processed by M, the longer the output produced by M on that prefix, which converges to $f(x)$, according to the definition of computability. More details of this proof can be found in full version. Moreover, if f is uniformly computable, then the modulus of continuity of M is in particular a modulus of continuity for f and is thus uniformly continuous.

Algorithm 1 Algorithm describing M.

```
   Input: x ∈ Σ^ω
1  out := ϵ ;   ;                          // this is written on the working tape
2  for i = 0 to +∞ do
3  |  for γ ∈ Γ do
4  |  |  if f(↑x[:i]) ⊆ ↑out.γ then
5  |  |  |  out := out.γ ; ;                        // append to the working tape
6  |  |  |  output γ ;   ;              // this is written on the output tape
```

$\Leftarrow$) The converse direction is less trivial and makes use of the reg-preserving assumption. Suppose that f is continuous. We design the machine M, represented as Algorithm 1, which is shown to compute f. This machine processes longer and longer prefixes $x[:i]$ of its input x (for loop at line 2), and tests (line 4) whether a symbol γ can be safely appended to the output. The test ensures that the invariant $\mathsf{out} \preceq f(x)$ is preserved at any point. Moreover, the continuity of f at x ensures that out is updated infinitely often. The only thing left to

obtain computability is that the test of line 4 is decidable. Let u and v be two words, deciding $f(\uparrow u) \subseteq \uparrow v$ is equivalent to deciding if $\mathsf{dom}(f) \cap \uparrow u \subseteq f^{-1}(\uparrow v)$. These sets are effectively regular since u, v are given and f is effectively reg-preserving, and $\mathsf{dom}(f) = f^{-1}(\Gamma^\omega)$. Since the constructions are effective, and the languages are regular, the inclusion is decidable.

We only have left to show that if f is moreover uniformly continuous, then M has a computable modulus of continuity. We start by showing that f has a computable modulus of continuity. Let us consider the predicate $P(i,j)\colon \forall x, y\ |x \wedge y| \geq j \Rightarrow |f(x) \wedge f(y)| \geq i$. Then we define $m\colon i \mapsto \min\{j \mid P(i,j)\}$. Since f is uniformly continuous, m is indeed well defined and is a modulus of continuity of f. To show that m is computable, we only have to show that $P(i,j)$ is decidable.

Let us consider the negation of $P(i,j)$: there exist $u, x_1, x_2, v_1, v_2, w_1, w_2$ such that $|u| = j$, and $f(ux_k) = v_k w_k$ for $k \in \{1,2\}$ with $|v_1| = |v_2| = i$ and $v_1 \neq v_2$. Hence to decide $\neg P(i,j)$, we only have to find two words $v_1 \neq v_2$ in Γ^i, such that $S \neq \emptyset$ where $S = \{vw \mid v \in \Sigma^j,\ \exists w_1, w_2,\ \text{s.t. } vw_1 \in f^{-1}(\uparrow v_1),\ vw_2 \in f^{-1}(\uparrow v_2)\}$. Since f is effectively reg-preserving, S is effectively regular. By searching exhaustively for words $v_1, v_2 \in \Gamma^i$ we get decidability of $P(i,j)$. We only have left to define a modulus of continuity for M. Let $m'\colon \mathbb{N} \to \mathbb{N}$ be defined by $m'(i) = m(i) + i$. If we read $m(i)$ symbols, we know we can output at least i symbols. Hence in each of the next i steps, we are guaranteed to output a letter. Hence m' is a modulus of continuity for M and f is uniformly computable. ◀

▶ Remark 7. Note that we focus on functions that are effectively reg-preserving, but Algorithm 1 is actually more general than that. The continuity-computability equivalence indeed carries over to any class of functions for which the test in line 4 is decidable.

4 A Characterisation of Continuity and Uniform Continuity

We provide here a characterisation of continuity (and uniform continuity) for reg-preserving functions (we don't need effectiveness here). The characterisation is based on a study of some particular properties of sequences and pairs of sequences which we define below:

▶ **Definition 8.** *Let $f\colon \Sigma^\omega \to \Gamma^\omega$.*

Let $(x_n)_{n\in\mathbb{N}}$ be a sequence of words in $\mathsf{dom}(f)$ converging to $x \in \Sigma^\omega$, such that $(f(x_n))_{n\in\mathbb{N}}$ is not convergent. Such a sequence is called a bad sequence *at x for f.*

Let $(x_n)_{n\in\mathbb{N}}$ and $(x'_n)_{n\in\mathbb{N}}$ be two sequences in $\mathsf{dom}(f)$ both converging to $x \in \Sigma^\omega$, such that either $(f(x_n))_{n\in\mathbb{N}}$ is not convergent, $(f(x'_n))_{n\in\mathbb{N}}$ is not convergent, or $\lim_n f(x_n) \neq \lim_n f(x'_n)$. Such a pair of sequences is called a bad pair *of sequences at x for f.*

A pair of sequences is synchronised *if it is of the form: $((uv^n wz^\omega)_n, (uv^n w' z'^\omega)_n)$*

▶ **Proposition 9.** *A function is* not *continuous if and only if it has a bad pair at some point of its domain. A function is* not *uniformly continuous if and only if it has a bad pair.*

Proof. The case of continuous functions is obtained just by definition. For uniform continuity, consider a function f with a bad pair $((x_n)_{n\in\mathbb{N}}, (x'_n)_{n\in\mathbb{N}})$, and let us show that it is not uniformly continuous. We can assume that both $(f(x_n))_{n\in\mathbb{N}}$ and $(f(x'_n))_{n\in\mathbb{N}}$ converge. Otherwise we can extract subsequences that converge, by compactness of Γ^ω. Moreover, since the pair is bad, one can assume that they converge to different limits $y \neq y'$. Let i be such that $y[i] \neq y'[i]$. For any j, one can find N such that for all $n \geq N$, $|x_n \wedge x'_n| \geq j$ since both sequences converge to x. Since $(f(x_n))_{n\in\mathbb{N}}$ converges to y, we can ensure that N is large enough so that for all $n \geq N$, $|f(x_n) \wedge y| \geq i$. We can also ensure that for $n \geq N$, $|f(x'_n) \wedge y'| \geq i$ holds. Let $n \geq N$, we have both $|x_n \wedge x'_n| \geq j$ and $|f(x_n) \wedge f(x'_n)| < i$, which means that f is not uniformly continuous.

Let f be a function which is not uniformly continuous, we want to exhibit a bad pair of f. According to the definition, there exists i such that for all j there exist x_j, x'_j with $|x_j \wedge x'_j| \geq j$ but $|f(x_j) \wedge f(x'_j)| < i$. By compactness of Σ^ω, there exists a subsequence of $(x_j)_{j\in\mathbb{N}}$ which is convergent. Let $\left(x_{\tau(j)}\right)_{j\in\mathbb{N}}$, with $\tau\colon \mathbb{N} \to \mathbb{N}$ increasing, denote such a subsequence. Then we have for all j that $|x_{\tau(j)} \wedge x'_{\tau(j)}| \geq \tau(j) \geq j$ and $|f(x_{\tau(j)}) \wedge f(x'_{\tau(j)})| < i$. Therefore up to renaming the sequences, we can assume that for all j, $|x_j \wedge x'_j| \geq j$ and $|f(x_j) \wedge f(x'_j)| < i$, with $(x_j)_{j\in\mathbb{N}}$ being convergent. By repeating the process of extracting subsequences, we can assume that $\left(x'_j\right)_{j\in\mathbb{N}}$, $(f(x_j))_{j\in\mathbb{N}}$, $\left(f(x'_j)\right)_{j\in\mathbb{N}}$ are also convergent. Since for any j, $|x_j \wedge x'_j| \geq j$, the two sequences converge to the same limit. In the end we obtain that $\left((x_j)_{j\in\mathbb{N}}, \left(x'_j\right)_{j\in\mathbb{N}}\right)$ is a bad pair for f at $\lim_j x_j = \lim_j x'_j$.

Note that in case $\mathsf{dom}(f)$ is not compact, then we may not have a subsequence of $(x_j)_{j\in\mathbb{N}}$ which converges in $\mathsf{dom}(f)$. However, the definition of bad pairs does not require convergence in the domain; it only asks for convergence to some x, which need not be in $\mathsf{dom}(f)$. ◀

The main result of this section is the following lemma which says that one can restrict to considering only *synchronised* bad pairs. In the following sections this characterisation will be used to decide continuity/uniform continuity.

▶ **Lemma 10** (Characterisation). *A reg-preserving function is not continuous if and only if it has a synchronised bad pair at some point of its domain. A reg-preserving function is not uniformly continuous if and only if it has a synchronised bad pair.*

Sketch of Proof. This lemma extends Proposition 9. It shows that, in the case of reg-preserving functions, one can restrict to considering synchronised pairs, which are much easier to deal with. The proof is done in several steps but due to a lack of space, we only sketch these steps, the full proof being given in full version [10].

First we show that for a reg-preserving function f, if there is a bad pair at some x, then there is one at some *regular* z, *i.e.* $z = uv^\omega$ for some finite words u, v. Moreover, for the case of non-uniform continuity, we show that z can be chosen so that $x \in \mathsf{dom}(f) \Leftrightarrow z \in \mathsf{dom}(f)$.

In the second step, since we have two sequences converging to regular z, we show how to replace the bad pair by a bad pair of *regular* sequences, still using the fact that f is reg-preserving. Finally, we prove that we can *synchronise* these two regular sequences and end up with a synchronised bad pair at z. ◀

5 Deciding Continuity and Uniform Continuity

We first show how to decide (uniform) continuity for rational and then for regular functions.

Rational case. We exhibit structural patterns which are shown to be satisfied by a one-way Büchi transducer iff the rational function it defines is not continuous (resp. not uniformly continuous). We express those patterns in the *pattern logic* defined in [15, Sec. 6], which is based on existential run quantifiers of the form $\exists \pi\colon p \xrightarrow{u|v} q$ where π is a run variable, p, q are state variables and u, v are word variables. Intuitively, there exists a run π from state p to state q on input u, producing output v. A one-way transducer is called *trim* if each of its states appears in some accepting run. Any one-way Büchi transducer can be trimmed in polynomial time. The structural patterns for trim transducers are given in Figures 2 and 3. The predicate $\mathsf{init}(p)$ expresses that p is initial while $\mathsf{acc}(p)$ expresses that it is accepting. The predicate $\mathsf{mismatch}$ expresses the existence of a mismatch between two words, as defined in Section 2.

$$\phi_{\text{cont}} = \begin{array}{l} \exists \pi_1 \colon p_1 \xrightarrow{u|u_1} q_1,\ \exists \pi_1' \colon q_1 \xrightarrow{v|v_1} q_1 \\ \exists \pi_2 \colon p_2 \xrightarrow{u|u_2} q_2,\ \exists \pi_2' \colon q_2 \xrightarrow{v|v_2} q_2,\ \exists \pi_2'' \colon q_2 \xrightarrow{w|w_2} r_2 \\ \big(\mathsf{init}(p_1) \wedge \mathsf{init}(p_2) \wedge \mathsf{acc}(q_1)\big) \wedge \\ \big(\mathsf{mismatch}(u_1, u_2) \vee (v_2 = \epsilon \wedge \mathsf{mismatch}(u_1, u_2 w_2))\big) \end{array}$$

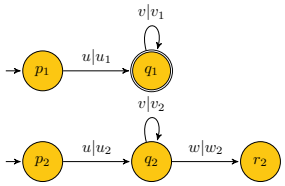

Figure 2 Pattern characterising non-continuity of rational functions given by *trim* one-way Büchi transducers.

$$\phi_{\text{u-cont}} = \begin{array}{l} \exists \pi_1 \colon p_1 \xrightarrow{u|u_1} q_1,\ \exists \pi_1' \colon q_1 \xrightarrow{v|v_1} q_1,\ \exists \pi_1'' \colon q_1 \xrightarrow{w|w_1} r_1 \\ \exists \pi_2 \colon p_2 \xrightarrow{u|u_2} q_2,\ \exists \pi_2' \colon q_2 \xrightarrow{v|v_2} q_2,\ \exists \pi_2'' \colon q_2 \xrightarrow{w|w_2} r_2 \\ \big(\mathsf{init}(p_1) \wedge \mathsf{init}(p_2)\big) \wedge \\ \big(\mathsf{mismatch}(u_1, u_2) \vee (v_1 = \epsilon \wedge \mathsf{mismatch}(u_1 w_1, u_2)) \\ \qquad \vee (v_1 = v_2 = \epsilon \wedge \mathsf{mismatch}(u_1 w_1, u_2 w_2))\big) \end{array}$$

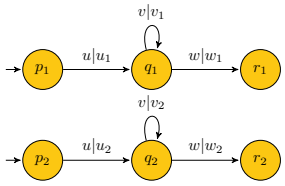

Figure 3 Pattern characterising non-*uniform* continuity of rational functions given by *trim* one-way Büchi transducers.

▶ **Lemma 11.** *A trim one-way Büchi transducer defines a non-continuous (resp. non-uniformly continuous) function if and only if it satisfies the formula ϕ_{cont} of Fig. 2 (resp. the formula $\phi_{u\text{-}cont}$ of Fig. 3).*

Sketch of Proof. Showing that the patterns of Figure 2 and Figure 3 induce non-continuity and non-uniform continuity, respectively, is quite simple. Indeed, the first pattern ϕ_{cont} is a witness that $(uv^n wz)_{n\in\mathbb{N}}$ is a bad sequence at a point uv^ω of its domain, for z a word with a final run from r_2, which entails non-continuity by Proposition 9 (if a sequence s is bad then (s, s) is bad). Similarly, the pattern $\phi_{\text{u-cont}}$ witnesses that the pair $\left((uv^n wz)_{n\in\mathbb{N}}, (uv^n w' z')_{n\in\mathbb{N}}\right)$ is synchronised and bad (with z, z' words that have a final run from r_1, r_2, respectively), which entails non-uniform continuity by Lemma 10.

For the other direction, we again use Lemma 10. From a synchronised bad pair, we can find a pair of runs with a synchronised loop, such that iterating the loop does not affect the existing mismatch between the outputs of the two runs, which is in essence what the pattern formulas of Figure 2 and Figure 3 state. The full proof is available in full version [10]. ◀

▶ **Theorem 12.** *Deciding if a one way Büchi transducer defines a continuous (resp. uniformly continuous) function can be done in NLogSpace.*

Proof. Let T be a one way Büchi transducer defining a function f. From Lemma 11, if T is trim, non-continuity of f is equivalent to T satisfying the formula ϕ_{cont} of Fig. 2. This formula is expressed in the syntax of the pattern logic from [15], where it is proved that model-checking pattern formulas against transducers can be done in NLogSpace [15, Thm. 6]. This yields the result.

If T is not trim, then we modify the formula ϕ_{cont} to additionally express that there must be some accepting run from r_2 on some input. Equivalently, we express that there exists a run from r_2 to some accepting state s, and a run looping in s, in the following way: we just add the quantifiers $\exists \pi_3 \colon r_2 \xrightarrow{\alpha|\beta} s\ \exists \pi_4 \colon s \xrightarrow{\gamma|\tau} s$ to ϕ_{cont} and the constraint $\mathsf{acc}(s)$ which requires s to be accepting.

The proof for non-uniform continuity, using formula $\phi_{\text{u-cont}}$ is similar. ◀

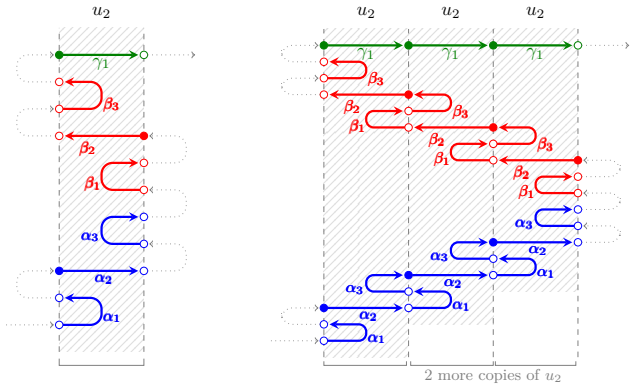

Figure 4 Pumping u_2. $\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3, \gamma_1$ are the outputs seen on u_2. The blue, red and green arrows are part of the run r while reading u_2.

Regular case. The case of regular functions is more intricate. We have to exploit the form of the output words produced by idempotent loops of two-way transducer runs. Idempotent loops always exist for sufficiently long inputs and indeed have a nice structure which allows one to characterise the form of the output words produced when iterating such loops [3]. A detailed definition of idempotent loops, based on the traversal monoid is in [3]. We have abstracted the main property of idempotent loops which is sufficient in our context, and for which it is not necessary to know the precise definition of idempotency. So, given a deterministic two-way transducer T on finite words (we need the notion only for finite words) and an input word $u_1u_2u_3$, we will say that u_2 is idempotent in (u_1, u_2, u_3) (or just idempotent when u_1, u_3 are clear from the context), if in the run r of T on $u_1u_2u_3$, the restriction of r to u_2 (which is a sequence of possibly disconnected runs on u_2) is idempotent *i.e.* we can pump u_2 any number of times in the context of u_1 and u_3 and still get a valid run of T [3]. See Figure 4, if the sequence of states visited before reading first position of u_2 (at first vertical dashed line in figure) and the sequence of states visited after reading u_2 (at second vertical dashed line), we can pump u_2, in other words concatenate the run shown in left to itself multiple times. In right side figure, the partial run is shown where u_2 is concatenated with itself twice. Observe that the outputs on the factors that is shown on each blue, red and green arrow remain same and the output words are concatenated in pumped word in a systematic manner.

Given a language of ω-words $L \subseteq \Sigma^\omega$, we denote by $\mathrm{Pref}(L)$ the set of finite prefixes of words in L, *i.e.* $\mathrm{Pref}(L) = \{u \in \Sigma^* \mid \exists v \in L, \text{ with } u \preceq v\}$. In order to deal with look-aheads more easily, we remove look-aheads by considering words annotated with look-ahead information. Given a $\mathsf{2DFT_{pla}}$ $(\mathcal{T}, P)$ over alphabet Σ and with a set of look-ahead states Q_P, realising a function f, we define $\widetilde{\mathcal{T}}$, a $\mathsf{2DBT}$ over $\Sigma \times Q_P$ which simulates $(\mathcal{T}, P)$ over words annotated with look-ahead states, and which accepts only words with a correct look-ahead annotation with respect to P (the formal definition can be found in full version [10]). We denote by $\tilde{f}$ the function it realises, in particular for all words $u \in \mathsf{dom}(f)$, there exists a unique annotated word $\tilde{u} \in \mathsf{dom}(\tilde{f})$ such that $\tilde{f}(\tilde{u}) = f(u)$, as P is prophetic. For any annotated word $\tilde{u}$, $\pi(\tilde{u}) = u$ stands for its Σ-projection.

From $\widetilde{\mathcal{T}}$, we define $\mathcal{T}_*$, a deterministic two-way transducer of finite words over the input alphabet $\Sigma \times Q_P$. Its domain is restricted to $\mathrm{Pref}(\mathsf{dom}(\tilde{f}))$ (which is a regular set) and it behaves just as $\widetilde{\mathcal{T}}$ until it reaches the right border of its input for the first time, after which it

accepts iff the input was indeed a prefix of $\mathsf{dom}(\tilde{f})$ which, as said before, is a regular property which can be checked by $\mathcal{T}_*$ while simulating $\widetilde{\mathcal{T}}$. We let f_* be the function realised by $\mathcal{T}_*$ (which depends on $\mathcal{T}$). We have that, for any infinite word $x \in \mathsf{dom}(\tilde{f})$, $\tilde{f}(x) = \lim_{u \prec x} f_*(u)$.

The following lemma is a first characterisation of non-continuity which we can get by exploiting the existence of synchronised bad pairs.

▶ **Lemma 13.** *Let $f\colon \Sigma^\omega \to \Gamma^\omega$ be a regular function defined by some deterministic two-way transducer $\mathcal{T}$ with look-ahead and let Q_P be the set of look-ahead states. Then f is not continuous (resp. uniformly continuous) iff there exist finite words $u_1, u_1', u_2, u_2', u_3, u_3' \in (\Sigma \times Q_P)^*$ such that $u_1u_2u_3, u_1'u_2'u_3' \in \mathsf{dom}(f_*)$ and*

1. *$\pi(u_1) = \pi(u_1')$, $\pi(u_2) = \pi(u_2')$, and $x = \pi(u_1)\pi(u_2)^\omega \in \mathsf{dom}(f)$ (resp. $x \in \Sigma^\omega$),*
2. *u_2 and u_2' are idempotent in (u_1, u_2, u_3) and (u_1', u_2', u_3') respectively (for $\mathcal{T}_*$),*
3. *there exists i such that for all $n \geq 1$, $f_*(u_1u_2^nu_3)[i] \neq f_*(u_1'{u_2'}^nu_3')[i]$.*

Sketch of Proof. For the if direction, since $\mathsf{dom}(f_*) = \mathrm{Pref}(\mathsf{dom}(\tilde{f}))$, for any n there are some $u_{4,n}, u_{4,n}'$ such that the words $x_n = u_1u_2^nu_{4,n}$ and $x_n' = u_1'{u_2'}^nu_{4,n}'$ are both in $\mathsf{dom}(\tilde{f})$. Moreover, the sequences $(\pi(x_n))_{n\in\mathbb{N}}$ and $(\pi(x_n'))_{n\in\mathbb{N}}$ both converge to x. However, since there is a mismatch between $f_*(u_1u_2^nu_3)$ and $f_*(u_1'{u_2'}^nu_3')$ at position i, and by definition of f_* we have $f_*(u_1u_2^nu_3) \preceq \tilde{f}(x_n)$ and $f_*(u_1'{u_2'}^nu_3') \preceq \tilde{f}(x_n')$, there is also one between $\tilde{f}(x_n)$ and $\tilde{f}(x_n')$ at position i. Thus the pair $\left((\pi(x_n))_{n\in\mathbb{N}}, (\pi(x_n'))_{n\in\mathbb{N}}\right)$ is a bad pair and we conclude by Lemma 10.

In the other direction, as for the rational case, we start from Lemma 10 stating that it suffices to check for a synchronised bad pair. Like in the rational case, we successively extract subsequences of the synchronised bad pair and at each step we need to preserve synchronicity as well as badness. The main idea is that if we iterate enough times the loop in the synchronised bad pair, we will end up with synchronised idempotent loops. The more detailed version is available in [10]. ◀

Given a deterministic two-way transducer T (*i.e.* with a trivial look-ahead) defining a function f and words $u_1, u_2, u_3 \in \Sigma^*$ such that $u_1u_2u_3 \in \mathrm{Pref}(\mathsf{dom}(T))$ and u_2 is idempotent for T, we say that u_2 is "producing" in (u_1, u_2, u_3) if the run of T on $u_1u_2u_3$ produces some output when reading at least one symbol of u_2, at some point in the run. If u_2 is producing, then $|f_*(u_1u_2^iu_3)| < |f_*(u_1u_2^{i+1}u_3)|$ for all $i \geq 1$.

Our goal is now to give another characterisation of (non-) continuity, which replaces the quantification on n in Lemma 13 by a property which does not need iteration, and therefore which is more amenable to an algorithmic check. It is based on the following key result.

▶ **Lemma 14.** *Let Σ be an alphabet such that $\# \notin \Sigma$. Let $g\colon \Sigma^\omega \to \Gamma^\omega$ be a regular function defined by some deterministic two-way transducer U. There exists a function $\rho_U\colon (\Sigma^*)^3 \to \Gamma^*$ defined on all tuples (u_1, u_2, u_3) such that u_2 is idempotent and $u_1u_2u_3 \in \mathsf{Pref}(\mathsf{dom}(g))$, and which satisfies the following conditions:*

1. *if u_2 is producing in (u_1, u_2, u_3), then $\rho_U(u_1, u_2, u_3) \prec \rho_U(u_1u_2, u_2, u_2u_3)$*
2. *for all $n \geq 1$, $\rho_U(u_1, u_2, u_3) \preceq g_*(u_1u_2^nu_3)$*
3. *for all $n \geq 1$, $\rho_U(u_1, u_2, u_3) = g_*(u_1u_2^nu_3)$ if u_2 is* not *producing in (u_1, u_2, u_3)*
4. *the finite word function $\rho_U'\colon u_1\#u_2\#u_3 \mapsto \rho_U(u_1, u_2, u_3)$ is (effectively) regular.*

Proof. The proof of Lemma 14 is based on a thorough study of the form of the output words produced by idempotent loops [3]. The whole proof, which requires technical notions, can be found in full version [10]. ◀

Note that the previous lemma is stated for transducers without look-ahead. It is however sufficient as we apply it to transducers of the form $\widetilde{\mathcal{T}}$. In particular, we use this lemma to characterise the continuity of a function defined by a $2\mathsf{DFT}_{\mathsf{pla}}$ $\mathcal{T}$ by using the function $\rho_{\widetilde{\mathcal{T}}}$.

Unlike in Lemma 13, in the following characterisation, we do not need to iterate the loop to check existence of a mismatch for all iterations, as we just need to inspect $\rho_{\widetilde{\mathcal{T}}}(u_1, u_2, u_3)$.

▶ **Lemma 15.** *Let $f\colon \Sigma^\omega \to \Gamma^\omega$ be a function defined by some deterministic two-way transducer $\mathcal{T}$ with look-ahead and let Q_P be the set of look-ahead states. The function f is not continuous (resp. not uniformly continuous) iff there exist $u_1, u_1', u_2, u_2', u_3, u_3' \in (\Sigma \times Q_P)^*$ such that $u_1u_2u_3, u_1'u_2'u_3' \in \mathsf{dom}(f_*)$ and*

1. *$\pi(u_1) = \pi(u_1')$, $\pi(u_2) = \pi(u_2')$, and $x = \pi(u_1)\pi(u_2)^\omega \in \mathsf{dom}(f)$ (resp. $x \in \Sigma^\omega$)*
2. *u_2 and u_2' are idempotent in (u_1, u_2, u_3) and (u_1', u_2', u_3') respectively (for $\widetilde{\mathcal{T}}$)*
3. *there is a mismatch between $\rho_{\widetilde{\mathcal{T}}}(u_1, u_2, u_3)$ and $\rho_{\widetilde{\mathcal{T}}}(u_1', u_2', u_3')$.*

Sketch of proof. We show how to replace condition 3 of Lemma 13 by condition 3 of this lemma. One direction is easy: if $\rho_{\widetilde{\mathcal{T}}}(u_1, u_2, u_3)[i] \neq \rho_{\widetilde{\mathcal{T}}}(u_1', u_2', u_3')[i]$ for some i, then by Condition 2 of Lemma 14, we get the result. Conversely, assume there is i such that $f_*(u_1u_2^nu_3)[i] \neq f_*(u_1'(u_2')^nu_3')[i]$ for all $n \geq 1$ and u_2, u_2' are both producing (the other cases are similar and done in full version). By Condition 1 of Lemma 14, $\rho_{\widetilde{\mathcal{T}}}(u_1, u_2, u_3) \prec \rho_{\widetilde{\mathcal{T}}}(u_1u_2, u_2, u_2u_3) \prec \cdots \prec \rho_{\widetilde{\mathcal{T}}}(u_1u_2^k, u_2, u_2^ku_3)$ for all $k \geq 1$, and similarly for the u_i'. Therefore, for large enough k, $\rho_{\widetilde{\mathcal{T}}}(u_1u_2^k, u_2, u_2^ku_3)$ and $\rho_{\widetilde{\mathcal{T}}}({u_1'u_2'}^k, u_2', {u_2'}^ku_3')$, have length at least i. By Condition 2, $x = \rho_{\widetilde{\mathcal{T}}}(u_1u_2^k, u_2, u_2^ku_3) \preceq f_*(u_1u_2^nu_3)$ and $x' = \rho_{\widetilde{\mathcal{T}}}({u_1'u_2'}^k, u_2', {u_2'}^ku_3') \preceq f_*({u_1'u_2'}^nu_3')$ for all $n \geq 2k+1$, from which we get $x[i] \neq x'[i]$. ◀

Finally, we show how to decide continuity by reduction to the emptiness problem of bounded-visit two-way Parikh automata [17, 14]. (Full details are in full version [10])

▶ **Theorem 16.** *Continuity and uniform continuity are decidable for regular functions.*

Sketch of proof. The proof is based on Lemma 15. First, we encode words u_1, u_1', u_2, u_2' as words over the alphabet $(\Sigma \times {Q_P}^2)$ to hard-code condition 1 of the lemma. In particular, we define the language L of words of the form $w_1\#w_2\#u_3\#u_3'$ such that $w_1, w_2 \in (\Sigma \times {Q_P}^2)^*$ represent u_1, u_1', u_2, u_2' and such that conditions 1 and 2 of the lemma are satisfied. Condition 2 and condition $\pi(u_1)\pi(u_2)^\omega \in \mathsf{dom}(f)$ are simple because they are regular properties of words, the domain of f being regular. For condition 3, we need counters to identify positions i and j such that $\rho_{\widetilde{\mathcal{T}}}(u_1, u_2, u_3)[i] \neq \rho_{\widetilde{\mathcal{T}}}(u_1, u_2, u_3)[j]$, and later on check that $i = j$. In particular, we rely on the model of two-way Parikh automata which extend two-way automata with counters which can be only incremented and tested at the end of the computation. If such automata visit any input position a bounded number of times, their emptiness is decidable [17, 14]. We show that L is definable by an automaton which (1) visits any input boundedly many times, and (2) simulates the transducer obtained by Lemma 14.4. ◀

6 Discussion and Further Directions

Summary. In this paper, we have studied two notions of computability for rational and regular functions, shown their correspondences to continuity notions which we proved to be decidable. The notion of uniform computability asks for the existence of a modulus of continuity, which tells how far one has to go in the input to produce a certain amount of output. It would be interesting to give a tight upper bound on modulus of continuity for regular functions, and we conjecture that it is always a linear (affine) function in that case.

Discussion on Church synthesis. This work is motivated by a synthesis problem: given a specification of a function of infinite words (as a transducer), does there exist an algorithm to compute it and if true, synthesize such an algorithm. We have established in the introduction that even in the setting of Church ω-regular synthesis, this question makes sense as some (functional and synchronous) ω-regular specifications may describe functions which are not even computable. Here we compare our work with Church synthesis and address some open question. The Church ω-regular synthesis problem is known be decidable [18]. The setting we consider in this paper is orthogonal: Church ω-regular synthesis considers *non*-functional specifications but they have to be synchronous, while we consider functional specifications but they can be represented by way more expressive automata devices (two-way transducers with look-ahead). Moreover, Church synthesis asks for computability by Mealy machines while our goal is to relax this notion to more general computability notions. A corollary of our results is that the Church ω-regular synthesis problem when the specification is functional and the function realising the specification is only required to be computable, can be decided in NLogSpace. An interesting open question that we do not solve here is the extension of this latter result to non-functional specifications. More precisely, we leave the following problem open: given an automaton over $\Sigma_i.\Sigma_o$ defining an ω-regular synchronous specification S, is S realisable by a computable function? This question was partially answered in [16], where S is assumed to be *total*, *i.e.* $\mathsf{dom}(S) = \Sigma_i^\omega$. Intuitively, it is shown that in this case, if S is realisable, then it is realisable by a bounded delay function, *i.e.* a function which can be implemented by a deterministic transducer which needs to read at most $i + K$ input symbols before outputting the ith output symbol, where K is a constant that depends only on f and not on the input. The open case where S is partial is more challenging. For example, the function S_{swap} has partial domain, is computable, but not bounded delay computable.

Other future directions. Another interesting direction is to find a transducer model which captures exactly the computable, and uniformly computable, rational and regular functions. For rational functions, the deterministic (one-way) transducers are not sufficient, already for uniform computability, as witnessed by the rational function which maps any word of the form $a^n b^\omega$ to itself, and any word $a^n c^\omega$ to $a^{2n} c^\omega$. For regular functions, we conjecture that 2DFT characterise the computable ones, but we have not been able to show it yet.

Finally, much of our work deals with reg-preserving functions in general. An interesting line of research would be to investigate continuity and uniform continuity for different classes of functions which have this property. One natural candidate is the class of *polyregular functions* introduced in [4] which enjoy several different characterisations and many nice properties, including being effectively reg-preserving. This means that continuity and computability also coincide, however deciding continuity seems challenging.

References

1 Alonzo Church. Logic, arithmetic and automata. In *Int. Congr. Math.*, pages 23–35, Stockholm, 1962.

2 Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular transformations of infinite strings. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 65–74. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.18.

3 Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. One-way definability of two-way word transducers. *Log. Methods Comput. Sci.*, 14(4), 2018. doi:10.23638/LMCS-14(4:22)2018.

4 Mikolaj Bojanczyk. Polyregular functions. *CoRR*, abs/1810.08760, 2018. arXiv:1810.08760.

5 Michaël Cadilhac, Olivier Carton, and Charles Paperman. Continuity and rational functions. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 115:1–115:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.115.

6 Michaël Cadilhac, Andreas Krebs, Michael Ludwig, and Charles Paperman. A circuit complexity approach to transductions. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science 2015 – 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part I*, volume 9234 of *Lecture Notes in Computer Science*, pages 141–153. Springer, 2015. doi:10.1007/978-3-662-48057-1_11.

7 Olivier Carton, Dominique Perrin, and Jean-Eric Pin. Automata and semigroups recognizing infinite words. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 133–168. Amsterdam University Press, 2008.

8 Swarat Chaudhuri, Sriram Sankaranarayanan, and Moshe Y. Vardi. Regular real analysis. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 509–518. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.57.

9 Christian Choffrut. Minimizing subsequential transducers: a survey. *Theor. Comput. Sci.*, 292(1):131–143, 2003. doi:10.1016/S0304-3975(01)00219-5.

10 Vrunda Dave, Emmanuel Filiot, Shankara Narayanan Krishna, and Nathan Lhote. Deciding the computability of regular functions over infinite words. *CoRR*, abs/1906.04199, 2019. arXiv:1906.04199.

11 Joost Engelfriet, Hendrik Jan Hoogeboom, and Bart Samwel. XML navigation and transformation by tree-walking automata and transducers with visible and invisible pebbles. *CoRR*, abs/1809.05730, 2018. arXiv:1809.05730.

12 Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. On computability of data word functions defined by transducers. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures – 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12077 of *Lecture Notes in Computer Science*, pages 217–236. Springer, 2020. doi:10.1007/978-3-030-45231-5_12.

13 Emmanuel Filiot. Logic-automata connections for transformations. In Mohua Banerjee and Shankara Narayanan Krishna, editors, *Logic and Its Applications – 6th Indian Conference, ICLA 2015, Mumbai, India, January 8-10, 2015. Proceedings*, volume 8923 of *Lecture Notes in Computer Science*, pages 30–57. Springer, 2015. doi:10.1007/978-3-662-45824-2_3.

14 Emmanuel Filiot, Shibashis Guha, and Nicolas Mazzocchi. Two-way parikh automata. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, volume 150 of *LIPIcs*, pages 40:1–40:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.40.

15 Emmanuel Filiot, Nicolas Mazzocchi, and Jean-François Raskin. A pattern logic for automata with outputs. In Mizuho Hoshi and Shinnosuke Seki, editors, *Developments in Language Theory – 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings*, volume 11088 of *Lecture Notes in Computer Science*, pages 304–317. Springer, 2018. doi:10.1007/978-3-319-98654-8_25.

16 Michael Holtmann, Lukasz Kaiser, and Wolfgang Thomas. Degrees of lookahead in regular infinite games. *Log. Methods Comput. Sci.*, 8(3), 2012. doi:10.2168/LMCS-8(3:24)2012.

17 Oscar H. Ibarra. Automata with reversal-bounded counters: A survey. In Helmut Jürgensen, Juhani Karhumäki, and Alexander Okhotin, editors, *Descriptional Complexity of Formal Systems – 16th International Workshop, DCFS 2014, Turku, Finland, August 5-8, 2014. Proceedings*, volume 8614 of *Lecture Notes in Computer Science*, pages 5–22. Springer, 2014. doi:10.1007/978-3-319-09704-6_2.

18 J.R. Büchi and L.H. Landweber. Solving sequential conditions finite-state strategies. *Trans. Ameri. Math. Soc.*, 138:295–311, 1969.

19 S. Rao Kosaraju. Regularity preserving functions. *SIGACT News*, 6(2):16–17, April 1974. doi:10.1145/1008304.1008306.

20 Jean-Éric Pin and Pedro V. Silva. On uniformly continuous functions for some profinite topologies. *Theor. Comput. Sci.*, 658:246–262, 2017. doi:10.1016/j.tcs.2016.06.013.

21 Christophe Prieur. How to decide continuity of rational functions on infinite words. *Theor. Comput. Sci.*, 276(1-2):445–447, 2002. doi:10.1016/S0304-3975(01)00307-3.

22 W. Rudin. *Principles of Mathematical Analysis*. International series in pure and applied mathematics. McGraw-Hill, 1976. URL: https://books.google.pl/books?id=kwqzPAAACAAJ.

23 Joel I. Seiferas and Robert McNaughton. Regularity-preserving relations. *Theor. Comput. Sci.*, 2(2):147–154, 1976. doi:10.1016/0304-3975(76)90030-X.

24 Richard Edwin Stearns and Juris Hartmanis. Regularity preserving modifications of regular expressions. *Inf. Control.*, 6(1):55–69, 1963. doi:10.1016/S0019-9958(63)90110-4.

25 Klaus Weihrauch. *Computable Analysis – An Introduction*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1st edition, 2000. doi:10.1007/978-3-642-56999-9.