Logical Methods in Computer Science
Volume 18, Issue 2, 2022, pp. 23:1–23:29
https://lmcs.episciences.org/

Submitted Jun. 16, 2021
Published Jun. 29, 2022

SYNTHESIS OF COMPUTABLE REGULAR FUNCTIONS OF INFINITE WORDS

VRUNDA DAVE [a], EMMANUEL FILIOT [b], SHANKARA NARAYANAN KRISHNA [a], AND NATHAN LHOTE [c]

[a] IIT Bombay, India
e-mail address: {vrunda,krishnas}@cse.iitb.ac.in

[b] Université Libre de Bruxelles, Belgium
e-mail address: efiliot@ulb.ac.be

[c] Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
e-mail address: nathan.lhote@lis-lab.fr

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.

Key words and phrases: transducers, infinite words, computability, continuity, synthesis.

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 a research associate at F.R.S.-FNRS.

This work is partially supported by the ERC Consolidator grant LIPA683080, as well as ANR DELTA (grant ANR-16-CE40-0007).

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.46298/LMCS-18(2:23)2022

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 [Alo62, JL69], 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 it alternatively reads an input symbol and produces an output symbol. 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 $S_{\mathsf{swap}}(u_1\sigma@u_2) = \sigma u_1@u_2$. The specification S_{swap} 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.

1.1. 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 three 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

[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 $S(u) = \{v\}$.

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 of x. This machine model has been defined in [Wei00, 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).

1.2. **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 and write it in working tape, until the first @ symbol is met. Now go back in the working tape to see the last symbol read before @, write the same in output tape. It is followed by further going back to the beginning in the working tape, and copying all the symbols of working tape except the last one from left to right into the output tape. This gives us the prefix σu_1. The suffix $@u_2$ can simply be produced by copying the content of input tape into output tape.

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 $f(u_1\sharp u_2\sharp\ldots) = \overline{u_1}\sharp\overline{u_2}\sharp\ldots$. 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 $f(u_1\sharp u_2\sharp\ldots) = u_1u_1\sharp u_2u_2\sharp\ldots$. Similarly as before, it is computable but also uniformly computable: to determine the i^{th} output symbol, it suffices to read an input prefix of length at most i. Indeed, let $u_1\sharp u_2\sharp\ldots\sharp u$ be a prefix of length i of the input x, then $u_1u_1\sharp u_2u_2\sharp\ldots\sharp u$ is a prefix of $f(x)$ of length $\geq i$.

1.3. **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 $M_1, M_2, \ldots$ of Turing machines (on finite word inputs), the function f_{halt} defined as $f_{\mathsf{halt}} : x \mapsto b_1b_2b_3\ldots$ where

$b_i \in \{0,1\}$ is such that $b_i = 1$ if and only if M_i halts on input ϵ, is not computable but (uniformly) continuous (as it is constant).

1.4. **Beyond synchronous functions: regular functions.** Functional specifications in the 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 [AFT12, 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 [CPP08, 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}.

1.5. **Contributions.** We call *effectively reg-preserving functions* those functions that effectively preserve regular languages by inverse image [SH63, Kos74, SM76, PS17]. This includes for instance rational functions, regular functions and the more general class of polyregular functions [Boj18, EHS21]. We first show that for effectively reg-preserving functions, computability and continuity coincide, respectively, uniform computability and uniform continuity (Section 3, Theorem 3.4). 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 5.2). Continuity and uniform continuity for rational functions were already known to be decidable in PTime, from Prieur [Pri02, Prop. 4]. However, Prieur's proof techniques do not transfer to the two-way case. We then prove that uniform continuity (and hence uniform computability) is decidable in PSpace for regular functions given by deterministic Büchi two-way transducers with look-ahead (Section 5, Theorem 5.7). Using the same techniques, we also get the decidability of continuity (and hence computability) for regular functions (Theorem 5.12) in PSpace. For both problems, we show that this upper-bound is tight, namely, both problems are PSpace-hard.

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, Lemma 4.4). Based on this, we derive a decision procedure for continuity of rational functions by checking in NLogSpace a structural transducer pattern. This pattern expresses properties of synchronised loops on two runs of the transducers. The case of regular functions is much more involved as loops in two-way automata or transducers are

more complex. We also give a characterisation via a structural pattern which we show to be checkable by a bounded-visit non-deterministic two-way Parikh automaton of polynomial size, yielding the PSpace upper bound as those automata can be checked in PSpace for emptiness [FGM19].

1.6. **Related work.** The notion of continuity has not been extensively studied in the transducers literature over infinite words. The work by Prieur [Pri02] is the closest to ours, while [CSV13] 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 [CCP17, CCP20]. Our notion of uniform continuity can be linked to continuity with respect to a particular language variety which was *not* studied in [CCP17] (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.* [Cho03]), 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 [CKLP15] 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 [EFR20]. 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 [EFR20] goes by reduction to the finite alphabet setting and uses our result presented in the paper [DFKL20].

Changes from the conference version. The results in this paper are an extension of the work presented at CONCUR 2020 [DFKL20]. In the conference version, Lemmas 4.4 and 5.1 had only a short proof sketch; this version contains the full technical details. A major change from the conference version is in Section 5.2. This has an added contribution, where we present a new proof for the decidability of continuity and uniform continuity for regular functions. To aid us in this, we define what is called a *critical pattern* (Section 5.2.3) and show that the existence of such patterns in two way transducers help in deciding the continuity of the corresponding regular function (Proposition 5.4). Finally, in the proof of Theorem 5.7 and Theorem 5.12, we analyze the complexity of the pattern detection, which in turn yields a (tight) upper-bound for the complexity of checking continuity and uniform continuity of regular functions; this was left open in the CONCUR 20 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 $u, v \in \Sigma^\omega$, $u \preceq v$ if and only if $u = v$). 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 $w = a_1 a_2 \ldots \in \Sigma^\omega$ 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 if $\mathsf{Inf}(\rho) \cap F \neq \emptyset$. A run is *accepting* if it is final and starts from an initial state. A word $w \in \Sigma^\omega$ is accepted ($w \in L(B)$) if 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 [CPP08, 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 if and only if 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 [CPP08, Thm. 7.2] that prophetic Büchi automata capture ω-regular languages.

2.1. **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 final 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* if 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 [AFT12, 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.

[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.

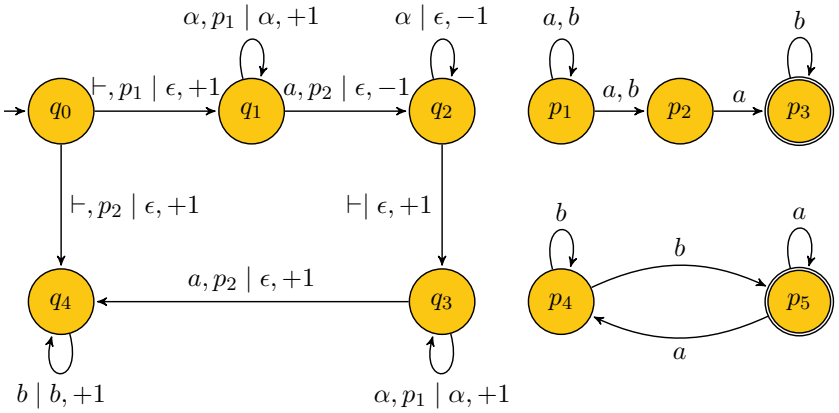

Figure 1: A $\mathsf{2DFT_{pla}}$ with automaton on the right implementing the look-ahead.

Let $\Sigma_\vdash = \Sigma \uplus \{\vdash\}$. Formally, a $\mathsf{2DFT_{pla}}$ 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 such that 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 $\mathsf{2DFT_{pla}}$ 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 $a_{i+1} a_{i+2} \dots \in L(A, p)$, 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 $\mathsf{2DFT_{pla}}$ $(\mathcal{T}, A)$ 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 $\mathsf{sup}\{i_n \mid 0 \leq n < |\rho|\} = \infty$. The output $[\![(\mathcal{T}, A)]\!](w)$ of a word w on run ρ is defined only when $\mathsf{sup}\{i_n \mid 0 \leq n < |\rho|\} = \infty$, and equals $\mathsf{out}(\rho)$. $\mathsf{2DFT_{pla}}$ are equivalent to $\mathsf{2DMT_{la}}$, and capture all regular functions. It is easy to see because different representations of ω-look-ahead are equiexpressive, the look-behind of $\mathsf{2DMT_{la}}$ can be eliminated while preserving the expressiveness, and one can encode the acceptance condition of $\mathsf{2DMT_{la}}$ as part of the look-ahead in $\mathsf{2DFT_{pla}}$ (see Appendix A for the proof).

Theorem 2.1. *A function $f \colon \Sigma^\omega \to \Gamma^\omega$ is regular if and only if it is 2DFT$_{pla}$ definable.*

Example 2.2. 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 $\mathsf{2DFT_{pla}}$ 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 $\mathcal{L}(A, p_1) = \Sigma^+ ab^\omega$,

$\mathcal{L}(A, p_2) = ab^\omega$, $\mathcal{L}(A, p_3) = b^\omega$, $\mathcal{L}(A, \{p_4, p_5\})$ is the set of words having infinitely many as. Each word in Σ^ω has a unique final run.

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 $\mathsf{2DFT}_{\mathsf{pla}}$.

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 3.1 (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.

Definition 3.2 (Continuity/Uniform continuity)**.** We interchangeably use the following two textbook definitions [Rud76] of 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 $x \in \mathsf{dom}(f)$.

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 3.3. 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 recognising a regular language $L \subseteq \Gamma^\omega$, produces an automaton recognising the language $f^{-1}(L) = \{u \mid f(u) \in L\}$.

Two well-studied classes (see *e.g.* [Fil15]) 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 3.4. *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. Assume that f is computable. We prove the continuity of f. Let M be the machine computing f. Let $x \in \mathsf{dom}(f)$ be the content on the input tape of M. 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. For all $i \geq 0$, define $\mathsf{Pref}_M(x, i)$ to be the smallest $j \geq 0$ such that, when M moves to the right of $x[:j]$ into cell $j+1$, it has output at least the first i symbols of $f(x)$. For any $i \geq 0$, choose $j = \mathsf{Pref}_M(x, i)$. Consider any $z \in \mathsf{dom}(f)$ such that $|x \wedge z| \geq j$. After reading j symbols of x, the machine M outputs at least i symbols of $f(x)$. Since $|x \wedge z| \geq j$, the first j symbols of x and z are the same, and M being deterministic, outputs the same i symbols on reading the first j symbols of z as well. These i symbols form the prefix for both $f(x), f(z)$, and hence, $|f(x) \wedge f(z)| \geq i$.

Thus, for every $x \in \mathsf{dom}(f)$ and for all i, there exists $j = \mathsf{Pref}_M(x, i)$ such that, for all $z \in \mathsf{dom}(f)$, if $|x \wedge z| \geq j$, then $|f(x) \wedge f(z)| \geq i$ implying continuity of f. 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 $m\colon i \mapsto \min\{j|\ P(i,j)\}$. 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 3.5. 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.

Topology preliminaries. A *regular word* (sometimes called ultimately periodic) over Σ is a word of the form uv^ω with $u \in \Sigma^*$ and $v \in \Sigma^+$. The set of regular words is denoted by $\mathsf{Reg}(\Sigma)$. The *topological closure* of a language $L \subseteq \Sigma^\omega$, denoted by $\bar{L}$, is the smallest language containing it and closed under taking the limit of converging sequences, *i.e.* $\bar{L} = \{x \mid\ \forall u \prec x, \exists y,\ uy \in L\}$. A language is *closed* if it is equal to its closure. A sequence of words $(x_n)_{n\in\mathbb{N}}$ is called regular if there exists $u, v, w \in \Sigma^*$, $z \in \Sigma^+$ such that for all $n \in \mathbb{N}$, $x_n = uv^n wz^\omega$. The proof of the characterisation of continuity (Lemma 4.4) requires the following folklore or easy to show propositions:

Proposition 4.1. *Let $L \subseteq \Sigma^\omega$ be regular language, then*

(1) *$\bar{L}$ is regular,*
(2) *$L \subseteq \overline{L \cap \mathsf{Reg}(\Sigma)}$ (i.e. the regular words are dense in a regular language),*
(3) *any regular word of $\bar{L}$ is the limit of a regular sequence of L*

Proof. Proof of Proposition 4.1.3: Let $L \subseteq \Sigma^\omega$ be regular. Let uv^ω be a regular word in $\bar{L}$. By regularity of L, there is a power of v, v^k such that for any words, w, x, $wv^k x \in L \Leftrightarrow wv^{2k}x \in L$.

Let us consider the language $K = \{x \mid uv^k x \in L\}$ which is non-empty since $uv^\omega \in \bar{L}$. Moreover, K is regular, and thus contains a regular word wz^ω. Hence we have that uv^ω is the limit of the sequence $\left(uv^{kn}wz^\omega\right)_{n\in\mathbb{N}}$ of L. □

Definition 4.2. 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: $\left((uv^n wz^\omega)_n, (uv^n w'z'^\omega)_n\right)$

Proposition 4.3. *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 $\left((x_n)_{n\in\mathbb{N}}, (x'_n)_{n\in\mathbb{N}}\right)$, 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 4.4 (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.*

Proof. This lemma extends Proposition 4.3. 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 only if direction is a simple consequence of Proposition 4.3. For the other direction, we first 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 a second step, since we have two sequences converging to a 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.

Let $f\colon \Sigma^\omega \to \Gamma^\omega$ be a reg-preserving function which is not uniformly continuous. Then according to Proposition 4.3, it has a bad pair $((x_n)_{n\in\mathbb{N}}, (x'_n)_{n\in\mathbb{N}})$ at some point x. If f is not continuous then we can assume that $x \in \mathsf{dom}(f)$. Our goal is to exhibit a synchronised bad pair at a point z, such that if $x \in \mathsf{dom}(f)$ then $z \in \mathsf{dom}(f)$.

By compactness of Γ^ω, we can extract subsequences such that both $(f(x_n))_{n\in\mathbb{N}}$ and $(f(x'_n))_{n\in\mathbb{N}}$ converge. Moreover, since the pair is bad, one can assume that they converge to different limits $y \neq y'$. Let i be such that $y[i] \neq y[i']$ and let $B_y = \{z \mid |y \wedge z| \geq i\} = \uparrow y[:i]$ and $B_{y'} = \{z \mid |y' \wedge z| \geq i\} = \uparrow y'[:i]$. By definition we have $B_y \cap B_{y'} = \emptyset$, and moreover both sets $B_y, B_{y'}$ are regular. Up to extracting subsequences, we can assume that for all n, $x_n \in f^{-1}(B_y)$ and $x'_n \in f^{-1}(B_{y'})$. This means that $x \in \overline{f^{-1}(B_y)} \cap \overline{f^{-1}(B_{y'})}$. Since f is reg-preserving, and from Proposition 4.1.1, the set $\overline{f^{-1}(B_y)} \cap \overline{f^{-1}(B_{y'})}$ is regular, and non-empty. Hence there exists a regular word $z \in \overline{f^{-1}(B_y)} \cap \overline{f^{-1}(B_{y'})}$. Moreover, since $\mathsf{dom}(f)$ is also regular, we can choose z so that $x \in \mathsf{dom}(f) \Leftrightarrow z \in \mathsf{dom}(f)$. Since $z \in \overline{f^{-1}(B_y)}$, there is a sequence $(z_n)_{n\in\mathbb{N}}$ of words in $f^{-1}(B_y)$ which converges to z. Furthermore, since $f^{-1}(B_y)$ is regular and since z is a regular word, we can assume, from Proposition 4.1.3, that the sequence $(z_n)_{n\in\mathbb{N}}$ is regular. Similarly, there is a regular sequence $(z'_n)_{n\in\mathbb{N}}$ in $f^{-1}(B_{y'})$ which converges to z.

If either sequence $(f(z_n))_{n\in\mathbb{N}}$ or $(f(z'_n))_{n\in\mathbb{N}}$ is not convergent then we are done, because one of the pairs $((z_n)_{n\in\mathbb{N}}, (z_n)_{n\in\mathbb{N}})$ or $((z'_n)_{n\in\mathbb{N}}, (z'_n)_{n\in\mathbb{N}})$ is bad and of course synchronised. If both sequences are convergent, then $\lim_n f(z_n) \in B_y$ and $\lim_n f(z'_n) \in B_{y'}$ (because B_y and $B_{y'}$ are both closed), which means that $|\lim_n f(z_n) \wedge \lim_n f(z'_n)| \leq i$, hence the pair $\big((f(z_n))_{n\in\mathbb{N}}, (f(z'_n))_{n\in\mathbb{N}}\big)$ is bad.

We only have left to show that we can synchronise the sequences. Let $z_n = uv^n wt^\omega$ and let $z'_n = u'v'^n w't'^\omega$

Let us first assume that z'_n is constant equal to z. Let $z' = u^{-1}z$, we have that $v^n z' = z'$ and $uv^n z' = z$ for any n. Thus the pair $\big((z_n)_{n\in\mathbb{N}}, (z)_{n\in\mathbb{N}}\big)$ is a synchronised bad pair.

Let us now assume that neither sequence is constant, which means that $|v|, |v'| > 0$. Without loss of generality, let us assume that $|u| \geq |u'|$, let $k \in \mathbb{N}$, let $p < |v'|$ be such that $|u'| + k|v'| + p = |u|$, let $v'' = v'[p+1:|v'|]v'[1:p]$ and let $w'' = v'[p+1:|v'|]w'$. Then we can write $z''_n = z'_{n+k+1} = u'(v')^k v'[1:p] \cdot (v'')^n \cdot w''t'^\omega = uv''^n w''t'^\omega$.

Note that $v^\omega = v''^\omega$, which means that $v^{|v''|} = v''^{|v|}$. Let $y_n = z_{|v''|n} = uv^{|v''|n} wt^\omega$ and let $y''_n = z''_{|v|n} = uv''^{|v|n} w't'^\omega$. Then the pair $\big((y_n)_{n\in\mathbb{N}}, (y''_n)_{n\in\mathbb{N}}\big)$ is a synchronised bad pair at z. □

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

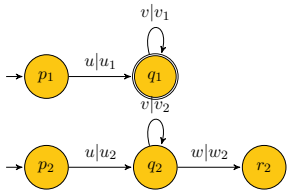

Figure 2: Pattern characterising non-continuity of rational functions given by *trim* one-way Büchi transducers.

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

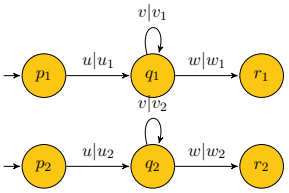

Figure 3: Pattern characterising non-*uniform* continuity of rational functions given by *trim* one-way Büchi transducers.

5. Deciding Continuity and Uniform Continuity

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

5.1. **Rational case.** We exhibit structural patterns which are shown to be satisfied by a one-way Büchi transducer if and only if the rational function it defines is not continuous (resp. not uniformly continuous). We express those patterns in the *pattern logic* defined in [FMR18, 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.

Lemma 5.1. *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 Figure 2 (resp. the formula $\phi_{u\text{-}cont}$ of Figure 3).*

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 4.3 (if a sequence s is bad then (s, s) is bad). Similarly, the pattern $\phi_{\text{u-cont}}$ witnesses that the pair $\big((uv^n wz)_{n\in\mathbb{N}}, (uv^n w' z')_{n\in\mathbb{N}}\big)$ is synchronised and bad (with z, z' words that have a final run from r_1, r_2, respectively), which entails non-uniform continuity.

For the other direction, we again use Lemma 4.4. 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. Let us consider the continuous case formally. The uniformly continuous case is similar. Let f be a function realised by a trim transducer T.

Let us first show that if the pattern of ϕ_{cont} appears, then we can exhibit a bad pair at a point of the domain. Let us assume ϕ_{cont} holds. Then the word uv^ω is in the domain of the function realised by the transducer since it has an accepting run. Let zt^ω be a word accepted from state r_2, which exists by trimness, and let us consider the pair $\left((uv^\omega)_{n\in\mathbb{N}}, (uv^n wzt^\omega)_{n\in\mathbb{N}}\right)$. We have $f(uv^\omega) = u_1 v_1^\omega$ and $u_2 \preceq \lim_n f(uv^n wzt^\omega)$. If $\mathsf{mismatch}(u_1, u_2)$ then the pair is bad, otherwise, we must have $v_2 = \epsilon$ and thus $u_2 w_2 \preceq \lim_n f(uv^n wzt^\omega)$. Since $\mathsf{mismatch}(u_1, u_2 w_2)$ we again have that the pair is bad.

Let us assume that the function f is not continuous at some point $x \in \mathsf{dom}(f)$. Then according to Lemma 4.4, there is a synchronised bad pair $\left((uv^n wz^\omega)_{n\in\mathbb{N}}, (uv^n w'z'^\omega)_{n\in\mathbb{N}}\right)$ converging to some point of $\mathsf{dom}(f)$.

Our goal is to exhibit a pattern as in ϕ_{cont}. If v is empty, then the sequences are constant which contradicts the functionality of T. So both the sequences converge to uv^ω, but their output sequences either do not converge or if converges, then not to $f(uv^\omega)$. Because of the mismatch, either $\left((uv^n wz^\omega)_{n\in\mathbb{N}}, (uv^\omega)_{n\in\mathbb{N}}\right)$ is a bad pair or $\left((uv^\omega)_{n\in\mathbb{N}}, (uv^n w'z'^\omega)_{n\in\mathbb{N}}\right)$ is a bad pair (or both). Without loss of generality, let $\left((uv^n wz^\omega)_{n\in\mathbb{N}}, (uv^\omega)_{n\in\mathbb{N}}\right)$ be a bad pair.

Let us consider an accepting run ρ of T over uv^ω. Since the run is accepting, there is an accepting state q_1 visited infinitely often. Let $l \in \{1, \ldots, |v|\}$ be such that, the state reached after reading $uv^n v[1\colon l]$ in the run ρ is q_1 for infinitely many ns. Let $v' = v[l+1\colon |v|]v[1\colon l]$, let $u' = uv[1\colon l]$ and let $w' = v[l+1\colon |v|]$, we have that $\left((u'v'^n w'z^\omega)_{n\in\mathbb{N}}, (u'v'^\omega)_{n\in\mathbb{N}}\right)$ is a bad pair. To simplify notations, we write this pair again as $\left((uv^n wz^\omega)_{n\in\mathbb{N}}, (uv^\omega)_{n\in\mathbb{N}}\right)$. We have gained that on the run over uv^ω, the accepting state q_1 is reached infinitely often after reading v factors. Now let k, l be integers such that $p_1 \xrightarrow{uv^k} q_1 \xrightarrow{v^l} q_1$, with p_1 an initial state. We consider l different sequences, for $r \in \{0, \ldots, l-1\}$ we define the sequence $s_r = \left(uv^{ln+r} wz^\omega\right)_{n\in\mathbb{N}}$. Since $\left((uv^n wz^\omega)_{n\in\mathbb{N}}, (uv^\omega)_{n\in\mathbb{N}}\right)$ is a bad pair, there must be a value r such that $\left(\left((uv^k)(v^l)^n(v^r w)z^\omega\right)_{n\in\mathbb{N}}, \left((uv^k)(v^l)^\omega\right)_{n\in\mathbb{N}}\right)$ is a bad pair. To simplify notations, again, we rename this pair $\left((uv^n wz^\omega)_{n\in\mathbb{N}}, (uv^\omega)_{n\in\mathbb{N}}\right)$, and we know that $p_1 \xrightarrow{u} q_1 \xrightarrow{v} q_1$, with p_1 initial and q_1 accepting.

For any m larger than the number of states $|Q|$ of T, let us consider the run of T over $uv^m wz^\omega$. Let i_m, j_m, k_m be such that we have the accepting run $p_m \xrightarrow{uv^{i_m}} q_m \xrightarrow{v^{j_m}} q_m \xrightarrow{v^{k_m} wz^\omega}$, with $i_m + j_m + k_m = m$ and $0 < j_m \leq |Q|$. And let $r_m = i_m + k_m \mod j_m$. Let us consider for some m, the sequence $\left((uv^{r_m})(v^{j_m})^n(wz)^\omega\right)_{n\in\mathbb{N}}$. By the presence of these loops, each of these sequences has a convergent image. There is actually a finite number of such sequences, since j_m and r_m take bounded values. Since the original pair is bad, this means that there must exist m such that the sequence $\left(f((uv^{r_m})(v^{j_m})^n(wz)^\omega)\right)_{n\in\mathbb{N}}$ does not converge to $f(uv^\omega)$. Let $u' = uv^{i_m}$, $v' = v^{j_m}$ and $w' = v^{k_m} w$, then $\left((u'v'^n w'z^\omega)_{n\in\mathbb{N}}, (u'v'^\omega)_{n\in\mathbb{N}}\right)$ is a bad pair. Once again we rename the pair $\left((uv^n wz^\omega)_{n\in\mathbb{N}}, (uv^\omega)_{n\in\mathbb{N}}\right)$ and now we have both $p_1 \xrightarrow{u} q_1 \xrightarrow{v} q_1$ and $p_2 \xrightarrow{u} q_2 \xrightarrow{v} q_2$, such that p_1, p_2 are initial, q_1 is accepting and wz^ω has a final run from q_2.

Now we have established the shape of the pattern, we only have left to show the mismatch properties.

$$p_1 \xrightarrow{u|u_1} q_1 \xrightarrow{v|v_1} q_1$$
$$p_2 \xrightarrow{u|u_2} q_2 \xrightarrow{v|v_2} q_2 \xrightarrow{wz^\omega|y_2}$$

Let us first assume that $v_2 \neq \epsilon$. Then $\lim_n f(uv^n wz^\omega) = u_2 v_2^\omega$. Since the pair is bad, there exists k such that $\mathsf{mismatch}(u_1 v_1^k, u_2 v_2^k)$. Hence, up to taking $u' = uv^k$, we have established the pattern:

$$\exists \pi_1 \colon p_1 \xrightarrow{u'|u'_1} q_1,\ \exists \pi'_1 \colon q_1 \xrightarrow{v|v_1} q_1$$
$$\exists \pi_2 \colon p_2 \xrightarrow{u'|u'_2} q_2,\ \exists \pi'_2 \colon q_2 \xrightarrow{v|v_2} q_2$$
$$\big(\mathsf{init}(p_1) \wedge \mathsf{init}(p_2) \wedge \mathsf{acc}(q_1)\big) \wedge \big(\mathsf{mismatch}(u'_1, u'_2)\big)$$

Let us now assume that $v_2 = \epsilon$. Then $\lim_n f(uv^n wz^\omega) = u_2 y_2$. Then there is a prefix w_2 of y_2 and an integer k such that $\mathsf{mismatch}(u_1 v_1^k, u_2 w_2)$. Hence, up to taking $u' = uv^k$, and w' a sufficiently long prefix of wz^ω we have established the pattern:

$$\exists \pi_1 \colon p_1 \xrightarrow{u|u_1} q_1,\ \exists \pi'_1 \colon q_1 \xrightarrow{v|v_1} q_1$$
$$\exists \pi_2 \colon p_2 \xrightarrow{u|u_2} q_2,\ \exists \pi'_2 \colon q_2 \xrightarrow{v|v_2} q_2,\ \exists \pi''_2 \colon q_2 \xrightarrow{w|w_2} r_2$$
$$\big(\mathsf{init}(p_1) \wedge \mathsf{init}(p_2) \wedge \mathsf{acc}(q_1)\big) \wedge \big((v_2 = \epsilon \wedge \mathsf{mismatch}(u_1, u_2 w_2))\big)$$

which concludes the proof. □

Theorem 5.2. *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 5.1, if T is trim, non-continuity of f is equivalent to T satisfying the formula ϕ_{cont} of Figure 2. This formula is expressed in the syntax of the pattern logic from [FMR18], where it is proved that model-checking pattern formulas against transducers can be done in NLogSpace [FMR18, 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. □

5.2. **Regular case.** As for the rational case, we first define a pattern which is satisfied by a $\mathsf{2DFT_{pla}}$ if and only if it defines a non-continuous function. Then we show how to check for the existence of this pattern. It is however more intricate than in the rational setting. First, we get rid of the look-ahead by working on words annotated with look-ahead information. We now make this notion formal.

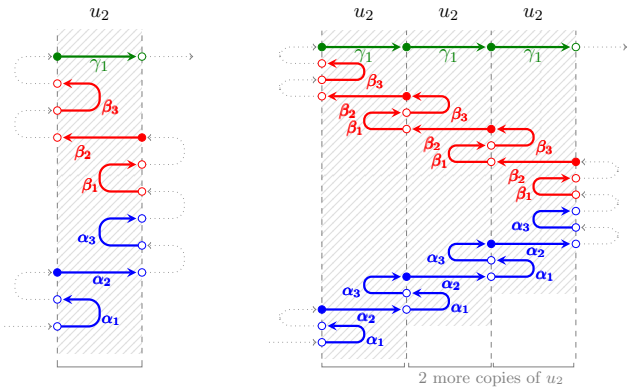

Figure 4: Pumping u_2. $\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3, \gamma_1$ are the outputs seen on u_2. The blue, red and green arrows are part of the run r while reading u_2.

5.2.1. *Look-ahead removal and annotated words.* Let f be realized by a $\mathsf{2DFT_{pla}}$ $\mathcal{T}$ over alphabet Σ with look-ahead automaton P. Let Q_P be the states of P. We define $\widetilde{\mathcal{T}}$, a 2DBT over $\Sigma \times Q_P$ which, informally, simulates $\mathcal{T}$ over words annotated with look-ahead states, and which accepts only words with a correct look-ahead annotation with respect to P. Formally, for an annotated word $u \in (\Sigma \times Q_P)^\infty$, we denote by $\pi_\Sigma(u) \in \Sigma^\infty$ its projection on Σ. An annotated word $x \in (\Sigma \times Q_P)^\omega$ is *well-labelled* if $\pi_{Q_P}(u)$ is the accepting run of P over u.

As P is prophetic, for all words $u \in \mathsf{dom}(f)$, there exists a unique well-annotated word, denoted $\tilde{u}$, which satisfies $\pi_\Sigma(\tilde{u}) = u$. We now let $\tilde{f}$ be the function defined, for all $u \in \mathsf{dom}(f)$, by $\tilde{f}(\tilde{u}) = f(u)$. The function $\tilde{f}$ is easily seen to be definable by a 2DBT which simulates $\mathcal{T}$ over the Σ-component of the input word and simulates P over the Q_P-component of the input word to check that the annotation is correct.

Proposition 5.3. *One can construct, in linear-time, a 2DBT denoted $\widetilde{T}$ which realises the function $\tilde{f}$.*

5.2.2. *Transition monoid of 2DBT.* In our pattern, we will be using the notion of transition monoid of a 2DBT, as the transition monoid of its underlying two-way automaton (obtained by ignoring the output words on transitions).

Let T be a 2DBT, we define its transition monoid M_T in the usual way (see *e.g.* [BGMP18] for details), but we also add information telling us if a word produces empty output or not. The set M_T contains functions $Q \times \{L, R\} \to Q \times \{L, R\} \times \{0, 1\}$. Informally, the transition morphism of T, $\phi_T \colon \Sigma^* \to M_T$ sends a word u to the function mapping *e.g.* (p, L) to $(q, R, 1)$ if there exists a run of T on u which enters u from the left in state p, stays within the inner positions of u until it exits u from the right in state q, and T has produced a non-empty output. This run is called a left-to-right traversal (LR-traversal for short). Left-to-left (LL-), right-to-right (RR-) and right-to-left (RL-) traversals are defined similarly. For instance a run from p to q is an LL-traversal on u, if it enters u from the left in state p, stays within the inner positions of u, until it exists u from the left in state q. If this run did not produce any output, then $\phi_T(u)$ contains the function mapping (p, L) to $(q, L, 0)$.

The transition monoid of a two-way automaton is defined similarly, except that the information on whether there is a production of non-empty output or not is not required, *i.e.* it is a set of functions of type $Q \times \{L, R\} \to Q \times \{L, R\}$.

5.2.3. *Critical Pattern.* Let $\mathcal{T}$ be a $\mathsf{2DFT_{pla}}$ and $\widetilde{T}$ be its associated 2DBT running over annotated words (see Section 5.2.1). Let $\rho = (q_0, i_0 = 0) \xrightarrow{b_{i_0}/\gamma_1} (q_1, i_1) \xrightarrow{b_{i_1}/\gamma_2} \cdots (q_n, i_n = l)$ be an initial run of $\widetilde{T}$ over a finite word uvw. We say that the factor v is *non-producing* in ρ if all the transitions occurring within v output ϵ. Otherwise v is *producing*. Formally, if in all the transitions $(q_j, i_j) \xrightarrow{b_{i_j}/\gamma_j} (q_{j+1}, i_{j+1})$ in run ρ such that $|u| \leq i_j < |uv|$ and $\gamma_j = \epsilon$, then v is non-producing.

We define the notion of *critical pattern* for the 2DBT $\widetilde{T}$. We say that $\widetilde{T}$ admits a critical pattern if there are words $u, v \in \Sigma^*$, $u_1, v_1, w_1, z_1, u_2, v_2, w_2, z_2 \in (\Sigma \times Q_P)^*$ such that

(1) $\pi_\Sigma(u_1) = \pi_\Sigma(u_2) = u$, $\pi_\Sigma(v_1) = \pi_\Sigma(v_2) = v$,
(2) v_1 and v_2 are idempotent[4] with respect to the transition monoid of $\widetilde{T}$,
(3) there exist two accepting runs of $\widetilde{T}$ on $u_1 v_1 w_1 z_1^\omega$ and $u_2 v_2 w_2 z_2^\omega$ which can be respectively decomposed into $\rho_1\lambda_1$ and $\rho_2\lambda_2$ such that for all $i = 1, 2$,
(4) ρ_i is over $u_i v_i w_i$ and does not end in v_i,
(5) v_i is not producing in ρ_i,
(6) there is a mismatch between $\mathsf{out}(\rho_1)$ and $\mathsf{out}(\rho_2)$.

We give the following characterisation of continuity and uniform continuity. Note that the patterns are similar to the one-way case.

Proposition 5.4. *Let T be a $\mathsf{2DFT_{pla}}$ realising a function f. Then the following holds:*

(1) *f is not uniformly continuous (resp. continuous).*
(2) *the 2DBT $\tilde{T}$ admits a critical pattern $u_1, u_2, v_1, v_2, w_1, z_1, w_2, z_2, \rho_1, \lambda_1, \rho_2, \lambda_2$ (resp. and $\pi_\Sigma(u_1 v_1^\omega) \in \mathsf{dom}(f)$)*

Proof. We will show that the existence of a critical pattern is equivalent to the existence of a synchronised bad pair. From Proposition 4.3 this will conclude the proof.

Let us first assume that we have a critical pattern $u_1, u_2, v_1, v_2, w_1, z_1, w_2, z_2, \rho_1, \lambda_1, \rho_2, \lambda_2$. Our goal is to prove that $\left((\pi_\Sigma(u_1 v_1^n w_1 z_1^\omega))_n, (\pi_\Sigma(u_2 v_2^n w_2 z_2^\omega))_n\right)$ is a bad pair. We define ρ_1^n, ρ_2^n the runs obtained from ρ_1, ρ_2, just by iterating the respective v_1, v_2 factors n times. Since v_1, v_2 are idempotents, reading v_i or v_i^n does not change the shape of the run ρ_i, nor the order of the outputs occurring outside of v_i. Moreover, since v_i is non-producing in both runs, the outputs over v_i and v_i^n are also the same. Hence we get $\mathsf{out}(\rho_1) = \mathsf{out}(\rho_1^n)$ and $\mathsf{out}(\rho_2) = \mathsf{out}(\rho_2^n)$. Thus the runs $\rho_1^n\lambda_1^n$ and $\rho_2^n\lambda_2^n$ produce outputs which mismatch at a position smaller than $\max(|\rho_1|, |\rho_2|)$ and the pair $\left((\pi_\Sigma(u_1 v_1^n w_1 z_1^\omega))_n, (\pi_\Sigma(u_2 v_2^n w_2 z_2^\omega))_n\right)$ is indeed bad.

Conversely, let us assume that $\left((uv^n x_1 y_1^\omega)_n, (uv^n x_2 y_2^\omega)_n\right)$ is a bad pair. Up to extracting subsequences, we can assume that v, y_1, y_2 are idempotent both for T and its look-ahead automaton. Thus we can obtain, up to taking a larger word u, a synchronised bad pair of the form $\left((\pi_\Sigma(u_1 v_1^n w_1 z_1^\omega))_n, (\pi_\Sigma(u_2 v_2^n w_2 z_2^\omega))_n\right)$ with $\pi_\Sigma(u_1) = \pi_\Sigma(u_2) = u$ and $\pi_\Sigma(v_1) = \pi_\Sigma(v_2) = v$. We also write $\pi_\Sigma(w_i) = x_i$ and $\pi_\Sigma(z_i) = y_i$ for $i \in \{1, 2\}$. Since the pair is bad there exists i such that for any m, there exists $n \geq m$ with $|f(uv^n x_1 y_1^\omega) \wedge f(uv^n x_2 y_2^\omega)| \leq i$.

[4] $\phi_{\tilde{T}}(v_i) = \phi_{\tilde{T}}(v_i v_i)$ for $i \in \{1, 2\}$

Let n be larger than $2i+2$ with $|f(uv^n x_1 y_1^\omega) \wedge f(uv^n x_2 y_2^\omega)| = j \leq i$. Let ρ_1 (resp. ρ_2) be the prefix of the run over $u_1 v_1^n w_1 z_1^\omega$ (resp. $u_2 v_2^n w_2 z_2^\omega$) which stops after producing the jth output. We make the following claim with respect to the sequence $u_1 v_1^{i+1} v_1^k v_1^{i+1} w_1 z_1^\omega$:

Claim 5.5. *Any transition of ρ_1 occurring within the factor v_1^k must produce empty outputs and ρ_1 ends outside of this factor.*

If we assume that the claim is true, then it is similarly true for ρ_2. Then we obtain a critical pattern $u_1 v_1^{i+1}, u_2 v_2^{i+1}, v_1^k, v_2^k, v_1^{i+1} w_1, z_1, v_1^{i+1} w_2, z_2, \rho_1, \lambda_1, \rho_2, \lambda_2$.

We only have left to show that the claim holds.

Proof of Claim 5.5. Let us assume towards a contradiction that a transition occurring within v_1^k in ρ_1 has non-empty output. Let ρ_1' the shortest prefix of the run λ over $u_1 v_1^{i+1} v_1^k v_1^{i+1} w_1 z 1^\omega$ which extends ρ_1 and ends outside of the v_1^k factor. Note that if ρ_1 already ends outside of this factor, $\rho_1' = \rho_1$. Let ρ be a factor of ρ_1' which contains such a producing transition and which starts and ends at a boundary of the v_1^k factor without leaving it.

Since $\tilde{T}$ is deterministic and since v_1 is idempotent, all v_1 factors (except possibly for the first and last ones) must also produce such a non-empty output in λ. These outputs must occur in λ either in increasing or decreasing order with respect to the order of the input word. If they occur in increasing order, the first $i+1$ v_1 factors produce non-empty outputs within ρ_1, which contradicts the assumption that $|\mathsf{out}(\rho_1)| \leq i$. Similarly if the outputs occur in decreasing order then the last $i+1$ v_1 factors produce non-empty outputs within ρ_1, which also yields a contradiction, hence we obtain that the claim must hold. □

This proves Proposition 5.4. □

5.2.4. *Deciding uniform continuity and continuity for regular functions.* We now show how to decide, in PSpace, whether a given $\mathsf{2DFT_{pla}}$ defines a uniform continuous function by checking whether its associated 2DBT over annoted words admits a critical pattern. Deciding continuity is based on the same ideas and shown in the last paragraph of this section. We reduce the problem to checking the emptiness of a non-deterministic two-way Parikh automaton [Iba14]. A *two-way Parikh automaton* (2PA) $\mathcal{P}$ of dimension d is a two-way automaton, running on *finite* words, extended with vectors of integers (including 0) of dimension d. Each dimension can be seen as a counter which can only be incremented. A run on a finite word is accepting if it reaches some accepting state and the (pairwise) sum of the vectors met along the transitions belong to some given semi-linear set represented as some existential Presburger formula[5]. The emptiness problem for 2PA is undecidable but decidable in PSpace for *bounded-visit* 2PA [FGM19]. A 2PA is bounded visit if there exists some k such that in any accepting run of $\mathcal{P}$, any input position is visited at most k times by that run. The 2PA we construct will be of polynomial size and bounded-visit, entailing PSpace decidability of uniform continuity for $\mathsf{2DFT_{pla}}$, following Proposition 5.3.

Lemma 5.6. *Given a $\mathsf{2DFT_{pla}}$ T, one can construct in polynomial time a bounded-visit 2PA $\mathcal{P}_T$ (of polynomial size) such that $L(\mathcal{P}_T) \neq \emptyset$ if and only if $\widetilde{T}$ admits a critical pattern.*

Before proving the lemma, let us state our main result.

[5]We recall that existential Presburger formulas are existentially quantified first-order formulas over the predicate $t_1 \leq t_2$ where t_i are terms over the constants $0, 1$ and the addition $+$.

Theorem 5.7. *Deciding whether a regular function given as a deterministic two-way transducer with prophetic Büchi look-ahead is uniformly continuous is* PSPACE-COMPLETE. *Hardness also holds for continuity.*

Proof. The upper-bound is a direct consequence of Proposition 5.3, Lemma 5.6, and the PSPACE upper-bound of [FGM19] for the non-emptiness of bounded-visit two-way Parikh automata.

The lower-bound is easily obtained by reducing the problem of checking the non-emptiness of the language of a deterministic two-way automaton over finite words, which is known to be PSPACE-hard[6]. Given a deterministic two-way automaton A on finite words over some alphabet Σ containing at least two letters a and b, we construct a $\mathsf{2DFT_{pla}}$ T defining the following function f. It is defined on all ω-words u in $(\Sigma \cup \{\#\})^\omega$ by $f(u) = a^\omega$ if $u = v\#w$ for some $v \in L(A)$ and w contains infinitely many a, otherwise by $f(u) = b^\omega$. Clearly, if $L(A)$ is empty, then f is the constant function mapping any word to b^ω and is therefore uniformly continuous (and continuous). Conversely, if there exists some $v \in L(A)$, then we claim that f is not continuous (and hence not uniformly continuous). Indeed, consider the family of words $u_n = v\#a^n\#^\omega$ for all $n \geq 0$ and some $a \in \Sigma$. Clearly, $f(u_n) = b^\omega$ for all n as u_n contains finitely many a. However $f(\lim_{n\to\infty} u_n) = f(v\#a^\omega) = a^\omega$, which is different from $\lim_{n\to\infty} f(u_n) = b^\omega$. Finally, it remains to show how to construct T from A. The idea is to use a look-ahead to check whether the input is of the form $v\#w$ where $v \in \Sigma^*$ and w contains infinitely many a. If that is the case, then T simulates A on v (using $\#$ as an endmarker). At the end of this simulation, if A reaches an accepting state, then T reads w and keeps on producing a. On the other hand, if A reaches a non-accepting state, then T keeps on producing b on the w part. If the look-ahead has failed (meaning that the input is not of the above form), then T keeps on producing b while reading its input in a one-way fashion. □

We now proceed to the proof of Lemma 5.6.

Encoding of the set of critical patterns as a language. We show how to encode the set of critical patterns as the language of a 2PA. To that end, let us see how a critical pattern can be encoded as a word. The 2PA will have to check properties $(1) - (6)$ of the definition of a critical pattern. The first condition is about projections. To check that u_1, u_2 project on the same Σ-word, as well as v_1, v_2, we just encode them as words over the alphabet $\Sigma \times Q_P \times Q_P$. Let us make this more formal. Let $\#$ be a fresh symbol. For any two alphabets A, B and any two words $w_1 \in A^*$ and $w_2 \in B^*$ of same length, we let $w_1 \otimes w_2 \in (A \times B)^*$ be their convolution, defined by $(w_1 \otimes w_2)[i] = (w_1[i], w_2[i])$ for any position i of w_1.

We now define a language $\mathrm{Crit}_{\mathcal{T}} \subseteq \vdash ((\Sigma \times Q_P^2)^*\#)^2((\Sigma \times Q_P)^*\#)^4 \dashv$ which consists of words of the form $w = \vdash (u \otimes a_1 \otimes a_2)\#(v \otimes b_1 \otimes b_2)\#w_1\#z_1\#w_2\#z_2\# \dashv$ where, if $u_i = u \otimes a_i$ and $v_i = u \otimes b_i$, then $u_1, v_1, w_1, z_1, u_2, v_2, w_2, z_2$ is a critical pattern.

The following proposition is immediate

Proposition 5.8. *$\mathcal{T}$ admits a critical pattern if and only if $\mathrm{Crit}_{\mathcal{T}} \neq \emptyset$.*

Note that this encoding makes sure condition (1) of the definition of critical pattern is ensured, but the other properties will have to be checked by the 2PA.

[6]This is a folklore result, easily obtained by reducing the PSPACE-hard problem of checking the non-emptiness of the intersection of the languages of n DFA, each pass of the two-way automaton over the whole input simulates one pass of each DFA.

Checking idempotency with a succinct two-way automaton. The 2PA has to check condition (2) of the critical patterns, *i.e.* idempotency with respect the transition monoid of the 2DBT $\widetilde{T}$. We show more generally, given a deterministic two-way automaton A, how a two-way automaton of polynomial size can recognise idempotent words with respect to the transition monoid of A. The proof of this result is not needed to understand the proof of the main lemma (Lemma 5.11), namely that the set $\mathrm{Crit}_{\mathcal{T}}$ is recognisable by a 2PA, and can be skipped in a first reading.

Lemma 5.9. *For any deterministic two-way automaton A (resp. deterministic two-way Büchi transducer T), one can construct in polynomial time a (bounded-visit) deterministic two-way automaton I_A which recognises the set of words $\#u\#$ such that u is idempotent with respect to A, and $\#$ is a fresh symbol.*

Proof. We prove the result for automata. Its adaptation to transducers is trivial, as the transition monoid of a transducer is the same as the transition monoid of its underlying automaton obtained by ignoring the outputs, without the information on whether some non-empty output is produced or not by a traversal. The following construction, done for automata, can be trivially adapted to account for this information.

Let Q be the set of states of A. Without loss of generality, we can assume that $Q = \{1, \dots, n\}$. Let $(i, j) \in Q^2$. We explain how a two-way automaton $A_{i,j}^{LR}$ can check whether (i, j) is is left-to-right traversal of u: it needs to start from the left of u in state i, and simulates A until it reaches a boundary of u: if it is the left boundary, it rejects, if it the right boundary, it accepts. Now, let us explain how a two-way automaton $B_{i,j}^{L,R}$ can check whether (i, j) is a left-to-right traversal of A over uu, by reading only $\#u\#$. Intuitively, the main idea is to simulate A and when A wants to go to the right of u following some transition t, then $B_{i,j}^{L,R}$ comes back to the left of u and applies t and proceeds with its simulation of A. More precisely, $B_{i,j}^{L,R}$ has two modes: M_1 and M_2. It stays in mode M_1 as long as A does not cross the right boundary of u or crosses the left boundary (in which case it stops its computation in a non-accepting state). As soon as it crosses the right boundary, using some transition t, $B_{i,j}^{L,R}$ moves to mode M_2, comes back to the left of u, and applies t. It then simulates A until it reaches the right boundary in state j in which case it stops its computation and accepts, or it reaches the left boundary in which case it stops its computation and rejects.

Note that in the above constructions, $A_{i,j}^{LR}$ and $B_{i,j}^{LR}$ are both deterministic, since A is deterministic. Therefore, they have bounded crossing. Moreover, they have a polynomial number of states. We construct in a similar way automata for each type of traversals (LR, RL, LL, RR) and each pair (i, j).

Then, clearly, u is idempotent if and only if the following holds

$$\bigwedge_{(i,j)\in Q^2, \mathrm{trav}\in\{LL,RL,LR,RR\}} \#u\# \in L(A_{i,j}^{\mathrm{trav}}) \Leftrightarrow \#u\# \in L(B_{i,j}^{\mathrm{trav}})$$

Since the automata are deterministic, they can be complemented in polynomial time. Moreover, intersection of two-way automata can be done in linear time by serial composition: the first automaton is simulated and if it eventually accepts, then the other automaton is simulated and must accept as well. Using these results, one can construct in polynomial time a deterministic two-way automaton implementing the conjunction above, concluding the proof. □

Checking acceptance of lasso words by a succinct two-way automaton over finite words. We prove a useful technical lemma, which intuitively says that it is possible to construct a two-way automaton of polynomial size accepting words of the form $u\#v$ such that uv^ω is accepted by a given Büchi two-way automaton A and v is idempotent with respect to the transition monoid of A. The statement of the lemma is slightly more general. The proof of this lemma is not needed to understand the proof of the main lemma, namely that the set $\mathrm{Crit}_{\mathcal{T}}$ is recognisable by a 2PA, and can be skipped in a first reading.

A *marked word* over an alphabet Σ is an ω-word over $\Sigma \times \{0,1\}$ which contains exactly one occurrence of 1. Given $i \geq 1$ and $x \in \Sigma^\omega$, we denote by $x \triangleright i$ the marked word such that for all $j \geq 1$, $(x \triangleright i)[j] = (x[j], 0)$ if $j \neq i$ and $(x \triangleright i)[j] = (x[j], 1)$ otherwise.

Lemma 5.10. *Let A be a non-deterministic (bounded-visit) two-way automaton over an alphabet Σ, $\# \notin \Sigma$ and q be a state of A. Let L_q be the language of words $(\vdash u\#v\dashv) \triangleright i$ such that $u, v \in \Sigma^*$, $2 \leq i \leq |u| + 1$, v is idempotent with respect to the transition monoid of A, and there exists an accepting run of A on uv^ω starting in configuration (q, i). Then, L_q is recognisable by a non-deterministic bounded-visit two-way automaton of polynomial size (in A).*

Proof. We define a two-way automaton P recognising L_q. The symbol $\#$ is needed to identify the left boundary of v. First, P checks that the input is valid, *i.e.* it is of the form $(\vdash u\#v\dashv) \triangleright i$ such that $2 \leq i \leq |u| + 1$ and v is idempotent with respect to the transition monoid of A. To check idempotency, we rely on the construction of Lemma 5.9.

If the validity checking succeeds, P comes back to the initial position of its input and check the existence of an accepting run as required by the statement of the lemma. Since this accepting run is on uv^ω and P can only read one occurrence of v, we first give a characterisation of such an accepting run which will turn useful to check its existence while accessing only one iteration of v. This characterisation intuitively says that there is an accepting loop on some finite iteration v^κ accessible from the marked position i in state q. More precisely, it can be seen that any accepting run r of A on uv^ω starting in configuration (q, i) can be decomposed as:

$$r = \rho_0 L_1 R_1 \dots L_n R_n \rho_1 \ell^\omega$$

where:

(1) n is smaller than the number of states of A
(2) ρ_0 is a run on u from configuration (q, i) to some configuration $(q_1, |u| + 1)$
(3) for all $1 \leq j \leq n$, L_j is a left-to-left traversal on v^ω starting in some configuration $(q_j, |u| + 1)$ and ending in some configuration $(p_j, |u|)$
(4) for all $1 \leq j \leq n$, R_j is a right-to-right traversal on u starting in configuration $(p_j, |u|)$ and ending in some configuration $(q_j, |u| + 1)$
(5) ρ_1 is a finite run visiting only positions of v^ω from configuration $(q_n, |u| + 1)$ to some configuration (p, j_1) such that $j_1 = |u| + |v|.m + 1$ for some m (*i.e.* j_1 is the initial position of some iteration of v)
(6) ℓ is a finite run visiting only positions of v^ω from configuration (p, j_1) to some configuration (p, j_2) such that $j_2 = j_1 + |v|m$ for some $m \geq 1$ (*i.e.* j_2 is the initial position of some iteration of v, and is at the right of j_1)

In the above characterisation, note that L_j, R_j are either LL- or RR-traversal starting at the right boundary of u. Since r is accepting, it visits the whole infinite input uv^ω and therefore there cannot be more such traversals than the number of states. Otherwise, this boundary would be visited twice in the same state by A, and therefore, it would be visited

infinitely many times, contradicting that the whole input is read. This justifies why we can take n smaller than the number of states.

This characterisation is not directly exploitable by P. We make the following observation: since v is idempotent, L_j is a left-to-left traversal on vv for all $1 \leq j \leq n$. Indeed, suppose that is not the case, it means that L_j visits all positions of $v^2\sigma$ at least once, where σ is the first symbol of v. Let s_1 be the state of L_j the first time it reaches the end of v, and s_2 be the state of L_j the first time it reaches the end of vv. In other words, there is a left-to-right traversal from q_j to s_1 on v and a left-to-right traversal from q_j to s_2 on vv. Since v is idempotent, we get that $s_1 = s_2$. This contradicts that L_j is a left-to-right traversal, since A then loops on $s_1 = s_2$ and moves forever to the right when initially starting to read v^ω on state q_j. This means that to check the existence of a left-to-left traversal on v^ω, one only needs to consider two iterations of v.

We now proceed to the construction of P. The automaton P guesses a decomposition as in the above characterisation by simulating A. The main difficulty is that it reads only one occurrence of v while A must be simulated on uv^ω. Let us first explain how the decomposition can be guessed. First, P accesses the marked position i and start simulating A from state q (ignoring the marking in $\{0,1\}$ since A is over alphabet Σ). If eventually A reaches the right boundary of u in some state q_1, P guesses some state p_1 and checks whether there exists a left-to-left traversal L_1 of A, from q_1 to some p_1, on v^ω (we explain later how this verification can be done by P). If it is the case, P again simulates A on u and if eventually A reaches the right boundary of u in some state q_2, once again, P checks whether there exists a left-to-left traversal of v^ω from q_2 to some p_2, and so on for at most n times. Non-deterministically, when P reaches the right boundary of u, it can decide to check for the existence of an accessible accepting loop in v^ω, *i.e.* runs ρ_1 and ℓ as in the above characterisation (we also explain later how this verification can be done by P). If such a loop is found, P accepts.

Let us now explain how the two verifications involved in the latter simulation can be done by P, namely: checking the existence of a left-to-left traversal on v^ω from some state, say s, and checking the existence of an accessible accepting loop. The main difficulty is that P has access to only one occurrence of v, but it can use two-wayness to overcome this problem. If A needs to cross the right boundary of v from the left (*i.e.* access the next iteration of v), P just rewinds its reading head to the first position of v, and if A needs to cross the left boundary of v from the right (*i.e.* access the previous iteration of v), then P moves its reading head to the last position of v. We call this simulation a *simulation modulo*. We use this idea of simulation modulo to check the existence of LL-traversals as well as ρ_1 and ℓ.

For LL-traversals, as explained before, one only needs to check the existence of an LL-traversal on vv. P therefore performs a simulation of A modulo and uses a bit of information to check whether it is in the first or second occurrence of v.

To check the existence of ρ_1, we can use similar arguments as before, based on the fact that v is idempotent, to prove that ρ_1 only visits the two first iterations of v. This is due to the fact that ρ_1 can be decomposed into a sequence of traversals on some powers of v but each of those traversal is a traversal on v by idempotency. For example, ρ_1 might be an LR-traversal on v^{κ_1} followed by a LL-traversal on v^{κ_2}, followed by a RR-traversal on v^{κ_3}, up to reaching configuration (p, j_1). However by idempotency, each of these traversals are actually traversals on v, so, $\kappa_j = 1$ for all j. Hence, only two iterations of v are needed (two and not one, because, for instance, an LR-traversal on v can be followed by a LL-traversal

on v, and hence two iterations are needed). Since two iterations of v are needed, P can check the existence of ρ_1 using some simulation modulo and one bit of information.

Finally, it remains for P to check for the existence of a loop visiting an accepting state. Using idempotency of v and similar arguments as before, it can be shown that such a loop can be found only on three iterations of v, *i.e.* on v^3. So, once again, P can perform a simulation modulo, and use two bits of information to check whether it is on the first, second or third copy of v.

Note that since A is bounded-visit, and since P only performs simulations of A on up to three iterations of v, it is bounded visit as well. □

Construction of the 2PA. We now have all the ingredients to construct a 2PA whose language is non-empty if and only if $\mathcal{T}$ admits a critical pattern. More precisely, we show the following:

Lemma 5.11. *The language $Crit_{\mathcal{T}}$ is recognisable by a 2PA C of polynomial size (in $\mathcal{T}$).*

Proof. We do not give a formal construction of C, which would be unnecessarily too technical. Instead, we explain how C can check the six conditions $(1)-(6)$ of the definition of a critical pattern when reading valid encodings. The dimension of C is 2 (*i.e.* its vectors are in $\mathbb{N}^2$ or equivalently, it has two incrementing counters). We say that the counters are *unchanged* by a transition if the vector on that transition is $(0,0)$.

First C checks, by doing a left-to-right pass on the whole input w, whether w is a valid encoding, *i.e.* whether $w \in \vdash((\Sigma \times Q_P^2)^*\#)^2((\Sigma \times Q_P)^*\#)^4\dashv$. If this is the case, then C comes back to the beginning of u and proceed by checking other properties, otherwise it rejects. Note that this first verification only needs a constant number of states. During this phase, counters are unchanged.

Let us now assume that C has validated the type of its input w, which is then of the form $w = \vdash(u \otimes a_1 \otimes a_2)\#(v \otimes b_1 \otimes b_2)\#w_1\#z_1\#w_2\#z_2\#\dashv$. Let $u_i = u \otimes a_i$ and $v_i = v \otimes v_i$. Note that C can navigate between each $\#$-free factors of w by just keeping in its state whether it is in the first, second, up to sixth factor. This information can be updated when the separator $\#$ is met and is polynomial. The second observation is that condition (1) (u_1 and u_2 projects on the same Σ-word, as well as v_1 and v_2) is necessarily satisfied by definition of the encoding.

Let us explain how C checks condition (2) about idempotency of v_1 and v_2. We rely on the construction of Lemma 5.9 applied to the 2DBT $\widetilde{\mathcal{T}}$ where outputs are ignored (therefore, it is a deterministic two-way automaton). If this second phase accepts, C proceeds to checking conditions $(3)-(6)$. Otherwise it rejects. Again, during this verification, counters are unchanged.

We now turn to conditions $(3)-(6)$. The 2PA C first constructs a finite run ρ_1 on $u_1v_1w_1$ which does not end in v_1. Since ρ_1 is existentially quantified in the definition of critical patterns, this is where non-determinism of C is needed. To do so, C simulates $\widetilde{\mathcal{T}}$ on $u_1v_1w_1$. More precisely, it reads $\vdash (u \otimes a_1 \otimes a_2)\#(v \otimes b_1 \otimes b_2)\#w_1\#$ but ignores the third components a_2 and b_2 as well as the $\#$ symbols. During this simulation, outputs of $\mathcal{T}$ are also ignored. If $\mathcal{T}$ wants to cross the left boundary of w_1, then C rejects, otherwise condition (4) is not satisfied. If $\mathcal{T}$ is in v_1 but produces a non-empty output, then C rejects as well, ensuring condition (5). Non-deterministically, C decides to stop the simulation, but only if $\mathcal{T}$ is not in v_1 (otherwise, this would violate condition (4)). If it does not stop the simulation, then C will never accept its input. Stopping the simulation means that C has constructed a

run ρ_1 which satisfies conditions (4) and (5) (for $i = 1$). We call this simulation phase S_1. To construct a run ρ_2 on $u_2v_2w_2$, C proceeds similarly as in S_1. We call this simulation S_2. Note that both simulation requires only a polynomial number of states.

It remains to explain (i) how to check condition (6) and (ii) how to check condition (3), *i.e.* whether ρ_i can be extended into accepting runs over $u_iv_iw_iz_i^\omega$, for $i = 1, 2$.

The two counters are used to check condition (6), *i.e.* to check the existence of a mismatch between the outputs of ρ_1 and ρ_2. To do so, the simulation S_i, $i = 1, 2$, is again divided into two modes: in the first mode, whenever $\mathcal{T}$ produces some output word α, the ith counter of C is incremented by $|\alpha|$, using the vector $(|\alpha|, 0)$ for $i = 1$, and $(0, |\alpha|)$ for $i = 2$. So, the ith counter counts the length of the outputs produced so far by $\mathcal{T}$ on $u_iv_iw_i$. Non-deterministically, it eventually decides to stop counting this length, and increments the counter c_i one last time by some $0 \leq \ell \leq |\alpha|$. It then keeps in memory the c_ith letter produced by $\mathcal{T}$, say, $\tau_i \in \Gamma$. After this non-deterministic guess, which corresponds to existentially quantify an output position, the ith counter is left unchanged during the whole computation of C. The automaton C will accept only if $c_1 = c_2$ and $d_1 \neq d_2$, ensuring the existence of a mismatch between $\mathsf{out}(\rho_1)$ and $\mathsf{out}(\rho_2)$. Again, this needs only a polynomial number of states: simulation S_i is divided into two modes (before and after the non-deterministic guess), so, C only needs to remember in which mode it is.

To conclude, it remains to show how condition (3) can be checked, *i.e.* how C can check that ρ_i can be extended into an accepting run over $u_iv_iw_iz_i^\omega$. This is a direct application of Lemma 5.10. □

Finally, Lemma 5.6 is a direct consequence of Lemma 5.11 and Proposition 5.8.

Deciding continuity for regular functions. We prove the following theorem:

Theorem 5.12. *Deciding whether a regular function given as a deterministic two-way transducer with prophetic Büchi look-ahead is continuous is* PSpace*-c.*

Proof. The lower bound has been established in Theorem 5.7. For the upper-bound, let $\mathcal{T}$ be a $\mathsf{2DFT_{pla}}$ defining a regular function f, with set of look-ahead states Q_P. According to Proposition 5.3, one needs to check whether $\widetilde{\mathcal{T}}$ admits a critical pattern $u_1, v_1, w_1, z_1, u_2, v_2, w_2, z_2$ such that $\pi_\Sigma(u_1v_1^\omega) \in \mathsf{dom}(f)$. We have seen in the proof of Theorem 5.7 how to construct a 2PA P of polynomial size which accepts the set of (encodings of) critical patterns. Additionally here, P needs to check whether $\pi_\Sigma(u_1v_1^\omega) \in \mathsf{dom}(f)$. The difficulty lies in that $\pi_\Sigma(u_1v_1^\omega) \in \mathsf{dom}(f)$ is not equivalent to $u_1v_1^\omega \in \mathsf{dom}(\tilde{f})$, because $u_1v_1^\omega$ may not be a good annotated word (see Section 5.2.1). So, it is not correct to check whether $u_1v_1^\omega \in \mathsf{dom}(\tilde{f})$.

To overcome this difficulty, we modify the notion of critical pattern into *strong* critical pattern, which also includes some good annotation of $\pi_\Sigma(u_1v_1^\omega)$. A strong critical pattern is a tuple of words $(u_1', v_1', w_1, z_1, u_2, v_2, w_2, z_2)$ such that $u_1', v_1' \in (\Sigma \times Q_P \times Q_P)^*$, $(\pi_{1,2}(u_1'), \pi_{1,2}(v_1'), w_1, z_1, u_2, v_2, w_2, z_2)$ is a critical pattern and $\pi_{1,3}(u_1'(v_1')^\omega) \in \mathsf{dom}(\tilde{f})$, where $\pi_{1,2}$ is a morphism which only keeps the first and second alphabet components while $\pi_{1,3}$ only keeps the first and third components. We show that f is continuous if and only if $\mathcal{T}$ admits a strong critical pattern. If $\mathcal{T}$ admits a strong critical pattern $(u_1', v_1', w_1, z_1, u_2, v_2, w_2, z_2)$, then for $u_1 = \pi_{1,2}(u_1')$ and $v_1 = \pi_{1,2}(v_1')$, we have that $(u_1, v_1, w_1, z_1, u_2, v_2, w_2, z_2)$ is a critical pattern. Moreover, $\pi_{1,3}(u_1'(v_1')^\omega) \in \mathsf{dom}(\tilde{f})$, which implies that $\pi_\Sigma(u_1'(v_1')^\omega) = \pi_\Sigma(u_1v_1^\omega) \in \mathsf{dom}(f)$. Hence f is continuous.

Conversely, if f is continuous, $\widetilde{\mathcal{T}}$ admits a critical pattern $pat = (u_1, v_1, w_1, z_1, u_2, v_2, w_2, z_2)$ such that $\pi_\Sigma(u_1 v_1^\omega) \in \mathsf{dom}(f)$. Since the look-ahead automaton is prophetic, its accepting run on $\pi_\Sigma(u_1 v_1^\omega)$ is of the form $r = r_1 r_2^\omega$ such that r_1 is a run on $\pi_\Sigma(u_1 v_1^{k_1})$ for some $k_1 \geq 0$ and r_2 is a run on $\pi_\Sigma(v_1^{k_2})$ for some $k_2 \geq 1$. We then let $u_1' = (u_1 v_1^{k_1}) \otimes r_1$ and $v_1' = (v_1^{k_2}) \otimes r_2$. We also let $u_2' = u_2 v_2^{k_1}$ and $v_2' = v_2^{k_2}$. We claim that $(u_1', v_1', w_1, z_1, u_2', v_2', w_2, z_2)$ is a strong critical pattern. Clearly, $\pi_{1,3}(u_1'(v_1')^\omega) \in \mathsf{dom}(\tilde{f})$, because the annotation of $\pi_{1,3}(u_1'(v_1')^\omega)$ is $r_1 r_2^\omega$ which is an accepting run of the look-ahead automaton. It remains to show that $pat' = (\pi_{1,2}(u_1') = u_1 v_1^{k_1}, \pi_{1,2}(v_1') = v_1^{k_2}, w_1, z_1, u_2' = u_2 v_2^{k_1}, v_2' = v_2^{k_2}, w_2, z_2)$ is a critical pattern for $\widetilde{\mathcal{T}}$. It is immediate as v_1 and v_2 are idempotent and non-producing, so, iterating them does not change the existence of accepting runs whose outputs mismatch.

We have established that f is continuous if and only if $\widetilde{\mathcal{T}}$ admits a strong critical pattern. As shown in Lemma 5.11, the set of (encodings of) critical patterns is recognisable by a bounded-visit 2PA of polynomial size. We need here to slightly modify the 2PA constructed in Lemma 5.11 so that it also checks whether $\pi_{1,3}(u_1'(v_1')^\omega)$ is in the domain of $\mathcal{T}$. Based on Lemma 5.10, we can construct a bounded-visit two-way automaton of polynomial size checking the latter property. When this automaton has successfully checked the latter property, it is then sequentially composed with a 2PA as in Lemma 5.11 checking whether the input (where the third components of the alphabets are ignored) forms a critical pattern. The resulting composition is a bounded-visit 2PA of polynomial size, whose emptiness can be checked in PSpace, as shown in [FGM19]. □

6. Conclusion

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 in NLogSpace and PSpace respectively. 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.

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 [Boj18] 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

[AFT12] 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. URL: https://ieeexplore.ieee.org/xpl/conhome/6275587/proceeding, doi:10.1109/LICS.2012.18.

[Alo62] Alonzo Church. Logic, arithmetic and automata. In *Int. Congr. Math.*, pages 23–35, Stockholm, 1962.

[BGMP18] 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.

[Boj18] Mikolaj Bojanczyk. Polyregular functions. *CoRR*, abs/1810.08760, 2018. URL: http://arxiv.org/abs/1810.08760, arXiv:1810.08760.

[CCP17] 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. URL: http://www.dagstuhl.de/dagpub/978-3-95977-041-5, doi:10.4230/LIPIcs.ICALP.2017.115.

[CCP20] Michaël Cadilhac, Olivier Carton, and Charles Paperman. Continuity of functional transducers: A profinite study of rational functions. *Log. Methods Comput. Sci.*, 16(1), 2020. doi:10.23638/LMCS-16(1:24)2020.

[Cho03] Christian Choffrut. Minimizing subsequential transducers: a survey. *Theor. Comput. Sci.*, 292(1):131–143, 2003. doi:10.1016/S0304-3975(01)00219-5.

[CKLP15] 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.

[CPP08] 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.

[CSV13] 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. URL: https://ieeexplore.ieee.org/xpl/conhome/6570844/proceeding, doi:10.1109/LICS.2013.57.

[DFKL20] Vrunda Dave, Emmanuel Filiot, Shankara Narayanan Krishna, and Nathan Lhote. Synthesis of computable regular functions of infinite words. In *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, pages 43:1–43:17, 2020. doi:10.4230/LIPIcs.CONCUR.2020.43.

[DGN18] Vrunda Dave, Paul Gastin, and Krishna Shankara Narayanan. Regular transducer expressions for regular transformations. *CoRR*, abs/1802.02094, 2018. URL: http://arxiv.org/abs/1802.02094, arXiv:1802.02094.

[EFR20] 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.

[EHS21] Joost Engelfriet, Hendrik Jan Hoogeboom, and Bart Samwel. XML navigation and transformation by tree-walking automata and transducers with visible and invisible pebbles. *Theor. Comput. Sci.*, 850:40–97, 2021. doi:10.1016/j.tcs.2020.10.030.

[FGM19] 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. URL: http://www.dagstuhl.de/dagpub/978-3-95977-131-3, doi:10.4230/LIPIcs.FSTTCS.2019.40.

[Fil15] 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.

[FMR18] 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.

[Iba14] 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.

[JL69] J.R. Büchi and L.H. Landweber. Solving sequential conditions finite-state strategies. *Trans. Ameri. Math. Soc.*, 138:295–311, 1969.

[Kos74] S. Rao Kosaraju. Regularity preserving functions. *SIGACT News*, 6(2):16–17, April 1974. doi:10.1145/1008304.1008306.

[Pri02] 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.

[PS17] 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.

[Rud76] W. Rudin. *Principles of Mathematical Analysis*. International series in pure and applied mathematics. McGraw-Hill, 1976. URL: https://books.google.pl/books?id=kwqzPAAACAAJ.

[SH63] 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.

[SM76] 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.

[Wei00] 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.

Appendix A. Proofs from Section 2

Proof of Theorem 2.1. In [AFT12], authors show that a function is regular iff it is definable by $\mathsf{2DMT_{la}}$. We show that a function is $\mathsf{2DMT_{la}}$ definable iff it is $\mathsf{2DFT_{pla}}$ definable. Recall that $\mathsf{2DMT_{la}}$ is defined as $(\mathcal{T}, A, B)$ where $\mathcal{T}$ is a two-way deterministic transducer with Muller acceptance, A is a regular look-ahead automaton, given by a Muller automaton and B is a regular look-behind given by a finite state automaton.

Given a $\mathsf{2DFT_{pla}}$ $(\mathcal{A}, P)$, where $\mathcal{A} = (Q, \Sigma, \Gamma, \delta_{\mathcal{A}}, q_0)$ and $P = (Q_P, \Sigma, \delta_P, Q_0, F)$, we construct a $\mathsf{2DMT_{la}}$ $(\mathcal{T}, A, B)$ as follows: For every state $p \in Q_P$, we construct an equivalent Muller automaton A_p with initial state s_p s.t. $\mathcal{L}(P, p) = \mathcal{L}(A_p)$. The Muller look-ahead automaton A used in the $\mathsf{2DMT_{la}}$ $(\mathcal{T}, A, B)$ is the disjoint union of the Muller automata A_p for all $p \in Q_P$. $\mathcal{T} = (Q, \Sigma, \Gamma, \delta_{\mathcal{T}}, q_0, 2^Q \backslash \emptyset)$, and $\delta_{\mathcal{T}}$ is obtained by modifying the transition function $\delta_{\mathcal{A}}(q, a, p) = (q', \gamma, d)$ as $\delta_{\mathcal{T}}(q, a, s_p) = (q', \gamma, d)$. Since the language accepted by two distinct states of a prophetic automaton are disjoint, the language accepted by A_p and A'_p are disjoint for $p \neq p'$. The look-behind automaton B accepts all of Σ^*. It is easy to see that $(\mathcal{T}, A, B)$ is deterministic : on any position i of the input word $a_1 a_2 \ldots$, and any state $q \in Q$, there is a unique A_p accepting $a_{i+1} a_{i+2} \ldots$. The domain of $(\mathcal{T}, A, B)$ is the same as that of $(\mathcal{A}, P)$, since the accepting states of A are the union of the accepting states of all the $A_p, p \in Q_P$. Since all the transitions in $\delta_{\mathcal{T}}$ have the same outputs as in $\delta_{\mathcal{A}}$ for each (q, a, p), the function computed by $(\mathcal{A}, P)$ is the same as that computed by $(\mathcal{T}, A, B)$.

For the other direction, given a $\mathsf{2DMT_{la}}$, it is easy to remove the look-behind by a product construction [DGN18]. Assume we start with such a modified $\mathsf{2DMT_{la}}$ $(\mathcal{T}, A)$ with no look-behind. Let $\mathcal{T} = (Q_{\mathcal{T}}, \Sigma, \Gamma, \delta_{\mathcal{T}}, s_{\mathcal{T}}, \mathcal{F}_{\mathcal{T}})$ and $A = (Q_A, \Sigma, \delta_A, s_A, \mathcal{F}_A)$ where $\mathcal{F}_{\mathcal{T}}, \mathcal{F}_A$ respectively are the Muller sets corresponding to $\mathcal{T}$ and A. We describe how to obtain a corresponding $\mathsf{2DFT_{pla}}$ $(\mathcal{A}, P)$ with P, a prophetic Büchi look-ahead automaton. $\mathcal{A}$ is described as $(Q_{\mathcal{T}}, \Sigma, \Gamma, \delta, p_0)$. The prophetic look-ahead automaton is described as follows. Corresponding to each state $q \in Q_A$, let P_q be a prophetic automaton such that $\mathcal{L}(A, q) = \mathcal{L}(P_q)$ (this is possible since prophetic automata capture ω-regular languages [CPP08]). Since $\mathcal{A}$ has no accepting condition, we also consider a prophetic look-ahead automaton to capture $\mathsf{dom}(\mathcal{T})$. The Muller acceptance of $\mathcal{T}$ can be translated to a Büchi acceptance condition, and let $P_{\mathsf{dom}(\mathcal{T})}$ represent the prophetic automaton such that $\mathcal{L}(P_{\mathsf{dom}(\mathcal{T})}) = \mathsf{dom}(\mathcal{T})$. Thanks to the fact that prophetic automata are closed under synchronized product, the prophetic automaton P we need, is the product of P_q for all $q \in Q_A$ and $P_{\mathsf{dom}(\mathcal{T})}$. Assuming an enumeration $q_1, \ldots, q_n$ of Q_A, the states of P are $|Q_A| + 1$ tuples where the first $|Q_A|$ entries correspond to states of $P_{q_1}, \ldots, P_{q_n}$, and the last entry is a state of $P_{\mathsf{dom}(\mathcal{T})}$. Using this prophetic automaton P and transitions $\delta_{\mathcal{T}}$, we obtain the transitions δ of $\mathcal{A}$ as follows. Consider a transition $\delta_{\mathcal{T}}(p, a, q_i) = (q', \gamma, d)$. Correspondingly in $\mathcal{A}$, we have $\delta(p, a, \kappa) = (q', \gamma, d)$ where κ is a $|Q_A| + 1$ tuple of states such that the ith entry of κ is an initial state of P_{q_i}. From the initial state $s_{\mathcal{T}}$, on reading $\vdash$, if we have $\delta_{\mathcal{T}}(p_0, \vdash, q_i) = (q', \gamma, d)$, then $\delta(p_0, \vdash, \kappa) = (q', \gamma, d)$ such that the ith entry of κ is an initial state of P_{q_i} and the last entry of κ is an initial state of $P_{\mathsf{dom}(\mathcal{T})}$.

To see why $(\mathcal{A}, P)$ is deterministic. For each state $q \in Q_{\mathcal{T}}$, for each position i in the input word $a_1 a_2 \ldots a_i \ldots$, there is a unique state $p \in Q_A$ such that $a_{i+1} a_{i+2} \ldots$ is accepted by A. By our construction, the language accepted from each $p \in Q_A$ is captured by the prophetic automaton P_p; by the property of the prophetic automaton P, we know that for any $q_1, q_2 \in Q_A$ with $q_1 \neq q_2$, $L(P_{q_1}) \cap L(P_{q_2}) = \emptyset$. Thus, in $(\mathcal{A}, P)$, for each state q of $\mathcal{A}$, there is a unique state $p \in P$ such that the suffix is accepted by $L(P)$; further, from the

initial state of $\mathcal{A}$, from $\vdash$, there is a unique state $p \in P$ which accepts $\mathsf{dom}((\mathcal{T}, A))$. Hence $\mathsf{dom}((\mathcal{A}, P))$ is exactly same as $\mathsf{dom}((\mathcal{T}, A))$. For each $\delta_{\mathcal{T}}(q, a, q_i) = (q', \gamma, d)$, we have the transition $\delta(q, a, \kappa) = (q', \gamma, d)$ with the ith entry of κ equal to the initial state of P_{q_i}, which preserves the outputs on checking that the suffix of the input from the present position is in $L(P_{q_i})$. Notice that the entries $j \neq i$ of κ are decided uniquely, since there is a unique state in each P_q from where each word has an accepting run. Hence, $(\mathcal{A}, P)$ and $(\mathcal{T}, A)$ capture the same function. □

Remark A.1. Example function g given in Section 2, after Theorem 2.1 cannot be realised without look-ahead, not even if we allow non-determinism.