Synthesizing Computable Functions from Rational Specifications over Infinite Words

Emmanuel Filiot
Université libre de Bruxelles, Brussels, Belgium

Sarah Winter
Université libre de Bruxelles, Brussels, Belgium

Abstract

The synthesis problem asks to automatically generate, if it exists, an algorithm from a specification of correct input-output pairs. In this paper, we consider the synthesis of computable functions of infinite words, for a classical Turing computability notion over infinite inputs. We consider specifications which are rational relations of infinite words, i.e., specifications defined by non-deterministic parity transducers. We prove that the synthesis problem of computable functions from rational specifications is undecidable. We provide an incomplete but sound reduction to some parity game, such that if Eve wins the game, then the rational specification is realizable by a computable function. We prove that this function is even computable by a deterministic two-way transducer.

We provide a sufficient condition under which the latter game reduction is complete. This entails the decidability of the synthesis problem of computable functions, which we proved to be ExpTime-complete, for a large subclass of rational specifications, namely deterministic rational specifications. This subclass contains the class of automatic relations over infinite words, a yardstick in reactive synthesis.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Automata over infinite objects; Theory of computation → Transducers

Keywords and phrases uniformization, synthesis, transducers, continuity, computability

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.43

Related Version A preliminary version of this work is hosted on arXiv under a different title.
Preliminary Version: `https://arxiv.org/abs/2103.05674v1`
Full Version: `https://arxiv.org/abs/2103.05674`

Funding *Emmanuel Filiot*: This work is partially supported by the MIS project F451019F (F.R.S.-FNRS). Emmanuel Filiot is a research associate at F.R.S.-FNRS.
Sarah Winter: Sarah Winter is a postdoctoral researcher at F.R.S.-FNRS.

Acknowledgements The authors want to thank Nathan Lhote for a discussion on two-way transducers and Martin Zimmermann for pointing out a reference.

1 Introduction

Program synthesis aims at automatically generating programs from specifications. This problem can be formalized as follows. There are four parameters: two sets of input and output domains I, O, a set $\mathcal{S}$ of relations (called specifications) from I to O, and a set $\mathcal{I}$ of (partial) functions (called implementations) from I to O. Then, given a specification $S \in \mathcal{S}$ defining the correct input/output relationships, the synthesis problem asks to check whether there exists a function $f \in \mathcal{I}$ satisfying S in the following sense: its graph is included in S and it has the same domain as S (i.e., f is defined on $x \in I$ iff $(x, y) \in S$ for some $y \in O$). Using a set-theoretic terminology, f is said to *uniformize* S. Moreover in synthesis, if such an f exists, then the synthesis algorithm should return (a finite presentation of) it.

Program synthesis quickly turns to be undecidable depending on the four parameters mentioned before. Therefore, research on synthesis either turn to developing efficient sound but incomplete methods, see for example the syntax-guided synthesis approach [2] or bounded synthesis [14, 15], or restrict the class of specifications $\mathcal{S}$ and/or the class of implementations $\mathcal{I}$. A well-known example of the latter approach is reactive synthesis, where $\mathcal{S}$ are automatic relations[1] over infinite words, and $\mathcal{I}$ are Mealy machines [6, 23, 10]. Infinite words (over a finite alphabet) are used to model infinite executions of reactive systems, and Mealy machines are used as a model of reactive systems processing bit streams.

In this paper, our goal is to synthesize, from specifications which are semantically binary relations of infinite words, stream-processing programs, which are semantically *streaming computable* functions of infinite words (just called *computable* functions in the sequel). Let us now make the computability notion we use more precise. Let Σ and Γ be to finite alphabets. A partial function $f\colon \Sigma^\omega \to \Gamma^\omega$, whose domain is denoted $\mathrm{dom}(f)$, is said to be computable, if there exists a deterministic (Turing) machine M with three tapes, a read-only one-way input tape, a two-way working tape, and a write-only output tape that works as follows: if the input tape holds an input sequence $\alpha \in \mathrm{dom}(f)$, then M outputs longer and longer prefixes of $f(\alpha)$ when reading longer and longer prefixes of α. A definition of this machine model can be found, for instance, in [25].

▶ **Example 1.** Over the alphabet $\Sigma = \Gamma = \{a, b, A, B\}$, consider the specification given by the relation $R_1 = \{(ux\alpha, xu\beta) \mid u\alpha, u\beta \in \{a, b\}^\omega, x \in \{A, B\}\}$. The relation R_1 is automatic: an automaton needs to check that the input prefix u occurs shifted by one position on the output, which is doable using only finite memory. Checking that the first output letter x also appears after u on the input can also be done by storing x in the state. Note that some acceptance condition (e.g., parity) is needed to make sure that x is met again on the input. There is no Mealy machine which can realize R_1, because Mealy machines operate in a synchronous manner: they read one input symbol and must deterministically produce one output symbol. Here, the first output symbol which has to be produced depends on the letter x which might appear arbitrarily far in the input sequence. However, R_1 can be uniformized by a computable function: there is an *algorithm* reading the input from left to right and which simply waits till the special symbol $x \in \{A, B\}$ is met on the input. Meanwhile, it stores longer and longer prefixes of u in memory (so it needs unbounded memory) and once x is met, it outputs xu. Then, whatever it reads on the input, it just copies it on the output (realizing the identity function over the remaining infinite suffix α). Note that this algorithm produces a correct output under the assumption that x is eventually read.

Contributions. We first investigate the synthesis of computable functions from *rational specifications*, which are those relations recognizable by non-deterministic finite state transducers, i.e., parity automata over a product of two free monoids. We however show this problem is undecidable (Proposition 4). We then give an incomplete but sound algorithm in Section 3, based on a reduction to ω-regular two-player games. Given a transducer $\mathcal{T}$ defining a specification $\mathcal{R}_\mathcal{T}$, we show how to effectively construct a two-player game $\mathcal{G}_\mathcal{T}$, proven to be solvable in ExpTime, such that if Eve wins $\mathcal{G}_\mathcal{T}$, then there exists a computable function which uniformizes the relation recognized by $\mathcal{T}$, which can even be computed by some input-deterministic **two-way** finite state transducer (a transducer which whenever it reads an input symbol, it deterministically produces none or several output symbols and either moves forward or backward on the input). It is easily seen that two-wayness is

[1] relations recognized by two-tape parity automata alternatively reading one input and one output symbol.

necessary: the relation R_1 cannot be uniformized by any deterministic device which moves only forward over the input and only uses finitely many states, as the whole prefix u has to be remembered before reaching x. However, a two-way finite-state device can do it: first, it scans the prefix up to x, comes back to the beginning of the input, knowing whether $x = A$ or $x = B$, and then can produce the output.

Intuitively, in the two-player game we construct, called unbounded delay game, Adam picks the input symbols while Eve picks the output symbols. Eve is allowed to delay her moves an arbitrarily number of steps, gaining some lookahead information on Adam's input. We use a finite abstraction to store the lookahead gained on Adam's moves. We show that any finite-memory winning strategy in this game can be translated into a function uniformizing the specification such that it is computable by an input-deterministic two-way transducer.

In Section 4, we provide a sufficient condition $\mathcal{P}$ on relations for which the game reduction is complete. In particular, we show that if a relation R satisfies $\mathcal{P}$, then Eve wins the game iff R can be uniformized by a computable function. A large subclass of rational relations satisfying this sufficient condition is the class of deterministic rational relations (DRAT, [24]). Deterministic rational relations are those relations recognizable by deterministic two-tape automata, one tape holding the input word while the other holds the output word. It strictly subsumes the class of automatic relations, and, unlike for automatic relations, the two heads are not required to move at the same speed. Furthermore, when the domain of the relation is topologically closed for the Cantor distance[2], we show that strategies in which Eve delays her moves at most a bounded number of steps are sufficient for Eve to win. Such a strategy can in turn be converted into an input-deterministic **one-way** transducer. This entails that for DRAT-specifications with a closed domain (such as for instance specifications with domain Σ^ω, i.e., total domains), if it is uniformizable by a computable function, then it is uniformizable by a function computable by an input-deterministic one-way transducer.

Based on the completeness result, we prove our main result, that the synthesis problem of computable functions from deterministic rational relations is EXPTIME-complete. Hardness also holds in the particular case of automatic relations of total domain.

Total versus partial domains. We would like to emphasize here on a subtle difference between our formulation of synthesis problems and the classical formulation in reactive synthesis. Classically in reactive synthesis, it is required that a controller produces for *every* input sequence an output sequence correct w.r.t. the specification. Consequently, specifications with partial domain are by default unrealizable. So, in this setting, the specification R_1 of the latter example is not realizable, simply because its domain is not total (words with none or at least two occurrences of a symbol in $\{A, B\}$ are not in its domain). In our definition, specifications with partial domain can still be realizable, because the synthesized function, if it exists, can be partial and must be defined only on inputs for which there exists at least one matching output in the specification. A well-known notion corresponding to this weaker definition is that of uniformization [21, 9, 7, 13], this is why we often use the terminology "uniformizes" instead of the more widely used terminology "realizes". The problem of synthesizing functions which uniformize quantitative specifications has been recently investigated in [1]. In [1], it was called the good-enough synthesis problem, a controller being good-enough if it is able to compute outputs for all inputs for which there

[2] A set $X \subseteq \Sigma^\omega$ is *closed* if the limit, if it exists, of any sequence $(x_i)_i$ of infinite words in X is in X. The limit here is defined based on the Cantor distance, which, for any two infinite words u, v, is 0 if $u = v$ and otherwise $2^{-\ell}$ where ℓ is the length of their longest common prefix.

exists a least one matching output by the specification. The uniformization setting can be seen as an assumption that the input the program receives is not any input, but belongs to some given language. Related to that, there is a number of works on reactive synthesis under assumptions on the behavior of the environment [8, 4, 22, 11, 5].

Related work. To the best of our knowledge, this work is the first contribution which addresses the synthesis of algorithms from specifications which are relations over infinite inputs, and such that these algorithms may need unbounded memory, as illustrated by the specification R_1 for which any infinite-input Turing-machine realizing the specification needs unbounded memory. There are however two related works, in some particular or different settings.

First, in [12], the synthesis of computable functions has been shown to be decidable in the particular case of *functional* relations, i.e., graphs of functions. The main contribution of [12] is to prove that checking whether a function represented by a non-deterministic two-way transducer is computable is decidable, and that computability coincides with continuity (for the Cantor distance) for this large class of functions. The techniques of [12] are different to ours, e.g., games are not needed because output symbols functionally depends on input ones, even in the specification, so, there is not choice to be made by Eve.

Second, Hosch and Landweber [19] proved decidability of the synthesis problem of Lipschitz-continuous functions from automatic relations with *total* domain, Holtmann, Kaiser and Thomas [17] proved decidability of the synthesis of continuous functions from automatic relations with *total* domain, and Klein and Zimmermann [20] proved ExpTime-completeness for the former problem. So, we inherit the lower bound because automatic relations are particular DRAT relations, and as we show in the last section of the paper, the synthesis problem of computable functions is the same as the synthesis problem of continuous functions. We obtain the same upper bound as [20] for a more general class of specifications, namely DRAT, and in the more general setting of specifications with partial domain. As we show, total vs. partial domains make an important difference: two-way transducers may be necessary in the former case, while one-way transducers are sufficient in the latter. [17, 20] also rely on a reduction to two-player games called delay games, but for which bounded delay are sufficient. However, our game is built such that it accounts for the fact that unbounded delays can be necessary and it also monitors the domain, which is not necessary in [17, 20] because specifications have total domain. Accordingly, the main differences between [20] and our delay games are their respective winning objectives and correctness proofs. Another difference is that our game applies to the general class of rational relations, which are *asynchronous* (several symbols, or none, can correspond to a single input symbol) in contrast to automatic relations which are synchronous by definition.

Omitted and sketched proofs can be found in full in the full version.

2 Preliminaries

Words, languages, and relations. Let $\mathbb{N}$ denote the set of non-negative integers. Let Σ and Γ denote *alphabets* of elements called *letters* or *symbols*. A *word* resp. *ω-word* over Σ is an empty or non-empty finite resp. infinite sequence of letters over Σ. The empty word is denoted by ε, the length of a word by $|\cdot|$. Usually, we denote finite words by u, v, w, etc., and infinite words by α, β, γ, etc. Given an (in)finite word $\alpha = a_0 a_1 \cdots$ over Σ with $a_0, a_1, \dots \in \Sigma$, let $\alpha(i)$ denote the letter a_i, $\alpha(i\colon j)$ denote the infix $a_i a_{i+1} \cdots a_j$, $\alpha(\colon i)$ the prefix $a_0 a_1 \cdots a_i$, and $\alpha(i\colon)$ the suffix $a_i a_{i+1} \cdots$ for $i \leq j \in \mathbb{N}$. For two (in)finite words

α, β, let $\alpha \wedge \beta$ denote their longest common prefix. Let Σ^*, Σ^+, and Σ^ω denote the set of finite, non-empty finite, and infinite words over Σ, respectively. Let Σ^∞ denote $\Sigma^* \cup \Sigma^\omega$. A *language* resp. *$\omega$-language* L is a subset of Σ^* resp. Σ^ω, its set of prefixes is denoted by $\text{Prefs}(L)$. A (binary) *relation* resp. *ω-relation* R is a subset of $\Sigma^* \times \Gamma^*$ resp. $\Sigma^\omega \times \Gamma^\omega$. An ω-relation is just called a relation when infiniteness is clear from the context. The domain $\text{dom}(R)$ of a (ω)-relation R is the set $\{\alpha \mid \exists \beta\ (\alpha, \beta) \in R\}$. It is *total* if $\text{dom}(R) = \Sigma^*$ resp. Σ^ω. Likewise, we define $\text{img}(R)$ the image of R, as the domain of its inverse. A relation R is *functional* if for each $u \in \text{dom}(R)$ there is at most one v such that $(u, v) \in R$. *By default in this paper, relations and functions are partial, i.e., are not necessarily total.*

Automata. A *parity automaton* is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \Delta, c)$, where Q is a finite set of states, Σ a finite alphabet, $q_0 \in Q$ an initial state, $\Delta \subseteq Q \times \Sigma \times Q$ a transition relation, and $c \colon Q \to \mathbb{N}$ is a function that maps states to *priorities*, also called *colors*. A parity automaton is *deterministic* if its transition relation Δ is given as a transition function δ. We denote by δ^* the usual extension of δ from letters to finite words. A *run of $\mathcal{A}$ on a word $w \in \Sigma^\infty$* is a word $\rho \in Q^\omega$ such that $(\rho(i), w(i), \rho(i+1)) \in \Delta$ for all $0 \leq i \leq |w|$. A run on ε is a single state. We say that ρ begins in $\rho(0)$ and ends in $\rho(|w|)$ if w is finite. We define $\text{Occ}(\rho)$ as the set of states that occur in ρ, $\text{Inf}(\rho)$ as the set of states that occur infinitely often in ρ, and $c(\rho)$ as $c(\rho(0))c(\rho(1))\cdots$. A run ρ is *accepting* if $\rho \in Q^\omega$, $\rho(0) = q_0$ and $\max \text{Inf}(c(\rho))$ is even. The language *recognized* by $\mathcal{A}$ is the set $L(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \text{there is an accepting run } \rho \text{ of } \mathcal{A} \text{ on } \alpha\}$. A language $L \subseteq \Sigma^\omega$ is called *regular* if L is recognizable by a parity automaton.

One-way transducers. A *transducer* (1NFT) is a tuple $\mathcal{T} = (Q, \Sigma, \Gamma, q_0, \Delta, c)$, where Q is finite state set, Σ and Γ are finite alphabets, $q_0 \in Q$ is an initial state, $\Delta \subseteq Q \times \Sigma^* \times \Gamma^* \times Q$ is a finite set of transitions, and $c \colon Q \to \mathbb{N}$ is a parity function. It is *input-deterministic* (1DFT) (also called *sequential* in the literature) if Δ is expressed as a function $Q \times \Sigma \to \Gamma^* \times Q$. A *finite non-empty run* ρ is a non-empty sequence of transitions of the form $(p_0, u_0, v_0, p_1)(p_1, u_1, v_1, p_2) \dots (p_{n-1}, u_{n-1}, v_{n-1}, p_n) \in \Delta^*$. The *input (resp. output)* of ρ is $\alpha = u_0 \cdots u_{n-1}$ (resp. $\beta = v_0 \cdots v_{n-1}$). As shorthand, we write $\mathcal{T} \colon p_0 \xrightarrow{\alpha/\beta} p_n$. An *empty run* is denoted as $\mathcal{T} \colon p \xrightarrow{\varepsilon/\varepsilon} p$ for all $p \in Q$. Similarly, we define an *infinite run*. A run is *accepting* if it is infinite, begins in the initial state and satisfies the parity condition. In this paper, we also assume that for any accepting run ρ, its input and output are both infinite. This can be syntactically ensured with the parity condition. The relation *recognized* by $\mathcal{T}$ is $R(\mathcal{T}) = \{(\alpha, \beta) \mid \text{there is an accepting run of } \mathcal{T} \text{ with input } \alpha \text{ and output } \beta\}$. Note that with the former assumption, we have $R(\mathcal{T}) \subseteq \Sigma^\omega \times \Gamma^\omega$. A relation is called *rational* if it is recognizable by a transducer, we denote by RAT the class of rational relations. A *sequential* function is a function whose graph is $R(\mathcal{T})$ for an input-deterministic transducer $\mathcal{T}$.

Two-way transducers. Given Σ, let $\Sigma_\vdash$ denote $\Sigma \uplus \{\vdash\}$, $\vdash$ is a new left-delimiter symbol. An *input-deterministic two-way transducer* (2DFT) is a tuple $\mathcal{T} = (Q, \Sigma_\vdash, \Gamma, q_0, \delta, c)$, where Q is a finite state set, Σ and Γ are finite alphabets, $q_0 \in Q$ is an initial state, $\delta \colon Q \times \Sigma \to \times \Gamma^* \times \{1, -1\} \times Q$ is a transition function, and $c \colon Q \to \mathbb{N}$ is a function that maps states to colors. A two-way transducer has a two-way read-only input tape and a one-way write-only output tape. Given an input sequence $\alpha \in \Sigma^\omega$, let $\alpha(-1) = \vdash$, the input tape holds $\vdash \alpha$. We denote a transition $\delta(p, a) = (\gamma, d, q)$ as a tuple (p, a, γ, d, q), and Δ denotes the tuple representation of δ. A *run* of $\mathcal{T}$ on $\alpha \in \Sigma^\omega$ is a sequence of transitions $(q_0, \alpha(i_0), \gamma_0, d_0, q_1)(q_1, \alpha(i_1), \gamma_1, d_1, q_2) \cdots \in \Delta^\omega$ such that $i_0 = 0$, and $i_{k+1} = i_k + d_k$ for all $k \in \mathbb{N}$. The *input* of ρ is α and the *output* of ρ is $\beta = \gamma_0\gamma_1 \cdots$. We define $c(\rho)$ as the sequence of colors $c(q_0)c(q_1)\cdots$, ρ is *accepting* if $\max \mathrm{Inf}(c(\rho))$ is even. The functional ω-relation *recognized* by the deterministic-two way transducer is defined as $R(\mathcal{T}) = \{(\alpha, \beta) \mid$ there is an accepting run of $\mathcal{T}$ with input α and output $\beta\}$.

Games. A *game arena* is a tuple $G = (V_0, V_1, v_0, A, E)$, where $V = V_0 \uplus V_1$ is a set of vertices, V_0 belongs to Eve and V_1 to Adam, v_0 is an initial vertex, A is a finite set of actions, and $E \subseteq V \times A \times V$ is a set of labeled edges such that $(v, a, v') \in E$ and $(v, a, v'') \in E$ implies that $v' = v''$ for all $v \in V$ and $a \in A$. We assume that the arena is deadlock-free. We use letters on edges as it is more convenient to have them at hand for the proofs, it is however not necessary. A *play* in G is an infinite sequence $v_0a_0v_1a_1\cdots$ such that $(v_i, a_i, v_{i+1}) \in E$ for all $i \in \mathbb{N}$. Note that a play is uniquely determined by its action sequence. A *game* is of the form $\mathcal{G} = (G, \mathit{Win})$, where G is a game arena and $\mathit{Win} \subseteq V^\omega$ is a winning condition. Eve wins a play $\alpha = v_0a_0v_1a_1\cdots$ if $v_0v_1\cdots \in \mathit{Win}$, otherwise Adam wins. For ease of presentation, we also write $\alpha \in \mathit{Win}$.

A *strategy* for Eve resp. Adam is a function $(VA)^*V_0 \to A$ resp. $(VA)^*V_1 \to A$ such that $\sigma(xv) = a$ with $x \in (VA)^*$, $v \in V$, and $a \in A$ implies that there is $v' \in V$ such that $(v, a, v') \in E$. A play $v_0a_0v_1a_1\cdots$ is consistent with a strategy σ for Eve resp. Adam if $\sigma(v_0a_0\cdots v_i) = a_i$ for all $i \in \mathbb{N}$ with $v_i \in V_0$ resp. $v_i \in V_1$. A strategy σ for Eve is a *winning strategy* if $\alpha \in \mathit{Win}$ for all plays α consistent with σ. A *strategy automaton* for Eve is a tuple $\mathcal{S} = (M, V, m_0, \delta, \mu)$, where M is a finite set of (memory) states, V is the alphabet, m_0 is an initial state, $\delta\colon M \times V \to M$ is the memory update function, and $\mu\colon M \times V_0 \to A$ is the next action function such that for all $v \in V_0$ and $m \in M$, there is $v' \in V$ with $(v, \mu(m, v), v') \in E$.

Problem statement. In this section, we introduce the problem we want to solve. Let Σ, Γ be two finite alphabets. Given a relation $R \subseteq \Sigma^\omega \times \Gamma^\omega$ and a (partial) function $f : \Sigma^\omega \to \Gamma^\omega$, f is said to *uniformize* R if $\mathrm{dom}(f) = \mathrm{dom}(R)$ and $(\alpha, f(\alpha)) \in R$ for all $\alpha \in \mathrm{dom}(R)$. We also say that R is uniformizable by f or that f is a uniformizer of R. We are interested in computable uniformizers, which we now introduce.

▶ **Definition 2** ([25] computable functions)**.** *A function $f\colon \Sigma^\omega \to \Gamma^\omega$ is called* computable *if there exists a deterministic multi-tape machine M that computes f in the following sense, M has a read-only one-way input tape, a two-way working tape, and a write-only one-way output tape. All tapes are infinite to the right, finite to the left. For any finite word $w \in \Sigma^*$, let $M(w)$ denote the output*[3] *of M on w. The function f is said to be computable if for all $\alpha \in \mathrm{dom}(f)$ and $i \in \mathbb{N}$ there exists $j \in \mathbb{N}$ such that $f(\alpha)(:i)$ is prefix of $M(\alpha(:j))$.*

Note that in the above definition, checking whether the infinite input belongs to the domain is not a requirement and should not be, because in general, it is impossible to do it reading only a finite prefix of the input. That is why in this definition, we assume that the input belongs to the domain of the function. It is a reasonable assumption. For instance, the inputs may have been produced by another program (e.g., a transducer) for which one has guarantees that they belong to some well-behaved (e.g., regular) language.

▶ **Example 3.** To begin with, consider the function $f_1\colon \{a, b, c\}^\omega \to \{b, c\}^\omega$ defined by $f_1(a^nba^\omega) = b^\omega$ and $f_1(a^nca^\omega) = c^\omega$ for all $n \in \mathbb{N}_{\geq 1}$. It is computable by a TM which on inputs of the form a^nxa^ω for $x \in \{b, c\}$, outputs nothing up to reading x, and then, depending on x, either outputs c or b whenever it reads an a in the remaining suffix a^ω.

[3] The finite word written on the output tape the first time M reaches the $|w|$th cell of the input tape

Consider the function $f_2 : \{a,b\}^\omega \to \{a,b\}^\omega$ defined by $f_2(\alpha) = a^\omega$ if α contains infinitely many a and $f(\alpha) = b^\omega$ otherwise for all $\alpha \in \{a,b\}^\omega$. It is rational but not computable, because to determine even the first output letter, an infinite lookahead is needed.

Let $\mathcal{S}$ be a class of relations. The *$\mathcal{S}$-synthesis problem* asks, given a relation $S \in \mathcal{S}$ (finitely represented), whether there exists a computable function which uniformizes S. If such a function exists, then the procedure must return a TM computing it. Our first result, proved using an easy adaptation of the proof of [7, Theorem 17], showing that it is undecidable whether a given rational relation of finite words has a sequential uniformizer, is an undecidability result.

▶ **Proposition 4.** *The* RAT*-synthesis problem is undecidable, even if restricted to the subclass of rational relations with total domain.*

Proof sketch. We sketch a reduction from Post's correspondence problem. Let $u_1, \dots, u_n$ and $v_1, \dots, v_n$ be a PCP instance. We construct the ω-rational relation R that contains pairs (α, β) of the form $\alpha = i_1 \cdots i_k \alpha'$ with $i_1 \cdots i_k \in \{1, \dots, n\}^*$ and $\alpha \in \{a,b\}^\omega$ and $\beta = u_{i_1} \cdots u_{i_k} a^\omega$ if α' contains infinitely many a and $\beta \neq v_{i_1} \cdots v_{i_k} \beta'$ otherwise. If the PCP has no solution, then the function $f \colon i_1 \cdots i_k \alpha' \mapsto u_{i_1} \cdots u_{i_k} a^\omega$ uniformizes R, because $u_{i_1} \cdots u_{i_k} \neq v_{i_1} \cdots v_{i_k}$. The function f is clearly computable. If the PCP has a solution, no computable function uniformizes R. If the integer sequence $i_1 \cdots i_k$ is the solution, then $u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$. Intuitively, for an input sequence starting with the solution, no prefix of the input sequence allows to determine whether the output must begin with $u_{i_1} \cdots u_{i_k}$ or is not allowed to begin with $u_{i_1} \cdots u_{i_k}$.

The relation R can be made complete by also allowing all "invalid" inputs together with any output, i.e., by adding all pairs $(\alpha, \beta) \in \{1, \dots, n, a, b\}^\omega \times \{1, \dots, n, a, b\}^\omega$ where the input sequence α is not of the form $i_1 \cdots i_k \alpha'$ with $i_1 \cdots i_k \in \{1, \dots, n\}^*$ and $\alpha \in \{a,b\}^\omega$, and any output sequence β. Any Turing machine that computes f can easily be adapted to verify whether the input valid. ◀

Next, we give a semi-decision procedure for solving the RAT-synthesis problem which is sound but not complete. In Section 4, we introduce a sufficient condition for completeness which yields a (sound and complete) decision procedure for large classes of rational relations.

3 Unbounded Delay Game

In this section, given a rational relation (as a transducer), we show how to construct a finite-state ω-regular two-player game called (unbounded) delay game. We prove that if Eve wins this game then there exists a computable function which uniformizes the relation. Moreover, this function is computable by an input-deterministic two-way transducer. We analyze the complexity of solving the game, which turns out to be in ExpTime. Solving these games yields an incomplete, but sound, decision procedure for the RAT-synthesis problem.

In the game, Adam provides inputs and Eve must produce outputs such that combination of inputs and outputs is in the relation. However, as seen in Example 1, Eve might need to wait arbitrarily long before she can safely produce output. Hence, as the game is finite, it can not store arbitrary long input words, and Eve's actions cannot produce arbitrarily long words neither. Instead, we finitely abstract input and output words using a notion we call profiles. Informally, a profile of an input word stores the effects of the word (together with some output word) on the states of the transducer (that specifies the relation) as well as the maximal priority seen along the induced state transformation. Such profiles contain sufficient information to express a winning condition which makes sure that given the word of input symbols provided by Adam, if Eve would output concrete output words instead of their abstraction, she would produce infinitely often non-empty output words whose concatenation, together with the input word, belongs to the relation.

State transformation profiles. Let $R \subseteq \Sigma^\omega \times \Gamma^\omega$ be a rational relation given by a transducer $\mathcal{T} = (Q_\mathcal{T}, \Sigma, \Gamma, q_0^\mathcal{T}, \Delta_\mathcal{T}, c_\mathcal{T})$, and $C_\mathcal{T} = \mathrm{img}(c_\mathcal{T})$ its set of used priorities. Let $\mathcal{D} = (Q_\mathcal{D}, \Sigma, q_0^\mathcal{D}, \delta_\mathcal{D}, c_\mathcal{D})$ be a deterministic parity automaton that recognizes $\mathrm{dom}(R)$, and $C_\mathcal{D} = \mathrm{img}(c_\mathcal{D})$ its set of used priorities; $\mathcal{D}$ can always be constructed from $\mathcal{T}$ by projecting away its outputs and by determinizing the resulting automaton.

Given $u \in \Sigma^*$, its *profile* P_u are all the possible state transformations it induces for any output. Formally, $P_u \subseteq Q_\mathcal{T} \times Q_\mathcal{T} \times C_\mathcal{T}$ is defined as $\big\{(p, q, c) \mid$ there is $v \in \Gamma^*$ and there is a run ρ of the form $\mathcal{T} \colon p \xrightarrow{u/v} q$ with $\max \mathrm{Occ}(\rho) = c\big\}$. Profiles can be multiplied as $P_1 \otimes P_2 = \{(p, r, \max\{m, n\}) \mid \exists q \colon (p, q, m) \in P_1, (q, r, n) \in P_2\}$. Given $u_1, u_2 \in \Sigma^*$, it is easy to verify that $P_{u_1 u_2} = P_{u_1} \otimes P_{u_2}$, and P_ε is neutral for $\otimes$.

Finite-state unbounded delay game. We now present a two-player ω-regular game $\mathcal{G}_\mathcal{T} = (G, \mathit{Win})$ such that if Eve has a winning strategy, then R has a computable uniformizer. In this game, Adam's actions are to provide input letters, letter-by-letter. Eve's goal is to construct a sequence of state transformations $(q_0, q_1, m_1)(q_1, q_2, m_2) \dots$ such that if the infinite input $\alpha \in \Sigma^\omega$ provided by Adam is in $\mathrm{dom}(R)$, then (i) the maximal priority seen infinitely often in $(m_i)_i$ is even and (ii) $\alpha = u_0 u_1 \cdots$ for some $u_i \in \Sigma^*$ such that $(q_i, q_{i+1}, m_{i+1}) \in P_{u_i}$ for all $i \geq 0$. As a consequence, all these finite runs can be concatenated to form an accepting run on $\alpha / v_0 v_1 \dots$, entailing $(\alpha, v_0 v_1 \dots) \in R$. One can then show that if Eve has a strategy to pick the state transformations while ensuring the latter property, then this strategy can be turned into a computable function, and conversely. Picking a state transformation is what we call a *producing action* for Eve. Since a state transformation picked by Eve may correspond to an arbitrarily long word u_i, she also has an action *skip* which allows her to wait before making such a producing action. Now, the difficulty for Eve is to decide when she makes producing actions, in other words, how to decompose the input α, only based on prefixes of α. To that end, before picking a state transformation, she may need to gather lookahead information from Adam. Consequently, the vertices of the game manipulates two consecutive profiles P_1 and P_2, with the invariant that P_1 is the profile of u_i while P_2 is the profile of u_{i+1}, when the input played so far by Adam is $u_0 \dots u_{i+1}$. When Eve knows enough, she picks a state transformation (q_i, q_{i+1}, m_i) in P_1, then P_1 becomes P_2 and P_2 is reset to P_ε. The inputs of Adam up to the next producing action of Eve form the word u_{i+2}, and so on. The vertices of the game also store information to decide whether the input belongs to the domain of R (states of $\mathcal{D}$), the parities m_i, as well as the states $q_0, q_1, \dots$. Formally, the game graph $G = (V, E)$ is composed of vertices of the form $\big(q, c, P_1, P_2, r\big) \times \{\forall, \exists\}$, where

- $q \in Q_\mathcal{T}$, *State reached on the combination of input and output sequence.*
- $c \in \{-1\} \cup C_\mathcal{T}$, *Priorities of the state transformations, -1 is used to indicate that no state transformation was chosen (skip action below).*
- P_1, P_2, *Profiles obtained from the given lookahead of the input word.*
- $r \in Q_\mathcal{D}$. *State reached on the given lookahead of the input word.*

From a vertex of the form $\big(q, c, P_1, P_2, r, \forall\big)$, Adam has the following actions:

- $\xrightarrow{a} \big(q, -1, P_1, P_2 \otimes P_a, \delta_\mathcal{D}(r, a), \exists\big)$, for all $a \in \Sigma$.
  Adam provides the next lookahead letter and P_2 is updated accordingly.

From a vertex of the form $\big(q, c, P_1, P_2, r, \exists\big)$, Eve has the following actions:

- $\xrightarrow{skip} \big(q, -1, P_1, P_2, r, \forall\big)$, and

  Eve makes a non-producing action, i.e., she waits for further lookahead on the input.
- $\xrightarrow{(q,q',m)} \big(q', m, P_2, P_\varepsilon, r, \forall\big)$, where $(q, q', m) \in P_1$.

  Eve makes a producing action: a state transformation from the first lookahead profile is chosen, the state transformation is applied, and the first profile is consumed.

The initial vertex of the game is $\big(q_0^{\mathcal{T}}, -1, P_\varepsilon, P_\varepsilon, q_0^{\mathcal{D}}, \forall\big)$.

Let us now define $Win \subseteq V^\omega$. The condition makes sure that if the input sequence provided by Adam is in the domain of R, then the sequence of state transformations can be used to build on accepting run of $\mathcal{T}$ on that input. $Win \subseteq V^\omega$ is the set of all plays γ satisfying the property

$$\max \mathrm{Inf}(col_{\mathcal{D}}(\gamma)) \text{ is even} \rightarrow \max \mathrm{Inf}(col_{\mathcal{T}}(\gamma)) \text{ is even},$$

where $col_{\mathcal{D}}(\gamma) = c_{\mathcal{D}}(\pi^5(\gamma))$, $col_{\mathcal{T}}(\gamma) = \pi^2(\gamma)$, and $\pi^i(\gamma)$ is the projection of γ onto the ith component of each vertex. It is not difficult to see that Win is ω-regular, e.g., one can design a parity automaton for it.

We explain the intuition behind Win. Our goal is to extract a computable function that uniformizes the relation from a winning strategy. Intuitively, there is a computable function that uniformizes R, if every input word $\alpha \in \mathrm{dom}(R)$ can be read letter-by-letter, and from time to time, a segment of output letters is produced, continuously building an infinite output word β such that $(\alpha, \beta) \in R$. We relate this to Win. Recall that R is defined by $\mathcal{T}$, and $\mathrm{dom}(R)$ by $\mathcal{D}$. Given a play γ, there is a unique input word $\alpha \in \Sigma^\omega$ that corresponds to γ. Since we are looking to build a computable function f with $\mathrm{dom}(f) = \mathrm{dom}(R)$, we care whether $\alpha \in \mathrm{dom}(R)$. The ω-word $col_{\mathcal{D}}(\gamma)$ is equal to $c(\rho_{\mathcal{D}})$, where $\rho_{\mathcal{D}}$ is the run of $\mathcal{D}$ on α. If $\max \mathrm{Inf}(col_{\mathcal{D}}(\gamma))$ is even, $\alpha \in L(\mathcal{D})$, i.e., $\alpha \in \mathrm{dom}(R)$. An output word $\beta \in \Gamma^\infty$ that corresponds to γ is only indirectly defined, instead the play defines a (possibly finite) sequence of state transformations that an output word β should induce together with α. How to extract a concrete β from γ is formally defined in the proof of Theorem 5. The ω-word $col_{\mathcal{T}}(\gamma)$ contains the relevant information to determine whether $(\alpha, \beta) \in R(\mathcal{T})$, i.e., $(\alpha, \beta) \in R$. In particular, if β is finite, $\max \mathrm{Inf}\, col_{\mathcal{T}}(\gamma)$ is -1, that means that only finitely many producing actions have been taken. If $\max \mathrm{Inf}\, col_{\mathcal{T}}(\gamma)$ is even, we have that $(\alpha, \beta) \in R$. Thus, Win expresses that if $\alpha \in L(\mathcal{D})$, then there is some $\beta \in \Gamma^\omega$, which can be built continuously while reading α such that $(\alpha, \beta) \in R$.

From winning strategies to uniformizers. We are ready to state our first positive result: If Eve has a winning strategy in the unbounded delay game $\mathcal{G}_{\mathcal{T}}$, then $R(\mathcal{T})$ is uniformizable by a computable function. In fact, we show a more precise result, namely, that if Eve has a winning strategy, then the relation is uniformizable by a function recognized by a deterministic two-way parity transducer. Additionally, if the domain of the relation is closed[4], then a deterministic one-way transducer suffices. Just as (one-way) transducers extend parity automata with outputs on their transitions, input-deterministic two-way transducers extend deterministic two-way parity automata with outputs. The reading tape is two-way, but the output tape is one-way. The class of functions recognizable by 2DFTs is smaller than the class of computable functions and enjoys many good algorithmic properties, e.g., decidability

[4] Recall Footnote 2.

Algorithm 1 Algorithm computing continuous function f that uniformizes R. The algorithm is described in the proof sketch of Theorem 5.

```
Input: $\alpha \in \Sigma^\omega$, $G$ game arena, $\mathcal{S} = (M, V, m_0, \delta, \mu)$ strategy automaton
Output: $\beta \in \Gamma^\infty$, if $\alpha \in \mathrm{dom}(R)$, then $(\alpha, \beta) \in R$
1  $m := m_0$ ;                         // current state of the strategy automaton
2  $u_1 := \varepsilon$ ;                                        // first input block
3  $u_2 := \varepsilon$ ;                                       // second input block
4  $s_{prev} := s_0$, initial vertex of $G$ ;           // previous vertex in the game
5  $s_{cur} := s_0$ ;                                   // current vertex in the game
6  $a := \alpha(0)$ ;                                      // current letter of $\alpha$
7  while true do
8      $u_2 := u_2.a$ ;
9      $s_{cur} := s$ if $s_{cur} \xrightarrow{a} s \in E$ ;     // update game vertex according to Adam's
        action
10     $m := \delta(m, s_{cur})$ ;  // strategy automaton is updated with Adam's action
11     $s_{prev} := s_{cur}$ ;
12     $s_{cur} := s$ if $s_{cur} \xrightarrow{\mu(m, s_{cur})} s \in E$ ;       // strategy automaton yields Eve's
        action, the updated game vertex is of the form $(\cdot, \cdot, P_{u_1}, P_{u_2}, \cdot, \cdot)$
13     $m := \delta(m, s_{cur})$ ;   // strategy automaton is updated with Eve's action
14     if $e := (s_{prev}, (p, q, c), s_{cur})$ is a producing edge then
15         choose output block $v_1 \in \Gamma^*$ such that $\mathcal{T}\colon p \xrightarrow{u_1/v_1} q$ with max prio $c$ ;
           ; // this choice can be made canonical by computing for instance
            the smallest word in lexicographic order satisfying this
            property
16         $u_1 := u_2$ ;
17         $u_2 := \varepsilon$ ;
18         $print(v_1)$ ;                                  // produce output block
19     end
20     $a := \alpha.nextLetter()$ ;                         // read next input letter
21 end
```

of the equivalence problem [3]. Note that any function recognizable by a 2DFT is computable, in the sense that it suffices to "execute" the 2DFT to get the output. So, from now on, we may freely say that a function is computable by a 2DFT.

▶ **Theorem 5.** *Let R be defined by a transducer $\mathcal{T}$. If Eve has a winning strategy in $\mathcal{G}_\mathcal{T}$, then R is uniformizable by a function computable by a 2DFT.*

Proof sketch. If Eve has a winning strategy in $\mathcal{G}_\mathcal{T}$, then she also has a finite-state winning strategy because the winning condition is ω-regular. From such a strategy we can build an algorithm (a Turing machine), see Algorithm 1, that computes a function f that uniformizes R. The high-level idea of the algorithm is to simulate the strategy, which abstracts inputs and outputs by profiles, and in parallel store concrete inputs and outputs corresponding to those profiles. This is possible as Turing machines have infinite storage capacity. In the algorithm, an input sequence α is read letter-by-letter and the corresponding play in $\mathcal{G}$ is simulated where Adam plays according to α and Eve according to her winning strategy. In the play, a lookahead $u_i \in \Sigma^*$ gained on Adam's input is stored as its profile P_{u_i}. In the algorithm, the lookahead u_i is stored concretely in addition to its profile P_{u_i}. When Eve takes an action she picks a state transformation (that should occur in the transducer) from $P_{u_{i-1}}$, the profile of the previous lookahead sequence u_{i-1}, also stored by the algorithm. The algorithm picks some $v_{i-1} \in \Gamma^*$ such that u_{i-1}/v_{i-1} induces the state transformation picked by Eve. A new non-empty lookahead $u_{i+1} \in \Sigma^*$ is built, stored as its profile $P_{u_{i+1}}$ in the play and as the concrete sequence u_{i+1} in the algorithm until Eve picks a state transformation from P_{u_i}, and so on. The lookaheads that are built are non-empty (except for the first one), and since Eve plays according to her winning strategy, the sequence of state transformations she picked can be used to build an accepting run of $\mathcal{T}$ on $(u_0u_1\dots, v_0v_1\dots)$, proving that the latter pair belongs to R.

Then we show that f can be actually recognized by a 2DFT. The main idea is to use two-wayness to encode finite lookahead over the input: the reading head goes forward to gather input information, and then must return to the initial place where the lookahead was needed to transform the input letter. The difficulty is for the 2DFT to return to the correct position, even though the lookahead can be arbitrarily long. In order to find the correct positions, we make use of a finite-state strategy automaton for Eve's winning strategy in the following sense. A (left-to-right) run of the strategy automaton on the input word yields a unique segmentation of the input, such that segments i and $i+1$ contain enough information to determine the output for segment i. The idea is to construct a 2DFT that simulates the strategy automaton in order to find the borders of the segments. If the 2DFT goes right, simulating a computation step of the deterministic strategy automaton is easy. Recovering the previous step of the strategy automaton when the 2DFT goes left is non-trivial, it is possible to compute this information using the Hopcroft-Ullman construction presented in [18]. We show that having the knowledge of the profiles of segments i and $i+1$ is enough to deterministically produce a matching output for segment i on-the-fly going from left-to-right over segment i again. ◀

We make some remarks about the form of the game, in particular the use of two lookahead profiles, instead of one. Assume we would have only one profile abstracting the lookahead over Adam inputs. For simplicity, assume the specification is automatic (i.e., letter-to-letter). Suppose, so far, Adam and Eve have alternated between providing an input letter and producing an output letter (in the finite-state game, Eve producing letter(s) corresponds to the abstract action of picking state transformations), but now, she needs to wait for more inputs before she can safely output something new. Suppose that Adam has provided some more input, say the word u, and Eve now has enough information about the input to be able to produce something new. Abstractly, it means that in the game, Adam has given the word u but only its profile P is stored. Eve might not be able to produce an output of the same length as u (for example, if producing the ith output letter depends on the $i+k$th input letter). So, she cannot consume the whole profile P (i.e., pick a state transformation in P). What she has to do, is to decompose the profile P into two profiles such that $P = P_1 \otimes P_2$ and pick a state transformation in P_1, and then continue the game with profile P_2 (and keep on updating it until she can again produce something). The problem is, firstly, that there is no unique way of decomposing P as $P_1 \otimes P_2$, and secondly, P_1 might not correspond to any prefix of u. That is why it is needed to have explicitly the decomposition at hand in the game construction.

▶ **Lemma 6.** *Deciding whether Eve has a winning strategy in $\mathcal{G}_{\mathcal{T}}$ is in* ExpTime.

Proof sketch. Two-player ω-regular games are decidable (see, e.g., [16]). The claimed upper bound is achieved by representing the winning condition as a deterministic parity automaton, carefully analyzing its size, and then solving a parity game. ◀

▶ Remark 7. The converse of Theorem 5 is not true.

Clearly, if the converse was true, the RAT-synthesis problem would be decidable, which is a contradiction to Proposition 4. We also give a small example that illustrates that uniformizability does not imply the existence of a winning strategy.

▶ **Example 8.** Consider the identity function $f\colon \{a,b\}^\omega \to \{a,b\}^\omega$ such that all inputs with either finitely many a or b are in the domain. A (badly designed) letter-to-letter transducer $\mathcal{T}$ that recognizes f has five states S, A, B, A', B', where S is the starting state, A, B (resp. A', B') are used to recognize finitely many b (resp. a), and from S, the first input/output letter non-deterministically either enters A or A'. In a play in $\mathcal{G}_\mathcal{T}$, at some point, Eve must make her first output choice, i.e., she starts to build a run of $\mathcal{T}$. This choice fixes whether the run is restricted to A, B or A', B'. No matter Eve's choice, Adam can respond with an infinite sequence of either only a (for A, B) or b (for A', B'), making it impossible to build an accepting run. Thus, Eve has no winning strategy, but clearly the function f is computable.

While in the above example, the point of failure is clearly the bad presentation of the specification, this is not the case in general. Recall the proof sketch of Proposition 4, where we provide a reduction from Post's correspondence problem. A non-deterministic transducer constructed from a given PCP instance $u_1, v_1, \dots, u_n, v_n$ can guess whether the input word contains infinitely many a, and accordingly either checks that the output begins with $u_{i_1} \cdots u_{i_k}$ for input sequences beginning with indices $i_1 \cdots i_k$, or checks that it does not begin with a prefix equal to $v_{i_1} \cdots v_{i_k}$. As detailed in the proof sketch, if the PCP instance has no solution, there is a computable uniformization, however, using the same argumentation as in the above example, such a transducer would make it impossible to have a winning strategy. In order to have a winning strategy, the transducer must be changed such that it checks at the same time whether the output starts with $u_{i_1} \cdots u_{i_k}$ and does not start with something equal to $v_{i_1} \cdots v_{i_k}$ for input sequences beginning with $i_1 \cdots i_k$. In general, depending on the PCP instance, it is not possible to make both checks in parallel.

We state a lemma about bounded delay.

▶ **Lemma 9.** *Let R be defined by a transducer $\mathcal{T}$. If there exists $\ell \geq 0$, such that Eve has a winning strategy in $\mathcal{G}_\mathcal{T}$ with at most ℓ consecutive* skip*-moves, then R is uniformizable by a function computable by a 1DFT.*

Intuitively, such a strategy yields a function computable by a 1DFT, because the needed lookahead (as it is bounded) can be stored in the state space.

4 A Sufficient Condition for Completeness

As we have seen in the previous section,

▶ Remark 10. Theorem 5 yields a semi-decision procedure for solving the RAT-synthesis problem (it is sound but not complete).

In this section, we show that the procedure is complete for two known and expressive classes of rational relations, namely the class of automatic relations (AUT), which are for example used as specifications in Church synthesis, as well as the class of deterministic rational relations (DRAT) [24] (to be formally defined below), see Corollary 13.

To arrive at these results, we define a structural restriction on transducers that turns out to be a sufficient condition for completeness. Let $\mathcal{T}$ be a transducer. An *input* (resp. *output*) state is a state p from which there exists an outgoing transition (p, u, v, q) such that $u \neq \varepsilon$ (resp. $v \neq \varepsilon$). The set of input (resp. output) states is called Q_i (resp. Q_o). A transducer $\mathcal{T}$ has *property* $\mathcal{P}$ if for all words $u \in \Sigma^*$, all $v_1, v_2 \in \Gamma^*$ such that v_1 is a prefix of v_2 the following holds:

$$\text{if } \mathcal{T}\colon p \xrightarrow{u/v_1} q, \mathcal{T}\colon p \xrightarrow{u/v_2} r, \text{ and } q, r \text{ are input states, then } q = r.$$

This property implies that, given $\alpha \in \Sigma^\omega, \beta \in \Gamma^\omega$ such that there is a run of $\mathcal{T}$ with input α and output β, for each prefix $u \in \Sigma^*$ of α there exists a unique prefix $v \in \Gamma^*$ of β that has to be produced while reading u before the remainder α' (let $\alpha = u\alpha'$) can be read. Furthermore, the target state of $\mathcal{T}\colon q_0 \xrightarrow{u/v}$ is unique and this state is exited only when further input (from α') has to be read. In simpler terms, given a prefix of an input word, it is uniquely determined how long the prefix of a given output word has to be so that further input can be processed and the reached state is unique. This implies that input prefixes together with (long enough) output prefixes are sufficient to determine the beginning of an accepting run if such a run exists. This allows us to show the following.

▶ **Theorem 11.** *Let R be defined by a transducer $\mathcal{T}$ with property $\mathcal{P}$. If R is uniformizable by a computable function, then Eve has a winning strategy in $\mathcal{G}_\mathcal{T}$.*

Proof sketch. In fact, we explain how to construct a winning strategy from a continuous (a computable function is always continuous, see Section 5) uniformizer f of R. Given $\alpha \in \mathrm{dom}(R)$ and $f(\alpha)$, we show that it is possible to decompose the input α into $u_0u_1\cdots$ and the output $f(\alpha)$ into $v_0v_1\cdots$ such that there exists an accepting run $\mathcal{T}\colon q_0 \xrightarrow{u_0/v_0} q_1 \xrightarrow{u_1/v_1} q_2\cdots$ where each q_i is an input state for $i > 0$. Moreover, this decomposition and run can be determined in a unique way and on-the-fly, in the sense that a factor u_i/v_i only depends on the factors $u_0/v_0,\ldots, u_{i-1}/v_{i-1}$. This makes it possible for Eve to pick a corresponding state transformation sequence $(q_0, q_1, c_0)(q_1, q_2, c_1)\cdots$ which is only dependent on *the so far seen* actions of Adam spelling $u_0u_1\cdots$. The main idea to determine the u_i is to look at the indices j for which the longest common prefix of the sets $S_j = \{f(\alpha(:j)\beta) \mid \alpha(:j)\beta \in \mathrm{dom}(R)\}$ strictly increases. Given u_i, the output $v_0v_1\cdots v_i$ is any common prefix of the sets S_j, such that a run $\mathcal{T}\colon q_i \xrightarrow{u_i/v_i}$ is defined and its target is an input state. The fact that $\mathcal{T}$ has property $\mathcal{P}$ guarantees that each of these runs has the same target, thus, the next state transformation (q_i, q_{i+1}, c_i) is uniquely determined. ◀

We turn to the setting of closed domains.

▶ **Lemma 12.** *Let R with* dom(R) *closed be defined by a transducer $\mathcal{T}$ with property $\mathcal{P}$. If R is uniformizable by a computable function, then there exists a computable $\ell \geq 0$ such that Eve has a winning strategy in $\mathcal{G}_\mathcal{T}$ with at most ℓ consecutive* skip*-moves.*

Proof sketch. Intuitively, the reason why bounded lookahead suffices in the setting of closed domains is that (basically at each point of time during a play) Adam's moves describe a series of longer and longer finite input words that "converge" to a valid infinite input word from the domain. Hence, Eve can not wait arbitrarily long to make producing moves, as such a play describes a valid infinite input sequence and a finite output sequence. ◀

We formally introduce AUT and DRAT. A relation is *deterministic rational* if it is recognized by a transducer where Q_i and Q_o partition its state space, and its transition relation Δ is a function $(Q_i \times \Sigma \times \{\varepsilon\} \to Q) \cup (Q_o \times \{\varepsilon\} \times \Gamma \to Q)$. It is *automatic* if additionally Δ strictly alternates between Q_i and Q_o states. It is easy to see that every DRAT-transducer (and a fortiori every AUT-transducer) satisfies the property $\mathcal{P}$. In general, given any transducer $\mathcal{T}$, we do not know if it is decidable whether $\mathcal{T}$ has property $\mathcal{P}$.

Main result. We now state our main result: Asking for the existence of a uniformization which is computable by a Turing machine or computable by an input-deterministic two-way transducer (2DFT), are equivalent questions, as long as specifications are DRAT relations. Moreover, these questions are decidable.

▶ **Corollary 13.** *Let R be defined by a* DRAT*-transducer $\mathcal{T}$. The following are equivalent:*
1. *R is uniformizable by a computable function.*
2. *R is uniformizable by a function computable by a 2DFT.*
3. *Eve has a winning strategy in $\mathcal{G}_{\mathcal{T}}$.*

Moreover, if $\mathrm{dom}(R)$ *is closed, then it is equivalent to R being uniformizable by a function computable by a 1DFT.*

Note that the above result also holds for the slightly more general case of relations given by transducers with property $\mathcal{P}$. We highlight two facts regarding closed domains.

▶ Remark 14. The set of infinite words over a finite alphabet is closed, i.e., every total domain is closed. Furthermore, it is decidable whether a domain (e.g., given by a Büchi automaton) is closed. It is a well-known fact that the topological closure of a Büchi language is a Büchi language (one can trim the automaton and declare all states to be accepting) and therefore one can check closedness by checking equivalency with its closure.

▶ **Theorem 15.** *The* AUT*- and* DRAT*-synthesis problems are* ExpTime-complete.

Proof. Membership in ExpTime directly follows from Lemma 6 and Corollary 13. In [20] it was shown that this problem is ExpTime-hard in the particular case of automatic relations with total domain, so the lower bound applies to our setting. ◀

5 Discussion

Continuous functions. We have shown that checking the existence of a computable function uniformizing a relation given by transducer with property $\mathcal{P}$ is decidable (a consequence of Theorems 5 and 11 and Lemma 6). The proofs of Theorems 5 and 11 use another notion, which is easier to manipulate mathematically than computability, that of continuity. A function is called *continuous* if

$$\forall \alpha \in \mathrm{dom}(f)\ \forall i \in \mathbb{N}\ \exists j \in \mathbb{N}\ \forall \beta \in \mathrm{dom}(f)\colon |\alpha \wedge \beta| \geq j \rightarrow |f(\alpha) \wedge f(\beta)| \geq i. \tag{1}$$

▶ **Example 16.** Consider the function f_1 of Example 3. f_1 is continuous, because the ith output symbol only depends on the $max(i, n+1)$ first input symbols. Consider the function f_2 of Example 3. The function f_2 is clearly rational, but it is not continuous. We verify that f_2 is not continuous, let α_n denote $a^n b^\omega$, we have that $|\alpha_n \wedge a^\omega| = n$ and $|f_2(\alpha_n) \wedge f_2(a^\omega)| = 0$ for all $n \in \mathbb{N}$. Thus, f_2 is not continuous.

The notions of computability and continuity are closely related. If a function $f\colon \Sigma^\omega \to \Gamma^\omega$ is computable, it is also continuous. This is not difficult to see when comparing the definitions of computable and continuous functions. The converse does not hold because the continuity definition does not have any computability requirements (see [12] for a counter-example). However, regarding synthesis, the two notions coincide:

▶ **Theorem 17.** *Let R be defined by $\mathcal{T}$ with property $\mathcal{P}$. The following are equivalent:*
1. *R is uniformizable by a continuous function.*
2. *R is uniformizable by a computable function.*

Proof. Indeed, any computable uniformizer is continuous. Theorem 11 states that if there exists a computable uniformizer, then there exists a winning strategy in the delay game. However, in the proof of this theorem, we show a stronger statement: If there exists a continuous uniformizer, then there exists a winning strategy in the delay game. Such a strategy can be assumed to have finite-memory (as finite-memory suffices to win games with ω-regular conditions). We have shown in the proof (sketch) of Theorem 5 how to translate a finite-state winning strategy into an algorithm (a Turing machine) that computes a function f which uniformizes the relation. ◀

Conclusion and future work. We investigated the synthesis of algorithms (aka. Turing machine computable functions) from rational specifications. While undecidable in general, we have proven decidability for DRAT (and a fortiori AUT). Furthermore, we have shown that the whole computation power of Turing machines is not needed, two-way transducers are sufficient (and necessary). As TMs reading heads are read-only left-to-right, converting a 2DFT into a TM requires that the TM stores longer and longer prefixes of the input in the working-tape for later access. This is the only use the TM needs to make of the working tape. This is a naive translation, and sometimes the working tape can be flushed (some prefixes of the input may possibly not be needed anymore). More generally, it is an interesting research direction to fine-tune the class of functions targeted by synthesis with respect to some constraints on the memory, including quantitative constraints.

Related to the latter research direction is the following open question: is the synthesis problem of functions computable by input-deterministic one-way (aka. sequential) transducers from deterministic rational relations decidable? It is already open for automatic relations. We have shown that if a rational relation with closed domain is uniformizable by a computable function, then also by a sequential function. However, closedness is not a sufficient condition: e.g., the function which maps any $a^n x c^\omega$ to $x c^\omega$ for $x \in \{\#, \$\}$, is sequential; a sequential transducer just has to erase the a^n part, but its domain is not closed. This problem is interesting because sequential transducers only require bounded memory to compute a function (in contrast to two-way transducers that require access to unboundedly large prefixes of the input).

References

1 Shaull Almagor and Orna Kupferman. Good-enough synthesis. In *International Conference on Computer Aided Verification*, pages 541–563. Springer, 2020.

2 Rajeev Alur, Rastislav Bodík, Eric Dallal, Dana Fisman, Pranav Garg, Garvit Juniwal, Hadas Kress-Gazit, P. Madhusudan, Milo M. K. Martin, Mukund Raghothaman, Shambwaditya Saha, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Dependable Software Systems Engineering*, pages 1–25. IEEE, 2015.

3 Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular transformations of infinite strings. In *2012 27th Annual IEEE Symposium on Logic in Computer Science*, pages 65–74. IEEE, 2012.

4 Roderick Bloem, Rüdiger Ehlers, and Robert Könighofer. Cooperative reactive synthesis. In Bernd Finkbeiner, Geguang Pu, and Lijun Zhang, editors, *Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings*, volume 9364 of *Lecture Notes in Computer Science*, pages 394–410. Springer, 2015.

5 Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-admissible synthesis. *Acta Informatica*, 54(1):41–83, 2017. doi:10.1007/s00236-016-0273-2.

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

7 Arnaud Carayol and Christof Löding. Uniformization in Automata Theory. In *Proceedings of the 14th Congress of Logic, Methodology and Philosophy of Science Nancy, July 19-26, 2011*, pages 153–178. London: College Publications, 2014.

8 Krishnendu Chatterjee and Thomas A. Henzinger. Assume-guarantee synthesis. In Orna Grumberg and Michael Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings*, volume 4424 of *Lecture Notes in Computer Science*, pages 261–275. Springer, 2007.

9 Christian Choffrut and Serge Grigorieff. Uniformization of rational relations. In *Jewels are Forever*, pages 59–71. Springer, 1999.

10 Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of model checking*, volume 10. Springer, 2018.

11 Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The complexity of rational synthesis. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 121:1–121:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

12 Vrunda Dave, Emmanuel Filiot, Shankara Narayanan Krishna, and Nathan Lhote. Synthesis of computable regular functions of infinite words. In *CONCUR*, volume 171 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

13 Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On equivalence and uniformisation problems for finite transducers. In *ICALP*, volume 55 of *LIPIcs*, pages 125:1–125:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

14 Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *Int. J. Softw. Tools Technol. Transf.*, 15(5-6):519–539, 2013. `doi:10.1007/s10009-012-0228-z`.

15 Carsten Gerstacker, Felix Klein, and Bernd Finkbeiner. Bounded synthesis of reactive programs. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 441–457. Springer, 2018. `doi:10.1007/978-3-030-01090-4_26`.

16 Erich Gradel and Wolfgang Thomas. *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer Science & Business Media, 2002.

17 Michael Holtmann, Łukasz Kaiser, and Wolfgang Thomas. Degrees of lookahead in regular infinite games. In *International Conference on Foundations of Software Science and Computational Structures*, pages 252–266. Springer, 2010.

18 John E Hopcroft and Jeffrey D Ullman. An approach to a unified theory of automata. *The Bell System Technical Journal*, 46(8):1793–1829, 1967.

19 F Hosch and Lawrence Landweber. Finite delay solutions for sequential conditions. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1972.

20 Felix Klein and Martin Zimmermann. How much lookahead is needed to win infinite games? *Log. Methods Comput. Sci.*, 12(3), 2016. `doi:10.2168/LMCS-12(3:4)2016`.

21 K. Kobayashi. Classification of formal languages by functional binary transductions. *Information and Control*, 15(1):95–109, July 1969.

22 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016.

23 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *ACM Symposium on Principles of Programming Languages (POPL)*. ACM, 1989.

24 Jacques Sakarovitch. *Elements of automata theory.* Cambridge University Press, 2009.

25 Klaus Weihrauch. *Computable analysis: an introduction.* Springer Science & Business Media, 2012.