Continuous Rational Functions Are Deterministic Regular

Olivier Carton
IRIF, Université Paris Cité, France

Gaëtan Douéneau-Tabot
IRIF, Université Paris Cité, France
Direction générale de l'armement – Ingénierie de projets, Paris, France

Abstract

A word-to-word function is rational if it can be realized by a non-deterministic one-way transducer. Over finite words, it is a classical result that any rational function is regular, i.e. it can be computed by a deterministic two-way transducer, or equivalently, by a deterministic streaming string transducer (a one-way automaton which manipulates string registers).

This result no longer holds for infinite words, since a non-deterministic one-way transducer can guess, and check along its run, properties such as infinitely many occurrences of some pattern, which is impossible for a deterministic machine. In this paper, we identify the class of rational functions over infinite words which are also computable by a deterministic two-way transducer. It coincides with the class of rational functions which are continuous, and this property can thus be decided. This solves an open question raised in a previous paper of Dave et al.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases infinite words, rational functions, determinization, continuity, streaming string transducers, two-way transducers

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.28

Related Version *Full Version*: https://arxiv.org/abs/2204.11235

Funding This work was supported by the DeLTA ANR project (ANR-16-CE40-0007).

1 Introduction

Transducers are finite-state machines obtained by adding outputs to finite automata. They are very useful in a lot of areas like coding, computer arithmetic, language processing or program analysis, and more generally in data stream processing. In this paper, we study transducers which compute partial functions. They are either deterministic, or non-deterministic but unambiguous (they have at most one accepting run on a given input).

Over finite words, a deterministic two-way transducer (2-dT) consists of a deterministic two-way automaton which can produce outputs. Such machines realize the class of *regular functions*, which is often considered as one of the functional counterparts of regular languages. It coincides with the class of functions definable by monadic second-order transductions [7], or copyless deterministic streaming string transducers (dSST), which is a model of one-way automata manipulating string registers [1]. On the other hand, the model of non-deterministic one-way transducers (1-nT) describe the well-known class of *rational functions*. It is well known that any rational function is regular, but the converse does not hold.

Infinite words. The class of *regular functions over infinite words* was defined in [2] using monadic second-order transductions. It coincides with the class of functions realized by 2-dT with ω-regular lookahead, or by copyless dSST with some Müller conditions. However, the use of ω-regular lookaheads (or Müller conditions for dSST) is necessary to capture the

expressive power of monadic second-order logic on infinite words, in order to check properties such as infinitely many occurrences of some pattern. Similarly, the model of 1-nT with Büchi acceptance conditions defines the subclass of *rational functions over infinite words*.

Even if regular and rational functions give very natural frameworks for specification (due to their connections with logic), not all these functions can effectively be computed by a deterministic machine without lookaheads. It turns out that the regular functions which can be computed by a deterministic Turing machine (doing an infinite computation on its infinite input) are exactly those which are continuous for the Cantor topology [5]. Furthermore continuity can be decided, which was has been known for rational functions since [10].

The authors of [5] conjecture that any continuous rational (or even regular) function can in fact be computed by a 2-dT (without lookahead), instead of a Turing machine. A partial answer was obtained in [8], whose results imply that 2-dT can be built for a subclass of rational functions defined by 1-nT where some forms of non-determinism are prohibited. Their proof is based on game-theoretic techniques.

Contributions. This paper shows that any continuous rational function over infinite words can be extended to a function which is computable by a 2-dT (without lookaheads). Since the converse also holds, this result completely characterizes rational functions which can be computed by 2-dTs, up to an extension of the domain. Furthermore, this property is decidable and our construction of a 2-dT is effective.

This result is tight, in the sense that two-way moves cannot be avoided. Indeed, one-way deterministic transducers (describing the class of *sequential functions*) cannot realize all continuous rational functions, even when only considering total functions (contrary to what happens for the subclass of rational functions studied in [8]).

In order to establish this theorem, we first study the expressive power of 2-dT over infinite words. We introduce the class of *deterministic regular functions* as the class of functions computed by 2-dT (as opposed to the regular functions, which are not entirely deterministic since they use lookaheads to guess the future). Following the aforementioned equivalences between two-way and register transducers, we prove that deterministic rational functions are exactly the functions which are realized by copyless dSST (without Müller conditions). Hence our problem is reduced to showing that any continuous rational function can be realized by a copyless dSST. Building a copyless dSST is also relevant for practical applications, since it corresponds to a streaming algorithm over infinite strings.

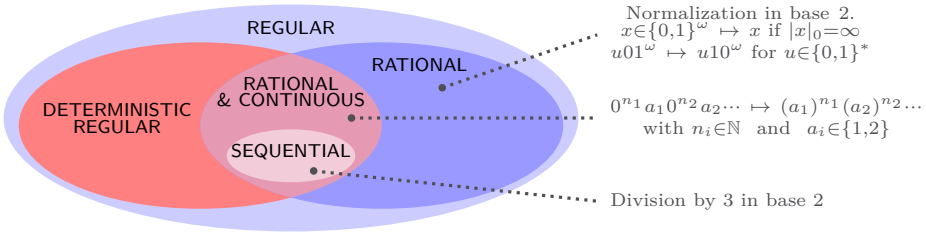


Figure 1 Classes of partial functions over infinite words studied in this paper.

Then we introduce various new concepts in order to transform a 1-nT computing a continuous function into a dSST. This determinization procedure is rather involved. The main difficulty is that even if the 1-nT is unambiguous, it might not check its guesses after reading only a finite number of letters. In other words, a given input can label several infinite runs, even if only one of them is accepting. However, a deterministic machine can never determine which run is the accepting one, since it requires to check whether a property occurs infinitely often. This intuition motivates our key definition of *compatible sets* among

the states of a 1-nT. Such sets are the sets of states which have a "common infinite future". The restriction of 1-nT considered in [8] leads to compatible sets which are always singletons (hence their condition defines a natural special case). We show that when the function computed by the 1-nT is continuous, the outputs produced along finite runs which end in a compatible set enjoy several combinatorial properties.

We finally describe how to build a dSST which realizes the continuous function given by a 1-nT. Its construction is is rather complex, and it crucially relies on the aforementioned properties of compatible sets. These sets are manipulated by the dSST in an original tree-like fashion. To the knowledge of the authors, this construction of this dSST is completely new (in particular, it is not based on the constructions of [5] nor of [8]).

Outline. We recall in Section 2 the definitions of rational functions and one-way transducers. In Section 3, we present the new class of deterministic regular functions and give the various transducer models which capture it. Our main result which relates continuous rational and deterministic regular functions is given in Section 4. The proof is sketched in sections 4 and 5.

2 Rational functions

Letters A, B denote alphabets, i.e. finite sets of letters. The set A^* (resp. A^+, A^ω) denotes the set of finite words (resp. non-empty finite words, infinite words) over the alphabet A. If $u \in A^* \cup A^\omega$, we let $|u| \in \mathbb{N} \cup \{\infty\}$ be its length. For $a \in A$, $|u|_a$ denotes the number of a in u. For $1 \leqslant i \leqslant |u|$, $u[i] \in A$ is the i-th letter of u. If $1 \leqslant i \leqslant j \leqslant |u|$, $u[i{:}j]$ stands for $u[i]u[i{+}1]\cdots$ until j. We write $w[i{:}]$ for $u[i{:}|u|]$. If $j > |u|$ we let $u[i{:}j] \coloneqq u[i{:}|u|]$. If $j < i$ we let $u[i{:}j] \coloneqq \varepsilon$. We write $u \sqsubseteq v$ (resp. $u \sqsubset v$) when u is a (resp. strict) prefix of v. Given two words u, v, we let $u \wedge v$ be their longest common prefix. We say that u, v are *mutual prefixes* if $u \sqsubseteq v$ or $v \sqsubseteq u$. In this case we let $u \vee v$ be the longest of them. A function f between two sets S, T is denoted by $f : S \to T$. If f is a *partial function* (i.e. possibly with non-total domain), it is denoted $f : S \rightharpoonup T$. Its domain is denoted $\mathsf{Dom}(f)$.

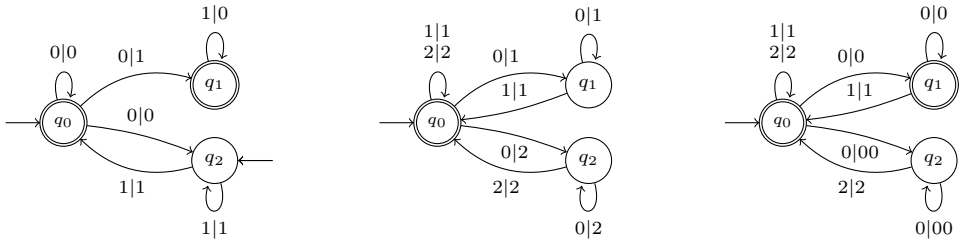
▶ **Definition 2.1.** *A* one-way non-deterministic transducer (1-nT) $\mathcal{T} = (A, B, Q, I, F, \Delta, \lambda)$ *is:*

- *a finite input (respectively output) alphabet A (respectively B);*
- *a finite set of states Q with $I \subseteq Q$ initial and $F \subseteq Q$ final;*
- *a transition relation $\Delta \subseteq Q \times A \times Q$;*
- *an output function $\lambda : \Delta \to B^*$ (defined for each transition).*

We write $q \xrightarrow{a|\alpha} q'$ whenever $(q, a, q') \in \Delta$ and $\lambda(q, a, q') = \alpha$. A *run* labelled by some $x \in A^* \cup A^\omega$ is a sequence of consecutive transitions $\rho \coloneqq q_0 \xrightarrow{x[1]|\alpha_1} q_1 \xrightarrow{x[2]|\alpha_2} q_2 \cdots$. The *output* of ρ is the word $\alpha_1\alpha_2\cdots \in A^* \cup A^\omega$. If $x \in A^\omega$, we also write $q_0 \xrightarrow{x|\alpha_1\alpha_2\cdots} \infty$ to denote an infinite run starting in q_0. The run ρ is *initial* if $q_0 \in I$, *final* if $x \in A^\omega$ and $q_i \in F$ infinitely often (Büchi condition), and *accepting* if both initial and final. $\mathcal{T}$ computes the *relation* $\{(x, y) : y \in B^\omega$ is output along an accepting run on $x\}$. It is *functional* if this relation is a (partial) function. In this case, $\mathcal{T}$ can be transformed in an equivalent *unambiguous* 1-nT (a transducer which has at most one accepting run on each $x \in A^\omega$) [3, Corollary 3]. A function $f : A^\omega \rightharpoonup B^\omega$ is said to be *rational* if it can be computed by a (unambiguous) 1-nT.

▶ **Example 2.2.** In Figure 2, we describe 1-nTs which compute the following functions:

- $\mathsf{normalize} : \{0,1\}^\omega \rightharpoonup \{0,1\}^\omega$ mapping $x \mapsto x$ if $|x|_0 = \infty$ and $u01^\omega \mapsto u10^\omega$ if $u \in \{0,1\}^*$;
- $\mathsf{replace} : \{0,1,2\}^\omega \rightharpoonup \{1,2\}^\omega$ with $\mathsf{Dom}(\mathsf{replace}) = \{x : |x|_1 = \infty$ or $|x|_2 = \infty\}$ and mapping $0^{n_1}a_1 0^{n_2}a_2 \cdots \mapsto a_1{}^{n_1+1}a_2{}^{n_2+1}\cdots$ if $a_i \in \{1,2\}$, $n_i \in \mathbb{N}$;
- $\mathsf{double} : \{0,1,2\}^\omega \to \{0,1,2\}^\omega$ mapping $0^{n_1}a_1 0^{n_2}a_2 \cdots \mapsto 0^{a_1 n_1}a_1 0^{a_2 n_2}a_2 \cdots$ and $0^{n_1}a_1 \cdots 0^{n_m}a_m 0^\omega \mapsto 0^{a_1 n_1}a_1 \cdots 0^{a_m n_m}a_m 0^\omega$ (if finitely many 1 or 2).



(a) 1-nT computing normalize. **(b)** 1-nT computing replace. **(c)** 1-nT computing double.

Figure 2 Unambiguous, clean and trim 1-nTs computing the functions of Example 2.2.

▶ Remark 2.3. The functions mentioned in Example 2.2 are not *sequential*, i.e. they cannot be computed by deterministic one-way transducers (i.e. deterministic 1-nTs).

A 1-nT is *trim* if any state is both accessible and co-accessible, or equivalently if it occurs in some accepting run. It is *clean* if the production along any accepting run is infinite.

▶ **Lemma 2.4.** *A trim* 1-nT *is clean if and only if for all $q \in F$, the existence of a cycle $q \xrightarrow{u|\alpha} q$ for $u \in A^+$ implies $\alpha \neq \varepsilon$. Given an unambiguous* 1-nT*, one can build an equivalent unambiguous, clean and trim* 1-nT*.*

3 Deterministic regular functions

We now introduce the new class of *deterministic regular* functions, which are computed by *deterministic two-way transducers*. Contrary to 1-nTs, such machines cannot test ω-regular properties of their input. Hence they describe continuous (and computable) functions.

▶ **Definition 3.1.** *A* deterministic two-way transducer (2-dT) $\mathcal{T} = (A, B, Q, q_0, \delta, \lambda)$ *is:*

- *an input alphabet A and an output alphabet B;*
- *a finite set of states Q with an initial state $q_0 \in Q$;*
- *a transition function $\delta : Q \times (A \uplus \{\vdash\}) \rightharpoonup Q \times \{\triangleleft, \triangleright\}$;*
- *an output function $\lambda : Q \times (A \uplus \{\vdash\}) \rightharpoonup B^*$ with same domain as δ.*

If the input is $x \in A^\omega$, then $\mathcal{T}$ is given as input the word $\vdash x$. The symbol $\vdash$ is used to mark the beginning of the input. We denote by $x[0] := \vdash$. A *configuration* over $\vdash x$ is a tuple (q, i) where $q \in Q$ is the current state and $i \geqslant 0$ is the current position of the reading head. The *transition relation* $\to$ is defined as follows. Given a configuration (q, i), let $(q', \star) := \delta(q, w[i])$. Then $(q, i) \to (q', i')$ whenever either $\star = \triangleleft$ and $i' = i - 1$ (move left), or $\star = \triangleright$ and $i' = i + 1$ (move right). A *run* is a (finite or infinite) sequence of configurations $(q_1, i_1) \to (q_2, i_2) \to \cdots$. An *accepting* run is an infinite run which starts in $(q_0, 0)$ and such that $i_n \to \infty$ when $n \to \infty$ (otherwise the transducer repeats the same loop).

The partial function $f : A^\omega \rightharpoonup B^\omega$ computed by $\mathcal{T}$ is defined as follows. Let $x \in A^\omega$ be such that there exists a (unique) accepting run $(q_0^x, i_0^x) \to (q_1^x, i_1^x) \to \cdots$ labelled by x. Let $y := \prod_{j=1}^{\infty} \lambda(q_j^x, w[i_j^x]) \in B^* \cup B^\omega$ be the concatenation of the outputs produced along this run. If $y \in B^\omega$, we define $f(x) := y$. Otherwise $f(x)$ is undefined.

▶ **Example 3.2.** The function replace from Example 2.2 can be computed by 2-dT. For each $i \geqslant 1$, this 2-dT crosses the block 0^{n_i} to determines a_i, and then crosses the block once more and outputs ${a_i}^{n_i+1}$. The function double can be computed using similar ideas. However, an important difference is that the 2-dT must output the block 0^{n_i} when it crosses it for the first time, in order to ensure that the production over 0^ω is 0^ω.

There exists deterministic regular functions which are not rational, for instance the function which reverses (mirror image) a prefix of its input.

Over finite words, it is known that two-way transducers are equivalent to copyless streaming string transducers [1]. Over infinite words, a similar equivalence holds between two-way transducers with lookahead and copyless streaming string transducers with Müller output conditions [2]. These models define the class of *regular functions* over infinite words. However, lookaheads enable two-way transducers to check ω-regular properties of their input (and thus non-computable behaviors). Hence our deterministic regular functions form a strict subclass of these regular functions over infinite words.

We now introduce a model of streaming string transducer to describe deterministic regular functions, in the spirit of the aforementioned results. In our setting, it consists of a one-way deterministic automaton with a finite set $\mathfrak{R}$ of registers that store words from B^*. We use a distinguished register $\mathfrak{out}$ to store the output produced when reading an infinite word. The registers are modified using *substitutions*, i.e. mappings $\mathfrak{R} \to (B \uplus \mathfrak{R})^*$. We denote by $\mathcal{S}_{\mathfrak{R}}^B$ the set of these substitutions. They can be extended morphically from $(B \uplus \mathfrak{R})^*$ to $(B \uplus \mathfrak{R})^*$ by preserving the elements of B. They can be composed (see Example 3.3).

▶ **Example 3.3.** Let $\mathfrak{R} = \{\mathfrak{r}, \mathfrak{s}\}$ and $B = \{b\}$. Consider $\sigma_1 := \mathfrak{r} \mapsto b, \mathfrak{s} \mapsto b\mathfrak{r}\mathfrak{s}b$ and $s_2 := \mathfrak{r} \mapsto \mathfrak{r}b, \mathfrak{s} \mapsto \mathfrak{r}\mathfrak{s}$, then $\sigma_1 \circ \sigma_2(\mathfrak{r}) = s_1(\mathfrak{r}b) = bb$ and $\sigma_1 \circ \sigma_2(\mathfrak{s}) = \sigma_1(\mathfrak{r}\mathfrak{s}) = bb\mathfrak{r}\mathfrak{s}b$.

▶ **Definition 3.4.** *A* deterministic streaming string transducer (dSST) *is:*

- *a finite input (resp. output) alphabet A (resp. B);*
- *a finite set of states Q with $q_0 \in Q$ initial;*
- *a transition function $\delta : Q \times A \rightharpoonup Q$;*
- *a finite set of registers $\mathfrak{R}$ with a distinguished output register $\mathfrak{out} \in \mathfrak{R}$;*
- *an update function $\lambda : Q \times A \rightharpoonup \mathcal{S}_{\mathfrak{R}}^B$ such that for all $(q, a) \in \mathsf{Dom}(\lambda) = \mathsf{Dom}(\delta)$:*
  - *$\lambda(q, a)(\mathfrak{out}) = \mathfrak{out} \cdots$;*
  - *there is no other occurence of $\mathfrak{out}$ in $\{\lambda(q, a)(\mathfrak{r}) : \mathfrak{r} \in \mathfrak{R}\}$.*

We denote it $\mathcal{T} = (A, B, Q, q_0, \delta, \mathfrak{R}, \mathfrak{out}, \lambda)$.

This machine defines a function $f : A^* \rightharpoonup B^*$ as follows. For $i \geqslant 0$ let $q_i^x := \delta(q_0, x[1{:}i])$ (when defined). For $i \geqslant 1$, we let $\lambda_i^x := \lambda(q_{i-1}^x, x[i])$ (when defined) and $\lambda_0^x(\mathfrak{r}) = \varepsilon$ for all $\mathfrak{r} \in \mathfrak{R}$. For $i \geqslant 0$, define the substitution $[\![\cdot]\!]_i^x := \lambda_0^x \circ \cdots \circ \lambda_i^x$. By construction we get $[\![\mathfrak{out}]\!]_i^x \sqsubseteq [\![\mathfrak{out}]\!]_{i+1}^x$ (when defined). If $[\![\mathfrak{out}]\!]_i^x$ is defined for all $i \geqslant 0$ and $|[\![\mathfrak{out}]\!]_i^x| \to +\infty$, we let $f(x) := \bigvee_i [\![\mathfrak{out}]\!]_i^x$ (it denotes the unique infinite word y such that $[\![\mathfrak{out}]\!]_i^x \sqsubseteq y$ for all $i \geqslant 0$). Otherwise $f(x)$ is undefined.

We say that a substitution $\sigma \in \mathcal{S}_{\mathfrak{R}}^B$ is *copyless* (resp. K-bounded) if for all $\mathfrak{r} \in \mathfrak{R}$, $\mathfrak{r}$ occurs at most once in $\{\sigma(\mathfrak{s}) : \mathfrak{s} \in \mathfrak{R}\}$ (resp. for all $\mathfrak{r}, \mathfrak{s} \in \mathfrak{R}$, $\mathfrak{r}$ occurs at most K times in $\sigma(\mathfrak{s})$).

▶ **Definition 3.5** (Copy restrictions). *We say that a* dSST *$\mathcal{T} = (A, B, Q, q_0, \delta, \mathfrak{R}, \mathfrak{out}, \lambda)$ is* copyless *(resp. K-*bounded*) if for all $x \in A^\omega$ and $i \leqslant j$ such that $\lambda_i^x \circ \cdots \circ \lambda_j^x$ is defined, this substitution is copyless (resp. K-bounded).*

▶ **Example 3.6.** The function replace from Example 2.2 can be computed by a copyless dSST. For all $i \geqslant 1$, it crosses the block 0^{n_i} and computes 1^{n_i} and 2^{n_i} in two registers. Once it sees a_i it adds in $\mathfrak{out}$ the register storing ${a_i}^{n_i}$. The function double can be computed using similar ideas. However, an important difference is that the dSST must directly output the block 0^{n_i} while crossing it, in order to ensure that the production over 0^ω is 0^ω.

The proof of the next result is quite involved, but it is largely inspired by the techniques used for regular functions over finite or infinite words (see e.g. [4, 6]).

▶ **Theorem 3.7.** *The following machines compute the same class of functions $A^\omega \multimap B^\omega$:*
1. *deterministic two-way transducers* (2-dT)*;*
2. *K-bounded deterministic streaming string transducers* (K*-bounded* dSST)*;*
3. *copyless deterministic streaming string transducers* (*copyless* dSST)*.*

Furthermore, all the conversions are effective.

▶ Remark 3.8. Even if this result is a variant of existing results over finite or infinite words, it requires a proof on its own. Indeed, the authors are not aware of a direct proof which would enable to deduce it from the existing similar results.

Let us now describe the domains of deterministic regular functions. We say that a language is *Büchi deterministic* if it is accepted by a deterministic Büchi automaton [9].

▶ **Proposition 3.9.** *If f is deterministic regular, then* Dom(f) *is Büchi deterministic.*

We finally give a closure property of deterministic regular functions under pre-composition.

▶ **Definition 3.10.** *A* restricted 1-nT *is a* 1-nT *whose states all are final.*

The semantics of a restricted 1-nT $\mathcal{N} = (A, B, Q, I, \Delta, \lambda)$ is defined so that it *always* computes a function $f : A^\omega \multimap B^\omega$. The domain Dom$(f)$ is the set of $x \in A^\omega$ such that $\mathcal{N}$ has a unique accepting run labelled by x, and such that the output along this unique run is infinite. In this case, we let $f(x)$ be the output of along this run. Intuitively, such a transducer expresses the ability to make non-deterministic guesses, as long as these guesses can be verified after reading a finite number of letters (i.e. there are no two possible infinite runs).

▶ **Theorem 3.11.** *Given a restricted* 1-nT *computing a function $f : A^\omega \multimap B^\omega$ and a deterministic regular function $g : B^\omega \multimap C^\omega$, $g \circ f$ is (effectively) deterministic regular.*

4 Continuous rational functions are deterministic regular

We now state the main result of this paper, which shows that a rational function can be extended to a deterministic regular function. Using an extension of the original function is necessary since not all ω-regular languages are Büchi deterministic (see Proposition 3.9). Note that Theorem 4.2 is in fact an equivalence, in the sense that a rational function which can be extended to a deterministic regular function is obviously continuous.

We recall that a function $f : A^\omega \multimap B^\omega$ is *continuous* if and only if for all $x \in$ Dom(f) and $n \geqslant 0$, there exists $p \geqslant 0$ such that $\forall y \in$ Dom(f), $|x \wedge y| \geqslant p \Rightarrow |f(x) \wedge f(y)| \geqslant n$.

▶ **Example 4.1.** The functions replace and double are continuous, but normalize is not.

▶ **Theorem 4.2.** *Given a continuous rational function $f : A^\omega \multimap B^\omega$, one can build a deterministic regular function f' which extends f (i.e. for all $x \in$ Dom(f), $f(x) = f'(x)$).*

To prove Theorem 4.2, it is enough by theorems 3.7 and 3.11 to show that f' can be computed as a composition of a restricted 1-nT and a K-bounded dSST (see Subsection 4.2, the construction will in fact give a 1-bounded transducer).

4.1 Properties of continuous rational functions

We first describe some structural properties of 1-nT computing continuous functions. In this subsection, we let $\mathcal{T} = (A, B, Q, I, F, \Delta, \lambda)$ be an unambiguous, clean and trim 1-nT computing a continuous function $f : A^\omega \multimap B^\omega$. It is well known that $\mathcal{T}$ verifies Lemma 4.3. This property is in fact equivalent to the continuity of f (see e.g. [10] or [5]).

▶ **Lemma 4.3.** *For all $q_1, q_2 \in I$, $q_1' \in F$, $q_2' \in Q$, $u \in A^*$, $u' \in A^+$, $\alpha_1, \alpha_1', \alpha_2, \alpha_2' \in B^*$ such that $q_i \xrightarrow{u|\alpha_i} q_i' \xrightarrow{u'|\alpha_i'} q_i'$ for $i \in \{1,2\}$ we have (note that $\alpha_1' \neq \varepsilon$ since $\mathcal{T}$ is clean):*

- *if $\alpha_2' \neq \varepsilon$, then $\alpha_1 {\alpha_1'}^\omega = \alpha_2 {\alpha_2'}^\omega$;*
- *if $\alpha_2' = \varepsilon$, $x \in A^\omega$, $\beta \in B^\omega$ and $q_2' \xrightarrow{x|\beta} \infty$ is final, then $\alpha_1 {\alpha_1'}^\omega = \alpha_2 \beta$.*

Empty cycles $q \xrightarrow{u|\varepsilon} q$ for $q \notin F$ cannot be avoided in a 1-nT. However, we shall see in Lemma 4.4 that such cycles can be avoided if the function is continuous. Formally, we say that the clean $\mathcal{T}$ is *productive* if the hypotheses of Lemma 4.3 imply $\alpha_2' \neq \varepsilon$.

▶ **Lemma 4.4.** *Given $\mathcal{T}$, one can build an equivalent unambiguous, trim and productive* 1-nT.

Compatible sets and steps. We now introduce the key notion of a *compatible set* which is a set of states having a "common future" and such that one of the future runs is accepting.

▶ **Definition 4.5** (Compatible set). *We say that a set of states $C \subseteq Q$ is* compatible *whenever there exists $x \in A^\omega$ and infinite runs ρ_q for each $q \in C$ labelled by x such that:*

- *$\forall q \in C$, ρ_q starts from q;*
- *$\exists q \in C$ such that ρ_q is final.*

Let Comp be the set of compatible sets. If $S \subseteq Q$, let $\mathsf{Comp}(S)$ be the set $2^S \cap \mathsf{Comp}$.

▶ **Definition 4.6** (Pre-step). *We say that C, u, D is a* pre-step *if $C, D \in \mathsf{Comp}$, $u \in A^*$ and for all $q \in D$, there exists a unique state $\mathsf{pre}^u_{C,D}(q) \in C$ such that $\mathsf{pre}^u_{C,D}(q) \xrightarrow{u} q$.*

Note that for all $D' \in \mathsf{Comp}(D)$, we have $\mathsf{pre}^u_{C,D}(D') \in \mathsf{Comp}$.

▶ **Definition 4.7** (Step). *We say that a pre-step C, u, D is a* step *if $\mathsf{pre}^u_{C,D}$ is surjective.*

Given $q \in D$, let $\mathsf{prod}^u_{C,D}(q)$ be the output $\alpha \in B^*$ produced along the run $\mathsf{pre}^u_{C,D}(q) \xrightarrow{u|\alpha} q$. We say that a (pre-)step is *initial* whenever $C \subseteq I$. We first claim that the productions along the runs of an initial step are mutual prefixes. Lemma 4.3 is crucial here.

▶ **Lemma 4.8.** *Let J, u, C be an initial step. Then $\mathsf{prod}^u_{J,C}(q)$ for $q \in C$ are mutual prefixes.*

▶ **Example 4.9.** In Figure 2b, if a step is initial, it is of the form $\{q_0\}, u, \{q_i\}$ for some $i \in \{0,1,2\}$. In Figure 2c, $\{q_0\}, 0^n, \{q_1, q_2\}$ is an initial step for all $n \geqslant 0$.

▶ **Definition 4.10** (Common, advance). *Let J, u, C be an initial step. We define:*

- *the* common $\mathsf{com}^u_{J,C} \in B^*$ *as the longest common prefix $\bigwedge_{q \in C} \mathsf{prod}^u_{J,C}(q)$;*
- *for all $q \in C$, its* advance $\mathsf{adv}^u_{J,C}(q) \in B^*$ *as $(\mathsf{com}^u_{J,C})^{-1} \mathsf{prod}^u_{J,C}(q)$;*
- *the* maximal advance $\mathsf{max\text{-}adv}^u_{J,C}$ *as the longest advance, i.e. $\bigvee_{q \in C} \mathsf{adv}^u_{J,C}(q)$.*

Definition 4.10 makes sense by Lemma 4.8, and furthermore $\mathsf{prod}^u_{J,C}(q) = \mathsf{com}^u_{J,C} \mathsf{adv}^u_{J,C}(q)$ for all $q \in C$. Now let $M \coloneqq \max_{q,q' \in Q, a \in A} |\lambda(q,a,q')|$ and $\Omega \coloneqq M|Q|^{|Q|}$. We say that a compatible set C is *separable* if there exists an initial step which ends in C, and such that the lengths of the productions along two of its runs differ of at least Ω.

▶ **Definition 4.11** (Separable set). *Let $C \in \mathsf{Comp}$, we say that C is* separable *if there exists an initial step J, u, C and $p, q \in C$ such that $\big||\mathsf{adv}^u_{J,C}(p)| - |\mathsf{adv}^u_{J,C}(q)|\big| > \Omega$.*

▶ Remark 4.12. In other words, it means that $|\mathsf{max\text{-}adv}^u_{J,C}| > \Omega$.

It is easy to see (by a pumping argument) that one can decide if a set is separable. We now show that the productions along the initial steps which end in a separable set are forced to "iterate" some value θ if the step is pursued. The following lemma is the key ingredient for showing that a rational function is deterministic regular (see Section 4).

▶ **Lemma 4.13** (Looping futures). *Let $C \in \mathsf{Comp}$ be separable and J, u, C be an initial step (not necessarily the one which makes C separable). There exists $\tau, \theta \in B^*$ with $|\tau| \leqslant 3\Omega$, and $|\theta| = \Omega!$ which can be uniquely determined from C and $\mathsf{adv}^u_{J,C}(q)$ for $q \in C$, such that:*

- $\tau \sqsubseteq \mathsf{max\text{-}adv}^u_{J,C} \sqsubseteq \tau\theta^\omega$;
- *for all step C, v, D and $q \in D$, $\mathsf{prod}^v_{C,D}(q) \sqsubseteq (\mathsf{adv}^u_{J,C}(p))^{-1}\tau\theta^\omega$ with $p := \mathsf{pre}^v_{C,D}(q)$.*

▶ Remark 4.14. Since $\mathsf{adv}^u_{J,C}(p) \sqsubseteq \mathsf{max\text{-}adv}^u_{J,C} \sqsubseteq \tau\theta^\omega$, the second item makes sense.

▶ **Example 4.15.** In Figure 2c, the compatible set $C := \{q_1, q_2\}$ is separable. For all step C, v, D we have $D = C$ thus $v = 0^n$, $\mathsf{prod}^{0^n}_{C,D}(q_1) = 0^n$ and $\mathsf{prod}^{0^n}_{C,D}(q_2) = 0^{2n}$.

4.2 Composition of a restricted 1-nT and a 1-bounded dSST

In the rest of this paper, we let $\mathcal{T} = (A, B, Q, I, F, \Delta, \lambda)$ be an unambiguous, productive and trim 1-nT computing a continuous $f : A^\omega \rightharpoonup B^\omega$. Our goal is to rewrite f as the composition of a restricted 1-nT and a 1-bounded dSST. We first build the restricted 1-nT, which computes an over-approximation of the accepting run of $\mathcal{T}$ in terms of compatible sets.

▶ **Lemma 4.16.** *One can build a restricted 1-nT $\mathcal{N}$ computing $g : A^\omega \rightharpoonup (\mathsf{Comp} \uplus A)^\omega$ such that $\mathsf{Dom}(f) \subseteq \mathsf{Dom}(g)$, and for all $x \in \mathsf{Dom}(g)$, $g(x) = C_0 x[1] C_1 x[2] C_2 \cdots$ where:*

- *$C_0 \subseteq I$ and for all $i \geqslant 0$, $C_i, x[i{+}1], C_{i+1}$ is a pre-step;*
- *if $x \in \mathsf{Dom}(f)$ then $\forall i \geqslant 0$, $q^x_i \in C_i$, where $q^x_0 \xrightarrow{x[1]} q^x_1 \xrightarrow{x[2]} \cdots$ is the accepting run of $\mathcal{T}$.*

Given $x \in \mathsf{Dom}(f)$, we denote by $C^x_0, C^x_1, \ldots$ the sequence of compatible sets produced by $\mathcal{N}$ in Lemma 4.16. We now describe a 1-bounded dSST $\mathcal{S}$ which, when given as input $g(x) \in (\mathsf{Comp} \uplus A)^\omega$ for $x \in \mathsf{Dom}(f)$, outputs $f(x)$ (this description is continued in Section 5).

Tree of compatibles. Given $C \in \mathsf{Comp}$, we define $\mathsf{tree}(C)$ as a finite set of words over $\mathsf{Comp}(C)$, which describes the decreasing chains for $\subset$. It can be identified with the set of all root-to-node paths of a tree labelled by elements of $\mathsf{Comp}(C)$, as shown in Example 4.18.

▶ **Definition 4.17** (Tree of compatibles). *Given $C \in \mathsf{Comp}$, we denote by $\mathsf{tree}(C)$ the set of words $\pi = C_1 \cdots C_n \in (\mathsf{Comp}(C))^+$ such that $C_1 = C$ and for all $1 \leqslant i \leqslant n{-}1$, $C_i \supset C_{i+1}$.*

▶ **Example 4.18.** If $C = \{1, 2, 3\}$ and $\mathsf{Comp}(C) = \{\{1, 2, 3\}, \{1, 2\}, \{2, 3\}, \{1\}, \{2\}, \{3\}\}$, then we have $\mathsf{tree}(C) = \{\{1, 2, 3\}\{1, 2\}\{1\}, \{1, 2, 3\}\{1, 2\}\{2\}, \{1, 2, 3\}\{2, 3\}\{2\}, \{1, 2, 3\}\{2, 3\}\{3\}, \{1, 2, 3\}\{1\}, \{1, 2, 3\}\{2\}, \{1, 2, 3\}\{3\}\}$. Its view as a tree is depicted in Figure 3.

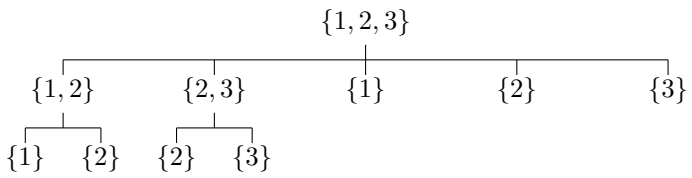


Figure 3 The tree of compatibles obtained from Example 4.18.

Information stored. The states of the dSST $\mathcal{S}$ are partitioned in two categories: the sets of the *separable mode* and the sets of of the *non-separable mode*. A configuration of the dSST $\mathcal{S}$ will always keep track of the following information:

- the content of a register $\mathfrak{out}$;
- two sets $\mathsf{J}, \mathsf{C} \in \mathsf{Comp}$ and a function $\mathsf{pre} : \mathsf{C} \to \mathsf{J}$ (stored in the state);
- a function $\mathsf{lag} : \mathsf{C} \to B^*$ such that $|\mathsf{lag}(q)| \leqslant 3\Omega$ for all $q \in \mathsf{C}$ (stored in the state).

Furthermore, when $\mathcal{S}$ is in a state of the separable mode, it will additionally store:

- a value $\theta \in B^*$ with $|\theta| = \Omega!$ (stored in the state);
- for all $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$ (note that $C_1 = \mathsf{C}$ by definition of $\mathsf{tree}(\mathsf{C})$):
  - a function $\mathsf{nb}_\pi : C_n \to [0{:}4]$ (stored in the state);
  - the content of a register $\mathfrak{out}_\pi$. For $\pi = \mathsf{C}$, we identify the register $\mathfrak{out}_\mathsf{C}$ with $\mathfrak{out}$;
- a function $\mathsf{last} : \mathsf{C} \to B^*$ such that $|\mathsf{last}(q)| < \Omega!$ forall $q \in \mathsf{C}$ (stored in the state).

If a configuration of $\mathcal{S}$ is clearly fixed, we abuse notations and denote by $\mathfrak{out}_\pi$ (resp. nb_π, lag, etc.) the value contained in register $\mathfrak{out}_\pi$ (resp. stored in the state) in this configuration. In a given configuration of $\mathcal{S}$, we say that $\pi \in \mathsf{tree}(\mathsf{C})$ is *close* if for all $\pi \sqsubset \pi' \in \mathsf{tree}(\mathsf{C})$, we have $\mathsf{nb}_{\pi'} = 0$ and $\mathfrak{out}_{\pi'} = \varepsilon$ (intuitively, the subtree rooted in π stores empty informations).

Invariants. The main idea for building $\mathcal{S}$ it the following. If C_i^x is a non-separable set, then the productions along the initial runs which end in C_i^x are mutual prefixes (by Lemma 4.8) which only differ from a bounded information. Hence the common part com of these runs is stored to $\mathfrak{out}$, and the adv are stored in the lag. If C_i^x becomes separable, then these runs still produce mutual prefixes, but two of them can differ by a large information. However by Lemma 4.13, they iterate some value θ. Hence the only relevant information is the number of θ which were produced along these runs. Formally, our construction ensures that the following invariants hold when $\mathcal{S}$ has just read $C_0^x x[1] C_1^x \cdots x[i] C_i^x$ for $i \geqslant 0$:

1. $\mathsf{C} = C_i^x$;
2. $\mathsf{J}, x[1{:}i], \mathsf{C}$ is an initial step and $\mathsf{pre} = \mathsf{pre}_{\mathsf{J},\mathsf{C}}$
3. if C is not separable, then $\mathcal{S}$ is in non-separable mode and:
   a. $\mathfrak{out} = \mathsf{com}_{\mathsf{J},\mathsf{C}}^{x[1:i]}$;
   b. $\mathsf{lag}(q) = \mathsf{adv}_{\mathsf{J},\mathsf{C}}^{x[1:i]}(q)$ for all $q \in \mathsf{C}$.
4. if C is separable, then $\mathcal{S}$ is in separable mode and:
   a. the $\mathsf{lag}(q)$ for $q \in Q$ are mutual prefixes, and so $\mathsf{max\text{-}lag} \coloneqq \bigvee_{q \in C} \mathsf{lag}(q)$ is defined. Furthermore, there exists $q \in \mathsf{C}$ such that $\mathsf{lag}(q) = \varepsilon$. We say that some $q \in \mathsf{C}$ is *lagging* if and only if $\mathsf{lag}(q) \sqsubset \mathsf{max\text{-}lag}$ (strict prefix), otherwise it is *not lagging*;
   b. if $\pi \in \mathsf{tree}(\mathsf{C})$ is such that $\pi \neq \mathsf{C}$ (i.e. $\mathfrak{out}_\pi \neq \mathfrak{out}$), then $\mathfrak{out}_\pi \in \theta^*$;
   c. for all $q \in \mathsf{C}$, $\mathsf{last}(q) \sqsubseteq \theta^\omega$ (if furthermore $|\mathsf{last}(q)| < \Omega!$, then $\mathsf{last}(q) \sqsubset \theta$);
   d. if q is lagging, then $\mathsf{last}(q) = \varepsilon$ and for all $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$ such that $q \in C_n$, we have $\mathsf{nb}_\pi(q) = 0$ and, if $\pi \neq \mathsf{C}$, $\mathfrak{out}_\pi = \varepsilon$;
   e. for all $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$, for $1 \leqslant i \leqslant n$ define $\pi_i \coloneqq C_1 \cdots C_i$. If $C_n = \{q\}$, then:
      - $\mathsf{prod}_{\mathsf{J},\mathsf{C}}^{x[1:i]}(q) = \mathfrak{out}\ \mathsf{lag}(q)$ if q is lagging;
      - $\mathsf{prod}_{\mathsf{J},\mathsf{C}}^{x[1:i]}(q) = \mathfrak{out}\ \mathsf{max\text{-}lag}\ \theta^{\mathsf{nb}_{\pi_1}(q)} \left(\prod_{i=2}^{n} \mathfrak{out}_{\pi_i}\, \theta^{\mathsf{nb}_{\pi_i}(q)}\right) \mathsf{last}(q)$ if q is not lagging.
   f. for all future steps C, u, D and for all $q \in D$, $\mathsf{prod}_{\mathsf{J},D}^{x[1:i]u}(q) \sqsubseteq \mathfrak{out}\ \mathsf{max\text{-}lag}\ \theta^\omega$;
   g. for all $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$ not close, let $J_n \coloneqq \mathsf{pre}_{\mathsf{J},\mathsf{C}}^{x[1:i]}(C_n) \subseteq \mathsf{J}$. Then $J_n, x[1{:}i], C_n$ is an initial step, which can be decomposed as an initial step $J_n, x[1{:}j], E$ and a step $E, x[j{+}1{:}i], C_n$ such that $|\mathsf{max\text{-}adv}_{J_n,E}^{x[1:j]}| \geqslant 4\Omega!$.

5 Description of the 1-bounded dSST for Subsection 4.2

In this section, we finally describe how the dSST $\mathcal{S}$ can preserve the invariants of Subsection 4.2, while being 1-bounded and outputting $f(x)$ when $x \in \mathsf{Dom}(f)$.

Let us first deal with the initialization of $\mathcal{S}$. When reading the first letter C_0^x of $g(x)$, $\mathcal{S}$ stores $\mathsf{J} \leftarrow C_0^x$, $\mathsf{C} \leftarrow C_0^x$ and $\mathsf{lag}(q) \leftarrow \varepsilon$ for all $q \in C_0^x$. This is enough if C_0^x is not separable. Otherwise, we let θ be given by Lemma 4.13 (applied to the initial simulation $C_0^x, \varepsilon, C_0^x$), $\mathsf{nb}_\pi(q) \leftarrow 0$ and $\mathfrak{out}_\pi \leftarrow \varepsilon$ for all $\pi = C_1 \cdots C_n \in \mathsf{tree}(C_0^x)$ and all $q \in C_n$.

▷ Claim 5.1. Invariants 1 to 4 (with $i = 0$) hold after this operation.

Assume now that the invariants hold for some $x \in \mathsf{Dom}(f)$ and $i \geqslant 0$. We describe how $\mathcal{S}$ updates its information when it reads $x[i{+}1]C_{i+1}^x$. Let $a \coloneqq x[i{+}1]$.

5.1 If C_i^x was not separable

In this case $\mathcal{S}$ was in the non-separable mode. We update $\mathsf{pre} \leftarrow \mathsf{pre} \circ \mathsf{pre}^a_{C_i^x, C_{i+1}^x}$, $\mathsf{C} \leftarrow C_{i+1}^x$ and $\mathsf{J} \leftarrow \mathsf{pre}(\mathsf{C})$. Since C_i^x, a, C_{i+1}^x was a pre-step, then $\mathsf{J}, x[1{:}i{+}1], \mathsf{C}$ is an initial step. For all $q \in C_{i+1}^x$, let $\delta_q \coloneqq \mathsf{lag}(\mathsf{pre}^a_{C_i^x, C_{i+1}^x}(q))\, \mathsf{prod}^a_{C_i^x, C_{i+1}^x}(q)$. Now let $c \coloneqq \bigwedge_{q \in Q} \delta_q$, we update $\mathfrak{out} \leftarrow \mathfrak{out}\, c$ and define $\alpha_q \coloneqq c^{-1}\delta_q$ for all $q \in \mathsf{C}$. It is easy to see that:

▷ Claim 5.2. $\mathfrak{out} = \mathsf{com}^{x[1:i+1]}_{\mathsf{J},\mathsf{C}}$ and $\alpha_q = \mathsf{adv}^{x[1:i+1]}_{\mathsf{J},\mathsf{C}}(q)$ for all $q \in \mathsf{C}$.

Finally we discuss two cases depending on the separability of $\mathsf{C} = C_{i+1}^x$:

- if C is not separable, then $\mathcal{S}$ stays in non-separable mode and it updates $\mathsf{lag}(q) \leftarrow \alpha_q$ for all $q \in \mathsf{C}$ (note that $|\alpha_q| \leqslant \Omega \leqslant 3\Omega$). We easily see that invariants 1, 2 and 3 hold.
- if C is separable, $\mathcal{S}$ goes to separable mode. By applying Lemma 4.13 to $\mathsf{J}, x[1{:}i{+}1], \mathsf{C}$ we get $\tau \in B^*$ with $k \coloneqq |\tau| \leqslant 3\Omega$. We update $\mathsf{lag}(q) \leftarrow \alpha_q[1{:}k]$ and $\mathsf{last}(q) \leftarrow \alpha_q[k{+}1{:}]$ for all $q \in \mathsf{C}$. The θ is given by Lemma 4.13, and we let $\mathsf{nb}_\pi(q) \leftarrow 0$ and $\mathfrak{out}_\pi \leftarrow \varepsilon$ for all $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$ (except for $\mathfrak{out}_\pi = \mathfrak{out}$ when $\pi = \mathsf{C} = C_{i+1}^x$) and all $q \in C_n$.
  ▶ **Lemma 5.3.** *Invariants 1, 2 and 4 hold in $i{+}1$ after this operation. Furthermore $|\theta| = \Omega!$, $|\mathsf{lag}(q)| \leqslant 3\Omega$ for all $q \in \mathsf{C}$, and for all $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$, $\mathsf{nb}_\pi = 0$.*
  Note that we may have $|\mathsf{last}(q)| \geqslant \Omega!$. In order to reduce their sizes, we apply the tool detailed in Subsection 5.2 (it will push the $\mathsf{last}(q)$ into the $\mathsf{nb}_\pi(q)$ and $\mathfrak{out}_\pi$).

5.2 Toolbox: reducing the size of $\mathsf{last}(q)$

In this subsection, we assume that $\mathcal{S}$ is in its separable mode and that invariants 2 and 4 hold in some $i \geqslant 0$. Furthermore, we suppose that $|\theta| = \Omega!$, $|\mathsf{lag}(q)| \leqslant 3\Omega$ for all $q \in \mathsf{C}$, and for all $C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$, $\mathsf{nb}_{C_1 \cdots C_n} : C_n \to [0{:}4]$. However last may be longer than it should.

From invariant 4c, there exists $n : \mathsf{C} \to \mathbb{N}$ such that $\mathsf{last}(q) = \theta^{n(q)}\delta_q$ with $\delta_q \sqsubset \theta$ for all $q \in \mathsf{C}$. We update $\mathsf{last}(q) \leftarrow \delta_q$ and $\mathsf{nb}_\mathsf{C}(q) \leftarrow \mathsf{nb}_\mathsf{C}(q) + n(q)$ for all $q \in \mathsf{C}$. Now, we have $|\mathsf{last}(q)| < \Omega!$ and $\mathsf{nb}_\pi(q) \leqslant 4$ when $\pi \neq \mathsf{C}$.

In order to reduce the value nb_C, we apply the function $\mathbf{down}(\mathsf{C})$ of Algorithm 1 which adds some θ in the $\mathfrak{out}_\pi$. Let us describe its base case informally. If $\mathsf{nb}_\mathsf{C}(q) > 0$ for all $q \in \mathsf{C}$, then no state is lagging by invariant 4d. Thus $\mathsf{lag}(q) = \mathsf{max\text{-}lag}$ for all $q \in \mathsf{C}$, and so $\mathsf{max\text{-}lag} = \varepsilon$ by invariant 4a. With the notations of invariant 4e (note that $\pi_1 = \mathsf{C}$), we get $\mathsf{prod}^{\mathsf{J},\mathsf{C}}_{x[1:i]}(q) = \mathfrak{out}\ \theta^{\mathsf{nb}_{\pi_1}(q)} \left(\prod_{i=2}^n \mathfrak{out}_{\pi_i}\, \theta^{\mathsf{nb}_{\pi_i}(q)}\right) \mathsf{last}(q)$ for all $q \in \mathsf{C}$. Thus we can produce in $\mathfrak{out}$ the value $\theta^m \coloneqq \bigwedge_{q \in \mathsf{C}} \theta^{\mathsf{nb}_\mathsf{C}(q)}$ (i.e. $m \leftarrow \min_{q \in \mathsf{C}} \mathsf{nb}_\mathsf{C}(q)$) and remove m to each $\mathsf{nb}_\mathsf{C}(q)$.

Algorithm 1 Sending down values in $\mathsf{tree}(\mathsf{C})$.

Function down(π)
  $C_1 \cdots C_n \leftarrow \pi$;
  /* 1. Add the common part of the buffers to the local output */
  $m \leftarrow \min_{q \in C_n} \mathsf{nb}_\pi(q)$;
  $\mathfrak{out}_\pi \leftarrow \mathfrak{out}_\pi\, \theta^m$;
  $\mathsf{nb}_\pi(q) \leftarrow \mathsf{nb}_\pi(q) - m$ for all $q \in C'$;
  /* 2. Check if some buffers $\mathsf{nb}_\pi(q)$ are still more than 4 */
  for $q \in C_n$ **do**
    if $\mathsf{nb}_\pi(q) > 4$ **then**
      for $C' \in \mathsf{Comp}(C_n)$ **such that** $C' \neq C_n$ **and** $q \in C'$ **do**
        $\mathsf{nb}_{\pi C'}(q) \leftarrow \mathsf{nb}_{\pi C'}(q) + (\mathsf{nb}_\pi(q) - 4)$;
      end
      $\mathsf{nb}_\pi(q) \leftarrow 4$;
    end
  end
  /* 3. Recursive calls */
  for $C' \in \mathsf{Comp}(C_n)$ **with** $C' \neq C_n$ **do**
    down$(\pi C')$;
  end

▶ **Lemma 5.4.** *Algorithm 1 is well defined. After the operation described in this subsection, invariants 2 and 4 hold, and furthermore we have* $|\theta| = \Omega!$, $|\mathsf{lag}(q)| \leqslant 3\Omega$ *and* $|\mathsf{last}(q)| < \Omega!$ *for all* $q \in \mathsf{C}$*, and for all* $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$, $\mathsf{nb}_\pi : C_n \to [0{:}4]$.

5.3 If C_i^x was separable

If C_i^x is separable, then $\mathcal{S}$ was in the separable mode by invariant 4. We first explain in Subsubsection 5.3.1 how to perform the update when C, a, C_{i+1}^x is a step (it corresponds to the "easy case" thanks to invariant 4f which deals with future steps). Then, we explain in Subsubsection 5.3.2 how the other case can be reduced to the first one, after a preprocessing which selects a subset $C' \subseteq \mathsf{C}$ such that C', a, C_{i+1}^x is a step.

5.3.1 Updating when C, a, C_{i+1}^x is a step

In the current subsubsection we assume that invariants 2 and 4 hold, that $\mathsf{C} \subseteq C_i^x$ is separable, and that C, a, C_{i+1}^x is a step. We show how to update the information stored by $\mathcal{S}$ in accordance with this step. Note that C_{i+1}^x is necessarily separable.

Since C will be modified, so will be $\mathsf{tree}(\mathsf{C})$, hence we begin with several register updates. For $\pi = D_1 \cdots D_n \in \mathsf{tree}(C_{i+1}^x)$, we define $C_i \coloneqq \mathsf{pre}^a_{\mathsf{C}, C_{i+1}^x}(D_i)$ for $1 \leqslant i \leqslant n$. Since we had a step then $C_1 = \mathsf{C}$, $C_i \in \mathsf{Comp}(\mathsf{C})$ and $C_1 \supseteq \cdots \supseteq C_n$. But we may not have $C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$ due to possible equalities. Let $1 = i_1 < \cdots < i_m \leqslant n$ be such that $C_{i_1} = \cdots = C_{i_2 - 1} \supset C_{i_2}$ and so on until $C_{i_m - 1} \supset C_{i_m} = \cdots = C_n$. Then $\rho \coloneqq C_{i_1} \cdots C_{i_m} \in \mathsf{tree}(\mathsf{C})$ and:

- if $i_m = n$, we let $\mathsf{nb}_\pi \leftarrow \mathsf{nb}_\rho \circ \mathsf{pre}^a_{\mathsf{C}, C_{i+1}^x}$ and $\mathfrak{out}_\pi \leftarrow \mathfrak{out}_\rho$;
- if $i_m < n$, we let $\mathsf{nb}_\pi \leftarrow 0$ and $\mathfrak{out}_\pi \leftarrow \varepsilon$.

For all $q \in C_{i+1}^x$, let $k_q \coloneqq |\mathsf{lag}(\mathsf{pre}^a_{\mathsf{C}, C_{i+1}^x}(q))^{-1}\, \mathsf{max\text{-}lag}|$ and:

- $\mathsf{lag}(q) \leftarrow \mathsf{lag}(\mathsf{pre}^a_{\mathsf{C}, C_{i+1}^x}(q))(\mathsf{prod}^a_{\mathsf{C}, C_{i+1}^x}(q)[1{:}k_q])$ (note that $\mathsf{max\text{-}lag}$ remains unchanged);
- $\mathsf{last}(q) \leftarrow \mathsf{last}(\mathsf{pre}^a_{\mathsf{C}, C_{i+1}^x}(q))(\mathsf{prod}^a_{\mathsf{C}, C_{i+1}^x}(q)[k_q{+}1{:}])$.

Now let $c \coloneqq \bigwedge_{q\in\mathsf{C}} \mathsf{lag}(q)$. We update $\mathsf{lag}(q) \leftarrow c^{-1}\,\mathsf{lag}(q)$ for all $q \in \mathsf{C}$ (therefore $\mathsf{max\text{-}lag}$ becomes $c^{-1}\,\mathsf{max\text{-}lag}$), $\mathfrak{out} \leftarrow \mathfrak{out}\ c$, $\mathsf{C} \leftarrow C^x_{i+1}$ and finally $\mathsf{pre} \leftarrow \mathsf{pre} \circ \mathsf{pre}^a_{\mathsf{C},C^x_{i+1}}$.

▶ **Lemma 5.5.** *After the operation described in this subsection, invariants 1, 2 and 4 hold, and* $|\theta| = \Omega!$, $|\,\mathsf{lag}(q)| \leqslant \Omega$ *for all* $q \in \mathsf{C}$, *and* $\mathsf{nb}_\pi : C_n \to [0{:}4]$ *for all* $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$.

However, we may have $|\,\mathsf{last}(q)| \geqslant \Omega!$. Thus we finally apply Subsection 5.2 once more.

5.3.2 Preprocessing when C, a, C^x_{i+1} is not a step

In the current subsubsection we assume that invariants 1, 2 and 4 hold in $i \geqslant 0$, that $\mathsf{C} = C^x_i$ is separable, and that C^x_i, a, C^x_{i+1} is *not* a step. Then let $C' \coloneqq \mathsf{pre}^a_{C^x_i,C^x_{i+1}}(C^x_{i+1}) \subset \mathsf{C}$ (an equality would give a step) and $\pi \coloneqq \mathsf{C}\,C' \in \mathsf{tree}(C)$. Two cases can occur.

If π is close. In this case, we have for all $\pi \sqsubset \pi' \in \mathsf{tree}(\mathsf{C})$ that $\mathsf{nb}_{\pi'} = 0$ and $\mathfrak{out}_{\pi'} = \varepsilon$. Therefore by invariant 4e we can describe the productions for all $q \in C'$ as follows:

- $\mathsf{prod}^{\mathsf{J},\mathsf{C}}_{x[1:i]}(q) = \mathfrak{out}\ \mathsf{lag}(q)$ if q is lagging;
- $\mathsf{prod}^{\mathsf{J},\mathsf{C}}_{x[1:i]}(q) = \mathfrak{out}\ \mathsf{max\text{-}lag}\ \mathfrak{out}_{\mathsf{C}\,C'}\ \theta^{\mathsf{nb}_\mathsf{C}(q)+\mathsf{nb}_{\mathsf{C}\,C'}(q)}\ \mathsf{last}(q)$ if q is not lagging.

Now two cases are possible, depending on whether there is a lagging state in C' or not:

- if there exists $q' \in C'$ which is lagging, then we must have $\mathfrak{out}_{\mathsf{C}\,C'} = \varepsilon$ by invariant 4d. For all $q \in C'$ let $\delta_q \coloneqq \mathsf{lag}(q)\theta^{\mathsf{nb}_\mathsf{C}(q)+\mathsf{nb}_{\mathsf{C}\,C'}(q)}\ \mathsf{last}(q)$ and let $c \coloneqq \bigwedge_{q\in C'} \delta_q$. Then we update $\mathfrak{out} \leftarrow \mathfrak{out}\ c$ and define $\alpha_q \coloneqq c^{-1}\delta_q$ for all $q \in C'$;
- if each $q \in C'$ is not lagging, we define $\delta_q \coloneqq \theta^{\mathsf{nb}_\mathsf{C}(q)+\mathsf{nb}_{\mathsf{C}\,C'}(q)}\,\mathsf{last}(q)$ and $c \coloneqq \bigwedge_{q\in C'} \delta_q$. Then we update $\mathfrak{out} \leftarrow \mathfrak{out}\ \mathsf{max\text{-}lag}\ \mathfrak{out}_{\mathsf{C}\,C'}\ c$ and define $\alpha_q \coloneqq c^{-1}\delta_q$ for all $q \in C'$;

We finally update $\mathsf{J} \leftarrow \mathsf{pre}(C')$, $\mathsf{C} \leftarrow C'$ and $\mathsf{pre} \leftarrow \mathsf{pre}|_{C'}$. It is easy to see that $\mathsf{J}, x[1{:}i], \mathsf{C}$ is a step and furthermore that we have computed com and adv.

▷ Claim 5.6. $\mathfrak{out} = \mathsf{com}^{x[1:i+1]}_{\mathsf{J},\mathsf{C}}$ and $\alpha_q = \mathsf{adv}^{x[1:i+1]}_{\mathsf{J},\mathsf{C}}(q)$ for all $q \in \mathsf{C}$.

This result exactly corresponds to Claim 5.2 from Subsection 5.1 (replace $i{+}1$ by i). Thus, to conclude, we just need to apply the operations described after Claim 5.2.

If π is not close. Let $c \coloneqq \bigwedge_{q\in C'} \mathsf{lag}(q)$, we update $\mathfrak{out} \leftarrow \mathfrak{out}\ c\ \mathfrak{out}_{\mathsf{C}\,C'}$ and for all $q \in C'$, $\mathsf{lag}(q) \leftarrow c^{-1}\,\mathsf{lag}(q)$ and $\mathsf{last}(q) \leftarrow \theta^{\mathsf{nb}_\mathsf{C}(q)}\,\mathsf{last}(q)$. Then, we update $\mathsf{nb}_{C'\pi} \leftarrow \mathsf{nb}_{\mathsf{C}\,C'\pi}$ and $\mathfrak{out}_{C'\pi} \leftarrow \mathfrak{out}_{\mathsf{C}\,C'\pi}$ for all $\pi \in (C')^{-1}\,\mathsf{tree}(C')$ (except for $\pi = \varepsilon$, in which case we have already updated $\mathfrak{out}_{C'} = \mathfrak{out}$ before). We finally update $\mathsf{J} \leftarrow \mathsf{pre}(C')$, $\mathsf{C} \leftarrow C'$ and $\mathsf{pre} \leftarrow \mathsf{pre}|_{C'}$.

▶ **Lemma 5.7.** *After the operation described in this subsection, invariants 2 and 4 hold, and* $|\theta| = \Omega!$, $|\,\mathsf{lag}(q)| \leqslant \Omega$ *for all* $q \in \mathsf{C}$, *and* $\mathsf{nb}_\pi : C_n \to [0{:}4]$ *for all* $\pi = C_1 \cdots C_n \in \mathsf{tree}(\mathsf{C})$. *Furthermore* C *is separable.*

▶ Remark 5.8. Contrary to the former cases, the main difficulty here is to show the preservation of invariant 4f. For this we essentially rely on invariant 4g and show that θ is still suitable.

Again, we may have $|\,\mathsf{last}(q)| \geqslant \Omega!$. Thus we finally apply Subsection 5.2 once more.

5.4 Boundedness and productivity of the construction

We first claim that $\mathcal{S}$ is a 1-bounded dSST, by construction.

▶ **Lemma 5.9.** *The* dSST $\mathcal{S}$ *is* 1*-bounded.*

It follows from invariants 1, 2 and 4e that for all $x \in \mathsf{Dom}(f)$, $\mathfrak{out}$ is always a prefix of $f(x)$ when $\mathcal{S}$ reads $g(x)$. To conclude the construction of $\mathcal{S}$, it remains to see that $\mathfrak{out}$ tends to an infinite word. The key ideas for showing Lemma 5.10 is to use the fact that $\mathcal{T}$ is productive, and that Algorithm 1 can only empty a buffer $\mathsf{nb}_{\mathsf{C}}(q)$ if it outputs a word.

▶ **Lemma 5.10.** *If $x \in \mathsf{Dom}(f)$, then $|\mathfrak{out}| \to \infty$ when $\mathcal{S}$ reads $g(x)$.*

6 Outlook

This paper provides a solution to an open problem. From a practical point of view, it allows to build a copyless streaming algorithm from a rational specification whenever it is possible (it is impossible when the rational function is not continuous). We conjecture that the techniques introduced in this paper can be extended to show that any continuous *regular* function is deterministic regular. Furthermore, they may also be used to study the rational or regular functions which are uniformly continuous for the Cantor topology, and capture them with a specific transducer model (another open problem of [5]).

References

1 Rajeev Alur and Pavol Cerný. Expressiveness of streaming string transducers. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl, 2010.

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

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

4 Luc Dartois, Ismaël Jecker, and Pierre-Alain Reynier. Aperiodic string transducers. *Int. J. Found. Comput. Sci.*, 29(5):801–824, 2018.

5 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)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

6 Gaëtan Douéneau-Tabot, Emmanuel Filiot, and Paul Gastin. Register transducers are marble transducers. In *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, 2020.

7 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic (TOCL)*, 2(2):216–254, 2001.

8 Emmanuel Filiot and Sarah Winter. Synthesizing computable functions from rational specifications over infinite words. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

9 Dominique Perrin and Jean-Éric Pin. *Infinite words: automata, semigroups, logic and games*. Academic Press, 2004.

10 Christophe Prieur. How to decide continuity of rational functions on infinite words. *Theoretical Computer Science*, 250(1-2):71–82, 2001.