Continuous rational functions are deterministic regular

Olivier Carton
Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

Gaëtan Douéneau-Tabot
Université Paris Cité, CNRS, IRIF, F-75013, Paris, France
Direction générale de l’armement - Ingénierie des 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 → Formal languages and automata theory

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

Digital Object Identifier 10.4230/LIPIcs...

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 [8], 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
Continuous rational functions are deterministic regular

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 [6]. Furthermore, continuity can be decided, which has been known for rational functions since [11].

The authors of [6] 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 [9], 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 [9]).

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.png)

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 [9] 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 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 [6] nor of [9]).

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. $A^+$, $A^\omega$) 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 $1 \leq i \leq |u|$, $u[i] \in A$ is the i-th letter of u. If $1 \leq i \leq j \leq |u|$, $u[i\ldots j]$ stands for $u[i]u[i+1]\ldots u[j]$. We write $u[i:]$ for $u[i\ldots |u|]$. If $j > |u|$ we let $u[i: j] := u[i\ldots |u|]$. If $j < i$ we let $u[i: j] := \varepsilon$. 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 $\text{Dom}(f)$.

Definition 2.1. A one-way non-deterministic transducer (1-nT) $T = (A, B, Q, I, F, \Delta, \lambda)$ 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 $q \xrightarrow{a, q'}$ whenever $(q, a, q') \in \Delta$ and $\lambda(q, a, q') = a$. A run labelled by some $x \in A^* \cup A^\omega$ is a sequence of consecutive transitions $\rho := q_0 \xrightarrow{x[1]\cdot a_1}, q_1 \xrightarrow{x[2]\cdot a_2}, q_2 \ldots$. The output of $\rho$ is the word $a_1a_2\ldots \in A^* \cup A^\omega$. If $x \in A^\omega$, we also write $q_0 \xrightarrow{x[1\ldots]} \infty$ to denote an infinite run starting in $q_0$. The run $\rho$ is initial if $q_0 \in I$, final if $x \in A^\omega$ and $q_t \in F$ infinitely often (Büchi condition), and accepting if both initial and final. $T$ computes the relation $\{(x, y) : y \in B^\omega \text{ is output along an accepting run on } x\}$. It is functional if this relation is a (partial) function. In this case, $T$ 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 $f : A^\omega \to B^\omega$ 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:
- normalize : $\{0,1\}^\omega \to \{0,1\}^\omega$ mapping $x \mapsto x$ if $|x|_0 = \infty$ and $u01^\omega \mapsto u10^\omega$ if $u \in \{0,1\}^*$;
Continuous rational functions are deterministic regular

- \( \text{replace} : \{0,1,2\}^\omega \to \{1,2\}^\omega \) with \( \text{Dom}(<\text{replace}>)=\{x : |x|_1=\infty \text{ or } |x|_2=\infty \} \) and mapping \( 0^n_1 a_1 0^n_2 a_2 \ldots \to a_1 n_1+1 a_2 n_2 +1 \ldots \) if \( a_i \in \{1,2\}, n_i \in \mathbb{N}; \)

- \( \text{double} : \{0,1,2\}^\omega \to \{0,1,2\}^\omega \) mapping \( 0^n_1 a_1 0^n_2 a_2 \ldots \to 0^{n_1} a_1 0^{n_2} a_2 \ldots \) and \( 0^n_1 a_1 \ldots 0^n_m a_m 0^\omega \to 0^{n_1} a_1 \ldots 0^{n_m} a_m 0^\omega \) (if finitely many 1 or 2).

![Figure 2](https://example.com/figure2.png)

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

- **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 \( q \xleftarrow{u[\alpha]} q \) 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 \( \omega \)-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 \( \delta : Q \times (A \cup \{\varepsilon\}) \to Q \times \{\varepsilon, \triangleright\} \);
  - an output function \( \lambda : Q \times (A \cup \{\varepsilon\}) \to B^* \) with same domain as \( \delta \).

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 \( i \geq 0 \) is the current position of the reading head. The transition relation \( \rightarrow \) is defined as follows. Given a configuration \((q,i)\), let \((q',\ast) := \delta(q,w[i])\). Then \((q,i) \rightarrow (q',i')\) whenever either \( \ast = \triangleleft \) and \( i' = i-1 \) (move left), or \( \ast = \triangleright \) and \( i' = i+1 \) (move right). A run is a (finite or infinite) sequence of configurations \((q_1,i_1) \rightarrow (q_2,i_2) \rightarrow \ldots\).

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 \( f : A^\omega \to B^\omega \) computed by \( \mathcal{T} \) is defined as follows. Let \( x \in A^\omega \) be such that there exists a (unique) accepting run \((q_0^x,i_0^x) \rightarrow (q_1^x,i_1^x) \rightarrow \ldots\) 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.
We denote it (when defined). For and thus non-computable behaviors. Hence our deterministic regular functions form a strict substitution is copyless (resp. which reverses (mirror image) a prefix of its input. There exists deterministic regular functions which are not rational, for instance the function Example 3.3.

Definition 3.5 (Copy restrictions). We say that a substitution \( s \) is copyless (resp. \( K \)-bounded) if for all \( x \in \mathbb{N}^d \) and \( i \leq j \) such that \( \lambda^i \circ \cdots \circ \lambda^j \) is defined, this substitution is copyless (resp. \( K \)-bounded).

Example 3.6. The function replace from Example 2.2 can be computed by 2-dT. For each \( i \geq 1 \), 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 \( 0^{\omega} \) when it crosses it for the first time, in order to ensure that the production over \( 0^{\omega} \) is \( 0^{\omega} \).

There exists deterministic regular functions which are not rational, for instance the function Example 3.3.

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 \( \delta : Q \times A \to Q \);
- a finite set of registers \( R \) with a distinguished output register \( \text{out} \in R \);
- an update function \( \lambda : Q \times A \to S^R_B \) such that for all \( (q,a) \in \text{Dom}(\lambda) = \text{Dom}(\delta) : \)
  - \( \lambda(q,a)(\text{out}) = \text{out} \cdots ; \)
  - there is no other occurrence of \( \text{out} \) in \( \{ \lambda(q,a)(r) : r \in R \} \).

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

This machine defines a function \( f : \mathbb{N}^d \to \mathbb{N} \) as follows. For \( i \geq 0 \) let \( q^i_0 := \delta(q_0, x[1:i]) \) (when defined). For \( i \geq 1 \), we let \( \lambda^i := \lambda(q^i_{i-1},x[i]) \) (when defined) and \( \lambda^0_0(x) = \varepsilon \) for all \( x \in R \). For \( i \geq 0 \), define the substitution \( [\lambda^i]_r := \lambda^0 \circ \cdots \circ \lambda^i \). By construction we get \( [\text{out}]_r \subseteq [\text{out}]_{i+1} \) (when defined). If \( [\text{out}]_r \) is defined for all \( i \geq 0 \) and \( [\text{out}]_r \to +\infty \), we let \( f(x) := V_i [\text{out}]_r \) (it denotes the unique infinite word \( y \) such that \( [\text{out}]_r \subseteq y \) for all \( i \geq 0 \)). Otherwise \( f(x) \) is undefined.

We say that a substitution \( \sigma \in S^R_B \) is copyless (resp. \( K \)-bounded) if for all \( r \in R \), \( r \) occurs at most once in \( \{ \sigma(s) : s \in R \} \) (resp. for all \( r \in R \), \( r \) occurs at most \( K \) times in \( \sigma(s) \)).
Continuous rational functions are deterministic regular

it sees $a_i$, it adds in out the register storing $a_i^n$. The function double can be computed using similar ideas. However, an important difference is that the dSST must directly output the block $0^m$ while crossing it, in order to ensure that the production over $0^\omega$ is $0^\omega$.

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. [5, 7]).

**Theorem 3.7.** The following machines compute the same class of functions $A^\omega \to B^\omega$:

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 [10].

**Proposition 3.9.** If $f$ is deterministic regular, then $\text{Dom}(f)$ 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 $N = (A, B, Q, I, \Delta, \lambda)$ is defined so that it always computes a function $f : A^\omega \to B^\omega$. The domain $\text{Dom}(f)$ is the set of $x \in A^\omega$ such that $N$ 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 $f : A^\omega \to B^\omega$ and a deterministic regular function $g : B^\omega \to C^\omega$, $g \circ f$ 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 $\omega$-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 $f : A^\omega \to B^\omega$ is continuous if and only if for all $x \in \text{Dom}(f)$ and $n \geq 0$, there exists $p \geq 0$ such that $\forall y \in \text{Dom}(f), |x \land y| \geq p \Rightarrow |f(x) \land f(y)| \geq n$.

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

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

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 $T = (A, B, Q, I, F, \Delta, \lambda)$ be an unambiguous, clean and trim 1-nT computing a continuous function $f : A^* \to B^*$. It is well known that $T$ verifies Lemma 4.3. This property is in fact equivalent to the continuity of $f$ (see e.g. [11] or [6]).

**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 $q'_i \xrightarrow{u_i \omega_i} q'_i \xrightarrow{u'_i \omega'_i} q'_i$ for $i \in \{1, 2\}$ we have (note that $\alpha_1' \neq \varepsilon$ since $T$ is clean):
- if $\alpha_2' \neq \varepsilon$, then $\alpha_1 \alpha_1' \omega = \alpha_2 \alpha_2' \omega$;
- if $\alpha_2' = \varepsilon$, $x \in A^\omega$, $\beta \in B^*$ and $q'_2 \xrightarrow{\varepsilon \omega_2} \infty$ is final, then $\alpha_1 \alpha_1' \omega = \alpha_2 \beta$.

Empty cycles $q \xrightarrow{\varepsilon \omega} q$ 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 $T$ is productive if the hypotheses of Lemma 4.3 imply $\alpha_2' \neq \varepsilon$.

**Lemma 4.4.** Given $T$, 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 $\rho_q$ for each $q \in C$ labelled by $x$ such that:
- $\forall q \in C$, $\rho_q$ starts from $q$;
- $\exists q \in C$ such that $\rho_q$ is final.

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

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

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

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

Given $q \in D$, let $\text{prod}^{u}_{C,D}(q)$ be the output $\alpha \in B^*$ produced along the run $\text{pre}^{u}_{C,D}(q) \xrightarrow{u} q$. 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 4.4 is crucial here.

**Lemma 4.8.** Let $J, u, C$ be an initial step. Then $\text{prod}^{u}_{J,C}(q)$ 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 $n \geq 0$.

**Definition 4.10 (Common, advance).** Let $J, u, C$ be an initial step. We define:
- the common $\text{com}^{u}_{J,C} \in B^*$ as the longest common prefix $\bigwedge_{q \in C} \text{prod}^{u}_{J,C}(q)$;
- for all $q \in C$, its advance $\text{adv}^{u}_{J,C}(q) \in B^*$ as $(\text{com}^{u}_{J,C})^{-1} \text{prod}^{u}_{J,C}(q)$;
- the maximal advance $\max\text{adv}^{u}_{J,C}$ as the longest advance, i.e. $\bigvee_{q \in C} \text{adv}^{u}_{J,C}(q)$.

Definition 4.10 makes sense by Lemma 4.8 and furthermore $\text{prod}^{u}_{J,C}(q) = \text{com}^{u}_{J,C} \text{adv}^{u}_{J,C}(q)$ for all $q \in C$. Now let $M := \max(10, \max_{q,q' \in Q, a \in A} |\lambda(q, a, q')|)$ and $\Omega := M|Q|^{|Q|}$. 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 $\Omega$.

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

O. Carton and G. Douéneau-Tabot XX:7
Remark 4.12. In other words, it means that $|\max\text{-adv}_{J,C}^\tau| > \Omega$.

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 $\theta$ 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 $C \in \text{Comp}$ be separable and $J, u, C$ be an initial step (not necessarily the one which makes $\text{Dom}(\pi)$ we have $\omega$ which computes an over-approximation of the accepting run of $\tau$.

The states of the $\text{Comp}$ are separable mode $\text{tree}$ the $\text{Definition 4.17}$ all root-to-node paths of a tree labelled by elements of $\text{comp}$.

Given $\text{Lemma 4.16}$. One can build a restricted $\text{1-nT}$ and a $\text{1-bounded dSST}$ computing a continuous function $f : A^\omega \rightarrow B^\omega$. Our goal is to rewrite $f$ as the composition of a restricted $\text{1-nT}$ and a $\text{1-bounded dSST}$. We first build the restricted $\text{1-nT}$, which computes an over-approximation of the accepting run of $\tau$ in terms of compatible sets.

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

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

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

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

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

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

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

- the content of a register $\text{out}$;
- two sets $J, C \in \text{Comp}$ and a function $\text{pre} : C \rightarrow J$ (stored in the state);
- a function $\text{lag} : C \rightarrow B^*$ such that $|\text{lag}(q)| \leq \Omega$ for all $q \in C$ (stored in the state);
- a value $\text{max-lag} \in B^*$ such that $|\text{max-lag}| \leq \Omega$ (stored in the state).
Following invariants hold when \( \text{Lemma 4.13} \), they iterate some value stored to
\( \text{out}_{\pi} \).
The main idea for building invariants.

In a given configuration of
\( \text{tree}(C) \) (note that
\( C_1 = C \) by definition of
\( \text{tree}(C) \)):
- a function
\( \text{nb}_{\pi} : C_n \to \{0:2\} \) (stored in the state);
- the content of a register
\( \text{out}_{\pi} \). For
\( \pi = C \), we identify the register
\( \text{out}_{\pi} \) with
\( \text{out} \);
- a function
\( \text{last} : C \to B^* \) such that
\( \| \text{last}(q) \| < \Omega! \) for all
\( q \in C \) (stored in the state).
If a configuration of
\( S \) is clearly fixed, we abuse notations and denote by
\( \text{out}_{\pi} \) (resp.
\( \text{nb}_{\pi} \),
\( \text{lag} \), etc.) the value contained in register
\( \text{out}_{\pi} \) (resp.
stored in the state) in this configuration.
In a given configuration of
\( S \), we say that
\( \pi \in \text{tree}(C) \) is close if for all
\( \pi \sqsubset \pi' \in \text{tree}(C) \), we have
\( \text{nb}_{\pi'} = 0 \) and
\( \text{out}_{\pi'} = \varepsilon \) (intuitively, the subtree rooted in
\( \pi \) stores empty informations).

**Invariants.**
The main idea for building
\( S \) it the following. If
\( C^*_i \) is a non-separable set, then the productions along the initial runs which end in
\( C^*_i \) are mutual prefixes (by
\( \text{Lemma 4.13} \)) which only differ from a bounded information. Hence the common part
\( \text{com} \) of these runs is stored to
\( \text{out} \), and the
\( \text{adv} \) are stored in the
\( \text{lag} \). If
\( C^*_i \) becomes separable, then these runs still produce mutual prefixes, but two of them can differ by a large information. However by
\( \text{Lemma 4.13} \) they iterate some value
\( \theta \). Hence the only relevant information is the number of
\( \theta \) which were produced along these runs. Formally, our construction ensures that the following invariants hold when
\( S \) has just read
\( C^*_0 x[1]C^*_1 \cdots x[i]C^*_i \) for
\( i \geq 0 

1. \quad C = C^*_i ;
2. \quad J.x[1:x], C \) is an initial step and
\( \text{pre} = \text{pre}_{j.C} 
3. \quad \text{if } C \text{ is not separable, then } S \text{ is in non-separable mode and:}
   \begin{enumerate}
   \item \quad \text{out} = \text{com}_{x[1:x]C} ;
   \item \quad \text{lag}(q) = \text{adv}_{x[1:x]C}(q) \) for all
\( q \in C.
   \end{enumerate}
4. \quad \text{if } C \text{ is separable, then } S \text{ is in separable mode and:}
   \begin{enumerate}
   \item \quad \text{for all } q \in Q \text{ lag}(q) \sqsubseteq \text{max-lag}. \text{ Furthermore, there exists } q \in C \text{ such that lag}(q) = \varepsilon .
\quad \text{We say that some } q \in C \text{ is lagging if and only if lag}(q) \sqsubseteq \text{max-lag} \) (strict prefix),
\quad otherwise we say that
\( q \) is not lagging;
   \item \quad \text{if } \pi \in \text{tree}(C) \text{ is such that } \pi \not\sqsubset C \) (i.e.
\( \text{out}_{\pi} \neq \text{out} \), then
\( \text{out}_{\pi} \in \theta^* 
   \item \quad \text{for all } q \in C, \text{last}(q) \sqsubseteq \theta^* \) (if furthermore
\( \| \text{last}(q) \| < \Omega! \), then
\( \text{last}(q) \sqsubset \theta 
   \item \quad \text{if } q \text{ is lagging, then } \text{last}(q) = \varepsilon \) and for all
\( \pi = C_1 \cdots C_n \in \text{tree}(C) \text{ such that } q \in C_n, \text{ we have } \text{nb}_{\pi}(q) = 0 \) and, if
\( \pi \not\sqsubset C, \text{out}_{\pi} = \varepsilon 
   \item \quad \text{for all } \pi = C_1 \cdots C_n \in \text{tree}(C), \text{ for } 1 \leq i \leq n \text{ define } \pi_i := C_1 \cdots C_i . \text{ If } C_n = \{q\}, \text{ then:}
   \quad = \prod_{j_i} x[1:j_i] \) (q) = \text{out lag}(q) \) if
\( q \) is lagging;
   \quad = \prod_{j_i} x[1:j_i] \) (q) = \text{out max-lag } \theta^{\text{nb}_{\pi}(q)} \) (prod_{j_i=2}^{n} \text{out}_{\pi_i} \theta^{\text{nb}_{\pi}(q)}) \) \text{last}(q) \) if
\( q \) is not lagging.
   \item \quad \text{for all future steps } C, u, D \text{ and for all } q \in D, \text{ prod}_{j_i=1}^{n} x[1:j_i] \) (q) \sqsubseteq \text{out max-lag } \theta^* ;
   \item \quad \text{for all } \pi = C_1 \cdots C_n \in \text{tree}(C) \text{ not close, let } J_n := \text{pre}_{x[1:j]C}(q) \subseteq J . \text{ Then } J_n, x[1:j], C_n \text{ is an initial step, which can be decomposed as an initial step } J_n, x[1:j], E \text{ and a step }
E, x[j+1:i], C_n \text{ such that } \| \text{max-adv} \) _{j_n, E} | \geq \Omega! .
   \end{enumerate}
5 Description of the 1-bounded dSST for Subsection 4.2

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

Let us first deal with the initialization of $S$. When reading the first letter $C^0_0$ of $q(x)$, $S$ stores $J \leftarrow C^0_0$, $C \leftarrow C^0_0$, and $\text{lag}(q) \leftarrow \varepsilon$ for all $q \in C^0_0$. There is no need to define $\text{max-lag}$ in this context. This is enough if $C^0_0$ is not separable. Otherwise, let $\theta$ (resp. $\text{max-lag}$) be given by the $\theta$ (resp. $\tau$) of Lemma 4.13 (applied to the initial simulation $C^0_0, \varepsilon, C^0_0$), $\text{nb}_\pi(q) \leftarrow 0$ and $\text{out}_\pi \leftarrow \varepsilon$ for all $\pi = C_1 \cdots C_n \in \text{tree}(C^0_0)$ and all $q \in C_n$. We also let $\text{last}(q) \leftarrow \varepsilon$ for all $q \in C^0_0$.

▷ Claim 5.1. Invariants 1 to 3 (with $i = 0$) hold after this operation.

Assume now that the invariants hold for some $x \in \text{Dom}(f)$ and $i \geq 0$. We describe how $S$ updates its information when it reads $x[i+1]|C^x_{i+1}$. Let $a := x[i+1]$.

5.1 If $C^x_{i+1}$ was not separable

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

▷ Claim 5.2. $\text{out} = \text{com}^f_{\text{pre}^x_{i+1}}$ and $\alpha_q = \text{adv}^f_{\text{pre}^x_{i+1}}(q)$ for all $Q \in C$.

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

- if $C$ is not separable, then $S$ stays in non-separable mode and it updates $\text{lag}(q) \leftarrow \alpha_q$ for all $Q \in C$ (note that $|\alpha_q| \leq \Omega \leq \Omega!$). We easily see that invariants 1 and 3 hold.

- if $C$ is separable, $S$ goes to separable mode. By applying Lemma 4.13 to $J, x[i+1], C$ we get $\tau \in B^*$ with $k := |\tau| \leq \Omega!$ and $\theta \in B^*$ with $|\theta| = \Omega!$. We update $\text{max-lag} \leftarrow \tau$, $\text{lag}(q) \leftarrow \alpha_q[1:k]$ and $\text{last}(q) \leftarrow \alpha_q[k+1:]$ for all $Q \in C$. We also let $\text{nb}_\pi(q) \leftarrow 0$ and $\text{out}_\pi \leftarrow \varepsilon$ for all $\pi = C_1 \cdots C_n \in \text{tree}(C)$ (except for $\text{out}_\pi \leftarrow \text{out}$ when $\pi = C = C^x_{i+1}$) and all $Q \in C_n$.

▷ Lemma 5.3. Invariants 1, 2 and 4 hold in $i+1$ after this operation. Furthermore $|\theta| = \Omega!$, $|\text{max-lag}| \leq \Omega!$ for all $Q \in C$, and for all $C_1 \cdots C_n \in \text{tree}(C)$, $\text{nb}_\pi = 0$.

Note that we may have $|\text{last}(q)| \geq \Omega!$. In order to reduce their sizes, we apply the tool detailed in Subsection 5.2. (it will push the last(q) into the nb_π(q) and out_π).

5.2 Toolbox: reducing the size of last(q)

In this subsection, we assume that $S$ is in its separable mode and that invariants 2 and 4 hold in some $i \geq 0$. Furthermore, we suppose that $|\theta| = \Omega!$, $|\text{max-lag}|, |\text{lag}(q)| \leq \Omega!$ for all $Q \in C$, and for all $C_1 \cdots C_n \in \text{tree}(C)$, $\text{nb}_\pi = 0$. However the last may be longer than they should. We are thus going to resize them.

From invariant 1, there exists $n : C \rightarrow \mathbb{N}$ such that $\text{last}(q) = \theta^n(q) \delta_q$ with $\delta_q \subseteq \theta$ for all $Q \in C$. We update $\text{last}(q) \leftarrow \delta_q$ and $\text{nb}_C(q) \leftarrow \text{nb}_C(q) + n(q)$ for all $Q \in C$. Now, we have $|\text{last}(q)| \leq \Omega!$ and $\text{nb}_\pi(q) \leq 2$ when $\pi \neq C$.

In order to reduce the value $\text{nb}_C$, we then apply the function $\text{down}(C)$ of Algorithm 1 which adds some $\theta$ in the out_π. Let us describe its base case informally. If $\text{nb}_C(q) > 0$ for all $Q \in C$, then no state is lagging by invariant 4d. Thus $\text{lag}(q) = \text{max-lag}$ for all $Q \in C$, and so
Function down(\(\pi\))

1. Add the common part of the buffers to the local output */
   \(m \leftarrow \min_{q \in C_n} nb_\pi(q)\);
   \(\text{out}_\pi \leftarrow \text{out}_\pi \theta^m\);
   \(\text{nb}_\pi(q) \leftarrow \text{nb}_\pi(q) - m \text{ for all } q \in C'\);

2. Check if some buffers \(\text{nb}_\pi(q)\) are still more than 2 */
   for \(q \in C_n\) do
     if \(\text{nb}_\pi(q) > 2\) then
       for \(C' \in \text{Comp}(C_n)\) such that \(C' \neq C_n\) and \(q \in C'\) do
         \(\text{nb}_{\pi C'}(q) \leftarrow \text{nb}_{\pi C'}(q) + (\text{nb}_\pi(q) - 2)\);
       end
     \end if
     \(\text{nb}_\pi(q) \leftarrow 2\);
   end
   /* 3. Recursive calls */
   for \(C' \in \text{Comp}(C_n)\) with \(C' \neq C_n\) do
     down(\(\pi\));
   end
end

max-lag = \(\varepsilon\) by invariant 4a. With the notations of invariant 4e (note that \(\pi_1 = C\)), we get
\(\prod_{x=1}^C \theta^{\text{nb}_x(q)}(\prod_{j \geq x} \text{out}_x(q)_{\theta^{\text{nb}_x(q)}})\) last(q) for all \(q \in C\). Thus we can produce in \(\text{out}\) the value \(\theta^m = \bigwedge_{q \in C} \theta^{\text{nb}_x(q)}\) (i.e. \(m \leftarrow \min_{q \in C} \text{nb}_C(q)\)) and remove \(m\) to each \(\text{nb}_C(q)\).

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

5.3 If \(C_t^x\) was separable

If \(C_t^x\) is separable, then \(S\) was in the separable mode by invariant 4. We first explain in Subsection 5.3.1 how to perform the update when \(C, a, C_{t+1}^x\) is a step (it corresponds to the "easy case" thanks to invariant 4 which deals with future steps). Then, we explain in Subsection 5.3.2 how the other case can be reduced to the first one, after a preprocessing which selects a subset \(C' \subseteq C\) such that \(C', a, C_{t+1}^x\) is a step.

5.3.1 Updating when \(C, a, C_{t+1}^x\) is a step

In the current subsection we assume that invariants 4 and 4a hold, that \(C \subseteq C_t^x\) is separable (we may not have \(C = C_t^x\) because of the preprocessing of Subsection 5.3.2), and that \(C, a, C_{t+1}^x\) is a step. We show how to update the information stored by \(S\) in accordance with this step. Note that \(C_{t+1}^x\) is necessarily separable.

Since \(C\) will be modified, so will be \(\text{tree}(C)\), hence we begin with several register updates.

For \(\pi = D_1 \cdots D_n \in \text{tree}(C_{t+1}^x)\), we define \(C_i := \text{pre}_D^{C_{t+1}^x}(D_i)\) for \(1 \leq i \leq n\). Since we had a step then \(C_1 = C, C_1 \in \text{Comp}(C)\) and \(C_1 \supseteq \cdots \supseteq C_n\). But we may not have \(C_1 \cdots C_n \in \text{tree}(C)\) due to possible equalities. Let \(1 = i_1 < \cdots < i_m \leq n\) be such that \(C_{i_1} = \cdots = C_{i_2} \supseteq C_{i_2}\) and so on until \(C_{i_m-1} \supseteq C_{i_m} = \cdots = C_n\). Then \(\rho := C_{i_1} \cdots C_{i_m} \in \text{tree}(C)\) and:
Continuous rational functions are deterministic regular

Now two cases are possible, depending on whether there is a lagging state in $\theta$.

For all $C$.

However, we may have $|\theta| \neq \Omega!$, in which case we have already updated $nb$.

If the new $\theta$ is close.

Then we update

If the new $\theta$ is not close.

Remark 5.8. CONTRARY TO THE FORMER CASES, THE MAIN DIFFICULTY HERE IS TO SHOW THE PRESERVATION OF INVARIANT 4f.

For this we critically rely on invariant 4g: the idea is to show that the invariants are separable, and that $\theta$ is still a step.

Lemma 5.5. After the operation described in this subsection, invariants $\mathbf{1}$, $\mathbf{2}$ and $\mathbf{3}$ hold, and $|\theta| = \Omega!$. |max-lag| is close.

Thus we finally apply Subsection 5.2 once more.

5.3.2 Preprocessing when $C, a, C_{i+1}$ is not a step

In the current subsubsection we assume that invariants $\mathbf{1}$, $\mathbf{2}$ and $\mathbf{3}$ hold in $i > 0$, that $C = C_i$ is separable, and that $C_{i+1}$ is not a step. Then let $C' := \text{pre}_{C_i, C_{i+1}}(C_{i+1}) \subset C$ (an equality would give a step) and $\pi := C C' \in \text{tree}(C)$.

We finally update

Claim 5.6. After the operation, $\text{out} = \text{com}_{C, C'}^{1+i+1}$ and $\alpha_q = \text{adv}_{C, C'}^{1+i+1}(q)$ for all $q \in C$.

This result exactly corresponds to Claim 5.2 from Subsection 5.1 (replace $i+1$ by $i$). Thus, we conclude, we just need to apply the operations described after Claim 5.2 (i.e. determining if the new $C$ is separable or not, and building the structure accordingly).

If $\pi$ is not close. Let $C := \bigwedge_{q \in C} \text{lag}(q)$, we update

Then we update

Lemma 5.7. After the operation described in this subsection, invariants $\mathbf{2}$ and $\mathbf{3}$ hold, and $|\theta| = \Omega!$. |max-lag| is close.

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 critically rely on invariant 4g: the idea is to show that $\theta$ is still a step.
suitable looping value, even if we have chosen a subset of our compatible set (observe that in Lemma 4.13, the value $\theta$ depends on the compatible set chosen).

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

### 5.4 Boundedness and productivity of the construction

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

▶ **Lemma 5.9.** The dSST $S$ is 1-bounded.

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

▶ **Lemma 5.10.** If $x \in \text{Dom}(f)$, then $|\text{out}| \to \infty$ when $S$ 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 [6]).

### 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.
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. Michal P. Chytil and Vojtěch Jákli. Serial composition of 2-way finite-state transducers and simple programs on strings. In *4th International Colloquium on Automata, Languages, and Programming, ICALP 1977*, pages 135–147. Springer, 1977.
5. Luc Dartois, Ismaël Jecker, and Pierre-Alain Reynier. Aperiodic string transducers. *Int. J. Found. Comput. Sci.*, 29(5):801–824, 2018.
6. 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.
7. Gaétan 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.
8. 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.
Continuous rational functions are deterministic regular

9 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.

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

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

12 John C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959.
We show in this section that the function \texttt{double} from Example 2.2 cannot be computed by a one-way deterministic transducer. It implies that one-wayness is not enough to compute continuous rational functions by deterministic transducers, even if the original function is total (contrary to what happens with the subclass of rational functions considered in [9]).

Now, assume that \texttt{double} is computed by a deterministic one way transducer. There exists \(N, M \geq 0, u_1, u_2, u_3, u_4, v_1 \in \{0, 1, 2\}^*\) such that for all \(n \geq 0\), \(\texttt{double}(0^{M+Nn}10^n) = u_1u_2u_3u_40^{M+Nn}10^n\) and \(\texttt{double}(0^{M+Nn}20^n) = u_1u_2u_3u_40^{2(M+Nn)}20^n\). A contradiction can easily be deduced.

**Proof of Theorem 3.7**

We show how to transform a 2-dT into a 1-bounded dsST. The main idea is to keep track of the right-to-right behavior of the 2-dT (the “crossing sequence”) on the prefix read so far. This proof is somehow standard for dsST, and its main ideas originate from [12] which first showed how to transform a two-way automaton into a one-way automaton.

Consider a 2-dT \(\mathcal{T} = (A, B, Q, q_0, \delta, \lambda)\) computing a partial function \(f : A^\omega \rightarrow B^\omega\). Let \(x \in A^\omega\). We denote by \(\rightarrow\) the transition relation of \(\mathcal{T}\) between configurations, and \(\rightarrow^+\) its transitive closure. Let \(Q_\perp := Q \cup \{\bot\}\). When reaching a position \(i \geq 0\) of the input \(\uparrow x\), the dsST will keep track the following information (see Figure 3):

- the state \(i^\perp \in Q_\perp\) that is “the state of \(\mathcal{T}\) the first time it goes to position \(i+1\)”. More formally, first\(i^\perp \in \Delta\) is the state such that \(\varphi^i_x := (q_0, 0) \rightarrow^+ (\text{first}^i_x, i+1)\) is the run which visits the position \(i+1\) for the first time (and \(\bot\) if such a run does not exist), which implies that \(x \not\in \text{Dom}(f)\). This information is coded in the state of the dsST;
- the concatenation of the outputs by \(\lambda\) along the run \(\varphi^i_x\), stored in the register \(\text{out}\);
- a function next\(i^\perp(q) : Q \rightarrow Q_\perp\), which gives for each \(q \in Q\) the state such that the run \(\rho^i_x(q) := (q, i) \rightarrow^+ (\text{next}^i_x(q), i+1)\) visits \(i+1\) for the first time when starting from \((q, i)\) (\(\bot\) if it does not exist). This information is coded in the state of the dsST;
- the concatenation of the outputs by \(\lambda\) along the run \(\rho^i_x(q)\) for all \(q \in Q\). This information is stored in a register \(\text{out}_q\) (it is empty if \(\text{next}^i_x(q) = \bot\)).
Continuous rational functions are deterministic regular

Updates when reading a letter. We have to show how the dSST can update this abstraction of the behavior of $T$. Assume that the invariants are computed correctly at position $i \geq 0$, we want to compute them for $i+1$. Let $a := x[i+1]$. We recursively define a function $\sigma_i^x : Q_\perp \to (Q_\perp)^* \cup (Q_\perp)^\omega$, which describes the sequence of states visited in position $i$ when starting from $q$ in position $i+1$, and before reaching position $i+2$:

1. if $q = \perp$ or $\delta(q, a)$ is undefined, then $\sigma_i^x(q) := \perp^\omega$;
2. otherwise if $\delta(q, a) = (q', \triangleright)$, then $\sigma_i^x(q) := \varepsilon$;
3. otherwise if $\delta(q, x[i+1]) = (q', \triangleleft)$, then $\sigma_i^x(q) := q' \, \sigma_i^x(\text{next}_i^x(q'))$.

Claim C.1. Let $q \in Q$. Then $p_1 \cdots p_n \in Q^*$ is a prefix of $\sigma_i^x(q)$ if and only if $(q, i+1) \to \rho_i^x(p_1) \to \cdots \to \rho_i^x(p_n)$ is a run.

Proof. Follows directly from the definitions of $\sigma_i^x(q)$ and $\rho_i^x(q)$.

Claim C.2. If next$_i^{x+1}(q) = \perp$, then $|\sigma_i^x(q)| = \infty$.

Claim C.3. If next$_i^{x+1}(q) = q' \neq \perp$, then $\sigma_i^x(q) = p_1 \cdots p_n$ with $n \leq |Q|$. Furthermore if $n = 0$ then $\delta(q, a) = (q', \triangleright)$, if $n > 0$ then $\delta(\text{next}_i^x(p_n), a) = (q', \triangleright)$, and $\rho_i^{x+1}(q) = (q, i+1) \to \rho_i^x(p_1) \to \cdots \to \rho_i^x(p_n) \to (q', i+2)$

Proof of claims C.2 and C.3. We have $\sigma_i^x(q) \in Q^* \cup Q^\omega \cup Q^\perp$ is a prefix of $\sigma_i^x(q)$ if and only if $\delta(\text{next}_i^x(p_n), a)$ undefined. In both cases this implies next$_i^{x+1}(q) = \perp$. On the other hand, if $p_1 \cdots p_n \in Q^*$ with $n > |Q|$ is a prefix of $\sigma_i^x(q)$, then the run of Claim C.1 above visits twice the same configuration. Thus it loops infinitely before seeing position $i+2$, which gives next$_i^{x+1}(q) = \perp$. Finally, the remaining case is when $\sigma_i^x(q) = p_1 \cdots p_n \in Q^*$ with $n \leq |Q|$. This means that $\delta(q, a) = (q', \triangleright)$ or $\delta(\text{next}_i^x(p_n), a) = (q', \triangleright)$ which, when added to the run of Claim C.1 gives the first visit of position $i+2$ in $\rho_i^{x+1}(q)$.

By computing the $|Q|$ first letters of $\sigma(q)$, the dSST can update the next$_i^{x+1}$ information. Furthermore, we define the register updates as follows:

- if next$_i^{x+1}(q) = \perp$, then out$^q_i \leftarrow \varepsilon$;
- otherwise $\sigma_i^x(q) = p_1 \cdots p_n$, and then out$^q_i \leftarrow \alpha_0 \, \text{out}_{p_1} \alpha_2 \cdots \text{out}_{p_n} \alpha_n$ where the $\alpha_j \in B^*$ are defined accordingly to $\rho_i^{x+1}(q)$ in Claim C.3

It remains to deal with the update of first$_i^{x+1}$ and out. By a similar argument, it is easy to see that first$_i^{x+1} \neq \perp$ if and only if $\sigma(\text{first}_i^x) = p_1 \cdots p_n \in Q^*$ with $n \leq |Q|$, and furthermore in that case $\varphi_i^{x+1} = \varphi_i^x \to \rho_i^{x+1}(\text{first}_i^x)$. We can thus update out$^q_i \leftarrow \text{out}_{\alpha_0} \, \text{out}_{p_1} \cdots \text{out}_{p_n} \alpha_n$ (where the $\alpha_j$ are the same as those of the update of out$^q_{\text{first}_i^x}$).
We first give a simple recursive algorithm to compute \( \lambda \) of \( \lambda_s \) for all \( i \geq 0 \) and furthermore output labels along the run \( \varphi^r \) tends to \( f(x) \in B^\omega \) when \( i \to +\infty \). Thus \( [\text{out}]^r \) tends to \( f(x) \).

Now assume that \( x \notin \text{Dom}(f) \). Then either first\(^r_x = - \) for some \( i \geq 0 \) (either because there is no infinite run of the 2-dT labelled by \( x \), or because this run is not accepting, i.e. it contains a loop), which will be detected by our dSST that will stop its computation. Or first\(^r_x \neq - \) for all \( i \geq 0 \), but the output labels of the \( \varphi^r \) do not tend to an infinite word, and therefore we get \( [\text{out}]^r \neq \infty \).

Correctness of the construction. Let \( x \in \text{Dom}(f) \), then by definition of the semantics of a 2-dT, we have first\(^r_x \neq - \) for all \( i \geq 0 \) and furthermore output labels along the run \( \varphi^r \) tends to \( f(x) \in B^\omega \) when \( i \to +\infty \). Thus \( [\text{out}]^r \) tends to \( f(x) \).

Claim C.4. Let \( x \in A^\omega \); 1 \( \leq j \leq i \) be such that \( \lambda^r_j \) is defined, then:

1. for all \( r \), \( s \in R \), \( r \) occurs at most once in \( \lambda^r_j \circ \ldots \circ \lambda^r_s (r) \);
2. if \( \text{out}_p \) occurs in \( \lambda^r_j \circ \ldots \circ \lambda^r_s (\text{out}_q) \), then next\(^r_q \neq - \) and \( (p, j) \) occurs in the run \( \rho^*_q \);
3. if \( \text{out}_p \) occurs in \( \lambda^r_j \circ \ldots \circ \lambda^r_s (\text{out}) \), then first\(^r_p \neq - \) and \( (p, j) \) occurs in the run \( \varphi^r \).

Proof. We show the three items by induction on \( i \geq 1 \). The base case for \( i = j \) follows from the definition of \( \rho \) and of the updates. Now consider the induction step from \( i \geq 1 \) to \( i+1 \). Item \( \square \) holds for \( r = \text{out} \) by definition of the updates \( \lambda \). Assume that \( \text{out}_p \) occurs in \( \lambda^r_j \circ \ldots \circ \lambda^r_s (\text{out}_q) \), we show simultaneously that items \( \square \) and \( \circ \) hold. First note that next\(^r_q \neq - \), since otherwise the update in \( x[i+1] \) is \( \text{out}_p \leftarrow e \). Let \( p_1 \ldots p_n = \sigma^r_{i+1} (q) \), then there exists \( a_0, \ldots, a_n \in R^* \) such that \( \lambda^r_j (\text{out}_q) = a_0 \text{out}_{p_1} \cdots \text{out}_{p_n} a_n \). Then by Claim C.4 the run \( \rho^*_{i+1} (q) \) is obtained by concatenating the \( \rho^*_{k} (p_k) \) for \( 1 \leq k \leq n \) in a disjoint way. Since \( j \leq i+1 \), then \( \text{out}_p \) occurs \( \lambda^r_j \cdots \lambda^r_s (\text{out}_{p_k}) \) for some \( 1 \leq k \leq n \). Then by induction hypothesis (item \( \square \), \( (p, j) \) occurs in \( \rho^*_{k+1} (q) \). But since there is no loop in \( \rho^*_{k+1} (q) \), then \( \text{out}_p \) only occurs in \( \lambda^r_j \cdots \lambda^r_s (\text{out}_{p_k}) \). By induction hypothesis (item \( \circ \), \( \text{out}_p \) occurs only once in this \( \lambda^r_j \cdots \lambda^r_s (\text{out}_{p_k}) \), and finally \( \text{out}_p \) occurs only once in \( \lambda^r_j \cdots \lambda^r_s (\text{out}) \). The proof of item \( \boxed{3} \) is similar.

C.2 From copyless dSSTs to 2-dTs

Let \( \mathcal{T} = (A, B, Q, q_0, \delta, R, \text{out}, \lambda) \) be a copyless dSST computing a function \( f : A^\omega \to B^\omega \). We first give a simple recursive algorithm to compute \( f \), and then show that this algorithm is correct and can be implemented by a 2-dT.

Algorithmic description. Given \( x \in \text{Dom}(f) \), let \( \rho := q_0^x \to q_1^x \to \cdots \) be the initial run of \( \mathcal{T} \) on \( x \) (with the convention that \( q_i^x = - \) if it is undefined). For \( i \geq 0 \), we also define \( \lambda^r_x := \lambda(q_{i-1}^x, x[i]) \) the substitution applied when reading \( x[i] \). We give in Algorithm 2 a function \text{substitute}(\alpha, i, \tau) \), producing \( [\alpha]_i^r \) when \( i \geq 0 \) and \( \alpha \in (B \cup R)^* \) as input \( \tau \) shall be used later, it is an additional information used by a 2-dT implementing the function. It makes recursive calls to compute the values of the registers which occur in \( \alpha \).
Algorithm 2  Computing recursively the values of a register

Function substitute(u, i, r)
/* α ∈ (B ⊎ R)* to be computed, i ≥ 0 current position */
for a in α[1],...,α[|α|] do
  if a ∈ B then
    Output a; /* Letter a ∈ B is output */
  else
    if i > 0 then
      substitute(λx_i(a), i−1, a); /* Recursion on λx_i(a) */
    end
  end
end

Function simulation
for i in {1,...,∞} do
  out α ← λx_i(out);
  substitute(α, i, out);
end

▷ Claim C.5 (Algorithm 2 is correct). Let x ∈ Aω and i ≥ 0 be such that q^x_i ≠ ⊥. Then for all α ∈ (R ⊎ B)* and r ∈ R, substitute(α, i, r) terminates and outputs [α]^x_i.

Proof. Immediate by induction on i ≥ 0.

▷ Remark C.6. We shall assume that Algorithm 2 blocks if q^x_i = ⊥ (since it happens if and only if λx_i is undefined).

We have also described in Algorithm 2 a function simulation which uses substitute. This function ranges over the positions 1, 2,... of the input, and for each of them it produces “the value added in out” at this position.

▷ Claim C.7. If x ∈ Dom(f), then simulation loops infinitely and outputs f(x).

▷ Claim C.8. If x ∉ Dom(f), then either simulation gets blocked at some point, or it produces a finite output.

Proof. Two cases are possible if x ∉ Dom(f). Either q^x_i = ⊥ for some i ≥ 0, and then simulation gets blocked before position i. Or q^x_i ≠ ⊥ for all i ≥ 0, but [out]^x_i tends to a finite word w ∈ B*. It is easy to see that simulation then produces w.

Implementation by a 2-dT. We now describe how to implement the function simulation by a 2-dT. First note that the machine needs to determine the substitution λ^x_i, hence the state q^x_i−1 (possibly ⊥) when in position i ≥ 1. For this, we add a lookbehind feature to our 2-dT, which enables it to choose its transition depending on a regular property of the prefix up to the current position. Over finite words, it is well known that given a 2-dT with lookbehind, one can build an equivalent 2-dT (see e.g. the “lookahead removal” techniques of [3]). We claim that the very same proof can also be applied to infinite words, since lookbehinds only concern a finite prefix of the input.

We now show how this 2-dT proceeds. The main loop of simulation on i ≥ 1 is executed by moving right on the input. At each position i ≥ 1 the 2-dT determines the value α ∈ (R ⊎ B)* such that λ^x_i(out) = out α (if it is defined) by using its lookbehind. Then it
processes each character \( a \in \alpha[1] \cdots \alpha[|\alpha|] \) (since \( \alpha \) is a bounded information, this loop is hard-coded in the states without moving). If \( a \in B \), it is output. Otherwise, the machine determines \( \beta := \lambda_i^r(a) \), moves left and executes \texttt{substitute}(\( \beta, i-1, a \)) by doing recursive calls. The case when \( i = 0 \) is detected by reading the initial letter \( + \).

However, a 2-\texttt{dT} (which has a bounded memory) cannot keep track of the “call stack” for \texttt{substitute}. Thus how can it determine the calling function when coming back from a recursive call? In fact, due to the copyless behavior, this information can be easily determined without a stack. Indeed, assume that the 2-\texttt{dT} has just finished executing \texttt{substitute}(\( \beta, i-1, a \)), then it moves right:

- if \( a = \texttt{out} \), then the call to \texttt{substitute} was done in \texttt{simulation} which had computed the value \( \lambda_i^r(\texttt{out}) = \texttt{out} \beta \). In this case, the 2-\texttt{dT} pursues the execution of \texttt{out} in \( i+1 \);
- otherwise there exists exactly one \( r \in R \) such that \( a \) occurs in \( \alpha := \lambda_i^r(r) \) (and this value can be determined). Then \texttt{substitute}(\( \beta, i-1, a \)) was called while executing \texttt{substitute}(\( \alpha, i, r \)) on position \( i \) and \( \beta = \lambda_i^r(a) \). Since \( a = \alpha[j] \) for a unique \( j \), then the index \( j \) can be determined and thus the 2-\texttt{dT} can pursue the computation.

Furthermore if \( x \in \text{Dom}(f) \), this machine visits its whole input.

### C.3 From \( K \)-bounded \( dSSTs \) to copycleess \( dSSTs \)

Let \( T = (A, B, Q, q_0, \delta, R, \texttt{out}, \lambda) \) be a \( K \)-bounded \( dSST \), we show how to transform it in an equivalent copyless \( dSST \). The proof techniques are adapted from those for \( dSST \) over finite words \([3, 4]\) or \( \omega \)-streaming string transducers \([2]\). However, these transformations usually add extra features to the \( dSST \), such as non-determinism or lookaheads, which we avoid here. Indeed, such features allow to check \( \omega \)-regular properties of the input, and it is precisely what we intend to get rid of in this paper.

**Generic proof ideas.** In order to transform \( T \) into a copyless \( dSST \), the natural idea is to keep \( K \) copies of each register. However, we cannot maintain \( K \) copies all the time: suppose that \( r \) is used to update both \( t_1 \) and \( t_2 \). If we have \( K \) copies of \( r \), we cannot produce, in a copyless way, \( K \) copies of \( t_1 \) and \( K \) copies of \( t_2 \).

This issue is solved as follows. Recall that \( q_i^r \) (resp. \( \lambda_i^r \)) is the state reached by \( T \) after reading \( x[1:i] \) (resp. the transition applied when reading \( x[i] \)) on input \( x \in A^\omega \). Let \( \mathcal{R} := R \setminus \{ \texttt{out} \} \) and \( r \in \mathcal{R} \). If \( i \geq 0 \) and \( x \in A^\omega \) are such that \( q_i^r \) is defined, we want to maintain \( \text{copies}_i^r(r) \) copies of the value \( \lambda_i^r(x^i) \), where \( \text{copies}_i^r(r) \leq K \) is the number of times that \( r \) will be used in \( \text{out} \) after position \( i \). Formally we define the following:

\textbf{Definition C.9.} Let \( x \in A^\omega \) and \( i \geq 0 \) such that \( q_i^r \) is defined. Given \( r \in \mathcal{R} \), we let:

\[
\text{copies}_i^r(r) := \max \{ |\lambda_i^r \circ \cdots \circ \lambda_j^r(\text{out})|_c : j \geq i \text{ and } \lambda_j^r \text{ is defined} \}.
\]

\textbf{Claim C.10.} \( \text{copies}_i^r(r) \leq K \) since \( T \) is \( K \)-bounded.

We now describe an inductive relation for the \( \text{copies}_i^r(r) \). Intuitively, Lemma \textbf{C.11} means that if \( \text{copies}_i^r(r) \) copies of \( r \) will be needed, then in the next transition these copies can be reparted between the registers. We postpone the proof of Lemma \textbf{C.11} to Subsubsection \textbf{C.3.3}.

\textbf{Lemma C.11.} Let \( x \in A^\omega \) and \( i \geq 0 \) such that \( q_i^x \) is defined. Then for all \( r \in \mathcal{R} \):

\[
\text{copies}_i^r(r) = |\lambda_i^r(\text{out})|_c + \sum_{s \in \mathcal{R}} |\lambda_{i+1}^r(s)|_c \times \text{copies}_{i+1}^r(s).
\]
Continuous rational functions are deterministic regular

However, this number \( c^\tau_i (\tau) \) cannot be determined after reading only \( x[1:i] \). It requires some information about \( x[i+1:] \) (that is typically why we would need a lookahead). Thus our copyless dSST will have to memorize a finite forest which describes the possible non-deterministic choices done to determine the values of \( \text{copies}^\tau_i (\tau) \). The copyless dSST will also keep track of the substitutions applied along the branches of this forest.

**Structure of the proof.** In order to make the construction of a copyless dSST more understandable, we first describe its behavior in a high-level algorithmic fashion in Subsubsection C.3.1. Then we justify in Subsubsection C.3.2 that this algorithm can be implemented by a copyless dSST. We finally give in Subsubsection C.3.3 the proof of Lemma C.11.

### C.3.1 Algorithmic description of the copyless dSST

We describe informally the behavior of the copyless dSST, denoted \( T' \). The goal of this section is to show the main ideas of the construction, without dealing with implementation details which shall be explained in Subsubsection C.3.2.

The key idea is that \( T' \), after reading position \( i \geq 0 \) of \( x \in A^\omega \), keeps track of a decomposition (see Definition C.12) which describes the substitution \( \lambda^x_i \) as a composition of \( K \)-bounded substitutions along the branches of a forest. We use the usual vocabulary for describing trees and forests: a leaf is a node which has children; the depth of a node is defined inductively when starting from the nodes with depth 0 (also called the roots); the height of the forest is the maximal depth of a node.

**Definition C.12** (Decomposition). Let \( x \in A^\omega \) and \( i \geq 0 \) be such that \( [\lambda^x_i]_i \) is defined. A decomposition of \( [\lambda^x_i]_i \) is:
1. a sequence \( 0 = i_0 < \cdots < i_m = i \) of positions;
2. a sequence \( \sigma_1, \ldots, \sigma_m \in \mathcal{S}_{\mathcal{R}'}^B \) of \( K \)-bounded substitutions such that for all \( 1 \leq \ell \leq m \), \( \sigma_\ell = \lambda_{i_\ell, i_{\ell+1}}^x \cdots \lambda_{i_1, i_0}^x \) restricted to \( \mathcal{R}' \);
3. a finite forest \( F \) whose nodes are labelled by functions \( g : \mathcal{R}' \to [0:K] \) such that:
   a. all the leaves have same depth \( m \) and distinct labels;
   b. there exists a leaf whose label is \( \text{copies}^x_i \);
   c. if \( h \) labels a node of depth \( 1 \leq \ell \leq m \) and \( g \) labels its parent, then for all \( \tau \in \mathcal{R}' \):
   \[
   g(\tau) = \sum_{s \in \mathcal{R}'} |\sigma_\ell(s)|_i \times h(s).
   \]

**Example C.13.** Assume that \( \mathcal{R}' = \{ \tau, s \} \), \( K = 5 \) and \( \lambda_1^\tau(\tau) = \tau, \lambda_1^\tau(s) = b s; \lambda_2^\tau(\tau) = s a s, \lambda_2^\tau(s) = s b; \lambda_3^\tau(\tau) = r, \lambda_3^\tau(s) = s \). We describe a possible decomposition of \( [\lambda^x_i]_i \):
1. the sequence \( i_0 = 0, i_1 = 1, i_2 = 3 \);
2. the substitutions \( \sigma_1, \sigma_2 \) defined by \( \sigma_1(\tau) = \tau, \sigma_1(s) = s b, \sigma_2(\tau) = s a s, \sigma_2(s) = s b \);
3. the forest depicted in Figure 5 (note that Equation 1 holds).

**Information stored by \( T' \).** The configuration of \( T' \) when in position \( i \geq 0 \) of \( x \in A^\omega \) stores some decomposition \( D^x_i \) of \( [\lambda^x_i]_i \) as follows, with \( m \leq L := (K+1)^{\mathcal{R}'} \):
1. the sequence of positions \( i_0 < \cdots < i_m \) is not stored by \( T' \) (its existence is an invariant which will be preserved along the computation);
2. the sequence of substitutions \( \sigma_1, \ldots, \sigma_m \) is stored depending on the forest \( F \), see below;
3. the forest \( F^x_i \) associated to \( D^x_i \). \( F^x_i \) has depth \( m \leq L \), hence its structure is a bounded information which can be stored in the state. Let us now explain how we deal with the
Initialization of the decomposition. When $i = 0$, the configuration of $T'$ describes the following decomposition $D^0_i$ of $[L]^T$ with $m = 0 ≤ L$:

1. the sequence $i_0 = 0$ (not stored explicitly);
2. no substitutions since $m = 0$;
3. the forest of depth 0 (i.e. it has only roots) with $L = (K + 1)|R'|$ nodes labelled by all possible functions $R' → [0:K]$. Conditions 3a and 3c of Definition C.12 are obviously true. Condition 3b follows from Claim C.10.

Updates of the decomposition. Assume that $λ^*_i + 1$ is defined (otherwise $T'$ gets blocked), and that $T'$ stores a decomposition $D^r_i$ of $[L]^T$. Then $T'$ transforms $D^r_i$ into a decomposition of $[L]^T_{i+1}$ as follows:

1. the new sequence of positions is $i_0 = 0 < \cdots < i_m = i < i_{m+1} = i + 1$. If $m+1 > L$, we first completely build this new decomposition, and then we apply the merging operation detailed in the next paragraph in order to reduce the depth to $L$;
2. the sequence $σ_1, \ldots, σ_m, σ_{m+1}$ where $σ_{m+1} := λ^*_i$. Note that the substitution $σ_{m+1}$ is $K$-bounded since $T$ is so;
3. before describing the new forest built from $F^x_i$, let us give some intuitions. Recall that $σ_{m+1}(\text{out}) = \text{out} α$ for some $α ∈ (B^* R')^x$. We want $T'$ to add $[α]_i$ to $\text{out}$ when reading $x[i+1]$. For this purpose, it needs to determine the value $λ^*_i; σ_1; \ldots; σ_m(r) = [r]_i$ for each $r ∈ R'$ which occurs in $α$. We thus define the functions $\text{used}_r : R' → [0:K]$ for $0 ≤ r ≤ m$, which describe “how many virtual copies” of the $σ_r(r)$ will be consumed to compute $[α]_{i+1}$. They are built by a decreasing induction:

$$\text{used}_0(r) := [σ_{m+1}(\text{out})]|_r$$ for all $r ∈ R'$;
$$\text{used}_r(r) := \sum_{s ∈ R'} [σ_{r+1}(s)]_i × \text{used}_{r+1}(s)$$ for all $r ∈ R'$, if $0 ≤ r < m$.

We now want to subtract the $\text{used}_r$ to the labels of the nodes, since they describe the number of copies that we need to “consume” to compute $λ^*_i; σ_1; \ldots; σ_m$. We can do that by the following recursive induction:

Proof. By Condition 3c on $F$ we have $g(r) = \sum_{s ∈ R'} [σ_{r+1}(s)]_i × h(s)$, therefore we get $g(r) − \text{used}_r(r) = \sum_{s ∈ R'} [σ_{r+1}(s)]_i × (h(s) − \text{used}_{r+1}(s))$ by definition of used.
Continuous rational functions are deterministic regular

We now describe in three steps how to build the new forest from $F^n_r$:

**Step 1: consuming the used $r$**. We replace each label $g$ in $F^n_r$ by $\bar{g}$ from Claim C.14 which may create negative labels. But since one leaf is labelled by $\text{copies}_r^n$, then by Lemma C.11 we see that $\text{copies}_r^n \geq 0$. Hence by Claim C.14 there is a branch whose labels are nonnegative. The copyless dSST shall use the “virtual copies of the $\sigma(r)$” stored along this branch to output $\lambda^n_r \circ \sigma_1 \circ \cdots \circ \sigma_m(\alpha)$ in a copyless fashion (see Subsubsection C.3.2).

**Step 2: adding level $m+1$**. For each leaf now labelled by $\bar{g}$, we create several children labelled by the $h : \mathbb{R}^r \to [0,K]$ such that for all $r \in \mathbb{R}^r$ we have:

$$ \bar{g}(r) = \sum_{s \in \mathbb{R}^r} |\sigma_{m+1}(s)|_r \times h(s). $$

For all $r \in \mathbb{R}$ and all created leaf labelled by $h$, the dSST creates $h(r)$ virtual copies of $\sigma_{m+1}(r) \in (\mathbb{R}^r \cup B)^*$ (which is a bounded information). Note that two created leaves cannot have the same label (otherwise it would be the case for their parents). Finally by Lemma C.11 the node labelled by $\text{copies}_{r+1}^n$ has a leaf labelled by $\text{copies}_{r+1}^n$.

**Step 3: removing errors**. Now it remains to deal with the fact that some nodes may have negative labels, and some leaves may have depth $\ell < m+1$. We thus remove all the nodes labelled by functions which take negative values, and their descendants. Finally, we trim the resulting forest by removing all nodes which are not ancestors of some leaf of depth $m+1$ (i.e. a leaf which has created in Step 2). It is easy to see that conditions $3a, 3c$ and $3b$ now hold.

**Merging operation: removing single children**. Let us now explain how to reduce the height of the decomposition obtained in the previous paragraph when $m+1 > L$.

▷ **Claim C.15.** If $m+1 > L$, there exists $1 \leq \ell \leq m$ such that all nodes of depth $\ell$ have exactly one children (in the forest built in the previous paragraph).

**Proof.** Assume that for all $1 \leq \ell \leq m$, some node of depth $\ell$ has at least two children. Since $m+1 > L$ and all leaves have the depth $m+1$, then our forest has more than $L$ leaves, which contradicts the fact that two leaves cannot have the same label.

The main idea is to “merge” $\sigma_\ell$ and $\sigma_{\ell+1}$ (which exists since $1 \leq \ell \leq m$). The decomposition of the previous paragraph is updated as follows to build $D^n_{\ell+1}$

1. the positions become $i_0 < i_1 < \cdots < i_{\ell-1} < i_{\ell+1} < \cdots < i_{m+1}$
2. the substitutions become $\sigma_1, \ldots, \sigma_{\ell-1}, \sigma_\ell \circ \sigma_{\ell+1}, \sigma_{\ell+2}, \ldots, \sigma_m$ (note that $\sigma_\ell \circ \sigma_{\ell+1}$ is $K$-bounded and corresponds to the restriction of $\lambda^n_{x_{\ell+1}} \circ \cdots \circ \lambda^n_{x_{\ell}}$);
3. before modifying the forest, we first show the following:

▷ **Claim C.16.** Assume that $h$ labels a node of depth $\ell+1$, and let $g$ be the label of its grandparent (for the forest built by the previous paragraph). Then for all $r \in \mathbb{R}^r$:

$$ g(r) = \sum_{s \in \mathbb{R}^r} |\sigma_\ell \circ \sigma_{\ell+1}(s)|_r \times h(s). $$

**Proof.** By Claim C.19 we have:

$$ \sum_{s \in \mathbb{R}^r} |\sigma_\ell \circ \sigma_{\ell+1}(s)|_r \times h(s) = \sum_{s \in \mathbb{R}^r} \sum_{t \in \mathbb{R}^r} |\sigma_{\ell}(t)|_r \times |\sigma_{\ell+1}(s)|_r \times h(s) $$

The result follows by permuting the sums and using condition 3c in $\ell$ and $\ell+1$. ▷
We thus transform the forest by merging each node of depth $\ell$ with its single child of depth $\ell+1$ (labelled by some $g$), and labelling the resulting node by $g$. Condition $\exists c$ still holds because of Claim C.16. Note that $T'$ also has to compute and store several copies of $\sigma_\ell \circ \sigma_{\ell+1}(r)$ for $r \in \mathcal{R}$. Subsection C.3.2 describes how to perform this update in a copyless fashion when starting from copies of $\sigma_\ell(s)$ and $\sigma_{\ell+1}(r)$.

Finally $T'$ has the same domain than $T$, since it gets blocked at position $i \geq 0$ if $\lambda_i^x$ is undefined, and otherwise it stores $[\text{out}]_x^T$ in its output $\text{out}$.

### C.3.2 Implementation details

In the previous subsection, we have described the behavior of the copyless dSST without detailing how, for each $g$ labelling a node of depth $1 \leq \ell \leq \ell_m$ and $r \in \mathcal{R'}$, this machine could “store $g(r)$ virtual copies of $\sigma_\ell(r) \in (\mathcal{R'} \uplus \mathcal{B})^*$”. We now explain it in detail.

#### Storing $K$-bounded substitutions.

We describe how a copyless dSST can store $K$-bounded substitutions (the ideas are mainly those of [7]). Let $\sigma \in \mathcal{S}_{\mathcal{R'}}^B$ be a $K$-bounded substitution, then for all $r \in \mathcal{R'}$ there exists $n \leq K' |\mathcal{R'}|$ such that $\sigma(r) = \alpha_0 \delta_1 \alpha_1 \cdots \delta_n \alpha_n$ with $\alpha_i \in B^*$, $t_i \in \mathcal{R'}$. We mainly have two informations in this expression:

- the sequence $s_1 s_2 \cdots s_n$ which describes where the former registers must be used;
- the sequence $\alpha_1 \cdots \alpha_n$ of (unbounded) words from $B^*$. Each of them must be stored in a register. Furthermore, we must keep track of the “mapping” between the registers and the $\alpha_i$, which is a bounded information.

We can now explain what we mean by “storing $g(r)$ virtual copies of $\sigma(r) \in (\mathcal{R'} \uplus \mathcal{B})^*$”: it means that we store $g(r)$ copies (in $g(r)$ distinct registers) of each $\alpha_i$. Note that if $g(r) \leq K$, we need at most $K^2 |\mathcal{R'}|$ registers. The sequence $s_1 s_2 \cdots s_n$ is stored the state.

#### Composing $K$-bounded substitutions.

With this representation, $T'$ is able to simulate the composition of two $K$-bounded substitutions, when their composition is itself $K$-bounded (which is always the case in the above merging operation).

**Claim C.17.** Assume that $\sigma, \sigma' \in \mathcal{S}_{\mathcal{R'}}^B$ and $\sigma \circ \sigma'$ are $K$-bounded and that $T'$ stores:

- $g(r)$ virtual copies of $\sigma(r)$ for $s \in \mathcal{R'}$, where $g : \mathcal{R'} \to [0:K]$;
- $g'(s)$ virtual copies of $\sigma'(s)$ for $s \in \mathcal{R'}$, where $g' : \mathcal{R'} \to [0:K]$.

Then there exists a copyless update of $T'$ for each $\alpha_i$. This result follows by summing over all $s \in \mathcal{R'}$.

**Proof.** In order to compute $g'(s)$ copies of $\sigma \circ \sigma'(s)$, we exactly need to used $|\sigma'(s)|_x \times g'(s)$ copies of $r$. The result follows by summing over all $s \in \mathcal{R'}$. ▶

Claim C.17 justifies how $T'$ can update is information in a copyless fashion when performing the merging operation. We still have to justify how $T'$ can compute $\lambda_0^x \circ \sigma_1 \circ \cdots \circ \sigma_m(\alpha)$ when it has to add something in $\text{out}$ in the above Step 1. As for the proof of Claim C.17 we exactly need to have used$_d(t)$ copies of $\sigma_d(t)$ for each $t \in \mathcal{R}$. This copies are taken along some root-to-leaf branch of the forest where removing the used$_d$ does not create negative labels $f$ (such a branch exists because copies$_d^T$ labels a leaf, as shown in Step 1). The remaining labels $f$ exactly correspond to the copies that were not used, hence they are still stored.

**Remark C.18.** To produce $[\alpha]_x^T$, we only need to consume the copies along one branch of $F_d^T$. However, to maintain a forest which is consistent with our decomposition $D_d^T$, we remove copies along all the branches, even if only one branch is truly used.
C.3.3  Proof of Lemma C.11

We first give a way to count the copies obtained when composing two substitutions.
▷ Claim C.19. Let $\sigma, \sigma' \in \mathcal{S}_R^I$, then for all $r, s \in \mathcal{R}$, $|\sigma \circ \sigma'(t)|_s = \sum_{t \in \mathcal{R}} |\sigma(t)|_s \times |\sigma'(t)|_t$.

Proof. We have $s'(t) = \alpha_0 t_1 \alpha_1 t_2 \ldots \alpha_n$ for some words $\alpha_i \in B^*$ and registers $t_i \in \mathcal{R}$. Therefore $|\sigma \circ \sigma'(t)|_s = \sum_{t \in \mathcal{R}} |\sigma(t)|_s \times |\sigma'(t)|_t$.

We then note that since $\text{out}$ is always updated under the form $\text{out} \alpha$, the number of copies of a given register in $\text{out}$ can only grow.

▷ Claim C.20. Given $r \in \mathcal{R}'$ and $i \geq 1$ such that $\lambda_i^r$ is defined, the function which maps $j \geq i-1 \mapsto |\lambda_i^r \circ \cdots \circ \lambda_j^r|_r$ is increasing (on its domain).

Proof. By Claim C.19, we have $|\lambda_i^r \circ \cdots \circ \lambda_{i+1}^r|_r \geq |\lambda_i^r \circ \cdots \circ \lambda_j^r|_r$. We are now ready to show Lemma C.11. If $j \geq i+1$ is such that $f_j$ is defined, we have the following for all $r \in \mathcal{R}'$ by Claim C.19:

$$|\lambda_{i+1}^r \circ \cdots \circ \lambda_j^r|_r = |\lambda_{i+1}^r|_r \times 1 + \sum_{s \in \mathcal{R}'} |\lambda_{i+2}^s|_r \times |\lambda_{i+2}^r \circ \cdots \circ \lambda_j^r|_s.$$ 

Now let $j_0 \geq i+1$ such that $|\lambda_{i+1}^r \circ \cdots \circ \lambda_{j_0}^r|_r$ is maximal. Since the function $j \mapsto |\lambda_{i+1}^r \circ \cdots \circ \lambda_j^r|_r$ is increasing by Claim C.19, we get for all $j \geq j_0$ (and when defined):

$$\text{copies}_j^r(t) = |\lambda_{i+1}^r \circ \cdots \circ \lambda_j^r|_r = |\lambda_{i+1}^r|_r \times 1 + \sum_{s \in \mathcal{R}'} |\lambda_{i+2}^s|_r \times |\lambda_{i+2}^r \circ \cdots \circ \lambda_j^r|_s.$$ 

And by the same argument of Claim C.19 applied to $|\lambda_{i+2}^r \circ \cdots \circ \lambda_j^r|_s$, we conclude:

$$\text{copies}_j^r(t) = |\lambda_{i+1}^r|_r \times 1 + \sum_{s \in \mathcal{R}'} |\lambda_{i+2}^s|_r \times \text{copies}_{i+1}^s(t).$$

D  Proof of Proposition 3.9

We show the stronger result which follows.

▷ Lemma D.1. If $f$ is deterministic regular, then $\text{Dom}(f)$ is Büchi deterministic. Conversely, if $L \subseteq \text{Dom}(f)$ is Büchi deterministic, then $f$ restricted to $L$ is deterministic regular.

Proof. We first show that the domain of a deterministic regular function is accepted by a Büchi deterministic automaton. For this let $T$ be a copyless dSST computing a deterministic regular function $f : A^* \rightarrow B^\omega$. We describe a deterministic Büchi automaton which follows the states of $T$ in a deterministic way (in particular, it gets blocked if $T$ gets blocked). It also memorizes for each $r \in \mathcal{R}$ if $[r]_r^T = \varepsilon$ or not (this is a bounded information which can be updated with a bounded memory). The automaton reaches an accepting state when $T$ adds a non-empty value into $\text{out}$. It is not hard to see that it exactly recognizes $\text{Dom}(f)$.

Now let $A$ be a deterministic Büchi automaton accepting a language $L \subseteq \text{Dom}(f)$. We build the product of $T$ and $A$ and follow the transition functions of both machines. The register updates are the same as in $T$ for all $r \in \mathcal{R} \setminus \{\text{out}\}$. For $\text{out}$, we store temporarily the values which are added in a new register $\text{out}'$, which is added to $\text{out}$ only when leaving an accepting state of $A$. Thus the output (which is always a prefix of the output of $T$) is infinite if and only if the input is accepted by $A$. ▷
Proof of Theorem 3.11

Let us consider a restricted 1-nT computing a function \( f : A^\omega \to B^\omega \) and a copyless dSST computing a deterministic regular function \( g : B^\omega \to C^\omega \). By making their cascade product (a standard construction for composing one way machines), we can easily build a copyless restricted non-deterministic streaming string transducer computing \( h := g \circ f \).

**Definition E.1.** A restricted non-deterministic streaming string transducer (restricted nSST) \( N = (A, C, Q, I, \Delta, R, \text{out}, \lambda) \) consists of:

- a finite input (resp. output) alphabet \( A \) (resp. \( C \));
- a finite set of states \( Q \) with \( I \subseteq Q \) initial;
- a transition relation \( \Delta \subseteq Q \times A \times Q \);
- a finite set of registers \( R \) with a distinguished output register \( \text{out} \in R \);
- an update function \( \lambda : \Delta \to S^C_R \) such that for all \( (q, a, q) \in \Delta \):
  - \( \lambda(q, a, q')(\text{out}) = \text{out} \cdots \);
  - there is no other occurrence of \( \text{out} \) in \( \{\lambda(q, a, q')(r) : r \in R\} \).

Given \( x \in A^\omega \cup A^* \), a (resp. initial, final, accepting) run of \( N \) labelled by \( x \) is a (resp. initial, final, accepting) run of the underlying one-way automaton \( (A, Q, I, F, \Delta) \) where \( F = Q \) (all states are accepting). Given an initial run \( \rho = q_0 \to q_1 \to \cdots \) composed of \( n \in \mathbb{N} \cup \{\infty\} \) transitions, we define for \( 0 \leq i \leq n \) the substitution \( \lambda^i_\rho \in S^C_R \) by \( \lambda^0_\rho(\tau) = \varepsilon \) for all \( \tau \in R \), and \( \lambda^i_\rho : \lambda(q_i, x[i]) \) for \( i \geq 1 \) We define the substitution \( \lambda^i_\rho, \rho \subseteq \lambda^0_0 \circ \cdots \circ \lambda^{i-1}_\rho \).

By construction one has that \([\text{out}]^\rho_i \subseteq [\text{out}]^{i+1}_\rho \) (when defined).

The restricted nSST computes a function \( h \) defined as follows. Let \( x \in A^\omega \) be such that there exists a unique accepting run \( \rho \) labelled by \( x \), and such that \( [\text{out}]^\rho_i \to \infty \). Then \( h(x) := V_{i+1}[\text{out}]^\rho_i \). Otherwise \( h(x) \) is undefined. We say that the restricted nSST is copyless if all the \( \lambda^i_\rho \) are copyless when \( \rho \) is an initial run. Equivalently (up to trimming), \( \lambda(q, a, q') \) is always copyless when defined.

**Building a 1-bounded dSST.** We now build a 1-bounded dSST \( T \) which computes an extension of \( h \) (i.e. whose domain contains \( \text{Dom}(h) \) and which coincides with \( h \) on this set).

We shall deal with the domain in the next paragraph. The general idea is to perform a subset construction on \( N \), while keeping track of the forest of all initial runs and the outputs that were produced along its branches. Without loss of generality, we assume that \( I = \{q_0\} \) is a singleton. Thus the forest of initial runs is a tree. If a single accepting run exists, then the first node of this tree which has at least two children must move forward infinitely often, which enables to produce the whole output of \( T \). The fact that we obtain a 1-bounded transducer and not a copyless one follows from the fact that the same value can be re-used when creating new branches (but the branches will never be merged later).

**Definition E.2 (Tree of runs).** Given \( x \in A^\omega \) and \( i \geq 0 \) we define the tree of runs \( T^x_i \) as a tree whose nodes are labelled by tuples \( (i_1, \rho, i_2) \) with \( 0 \leq i_1 \leq i_2 \) and \( \rho \) is a run labelled by \( x[i_1+1:i_2] \). It is built by induction as follows:

- \( T^x_0 \) consists of a single node labelled by \( (0, \rho, 0) \) where by \( \rho = q_0 \) is a run labelled by \( \varepsilon \);
- if \( T^x_i \) is built, we obtain \( T^x_{i+1} \) by doing the following operations where \( a := x[i+1] \):
  - **New transition:** below each leaf labelled by \( (i_1, \rho, i) \) with \( \rho = q_1 \to \cdots \to q_i \), and for all transitions \( (q_i, a, q_{i+1}) \in \Delta \), we add a leaf labelled by \( (i, q_i \to q_{i+1}, i+1) \);
  - **Removing ambiguity:** for all \( q' \in Q \), if at least two leaves created by the previous step are such that \( q' = q_{i+1} \), then we remove all the created leaves such that \( q' = q_{i+1} \) (indeed,
Continuous rational functions are deterministic regular

if there exists an infinite run starting on \( q' \) and labelled by \( x[i+2:] \), then \( x \notin \text{Dom}(f) \) since it has two accepting runs;

**Trimming the tree:** we remove all nodes in \( T^{x}_{i+1} \) which are not an ancestor of a new leaf;

**Merging single nodes:** for any node (labelled by \((i_{1}, q_{1} \rightarrow \cdots \rightarrow q_{k}, i_{2})\)) which has only one child (labelled by \((i_{2}, q_{2} \rightarrow \cdots \rightarrow q_{k}, i_{3})\)) we merge these two nodes together with common label \((i_{1}, q_{1} \rightarrow \cdots \rightarrow q_{k} \rightarrow \cdots \rightarrow q_{k}, i_{3})\).

The following properties immediately follow from the construction.

▷ **Claim E.3.** There exists \( L \geq 0 \), such that for all \( x \in A^{\omega} \), \( i \geq 0 \), \( T^{x}_{i} \) has at most \( L \) nodes.

▷ **Claim E.4.** If \( x \in \text{Dom}(f) \), there exists a root-to-leaf branch of \( T^{x}_{i} \) whose labels describe the beginning of the unique accepting run labelled by \( x \).

We also note that if \( x \in \text{Dom}(f) \), the run stored in the root of \( T^{x}_{i} \) becomes longer and longer.

▷ **Claim E.5.** If \( x \in \text{Dom}(f) \) and \( (0, r_i, r_i) \) is the label of the root of \( T^{x}_{i} \), then \( r_i \to \infty \).

**Proof.** Assume that \( r_i \) is ultimately constant. Then the root of \( T^{x}_{i} \) always has at least two children, which implies that there exists two distinct infinite runs labelled by \( x \). ◀

Let us finally describe how \( T^{x}_{i} \) is used to build a 1-bounded dSST \( \mathcal{T} \). When in position \( i > 0 \) of its input \( x \in A^{\omega} \), the dSST stores the following information:

- the structure of \( T^{x}_{i} \), i.e. the tree without its labels (stored in the state);
- for each leaf of \( T^{x}_{i} \) labelled by \((i_{1}, q_{1} \rightarrow \cdots \rightarrow q_{k}, i)\), the last state \( q_{k} \) (stored in the state);
- for each node of \( T^{x}_{i} \) labelled by \((j, \rho_{n}, j')\), let \( \rho := \rho_{1} \cdots \rho_{n} \) be the run labelling the branch of \( T^{x}_{i} \) starting in the root and ending in \((j, \rho_{n}, j')\). Let \( \sigma_{n} := \lambda_{i+1}^{n} \circ \cdots \circ \lambda_{i}^{n} \) be the substitution applied along \( \rho_{n} \). Let \( \alpha_{i} \in ((\mathcal{R} \setminus \{\text{out}\}) \cup C)^{+} \) be such that \( \sigma_{n}(\text{out}) = \text{out} \alpha_{i} \), then we store \([\alpha]_{\rho_{1}^{n} \circ \cdots \circ \rho_{n-1}}^{i} \) (it is the value which is added in \( \text{out} \) when following \( \rho_{n} \)).

Furthermore we do the following for the root and the leaves:

- if \((j, \rho_{n}, j')\) is the root of \( T^{x}_{i} \), then \([\alpha]_{\rho_{1}^{n} \circ \cdots \circ \rho_{n-1}}^{i} \) is stored in the register \( \text{out} \) of \( \mathcal{T} \);
- if \((j, \rho_{n}, j')\) is a leaf, we also store for all \( r \in \mathcal{R} \setminus \{\text{out}\} \), the value \([r]_{j'}^{i} \) (it is the value of \( r \) after executing \( \rho \)).

It is clear that for \( x \in \text{Dom}(h) \), the value of \text{out} \ in \( \mathcal{T} \) in always a prefix of \( h(x) \). Furthermore, this value tends to an infinite word by Claim E.5 and the semantics of restricted nSST. The updates of \( \mathcal{T} \) can be performed by following the operations which build \( T^{x}_{i} \) in Definition E.2.

Furthermore, the machine can be built in a 1-bounded way. Indeed, \( \mathcal{N} \) was copyless, hence we can check that given a branch of \( T^{x}_{i} \), there is no need to make copies in order to update the information stored by \( \mathcal{T} \) along this branch (hence we only create copies if a leaf creates several children, but they correspond to several distinct futures).

**Domains.** We have built a 1-bounded dSST computing an extension of \( h \). We now show that its domain can be restricted to \( \text{Dom}(h) \). This result follows from lemmas D.1 and E.6.

▷ **Lemma E.6.** If \( h \) is computed by a restricted nSST, then \( \text{Dom}(h) \) is Büchi deterministic.

**Proof.** The idea is to build a deterministic Büchi automaton which keeps track of \( T^{x}_{i} \) as \( \mathcal{T} \) does. However, checking that \( r_{i} \to \infty \) (see Claim E.5) and that the out of \( \mathcal{T} \) tends to an infinite value is not sufficient for \( x \in A^{\omega} \) to be in \( \text{Dom}(f) \). Indeed, there may exist infinite runs which we removed in the operation “Removing ambiguity” of Definition E.2. Hence if \( q' \) is defined as in “Removing ambiguity”, we also have to check all runs which start in \( q' \) and are labelled by a prefix of \( x[i+2:] \) are finite. This can be done by simulating these runs, and reaching an accepting state only when this simulation gets blocked. ◀
F  Proof of Lemma 4.3

Let $x \in A^*$ and $\beta \in B^*$ be such that $q'_2 \xrightarrow{\beta} x$ is final (such a run exists since the transducer is trim and clean). Therefore, for all $n \geq 0$ we have $f(uu^n x) = \alpha_2 q'_2 \beta$. On the other hand $f(uu^n x) = \alpha_1 q_1'$ because $q_1 \in F$. By continuity in $uu^n x \in \text{Dom}(f)$, for all $p \geq 0$ we have $|f(uu^n x) \wedge f(uu^n)| \geq p$ for $n$ large enough. The result follows directly.

G  Proof of Lemma 4.4

Let $T = (A, B, Q, I, F, \Delta, \lambda)$ be a trim, unambiguous and clean 1-nT which computes a continuous function $f : A^* \rightarrow B^*$. Note that Lemma 4.3 holds.

Definition G.1. We say that $q'_2 \in Q \setminus F$ is constant if there exists $q_1, q_2 \in I$, $q'_1 \in F$, $u \in A^*$, $u' \in A^+$, $\alpha_1, \alpha_2, \alpha'_2 \in B^+$ with $\alpha'_2 = \varepsilon$ such that $q_1 \xrightarrow{u | \alpha_1} q'_1 \xrightarrow{\alpha_2} q'_2$.

Since $T$ is clean, the existence of constant states is clearly equivalent to the non-productivity of $T$. Thus we want to avoid such states. Let us now justify the “constant” terminology.

Claim G.2. We can compute the set of constant states. Furthermore given a constant state $q$, we can compute $\alpha_2 \in B^+$, $\alpha'_2 \in B^*$ such that for all final run $q \xrightarrow{\alpha_2} \beta = \alpha_2 \alpha'_2 \varepsilon$.

Proof. If $q'_2$ is constant, we can enforce $|u|, |u'| \leq |Q|$ (see e.g. Lemma H.1 which states a more general result). Hence we can decide if a state is constant. Furthermore, if $q'_2$ is constant then by Lemma 4.3, we have $\beta = (\alpha_2)^{-1} \alpha_1 \alpha'_2$ for all $q \xrightarrow{\alpha_2} \varepsilon$.

Given $q$ constant, we label a final run starting in $q$. Let $T_q = (A, B, Q_q, I_q, F_q, \Delta_q, \lambda_q)$ be:

- $Q_q := \{q\} \cup \{q' : q \xrightarrow{*} q'\} \times \{1\}$;
- $I_q := \{q\}$ and $F_q := \{q' : q' \in F\}$;
- $\Delta_q := \{(q, a, (q', 1)) : (q, a, q') \in \Delta\} \cup \{((q', 1), a, (q'', 1)) : (q', a, q'') \in \Delta\}$;
- $\lambda_q(q, a, (q', 1)) = \lambda_q$ and $\lambda_q((q', 1), a, (q'', 1)) = \alpha'_q$.

Claim G.3. If $q \xrightarrow{\alpha} q$ in $T_q$, then $u \neq \varepsilon$ implies $\alpha \neq \varepsilon$. Furthermore $T_q$ is unambiguous and it computes $f_q : A^* \rightarrow B^*$ such that $x \in \text{Dom}(f_q)$ if and only if there exists a (unique) final run $q \xrightarrow{\alpha q} \beta$ in $T$, and then $f_q(x) = \beta$.

Proof. It is clear that $T_q$ has no loop which produces $\varepsilon$ since $\alpha'_q \neq \varepsilon$. Furthermore, an accepting run of $T_q$ labelled by $x$ is necessarily of the form $q \xrightarrow{\alpha q} (q_1, 1) \xrightarrow{\alpha_2 q} (q_2, 1) \cdots$, where $q \xrightarrow{\alpha_1} q_1$, $\xrightarrow{\alpha_2} q_2 \cdots$ is final in $T$. The converse also holds. Hence $T_q$ is unambiguous since $T$ is unambiguous and trim, and furthermore it computes the function $f_q$.

Finally, let us build $T' = (A, B, Q', I', F', \Delta', \lambda')$ which computes $f$ and is productive and unambiguous (the trimming can be done after). It consists of the disjoint union of $T$ and $T_q$ for $q$ constant. Its initial states are those of $T$, and the final states are both those of $T$ and $T_q$ for $q$ constant. Furthermore, for $q$ constant, all the outgoing transitions from $q$ in $T$ are removed, and $q$ is merged with the initial state of $T_q$ (we are forced to go in $T_q$).

Claim G.4. Let $x \in \text{Dom}(f)$ and $\rho$ be the accepting run of $T$ labelled by $x$. Assume that it never visits a constant state. Then $\rho$ is also an accepting run in $T'$ (which stays in $T$) with same label and output than in $T$. Conversely, if $\rho$ is an accepting run of $T'$ which stays in $T$, labelled by $x \in A^*$, then $\rho$ is an accepting run of $T$ which never visits a constant state, with same label and output than in $T'$.  

Claim G.5. Let \( x \in \text{Dom}(f) \) and \( \rho = q_0 \xrightarrow{[1]} q_1 \cdots \) be the accepting run of \( \mathcal{T} \) labelled by \( x \). Assume that it visits a constant state for the first time in \( q_i \) for \( i \geq 0 \). Then \( \rho' := q_0 \xrightarrow{[1]} q_1 \xrightarrow{[i+1]} (q_{i+1}, 1) \cdots \) is also an accepting run in \( \mathcal{T}' \) with same label and output than \( \rho \). Conversely, if \( \rho' \) is an accepting run of \( \mathcal{T}' \) labelled by \( x \in A^\omega \), which goes into some \( T_q \) at some point, then it stays in this \( T_q \) and is of the form \( \rho' := q_0 \xrightarrow{[1]} q_1 \xrightarrow{[i+1]} (q_{i+1}, 1) \cdots \). where \( q_i = q \) is constant in \( \mathcal{T} \). Then \( \rho = q_0 \xrightarrow{[1]} q_1 \xrightarrow{[2]} q_2 \cdots \) is an accepting run in \( \mathcal{T} \) which visits a constant state, and with the same output as \( \rho' \).

The two above claims enable us to show that \( \mathcal{T}' \) is unambiguous and that it computes the function \( f \). It is still clean, we now check that it is productive. Assume that there exists \( q_1, q_2 \in I', q_1' \in F', q_2' \in Q \times F, u \in A^*, u' \in A^+, \alpha_1, \alpha_2, \alpha_2' \in B^* \) with \( \alpha_2' = \varepsilon \) such that \( q_1 \xrightarrow{\alpha_1 \alpha_2} q_1' \xrightarrow{\alpha_1 \alpha_2} q_2' \). Then \( q_2' \) is not in one of the \( \mathcal{T}_q \), because these machines have no \( \varepsilon \)-loops by Claim G.3. Thus we had a run \( q_2 \xrightarrow{\alpha_2} q_2' \xrightarrow{\alpha_2} q_2' \in \mathcal{T} \) and so \( q_2' \) was not constant in \( \mathcal{T} \) (because otherwise it would be the initial state of \( \mathcal{T}_q \)). Now:

- either \( q_1 \xrightarrow{\alpha_1} q_1' \xrightarrow{\alpha_1} q_1' \) stays in \( \mathcal{T} \). Since \( q_1' \in F', q_1' \in F \) and this statement
  contradicts the fact that \( q_2' \) is not constant in \( \mathcal{T} \);
- or \( q_1 \xrightarrow{\alpha_1} q_1' \xrightarrow{\alpha_1} q_1' \) goes in \( \mathcal{T}_q \) at some point, for some constant state \( q \). Assume without loss of generality that \( q \not\in T_q \), then \( q_1' = (q''_1, 1) \) and \( q_1 \xrightarrow{\alpha_1} q_1' \xrightarrow{\alpha_2} q''_1 \) is a run in \( \mathcal{T} \). Then \( q''_1 \in F \) since \( q_1'' \in F_q \), hence \( \varepsilon_1 = \varepsilon \) because \( \mathcal{T} \) was clean. This contradicts the fact that \( q_2' \) is not constant in \( \mathcal{T} \).

The goal of this section is to show lemmas 4.8 and 4.13. For this purpose, we establish several properties of the 1-nT \( \mathcal{T} = (A, B, Q, I, F, \Delta, \lambda) \) which is assumed to compute a continuous function. The two results will (respectively) be consequences of lemmas H.2 and H.6.

H Proofs of lemmas 4.8 and 4.13

We first give a pumping-like characterization of compatible sets (which implies that one can decide if a set is compatible).

Lemma H.1 (Characterization of compatibility). The set \( C \) is compatible if and only if there exists a function \( d : C \to Q \), and \( u, u' \in A^* \) such that the following holds:

- for all \( q \in C \), \( q \xrightarrow{\varepsilon} d(q) \xrightarrow{\alpha} d(q) \);
- there exists \( q \in C \) such that \( d(q) \) is accepting,
- \( u' \neq \varepsilon \) and \( |u|, |u'| \leq |Q|^{|Q|} \).

Proof. It is clear that this condition implies compatibility. Now assume that there exists \( x \in A^\omega \) and infinite runs \( \rho_q \) for \( q \in C \) labelled by \( x \) such that \( \forall q \in C, \rho_q \) begins in \( q \), and furthermore \( \rho_q \) is final for some \( p \in Q \). Therefore we have \( \rho_q(i) \in F \) infinitely often, and by a pigeonhole argument we get \( i < j \in \mathbb{N} \) such that \( \rho_q(i) = \rho_q(j) \) for all \( q \in C \), and \( \rho_q(i) \in F \). We define \( d(q) := \rho_q(i) \) for \( q \in C \). Finally we get \( q \xrightarrow{\alpha(u)} d(q) \xrightarrow{\alpha(u')} d(q) \) for all \( q \in C \).

Now if \( |x[i+1]| \geq |Q|^{|Q|} + 1 \), by a similar pumping argument we factor \( x[i+1] = vv'v'' \) such that \( \varepsilon' \neq \varepsilon \) and \( q \xrightarrow{vv'} d(q) \xrightarrow{\alpha(u)} d(q) \) for all \( q \in C \). By induction we obtain \( |u| \leq |Q|^{|Q|} \) and a similar reasoning gives \( 1 \leq |u'| \leq |Q|^{|Q|} \).

We now introduce the notion of end, which allows to complete initials runs by some future.

Lemma H.2 (End). Let \( C \in \text{Comp} \), there exists a function \( \text{end}_C : C \to B^\omega \) such that for all initial step \( J, u, C \) and \( p, q \in C \) we have

\[
\prod_{J, C}^p (\text{end}_C(q)) = \prod_{J, C}^p (\text{end}_C(q)) \text{end}_C(q).
\]
Proof. Since \( C \) is compatible we get by Lemma H.1 words \( v \in A^+, v' \in A^+ \) and a function 
\( d : C \rightarrow Q \) such that for all \( q \in C, q \xrightarrow{\alpha(q),\beta(q)} d(q) \) with \( \alpha(q), \alpha'(q) \in B^+ \). Since \( \mathcal{T} \) is trim and clean, for all \( q \in C \) there exists \( x(q) \in A^+, \beta(q) \in B^2 \) such that \( d(q) \xrightarrow{\alpha(q),\beta(q)} \infty \) is accepting. We define \( \text{end}_C(q) := \alpha(q)\alpha'(q)^\ast \) if \( \alpha'(q) \neq \varepsilon \) and \( \text{end}_C(q) := \alpha(q)\beta(q) \) otherwise. Let us now justify that \( \text{end}_C \) verifies our equalities. By Lemma H.1 there is some \( p \in C \) such that \( d(p) \) is final. By transitivity it is enough to show that for all \( q \in C \) we have 
\( \text{prod}^u_{J,C}(p) \text{end}_C(p) = \text{prod}^u_{J,C}(q) \text{end}_C(q) \), which is a direct consequence of Lemma H.3. ▲

The notions of common and advance are presented in Definition 4.10. They enable us to reformulate Lemma H.2 as follows.

\[ \square \]

\textbf{Lemma H.3.} Let \( J, v, D \) be an initial step and \( C, v, D \) be a step. Then for all \( p, q \in D \):
\[ \text{adv}^v_{J,C}(\text{pre}^v_{C,D}(p)) \text{prod}^v_{C,D}(q) \text{end}_D(q) = \text{adv}^v_{J,C}(\text{pre}^v_{C,D}(q)) \text{prod}^v_{C,D}(D)(p) \text{end}_D(p). \]

Proof. By Lemma H.2 we have \( \text{prod}^v_{J,C}(p) \text{end}_D(p) = \text{prod}^v_{J,C}(q) \text{end}_D(q) \) which by splitting gives 
\( \text{prod}^v_{J,C}(\text{pre}^v_{C,D}(p)) \text{prod}^v_{C,D}(q) \text{end}_D(p) = \text{prod}^v_{J,C}(\text{pre}^v_{C,D}(q)) \text{prod}^v_{C,D}(q) \text{end}_D(q). \) ▲

\textbf{H.2 Separable compatible sets}

The notion of separable compatible set is presented in Definition 4.11. We first give a pumping-like characterization of these sets (which implies that one can decide if a set is separable).

\[ \square \]

\textbf{Lemma H.4 (Characterization of separability).} A set \( C \in \text{Comp} \) is separable if and only if there exists two functions \( i : C \rightarrow I \) and \( \ell : C \rightarrow Q \), \( u, u', u'' \in A^+ \) and three functions \( \alpha, \alpha', \alpha'' : C \rightarrow B^+ \) such that the following holds:
\[ \begin{align*}
&\text{for all } q \in C, \ i(q) \xrightarrow{u(q),\ell(q)} u'(q), u''(q), \ \ell(q), u''(q), q; \\
&u' \neq \varepsilon \text{ and } |u, |u'||, |u''| \leq |Q|^{|Q|}; \\
&\text{there exists } p, q \in C \text{ such that } |\alpha(p)| \neq |\alpha'(q)|. 
\end{align*} \]

Proof. If the conditions holds, then by iterating the loop the set is separable. Conversely, let \( J, v, C \) and \( p, q \in C \) be such that \( || \text{prod}^p_{J,C}(p) || > |M|Q|^{|Q|} \). Suppose by symmetry that \( | \text{prod}^p_{J,C}(p) \rangle > | \text{prod}^p_{J,C}(q) \rangle + |M|Q|^{|Q|} \). Thus \( |v| > |Q|^{|Q|} \). By pumping we can factor \( v = uu'u'' \) with \( 0 < |u'| \leq |Q|^{|Q|} \) such that \( i(q) \xrightarrow{u(q),\ell(q)} u'(q), u''(q), \ell(q), u''(q) \) for all \( r \in Q \). Now, if \( |\alpha'(p)| = |\alpha'(q)| \), we can remove the loop and get the result by induction since \( |uu'| < |v| \) and \( |\alpha(p)\alpha''(p)| > |\alpha(q)\alpha''(q)| + |M|Q|^{|Q|} \). Otherwise \( |\alpha'(p)| \neq |\alpha'(q)| \) and we can enforce \( |u, |u'|| \leq |Q|^{|Q|} \) by a similar pumping argument. ▲

\[ \square \]

\textbf{Remark H.5.} Since \( \Omega := |M|Q|^{|Q|} \), observe that \( |\alpha(q)|, |\alpha'(q)|, |\alpha''(q)| \leq \Omega \).

We now state the strong version of Lemma 4.13. Its proof is given in Subsection H.3.

\[ \square \]

\textbf{Lemma H.6 (Looping futures - strong version).} Let \( C \in \text{Comp} \) be separable and \( J, v, C \) be an initial step. There exists \( r, \theta \in B^+ \) with \( |\theta| = \Omega! \) and \( |r| \leq \Omega! \), which can be uniquely determined from \( C \) and \( \text{adv}^v_{J,C}(q) \) for \( q \in C \), such that:
\[ \begin{align*}
&\text{max-adv}^v_{J,C} \subseteq \tau \theta^\ast; \\
&\text{for all step } C, v, D \text{ and } q \in D, \text{prod}^v_{J,C}(q) \text{end}_D(q) = (\text{adv}^v_{J,C}(\text{pre}^v_{C,D}(q)))^{-1} \tau \theta^\ast. 
\end{align*} \]
**H.3 Proof of Lemma H.6**

Since $C$ is separable, we get $p, q \in C$ verifying the conditions of Lemma H.4. Assume by symmetry that $0 \leq |\alpha'(q)| < |\alpha'(p)| \leq \Omega$. Note that $i(C), uu''u'', C$ is an initial step for all $n \geq 0$. From this observation, we deduce Sublemma H.7.

**Sublemma H.7.** There exists $\beta, \theta \in B^*$ such that $|\beta| \leq \Omega$ and $|\theta| = \Omega!$, and $N, K \geq 0$ such that for all $n$ large enough, we have:

$$\beta\theta^n-K \subseteq \left(\prod_{i(C),C} u''u''u''(q)\right)^{-1} \prod_{i(C),C} u''u''u''(p).$$

Furthermore, the values $\beta$ and $\theta$ can be computed from $i(C), C, u, u'$ and $u''$.

**Proof.** Let us first observe that for $n \geq 0$ large enough, we have:

$$\pi_n := \left(\prod_{i(C),C} u''u''u''(q)\right)^{-1} \prod_{i(C),C} u''u''u''(p) = \alpha(p)\alpha'(p)\alpha''(p)[t_n]$$

where $t_n := |\alpha(q)| + n \times |\alpha'(q)| + |\alpha''(q)|$. If $|\alpha'(q)| = 0$ the result is clear. From now on, we assume that $|\alpha'(q)| > 0$. Let us consider $n|\alpha'(p)|$ iterations of the loop, then:

$$\pi_{n|\alpha'(p)|} = \left(\alpha'(p)^{n|\alpha'(p)|}\alpha''(p)\right) [\alpha(q)] + |\alpha''(q)| - |\alpha(p)| + n|\alpha'(p)||\alpha'(q)|]$$

$$= \left(\alpha'(p)^{n-M|\alpha'(p)|-|\alpha''(q)|}\alpha''(p)\right) [\alpha(q)] + |\alpha''(q)| - |\alpha(p)| + M|\alpha'(p)||\alpha'(q)|]$$

where $M$ is fixed such that $|\alpha(q)| + |\alpha''(q)| - |\alpha(p)| + M|\alpha'(p)||\alpha'(q)| \geq 0$. The result easily follows by choosing $\theta := \varphi^{N|\alpha'(p)|}$ where $\varphi$ is defined as a conjugate of $\alpha'(p)$ (shifted of $|\alpha(q)| + |\alpha''(q)| - |\alpha(p)| + M|\alpha'(p)||\alpha'(q)|]$, and $\beta = \varepsilon$.

From this result, we now deduce that the futures have a looping behavior.

**Sublemma H.8.** For all step $C, v, D$ and for all $\tau \in D$, if $r := \pre_n^{\tau} C, D(\tau)$ then we have $
ord\prod_\tau C, D(\tau) = (\ord_\tau C, D(r))^{-1} (\ord_\tau C, D(q)) \beta\theta\omega$.

**Proof.** Let $\overline{p}, \overline{q}$ be such that $\pre_n^{\tau} C, D(\overline{p}) = p$ and $\pre_n^{\tau} C, D(\overline{q}) = q$. We get from Lemma H.3 applied to $i(C), uu''u'', C$ and $C, v, D$ that:

$$\prod_{C, D}(\overline{q}) \ord_{C, D}(\overline{q}) = \left(\prod_{i(C),C} u''u''u''(q)\right)^{-1} \prod_{i(C),C} u''u''u''(p) \ord_{C, D}(\overline{p}) \ord_{C, D}(\overline{q}) \ord_{C, D}(\overline{q}).$$

For $n$ large enough, Sublemma H.7 shows $\beta\theta^n-K \subseteq \prod_{C, D}(\overline{q}) \ord_{C, D}(\overline{q})$. Therefore, $\beta\theta^n-K \subseteq \prod_{C, D}(\overline{q}) \ord_{C, D}(\overline{q})$. Hence $\prod_{C, D}(\overline{q}) \ord_{C, D}(\overline{q}) = \beta\theta\omega$ because we can chose arbitrarily large $n \geq 0$.

Now, let us apply Lemma H.3 to $i(C), uu'u'', C$ and $C, v, D$, we get

$$\prod_{C, D}(\overline{q}) \ord_{C, D}(\overline{q}) = (\ord_\tau C, D(r))^{-1} \ord_\tau C, D(q) \ord_{C, D}(\overline{q}) \ord_{C, D}(\overline{q}) = \beta\theta\omega$$

and the result follows immediately.

Let us now consider what happens with the step $J, w, C$. Let $r \in C$ (resp. $s \in C$) be such that $\ord_{\tau} C, D(r) = \varepsilon$ (resp. $\ord_{\tau} C, D(s) = \max_{\tau} \ord_{\tau} C$), i.e. the run ending in $r$ (resp. in $s$) has the smallest (resp. the longest) production.

\[\Box\]
Let $C, v, D$ be a step and $\tau \in D$ (resp. $\tau$) be such that $s = \text{pre}_{C, D}(\tau)$ (resp. $r = \text{pre}_{C, D}(\tau)$).

Note that there exists such a step (at least the empty one), and furthermore:

$$
\max_{J, C} \prod_{C, D}(\tau) \text{ end}_D(\tau) = \text{adv}_{J, C}(s) \prod_{C, D}(\tau) \text{ end}_D(\tau)
$$

by choice of $s$;

$$
= \text{adv}_{J, C}(r) \prod_{C, D}(\tau) \text{ end}_D(\tau)
$$

by Lemma H.13;

$$
= \varepsilon \prod_{C, D}(\tau) \text{ end}_D(\tau)
$$

by choice of $r$;

$$
(\text{adv}_{i(C, C)}(r))^{-1} (\text{adv}_{i(C, C)}(q)) \beta \theta^\omega
$$

by Sublemma H.16.

Let $m := |(\text{adv}_{i(C, C)}(q))| - |(\text{adv}_{i(C, C)}(r))| + |\beta|$, then $-3\Omega \leq m \leq 4\Omega$ (indeed $|\beta| \leq \Omega$, and furthermore $|\text{adv}_{i(C, C)}(q)| \leq 3\Omega$ and $|\text{adv}_{i(C, C)}(r)| \leq 3\Omega$ because $|uu''| \leq 3|Q|^{|Q|}$).

Now, observe that $|\theta| = \Omega! \geq 4\Omega$ because $\Omega = M|Q|^{|Q|}$ and because we have chosen $M \geq 10$. We finally make a case disjunction depending on the sign of $m$:

- if $m \geq 0$, we let $\tau := (\text{adv}_{i(C, C)}(r))^{-1} (\text{adv}_{i(C, C)}(q)) \beta$;
- if $m < 0$, we let $\tau := (\text{adv}_{i(C, C)}(r))^{-1} (\text{adv}_{i(C, C)}(q)) \beta \psi$.

Note that $|\tau| \leq \Omega!$ and that it only depends on $\beta$ (i.e. on the step $i(C, uu''C)$ and on $r$ (thus on the advances of $J, w, C$), but not on the “future” step $C, v, D$ that we have selected).

Hence, for all step $C, v, D$ we have:

$$
\max_{J, C} \prod_{C, D}(\tau) \text{ end}_D(\tau) = \tau \theta^\omega.
$$

and Lemma H.16 immediately follows.

1 Proof of Lemma 4.16

We denote by Parts the powerset $2^Q$. We fix a total ordering $\prec$ on Parts.

1.1 Properties of compatible sets

We begin this proof by giving some basic properties of compatible sets.

Definition 1.1. If $u \in A^*$ and $S \subseteq Q$, we let $S \cdot u := \{ q' : \exists q \in S, q \uparrow_\omega, q' \subseteq Q \}$.

Lemma 1.2 (Compatible sets cover the future). Let $x \in \text{Dom}(f)$, $i \geq 0$ and $q^x_i \in S \subseteq Q$. Then there exists $C \in \text{Comp}(S)$ and $j \geq i$ such that $C \cdot x[i+1:j] = S \cdot x[i+1:j]$.

Proof. Assume by contradiction that the property of Lemma 1.2 does not hold. Let $P$ be the set of subsets $C \subseteq S$ such that $q^x_i \in C$, and for all $p \in P$ there exists an infinite run starting in $p$ and labelled by $x[i+1:]$. Then $P \subseteq \text{Comp}(S)$ and $\{q^x_i\} \subseteq P$.

Now consider a set $C \in P$ such that $|C| = \max_{C' \in P} |C'|$. Since $C \in \text{Comp}(S)$ then $\forall j \geq i$ we must have $C \cdot x[i+1:j] \neq S \cdot x[i+1:j]$, thus $(S \setminus C) \cdot x[i+1:j] \neq \emptyset$ (because $S \cdot x[i+1:j] = (C \cdot x[i+1:j]) \cup ((S \setminus C) \cdot x[i+1:j])$). Hence the tree of all runs starting from $S \setminus C$ and labelled by $x$ is infinite, thus by Kruskal’s lemma it has an infinite branch, i.e. there exists a state $p \in S \setminus C$ and an infinite run starting in $p$ and labelled by $x$. Finally $C \cup \{p\} \in P$, which contradicts the maximality of $|C|$.

Using this result, one can define $\text{cover}_I(S)$ to be the smallest set for $\prec$ among the elements of $\text{Comp}(S)$ whose future completely covers the future of $S$ as quickly as possible. Formally we have Definition 1.3 (which makes sense by Lemma 1.2).

Definition 1.3 (Time and cover). Let $x \in \text{Dom}(f)$, $i \geq 0$ and $q^x_i \in S \subseteq Q$, we define:

- $\text{time}_I(S) := \min \{ j \geq i : \exists C \in \text{Comp}(S) \text{ such that } C \cdot x[i+1:j] = S \cdot x[i+1:j] \}$
Continuous rational functions are deterministic regular

\[ \text{cover}_i^x (S) \] the minimal element for \( \prec \) in the (non-empty) set:
\[ \{ C \in \text{Comp}(S) : C \cdot x[i+1] \cdot \text{time}_i^x (S) = S \cdot x[i+1] \cdot \text{time}_i^x (S) \} \] .

We now define the sequence \( \text{Good}_i^x \) \( i \in \mathbb{N} \) which is obtained by applying successively the functions \( \text{cover}_i^x \) \( i \) when starting from \( I \).

- \[ \text{Definition I.4} \] (Good). Let \( x \in \text{Dom}(f) \), the sequence \( \text{Good}_i^x \) \( i \in \mathbb{N} \) is defined by:
  - \( \text{Good}_0 = \text{cover}_0^x (I) \);
  - for \( i \geq 0 \), \( \text{Good}_{i+1}^x = \text{cover}_{i+1}^x (\text{Good}_i^x \cdot x[i+1]) \).

- \[ \text{Lemma I.5} \] For all \( i \geq 0 \), \( \text{Good}_i^x \) is well defined, \( \text{Good}_i^x \subseteq I \cdot x[1:i] \) and \( q_i^x \in \text{Good}_i^x \).

\[ \text{Proof.} \] The result is shown by induction on \( i \geq 0 \). Assume that it holds for \( i \geq 0 \) (the base case is very similar) then \( q_{i+1}^x \in \text{Good}_{i+1}^x \cdot x[i+1] \) since \( q_i^x \in \text{Good}_i^x \). Therefore \( \text{Good}_{i+1}^x \) is well defined by Definition \[ \text{I.3} \] . Furthermore, by induction hypothesis and definition of \( \text{cover}_{i+1}^x \) we have \( \text{Good}_{i+1}^x \subseteq \text{Good}_i^x \cdot x[i+1] \subseteq I \cdot x[1:i] \cdot x[i+1] = I \cdot x[i+1] \).

It remains to show that \( q_{i+1}^x \in \text{Good}_{i+1}^x \). Indeed, by definition of \( \text{cover}_{i+1} \) there exists \( j \geq i+1 \) such that \( \text{Good}_{i+1}^x \cdot x[i+2:j] = \text{Good}_{i+1}^x \cdot x[i+2:j] \). But we necessarily have \( q_{i}^x \in \text{Good}_i^x \cdot x[i+1] \cdot x[i+2:j] \) because \( q_{i+1}^x \in \text{Good}_{i+1}^x \cdot x[i+1] \). Therefore we can find a run of \( T \) of the form \( p_{i+1} \rightarrow p_{i+2} \rightarrow \cdots \rightarrow p_j \) labelled by \( x[i+2:j] \) where \( p_{i+1} \in \text{Good}_{i+1}^x \) and \( p_j = q_j^x \). Since \( \text{Good}_{i+1}^x \subseteq I \cdot x[1:i+1] \) and \( T \) is unambiguous, one has \( p_{i+1} = q_{i+1}^x \). \[ \text{Remark I.6} \] Since \( \text{Good}_i^x \subseteq I \cdot x[1:i] \) and \( \text{Good}_{i+1}^x \subseteq \text{Good}_i^x \cdot x[i+1] \) and because the transducer \( T \) is trim and unambiguous, we have that \( \text{Good}_i^x, x[i], \text{Good}_{i+1}^x \) is a pre-step.

In order to show Lemma \[ \text{4.16} \] we prove that the sequence \( \text{(Good)_i^x} \) can be computed by a restricted 1-nT.

- \[ \text{Proposition I.7} \] One can build a restricted 1-nT \( A \) computing some \( f : A^\omega \rightarrow (A \psi \text{Comp})^\omega \) such that if \( x \in \text{Dom}(f) \), then \( f'(x) = \text{Good}_i^x \cdot x[1] \cdot \text{Good}_i^x \cdot x[2] \cdots \).

\[ \text{Proof of Lemma I.6} \] Immediate by definition of \( \text{(Good)_i^x} \) and Lemma I.5 .

I.2 Description of the restricted 1-nT

We now describe how the restricted 1-nTA of Proposition I.7 is built.

\[ \text{States.} \] A state of \( A \) is a tuple \( (S, C, S) \) where:
  - \( S \in \text{Parts} \) and \( C \in \text{Comp}(S) \) (hint: we want \( C \) to be \( \text{Good}_i^x \) after reading \( x[1:i] \));
  - \( S \) is a set of tuples \( (S^h, C^h, T, g) \) where:
    - \( S^h \in \text{Parts} \), \( C^h \in \text{Comp}(S^h) \) and \( T \in \text{Parts} \);
    - \( g : \text{Comp}(S^h) \rightarrow 2^T \) (hint: it will store some "history" about the \( C \) and \( S \) visited).

- \[ \text{Definition I.8} \] (Indicator function of compatible subsets). Let \( S \in \text{Parts} \), we define the function \( \chi_S : \text{Comp}(S) \rightarrow 2^S, X \mapsto \{ X \} \).

The initial states of \( A \) are those of the form \( (I, C, \{(I, C, I, \chi_I)\}) \) for \( C \in \text{Comp}(I) \). Intuitively, these \( C \in \text{Comp}(I) \) describe the possible candidates for \( \text{Good}_0^x = \text{cover}_0^x (I) \).
Transitions. Let us now describe formally the transitions. If \( a \in A \), there is an \( a \)-labelled transition from \((S_1,C_1,\delta_1)\) to \((S_2,C_2,\delta_2)\) if the following conditions hold:
1. \( S_2 = C_1 \cdot a \);
2. \( \delta_2 = \{(S^h,C^h,T \cdot a,g \cdot a) : (S^h,C^h,T,g) \in \delta_1\} \cup \{(S_2,C_2,\xi_{S_2})\} \) where:
   \[ g \cdot a : \text{Comp}(S^h) \rightarrow 2^{T \cdot a}, X \mapsto g(X) \cdot a; \]
3. furthermore we add two restrictions which aim at “forcing” the choice of \( \text{Good}^x \):
   a. if \( S_1 \in \text{Comp} \) then \( C_1 = S_1 \);
   b. if \((S^h,C^h,T,g) \in \delta_1\) with \( g^{-1}(T) = \emptyset \) but \( (g \cdot a)^{-1}(T \cdot a) \neq \emptyset \), then \( C^h \) is the minimal element for \( x \) in \((g \cdot a)^{-1}(T \cdot a) \neq \emptyset \).

Output. The word produced on an \( a \)-transition coming in \((S,C,\delta)\) is \( aC \in (A \uplus \text{Comp})^* \). We add a specific production for the initial states (which does not modify the construction).

1.3 Correctness of the construction

Since the output only describes the \( C \) visited, it remains to show the following result.

\[ \text{Lemma I.9.} \text{ Let } x \in \text{Dom}(f), \text{ then } A \text{ has only one final run labelled by } x. \text{ It is of the form } (S_0, \text{Good}_0, \delta_0) \rightarrow (S_1, \text{Good}_1, \delta_1) \rightarrow \cdots \text{ for some } S_0, S_1, \ldots \text{ and } \delta_0, \delta_1, \ldots. \]

The rest of this subsection is devoted to the proof of this result. We first give an extension of the \( g \cdot a \) defined to act on a function as in condition 2 of the transitions of \( A \).

\[ \text{Definition I.10 (Monoid action over functions). Let } g : \text{Comp}(S^h) \rightarrow 2^T \text{ and } u \in A^*, \text{ then we define } g \cdot u : \text{Comp}(S^h) \rightarrow 2^{T \cdot u}, X \mapsto X \cdot u. \]

\[ \text{Claim I.11.} \text{ If } u \in A^* \text{ and } a \in A \text{ then } g \cdot u \cdot a = g \cdot u a. \text{ Furthermore } g \cdot \epsilon = g. \]

We now show that the \( \delta_i \) along an accepting run store some “history” about the \( S_i \) and \( C_i \).

\[ \text{Sublemma I.12.} \text{ Let } (S_0,C_0,\delta_0) \rightarrow (S_1,C_1,\delta_1) \rightarrow \cdots \text{ be a accepting run of } A \text{ labelled by } x \in A^\omega, \text{ then for all } n \geq 0: \]

\[ \begin{align*}
   &S_0 = I \text{ and } S_{n+1} = C_n \cdot x[n+1]; \\
   &\forall n \geq 0, C_n \in \text{Comp}(S_n); \\
   &\forall n \geq 0 \delta_n = \{(S_i,C_i,S_i \cdot x[i+1:n],\chi_{S_i} \cdot x[i+1:n]) : 0 \leq i < n\}. \\
\end{align*} \]

\[ \text{Proof.} \text{ The result is shown by induction on } n \geq 0. \text{ The base case follows from the definition of initial states. Assume that it holds for some } n \geq 0, \text{ then } S_{n+1} = S_n \cdot a \text{ by condition } 1 \text{ on transitions, } C_{n+1} \in \text{Comp}(S_{n+1}) \text{ by definition of states, and finally:} \]

\[ \delta_{n+1} = \{(S_i,C_i,S_i \cdot x[i+1:n+1],\chi_{S_i} \cdot x[i+1:n+1]) : 0 \leq i < n\} \]

\[ \cup \{(S_{n+1},C_{n+1},S_{n+1} \cdot \epsilon,\chi_{S_{n+1}} \cdot \epsilon)\}. \]

by condition 2 on transitions and Claim I.11.

Given \( x \in \text{Dom}(f) \), we now completely describe the unique accepting run of \( A \) labelled by \( x \). Let \( (S^x_n)_{n \in \text{Parts}} \) defined by \( S^x_0 := I \) and for \( n \geq 0 \) let \( S^x_{n+1} := \text{Good}^x \cdot x[n+1] \). Furthermore for \( n \geq 0 \) we define \( \delta^x_n := \{(S^x_i,\text{Good}^x,S^x_i \cdot x[i+1:n],\chi_{S^x_i} \cdot x[i+1:n]) : 0 \leq i < n\}. \)

\[ \text{Sublemma I.13.} \text{ Given } x \in \text{Dom}(f), \text{ then } \rho^x_s = (S^x_0,\text{Good}^x,\delta^x_0) \rightarrow (S^x_1,\text{Good}^x,\delta^x_1) \rightarrow \cdots \text{ is an accepting run of } A \text{ labelled by } x. \]
Proof. Each \((S^x_n, \text{Good}^x_n, \delta^x_n)\) is a state since \(\text{Good}^x_n \in \text{Comp}(S^x_n)\). The first state is initial since \(S^x_0 = I\) and \(\text{Good}^x_0 \in \text{Comp}(I)\). \(\delta^x_0 = \{I, \text{Good}^x_0, I, \chi_1\}\). Condition [1] holds since \(S^x_{n+1} = \text{Good}^x_n \cdot x[n+1]\) for \(n \geq 0\). Condition [2] holds, since by definition and Claim [11]
\[
S^x_{n+1} = \{ (S^x_i, \text{Good}^x_i, S^x_{i+1}): (S^x_i, \text{Good}^x_i, S^x_{i+1}) \text{ is a state} \}
\]

Now let us show that given \(n \geq 0\), conditions [3a] and [3b] for transitions hold between \((S^x_n, \text{Good}^x_n, \delta^x_n)\) and \((S^x_{n+1}, \text{Good}^x_{n+1}, \delta^x_{n+1})\). Indeed:
- if \(S^x_n \in \text{Comp}\), then \(S^x_n \in \text{Comp}(S^x_n)\) such that \(C \cdot x = S^x_n \cdot x\). Hence \(\text{time}^0_n = n\) and \(\text{Good}^x_n = \text{cover}^x_n(S^x_n) = S^x_n\). Thus condition [3a] holds;
- assume that \(\exists n \geq i\) such that \((X^x_i \cdot x[i+1:n])^{-1}(S^x_i \cdot x[i+1:n]) = \emptyset\) and such that:
\[
(X^x_i \cdot x[i+1:n])^{-1}(S^x_i \cdot x[i+1:n]) \neq \emptyset
\]
By definition of \(g \cdot u\) and \(\chi\) we get for \(j \geq i\):
\[
(X^x_i \cdot x[i+1:j])^{-1}(X^x_i \cdot x[i+1:n]) = \{ Y \subseteq \text{Comp}(X^x_i) : X \cdot x[i+1:j] = Y \}
\]
therefore for \(Y = S^x_i \cdot x[i+1:j]\) we have:
\[
(X^x_i \cdot x[i+1:j])^{-1}(S^x_i \cdot x[i+1:j]) = \{ C \in \text{Comp}(S^x_i) : C \cdot x[i+1:j] = S^x_i \cdot x[i+1:j] \}
\]
Hence our hypotheses yield \(\text{time}^x_i(S^x_i) = n+1\) since \(\text{Definition 1.3}\) holds and furthermore \(\text{Good}^x_{n+1}\) is the minimal element for \(\preceq\) in the set \((X^x_i \cdot x[i+1:n])^{-1}(S^x_i \cdot x[i+1:n+1])\). Therefore the run is accepting.

**Sublemma 1.14.** Given \(x \in \text{Dom}(f)\), the run \(\rho_x^i\) is the unique accepting run labelled by \(x\).

Proof. Let us consider an accepting run \(\rho = (S_0, C_0, \delta_0) \rightarrow (S_1, C_1, \delta_1) \rightarrow \cdots\) and suppose that it coincides with \(\rho_x^i\) until the state \((S_{i-1}, C_{i-1}, \delta_{i-1})\). Then one has \(S_i = S^x_i\) since:
- either \(i = 0\) and \((S_0, C_0, \delta_0)\) is initial, thus \(S_0 = I = S^0_0\);
- or \(i > 1\) and by Sublemma 1.12 we have \(S_i = C_{i-1} \cdot x[i] = \text{Good}^x_{i-1} \cdot x[i] = S^x_i\).

Furthermore \(C_i \in \text{Comp}(S^x_i)\). Now let \(n := \text{time}^x_i(S^x_i)\), two cases may occur:
- either \(n = i\) which means that \(S^x_i \in \text{Comp}(S^x_i)\) hence \(\text{Good}^x_i = \text{cover}^x_i(S^x_i) = S^x_i\). By condition [3a] on transitions (and since there is an outgoing transition from the state \((S_i, C_i, \delta_i)\)) we conclude that \(C_i = S^x_i = \text{Good}^x_i\);
- or \(n > i\). Hence by Equation 3 \((X^x_i \cdot x[i+1:n-1])^{-1}(S^x_i \cdot x[i+1:n-1]) = \emptyset\) and \(\text{Good}^x_i\) is the smallest element for \(<\circ (X^x_i \cdot x[i+1:n])^{-1}(S^x_i \cdot x[i+1:n]) \neq \emptyset\). But by Sublemma 1.12 we have \((S^x_i, C_i, S^x_{i+1} \cdot x[i+1:n]) \in \delta_{i+1}\). Considering the transition from \(n-1\) to \(n\), we must have by condition [3a] that \(C_i = \text{Good}^x_i\) and finally \(\delta_i = \delta^x_i\) by Sublemma 1.12
max-lag) and one of them is empty. Invariants 4b and 4d are obvious due to emptiness of the nb\textsubscript{x} and out\textsubscript{x}. For Invariant 4c we use Remark 4.14. Invariant 4e holds by definition of lag(q) and emptiness of the nb\textsubscript{x} and out\textsubscript{x}. For Invariant 4f we use Lemma 4.13 which gives for all step C, u, D and q ∈ D, \(prod^{z[1:i]}_{J,C}(q) ⊆ (adv^{z[1:i]}_{J,C}(pre^{z[1:i]}_{J,D}(q)))^{-1}πθ\). Therefore by adding \(prod^{z[1:i]}_{J,C}(pre^{z[1:i]}_{J,D}(q))\) on both sides we get \(prod^{z[1:i]}_{J,D}(q) ⊆ com^{z[1:i]}_{J,C}(q)π\theta\). We conclude because max-lag = τ. Finally invariant 4g follows since all paths π ≠ C are close.

### J.2 Proof of Lemma 5.4

Invariant 3 is preserved along the operation, since we never modify J or C. It is clear that invariant 4f holds before using the function down(C). Indeed, we do not modify the lag(q), thus the lagging states are the same, and furthermore we do not create non-close paths.

It is clear that Algorithm 1 is well defined since its recursive calls follow the definition of tree(C) (in a strictly decreasing way). For all π ∈ tree(C), let \(n_π\) (resp. \(o_π\)) be the value of nb\textsubscript{x} (resp. out\textsubscript{x}) before launching the function down(C).

#### ▶ Sublemma J.1

Let π = C\textsubscript{1} · · · C\textsubscript{n} ∈ tree(C). During the execution of down(C), the following invariants hold just before down(π) makes its recursive calls:

1. for all π′ = C\textsubscript{1} · · · C\textsubscript{n}′ ⊆ π, nb\textsubscript{x} : C\textsubscript{u} → \{0:2\};
2. for all C′ ∈ Comp(C\textsubscript{n}), C′ ≠ C\textsubscript{n} and all πC′ ⊆ π′, we have nb\textsubscript{x} = n\textsubscript{πC′} and out\textsubscript{x} = o\textsubscript{πC′};
3. if down(π) has performed the update nb\textsubscript{x}C′(q) ← nb\textsubscript{x}C′(q) + (nb\textsubscript{x}(q) - 4), for some C′ and q ∈ C\textsubscript{n}, then for all π′ ⊆ π, we have nb\textsubscript{x}(q) = 2.
4. if q ∈ C was lagging before launching down(C), then for all π′ = C\textsubscript{1}′ · · · C\textsubscript{n}′ ∈ tree(C) such that q ∈ C\textsubscript{n′}, we have nb\textsubscript{x}(q) = 0 and, if π′ ≠ C, out\textsubscript{x} = ε;
5. for all π ⊆ π′ = C\textsubscript{1} · · · C\textsubscript{n}′ ∈ tree(C) such that C\textsubscript{n′} = \{q\}, if π\textsubscript{i} denotes C\textsubscript{1} · · · C\textsubscript{i} then:

\[θ^{nb\textsubscript{x}} \left( \prod_{i=2}^{n′} out\textsubscript{x} \right) \equiv θ^{n\textsubscript{π}} \left( \prod_{i=2}^{n′} o\textsubscript{x} \right) \]

6. for all π′ = C\textsubscript{1} · · · C\textsubscript{n}′ ⊆ π which is not close and was close before launching down(C), let J′ := pre\textsuperscript{z[1:i]}\textsubscript{J,C}(C\textsubscript{n′}) ⊆ J. Then J′, π[1:i]:C\textsubscript{n′} is an initial step and | max-adv\textsuperscript{z[1:i]}\textsubscript{J,C} | ≥ 4Ω!

**Proof.** Invariant 3 is clear since step 2 forces nb\textsubscript{x} = 2 if nb\textsubscript{x} > 2. Invariants 4b, 4c, and 4d result from a simple but bureaucratic verification. For invariant 4e, note that if nb\textsubscript{x}C′(q) ← nb\textsubscript{x}C′(q) + (nb\textsubscript{x}(q) - 2) is performed, then we had nb\textsubscript{x}(q) = 2 (so now nb\textsubscript{x}(q) = 2) before executing down(C). Therefore using invariants 4b and 4d, we conclude that we had nb\textsubscript{x}(q) = 4 for all π′ ⊆ π before lauching down(π).

Let us show that invariant 4f is preserved. First, if π′ ⊆ π is not close before down(π) makes its recursive calls, then it was not close before launching down(π). Assume now that π is not close, but was close before launching down(π). By invariant 4f, we conclude that π was close before launching down(π). Hence it means that down(π) performed a nb\textsubscript{x}C′(q) ← nb\textsubscript{x}C′(q) + (nb\textsubscript{x}(q) - 2). Hence for all π′ ⊆ π we have nb\textsubscript{x}(q) = 2 by invariant 4e. Before executing the first “for” loop of down(π), we had nb\textsubscript{x}(q′) = 0 for some q′ ∈ C\textsubscript{0}, and thus nb\textsubscript{x}(q′)(q′) = nb\textsubscript{x}(q′) = 0 after this loop (by construction and since π was close). Let π\textsubscript{i} := C\textsubscript{i} · · · C\textsubscript{i}, for 1 ≤ i ≤ n. From invariant 4e of S, and invariants 4f and 4g we get:

\[prod^{z[1:i]}_{J,C}(q) = \text{out-lag} \ θ^{nb\textsubscript{x}}(\prod_{i=2}^{n} out\textsubscript{x} \ θ^{nb\textsubscript{x}}) \ θ^{nb\textsubscript{x}}(q) \ θ^{nb\textsubscript{x}}(\prod_{i=2}^{n} out\textsubscript{x} \ θ^{nb\textsubscript{x}}) \]
Continuous rational functions are deterministic regular

Therefore we get:

\[
\prod_{i=2}^{n} \text{out}_{s_{i}} \theta^{n_{b_{s_{i}}}(q')} \bigg|_{\prod_{i=1}^{n} \text{out}_{s_{i}} \theta^{n_{b_{s_{i}}}(q')} \theta^{n_{b_{s_{i}}}(q')} \text{last}(q')} \]

which concludes the proof.

For invariant 4d, let us consider a lagging state \( s \). Thus we use invariants 4a, 4e and 4f to show that the \( \text{last}(q) \) is clear that invariants 1 and 2 hold after this step. We now use \( J \), \( \text{lag} \), etc. to denote the values stored before the operation, and \( J \), \( \text{lag} \) etc. (with a bar) after the operation.

**Invariants 4a, 4b and 4c** To show invariant 4a we note that \( k_{q} \neq 0 \) if and only if \( \text{pre}_{\mathcal{C}_{i+1}}^{2}(q) \) was lagging. Thus we use invariants 4a, 4e and 4f to show that the \( \text{lag} \) are prefixes of \( \text{max-lag} \). For invariant 4b it is sufficient to see that the \( \text{out}_{\mathcal{C}} \) are obtained from the \( \text{out}_{\mathcal{C}} \). We just have to note that \( \text{lag} \) is not used anywhere except for \( \text{lag} \). Indeed (with the notations of Subsubsection 5.3.1), if \( \rho = \mathcal{C} \) and \( m = n \), then \( i_{m} = i_{1} = 1 = n \) and by definition of \( \text{tree}(\mathcal{C}) \) we have \( \pi = \mathcal{C} \). For invariant 4e we use invariants 4d and 4f which imply that for all \( q \in \mathcal{T} \):

- if \( \text{pre}_{\mathcal{C}_{i+1}}^{2}(q) \) was lagging then \( \prod_{i=1}^{n} \mathcal{O}_{\mathcal{C}}(q) \subseteq (\text{lag}(\text{pre}_{\mathcal{C}_{i+1}}^{2}(q)))^{-1} \text{max-lag} \theta^{\omega} \);
- otherwise, \( \prod_{i=1}^{n} \mathcal{O}_{\mathcal{C}}(q) \subseteq (\text{last}(\text{pre}_{\mathcal{C}_{i+1}}^{2}(q)))^{-1} \theta^{\omega} \).

Thus \( \prod_{i=1}^{n} \mathcal{O}_{\mathcal{C}_{i+1}}(q)[k_{q}+1] \subseteq (\text{last}(\text{pre}_{\mathcal{C}_{i+1}}^{2}(q)))^{-1} \theta^{\omega} \) by invariant 4d, and so \( \text{last}(q) \subseteq \theta^{\omega} \).

**Invariant 4d** For invariant 4d let us consider a lagging state \( q \in \mathcal{T} \). Then \( \text{pre}_{\mathcal{C}_{i+1}}^{2}(q) \) was also lagging and \( |\prod_{i=1}^{n} \mathcal{O}_{\mathcal{C}_{i+1}}(q)| < k_{q} \). Thus \( \text{last}(q) = \varepsilon \). Now, let \( \pi = D_{1} \cdots D_{n} \in \text{tree}(\mathcal{C}_{i+1}) \) be such that \( q \in D_{n} \), with the notations of Subsubsection 5.3.1 we have:

- either \( i_{m} < n \), then \( n_{\text{lag}} = 0 \) and \( \text{lag} \); or \( i_{m} = n \). Since \( C_{i_{m}} = \mathcal{C} = \text{pre}_{\mathcal{C}_{i+1}}^{2}(D_{n}) \), we get \( \text{pre}_{\mathcal{C}_{i+1}}^{2}(q) \in C_{i_{m}} \) and so \( n_{\text{lag}} = 0 \).

Furthermore, if \( D_{n} \neq \mathcal{T} \), then (see before) \( C_{i_{m}} = \mathcal{C} \neq C \) thus \( \text{lag} = \varepsilon \).
Invariant 4e. The proof is similar to that of invariant 4d. The important result is that given \( \pi = D_1 \cdots D_n \in \text{tree}(C) \) such that \( D_n = \{ q \} \), then (with the notations of Subsubsection 5.3.1)

if \( \pi_i := D_1 \cdots D_i \) for \( 1 \leq i \leq n \) and \( \rho_j := C_{i+1} \cdots C_j \) for \( 1 \leq j \leq m \), we get:

- \( \text{out}_{\pi_i} = \varepsilon \) and \( \text{nb}_{\pi_i} = 0 \) if \( i \notin \{ i_1, \ldots, i_m \} \);
- \( \text{out}_{\pi_{i,j}} = \text{out}_{\rho_j} \) for \( 1 < j \leq m \) and \( \text{nb}_{\pi_{i,j}} = \text{nb}_{\rho_j} \odot \text{pre}_{C,\pi}^e \) if \( 1 \leq j \leq m \)

Hence \( \theta_{\text{nb},v_i}(q) \prod_{j=2}^n \text{out}_{\pi_j} \theta_{\text{nb},v_i}(q) = \theta_{\text{nb},v_i}(\text{pre}_{C,\pi}(q)) \prod_{j=2}^m \text{out}_{\rho_j} \theta_{\text{nb},v_i}(\text{pre}_{C,\pi}(q)) \).

Invariant 4f. Let \( C, u, D \) be a step, then \( C, au, D \) is also a step. Therefore we apply invariant 4f at the previous stage and the result follows since \( \text{out}_{\max-lag} \theta^\pi = \text{out}_{\max-lag} \theta^\sigma \).

J.4 Proof of Lemma 5.7

Let us show that invariants 2 and 13 hold after this operation. We use J, out, etc. to denote the values stored before the operation, and J, out, etc. (with a bar) to denote them after. The case of invariant 2 is trivial because \( C \in \text{Comp} \) and \( J = \text{pre}_{I,C}(\pi C) \). Furthermore, \( C \) is separable by invariants 14 and Lemma 14.3.

Invariants 4a to 4d. The \( \text{lag}(q) \) for \( q \in C' \) are prefixes of \( \text{max-lag} \) and furthermore \( \text{lag}(q) = \varepsilon \) for some \( q \in C' \), because of the definition of \( \varepsilon \). Thus invariant 4a holds. Invariant 4b holds because the \( \text{out}_{\pi} \) for \( \pi \notin C \) are obtained from the \( \text{out} \) for \( \pi \notin C \). Invariant 4c is also trivial.

Since \( C' = \text{Comp} \) was not close, there was some \( p \in C' \) which was not lagging. Therefore some \( q \in C' \) is now lagging if and only if it was lagging before the operation. Let us consider such a \( q \in C' \) if it exists, and let \( \pi = C_1 \cdots C_n \in \text{tree}(C) \) be such that \( q \in C_n \). The update gives \( \text{nb}_{C_n} = \text{nb}_{C'} \) which must be null since \( q \) was lagging. The cases of \( \text{out}_{\pi} \) (for \( \pi \notin C \)) and \( \text{last}(q) \) are similar. Thus invariant 4d holds.

Invariant 4e. Let us now show invariant 4e. Two cases occur given \( q \in C \):

- either \( q \in C \) is lagging, then it was lagging before and so we had \( \text{prod}^I_{x_1}(C)(q) = \text{out lag}(q) \).
- further, since \( q \in C' \) was lagging we necessarily had \( \text{out}_{\pi}(q) = \varepsilon \) by invariant 4d

Since the update gives \( \text{out} = \text{out}_{\pi} \text{out}_{C,\pi} \) and \( \text{lag}(q) = e^{-1}(\text{lag}(q)) \), we conclude that \( \text{out lag}(q) = \text{out lag}(q) \);

or \( q \in C \) is not lagging, let \( \pi = C_1 \cdots C_n \in \text{tree}(C) \) be such that \( C_n = \{ q \} \). Then let \( \pi := C_1 \cdots C_n \in \text{tree}(C) \), if \( \pi_i := C_1 \cdots C_i \) we had:

\[
\text{prod}^I_{x_1}(C)(q) = \text{out max-lag} \theta_{\text{nb},C}(q) \left( \prod_{i=1}^n \text{out}_{\pi_i} \theta_{\text{nb},C}(q) \right) \text{last}(q).
\]

Now if \( \pi_{i} := C_1 \cdots C_i \), the update gives \( \text{out}_{\text{pi}} = \text{out}_{\pi_{i}} \) and \( \text{nb}_{\text{pi}} = \text{nb}_{\pi_{i}} \), if \( i > 1 \); and \( \text{out} \leftarrow \text{out}_{\pi_{i}} \) and \( \text{nb}_{\text{pi}} \leftarrow \text{nb}_{\pi_{i}} \) for \( i = 1 \). Therefore we have:

\[
\text{prod}^I_{x_1}(C)(q) = \text{out max-lag} \left( \prod_{i=2}^n \text{out}_{\pi_i} \theta_{\text{nb},C}(q) \right) \theta_{\text{nb},C}(q) \text{last}(q).
\]
The result follows by update of last(q) and since max-lag = c−1 max-lag.

**Invariant 4g.** Let \( \mathcal{T}, u, D \) be a step, we show that for all \( q \in D \): \( \prod_{J,D}^{\delta}[1:i]u(q) \subseteq \text{out} \max-lag \theta^\omega \).

By invariant 4g we can decompose the step \( J, x[1:j], \mathcal{T} \) in an initial step \( J, x[1:j], E \) and a step \( E, x[j+1:], \mathcal{T} \) such that \( |\max-\text{adv}^{[1:j]}_{J,E}| \geq 2\Omega! \). By applying Lemma 4.13 to \( J, x[1:j], E \), it follows that there exists \( \varphi, \tau, \gamma \) with \( |\varphi| = 2\Omega! \) and \( |\tau| \leq 2\Omega! \) such that \( \max-\text{adv}^{[1:j]}_{J,E} = \tau \gamma \) (thus \( |\gamma| = |\max-\text{adv}^{[1:j]}_{J,E}| - |\tau| \geq 2\Omega! - 2\Omega! \geq 2\Omega! \)). \( \gamma \subseteq \varphi^\omega \) and for all step \( E, x[j+1:]u, D \) such that:

\[
\prod_{E,D}^{[j+1:j]+1}u(q) \subseteq (\text{adv}^{[1:j]}_{J,E}(\prod_{E,D}^{[j+1:j]+1}u(q)))^{-1} \tau \varphi^\omega.
\]

and therefore by adding \( \prod_{J,E}^{[1:j]+1}(\prod_{E,D}^{[j+1:j]}u(q)) \) on both sides:

\[
\prod_{J,D}^{[1:j]+1}u(q) \subseteq \min^{[1:j]}_{J,D}(\tau \varphi^\omega).
\]

To conclude, it is thus sufficient to show the following result:

**Sublemma J.2.** \( \min^{[1:j]}_{J,D}(\tau \varphi^\omega) = \text{out} \max-lag \theta^\omega \).

**Proof.** Since \( C \mathcal{T} \in \text{tree}(C) \) was not close, there was some \( \pi \neq \varepsilon \) such that \( C \mathcal{T}, \pi \in \text{tree}(C) \) and \( \text{nb}_\pi \neq 0 \) or \( \text{out}_\pi \neq \varepsilon \). Therefore there is \( q \in \mathcal{T} \) which was not lagging and such that:

\[
\prod_{J,C}^{[1:j]}(q) = \prod_{J,C}^{[1:j]}(q) = \text{out} \max-lag \theta^n(q) \text{last}(q).
\]

for some \( n(q) > 0 \). Let us consider the \( q \in \mathcal{T} \) such that \( (n(q), |\text{last}(q)|) \) is maximal. Then for all \( p \in \mathcal{T} \) we get \( \prod_{J,C}^{[1:j]}(p) \subseteq \prod_{J,C}^{[1:j]}(q) \), thus (use Equation 4 for the right handside):

\[
\min^{[1:j]}_{J,E}(\tau \gamma) = \min^{[1:j]}_{J,E}(\max-\text{adv}^{[1:j]}_{F,E}) \subseteq \text{out} \max-lag \theta^n(\text{last}(q)) \subseteq \min^{[1:j]}_{J,D}(\tau \varphi^\omega).
\]

Two cases can occur depending on the sign of \( k := |\text{out} \max-lag| - |\min^{[1:j]}_{J,D}(\tau \gamma)| \):

- either \( k \geq 0 \), then \( \text{out} \max-lag = \min^{[1:j]}_{J,D}(\tau(\varphi^\omega[1:k])) \) and \( \theta = \varphi^\omega[k+1:k+1+\Omega!] \);
- or \( k < 0 \), then \( \text{out} \max-lag(\theta^n(\text{last}(q))[k+1:]) = \min^{[1:j]}_{J,D}(\tau \gamma) \) and \( \gamma \subseteq \theta^n(\text{last}(q))[k+1:] \).

But since \( |\gamma| \geq 2\Omega! \) and \( \gamma \subseteq \varphi^\omega \), we conclude that \( \varphi = \theta^n[k+1:k+1+\Omega!] \).

In both cases, we conclude that \( \min^{[1:j]}_{J,D}(\tau \varphi^\omega) = \text{out} \max-lag \theta^\omega \).

**Invariant 4g.** Let us consider \( \pi = C_1 \cdots C_n \in \text{tree}(\mathcal{T}) \), \( \pi \) not close after the operation. Then there exists \( \pi \subseteq \pi' = C_1 \cdots C_n' \in \text{tree}(\mathcal{T}) \) such that \( \text{nb}_{\pi'} \neq 0 \) or \( \text{out}_{\pi'} \neq \varepsilon \). Let \( \pi' = C_1 \cdots C_n' \), then \( \mathcal{T} \subseteq \pi' \), and by the updates we get \( \text{nb}_{\pi'} = \text{nb}_{\pi} \) and \( \text{out}_{\pi'} = \text{out}_{\pi} \).

Finally, since invariant 4g held before the operation, it still holds.

**K** Proofs of section 5.4: boundedness and productivity of \( S \)

For \( x \in A^\omega \) and \( i \geq 0 \), we denote by \( \lceil \text{out}_x \rceil^*_i \), etc. the values of the registers of \( S \) after reading \( C_0^x \cdots x[i]C_i^x \) (when defined).
K.1 Proof of Lemma 5.9: 1-boundedness of $S$

Given $\pi \in \text{tree}(C_i^x)$, we say that $[\text{out}_i]_{i'}^x$ is 1-bounded if for all $0 \leq i' \leq i$ and $\pi' \in \text{tree}(C_i^x)$, $\text{out}_{i'}$ occurs at most once in $\sigma(\text{out}_i)$, where $\sigma$ is the substitution applied by $\mathcal{S}$ when reading $x[i'+1]C_{i'+1} \cdots C_i^x$. Given $\pi, \rho \in \text{tree}(C_i^x)$, we say that $[\text{out}_i]_{i'}^x$ and $[\text{out}_i]_{i'}^x$ have no shared memory if for all $0 \leq i' \leq i$ and $\pi' \in \text{tree}(C_i^x)$, $\text{out}_{i'}$ does not occur both in $\sigma(\text{out}_i)$ and $\sigma(\text{out}_\rho)$. Lemma 5.9 immediately follows from Sublemma K.1.

Sublemma K.1. For all $\pi \in \text{tree}(C_i^x)$, $[\text{out}_i]_{i'}^x$ is 1-bounded. Furthermore, if $\rho \in \text{tree}(C_i^x)$ is such that $\pi \subset \rho$, then $[\text{out}_i]_{i'}^x$ and $[\text{out}_i]_{i'}^x$ have no shared memory.

The rest of this subsection is devoted to the proof of Sublemma K.1 by induction on $i \geq 0$. The result is obvious for $i = 0$. Assume now by induction that it holds for some $i \geq 0$.

First note that Subsection 5.2 only adds constant values in the register, hence its applications will always preserve our property. Now, if $C_i^x$ was separable, the transition of $\mathcal{S}$ uses Subsection 5.1. The value $[\text{out}_i]_{i+1}^x$ is obtained using $[\text{out}_i]^x$ once, plus constant values. Furthermore, if $C_{i+1}^x$ is not separable, then each value $[\text{out}_i]_{i+1}^x$ is built from constant values, hence they are 1-bounded and they share no memory. The result holds in $i+1$.

Now if $C_{i+1}^x$ is not separable, the transition may first apply Subsubsection 5.3.2. If $\pi$ is close, then the situation is similar to that of Subsection 5.1 except that we may use $\text{out}_\mathcal{C}^C$ to update $\text{out}$. However, since $[\text{out}_\mathcal{C}^C]_{i+1}^x$ and $[\text{out}_i]_{i+1}^x$ are 1-bounded and have no shared memory by induction hypothesis, then the resulting value of $\text{out}$ is 1-bounded. Now if $\pi$ is not close, the argument for $\text{out}$ is similar. Furthermore, we update $\text{out}_{\mathcal{C}^C} \leftarrow \text{out}_\mathcal{C}^C \cdot \pi$ for $\pi \neq \varepsilon$, which clearly preserves the fact that those registers are 1-bounded and have no shared memory. Furthermore, they also have no shared memory with $[\text{out}_i]_{i+1}^x$.

Let us finally consider the application of Subsubsection 5.3.1. The updates clearly preserve 1-boundedness since there are no concatenations. Let $\pi_n := D_1 \cdots D_n \in \text{tree}(C_{i+1}^x)$, we show that if $\pi_{n'} := D_1 \cdots D_{n'}$ for $n' \leq n$, then $[\text{out}_{\pi_n}]_{i+1}^x$ and $[\text{out}_{\pi_{n'}}]_{i+1}^x$ have no shared memory. Let $1 = i_1 < \cdots < i_m \leq n$ be given by Subsubsection 5.3.1 for $\pi_n$. If $i_m < n$ the result is clear since $\text{out}_{\pi_n} \leftarrow \varepsilon$ (and we may finally add constant values in it by Subsection 5.2). Otherwise $\text{out}_{\pi_n} \leftarrow \text{out}_{C_{i_1} \cdots C_{i_m}}$. If $n' \neq i_m$ for some $1 \leq m' \leq m$, then the result is clear since $\text{out}_{\pi_{n'}} \leftarrow \varepsilon$. Otherwise $\text{out}_{\pi_{n'}} \leftarrow \text{out}_{C_{i_1} \cdots C_{i_m}}$. But necessarily $m' < m$ (because $n' < n$) and so by induction hypothesis the former values of $\text{out}_{C_{i_1} \cdots C_{i_m}}$ and $\text{out}_{C_{i_1} \cdots C_{i_m}}$ shared no memory. The result follows since Subsection 5.2 only adds constant values.

K.2 Proof of Lemma 5.10: productivity of $\mathcal{S}$

Let us fix a word $x \in \text{Dom}(f)$, we want to show that when $\mathcal{S}$ reads $g(x) = C_0^n x[1] \cdots$, we have $\|\text{out}_i^x\| \to \infty$. Let us first suppose that the transitions of $\mathcal{S}$ on $g(x)$ use Subsection 5.1 or the “close” paragraph of Subsubsection 5.3.1 infinitely often. Then for infinitely many $i \geq 0$ we have $\|\text{out}_i^x\| = \text{size}(i, \mathcal{I})$, and the $\alpha_q = \text{adv}_{1,J}^{\mathcal{C},2}(q)$ have a size bounded by $\Omega + 2 \times 2\Omega$. Since $\text{prod}_{1,J}^{\mathcal{C}}(q_i^x)$ tends to $f(x)$, we conclude that $\text{out}$ also tends to an infinite word.

Now assume that there exists $N \geq 0$ such that when reading the suffix of its input $C_N^x x[N]C_{N+1}^x x[N+1] \cdots$, $\mathcal{S}$ only uses Subsubsection 5.3.2 in the “non-close” case, and Subsubsection 5.3.1 when doing its transitions. The rest of the proof is done by contradiction: we assume that $\mathcal{S}$ only adds empty words in $\text{out}$ when performing those transitions.

A key ingredient to reach a contradiction will be the fact that $\mathcal{T}$ is productive.

Sublemma K.2 (Productivity). Let $J, u, S$ be an initial step and $S, u', S$ be a step such that $u' \neq \varepsilon$, $\text{pre}_{S,S}^J(u') : S \to S$ is the identity function, and some $q'_1 \in S$ is accepting. Then $\text{prod}_{S,S}^J(q'_1) \neq \varepsilon$ for all $q \in S$. 

O. Carton and G. Douéneau-Tabot XX:39 MFCS 2022
Continuous rational functions are deterministic regular

Proof. Let \( q'_2 \in S \). If it is accepting the result follows since a productive 1-nT is clean. Otherwise, by definition of steps there exists (uniques) \( q_1, q_2 \in I \), \( \alpha_1, \alpha_2 \in B^* \) such that \( \nu(\alpha_1, q_1) = \nu(\alpha_2, q_2) \) and \( q'_i \) for \( i \in \{1, 2\} \). These are the conditions of Lemma 4.3, hence since \( T \) is productive we get \( \prod_{x, S, S}(q'_2) \neq \epsilon \).

▶ Definition K.3. For all \( i \geq 0 \), let \( S_i^x := \bigcap_{i' \geq i} \text{pre}_{C_i^1, C_i^0}(C_{i'}^x) \).

▶ Sublemma K.4. For all \( i \geq 0 \), \( q_i^x \in S_i^x \) and \( S_i^x, x[i+1], S_{i+1}^x \) is a step.

Proof. For all \( i' \geq i \), we have \( q_i^x = \text{pre}_{C_i^1, C_i^0}(q_{i'}^x) \) and \( q_i^x \in C_i^x \). Hence \( q_i^x \in S_i^x \). Let us now show that \( p \in S_i^x \) if and only if there exists a sequence \( (p_{i'})_{i' \geq i} \) such that \( p_i = p, p_{i'} \in C_{i'}^x \) and \( p_{i'} = \text{pre}_{C_{i'}^1, C_{i'}^0}(p_{i'+1}) \) for all \( i' \geq i \). The “if” direction is obvious. Conversely, if \( p \in S_i^x \), then for all \( n \geq i \), there exists a finite sequence \( (p_{i'})_{i' \leq i' \leq n} \) such that \( p_i = p, p_{i'} \in C_{i'}^x \) and \( p_{i'} = \text{pre}_{C_{i'}^1, C_{i'}^0}(p_{i'+1}) \). By König’s lemma (see \( \text{pre} \) as the ancestor relation in a tree), we can build an infinite sequence \( (p_{i'})_{i' \geq i} \). From this characterization, it follows that \( S_i^x \in \text{Comp} \), and \( \text{pre}_{C_i^1, C_i^0}(S_{i+1}^x) = S_i^x \), which implies that \( S_i^x, x[i+1], S_{i+1}^x \) is a step.

We claim that the \( S_i^x \) completely “cover” the set \( C_i^x \) at some point in the future.

▶ Sublemma K.5. For all \( i \geq 0 \), there exists \( i' \geq i \) such that \( \text{pre}_{C_i^1, C_i^0}(C_{i'}^x) = S_i^x \).

Proof. Since \( C_i^x, x[i+1], C_{i+1}^x \) is a pre-step, \( \text{pre}_{C_i^1, C_i^0}(C_{i'}^x) \supseteq \text{pre}_{C_i^1, C_i^0}(C_{i'}^{x[i+1]}) \). Hence \( (\text{pre}_{C_i^1, C_i^0}(C_{i'}^x))_{i' \geq i} \) is ultimately constant, and by Definition K.3 its limit is \( S_i^x \).

Finally, let us extract a sequence of positions where \( S_i^x \) is constant.

▶ Sublemma K.6. There exists a sequence \( n \leq \ell_1 < \ell_2 \cdots \) such that \( S_{\ell_1}^x = S_{\ell_2}^x = \cdots = S \), \( q_{\ell_1}^x = q_{\ell_2}^x = \cdots \in F \) and \( \text{pre}_{S, S}(x[\ell_1+1, \ell_2]) \) is the identity function for all \( j \leq \ell_2 \).

Proof. Since \( (q_i^x)_{i \geq 0} \) is accepting, one can extract an infinite sequence \( n \leq \ell_1 < \ell_2 \cdots \) such that \( q_{\ell_1}^x = q_{\ell_2}^x = \cdots \in F \). Up to extracting a subsequence with Ramsey’s theorem for singletons (i.e. the pigeonhole principle), we can assume that \( S_{\ell_1}^x = S_{\ell_2}^x = \cdots = S \). Up to extracting a subsequence using Ramsey’s theorem for pairs (color a pair \( j \leq j' \) by \( \text{pre}_{S, S}(x[\ell_1+1, \ell_2]) \), which is a permutation of \( S \), \( \text{pre}_{S, S}(x[\ell_1+1, \ell_2]) \), a step), we can assume that \( \text{pre}_{S, S}(x[\ell_1+1, \ell_2]) \) for \( j \leq j' \) is the identity function.

By sublemmas K.2 and K.6, we get \( \prod_{x, \ell_1, C_{\ell_1}^0}(q) = \prod_{x, S, S}(q) \neq \epsilon \) for all \( j \geq 1 \) and all \( q \in S \). Therefore \( \prod_{x, \ell_1, C_{\ell_1}^0}(q) \geq K \) for all \( K \geq 1 \) and \( q \in S \).

Let us now fix \( K = 2\Omega! \), \( i = \ell_1 \) and \( j' := \ell_1 + K \). By Sublemma K.5, there exists \( i'' \geq i \) such that \( \text{pre}_{C_i^1, C_{i'}^0}(C_{i'}^x) = S_i^x \). Hence for all \( q \in C_i^x \), if \( q' := \text{pre}_{C_i^1, C_{i'}^0}(C_{i'}^x) \) we get \( \prod_{x, C_{i'}^0, C_i^0}(q') \geq \prod_{x, S, S}(q) \geq 2\Omega! \).

The last operation which was applied by \( S \) when reading from \( x[i'' \cdots] \) is Algorithm 1. By definition of \( n \) in down(\( C_0^x \)), there exists \( q \in C_0^x \) such that \( [\text{nb}_{C_0^x}(q)]_{x}^x = 0 \). Thus we can apply Sublemma K.7 with \( i_1 = i \) and \( i_2 = i'' \), and Remark K.8 yields a contradiction.

▶ Sublemma K.7. Let \( n \leq i_1 \leq i_2 \). Let \( q_2 \in C_1^x \) be such that \( [\text{nb}_{C_1^x}(q_2)]_{x}^x = 0 \). Let us define \( q_1 := \text{pre}_{C_1^1, C_1^0}(q_2) \), then \( [\text{nb}_{C_1^x}(q_1)]_{x}^x = 0 \) and:

\[
\prod_{x, i_1, C_{i_1}^0}(q_2) = \|\text{last}\|_i^x(q_2) - \|\text{last}\|_i^x(q_1) + \|\text{lag}\|_i^x(q_2) - \|\text{lag}\|_i^x(q_1).
\]
Remark K.8. In particular, we get $\prod_{C_{i_1}^{x}, C_{i_2}^{x}}^{(x_{i_1}+1,x_{i_2})}(q_2) < 2\Omega!$ because the size of lag is bounded by $\Omega!$ and the size of last is (strictly) bounded by $\Omega!$.

Proof. The proof consists in a decreasing induction on $i_1 \leq i_2$. The base case being trivial, let us show it for $i_0 := i_1 - 1$. Let $q_2 \in C_{i_2}^{x}$ be such that $[nb_{C_{i_2}^{x}}(q_2)]_{i_2}^x = 0$ and $q_1 := \text{pre}_{C_{i_1}^{x}, C_{i_2}^{x}}^{(x_{i_1}+1,x_{i_2})}(q_2)$. By induction hypothesis, Sublemma K.7 holds. Now let us consider the transition of $S$ from $C_{i_2}^{x}$ to $C_{i_1}^{x}$. It obtained by possibly applying Subsubsection 5.3.2 (in the “non-close” case) and then Subsubsection 5.3.1. We study the preservation of our property along these operations, starting from the last one (we backtrack on the computation).

Last operation: applying Subsection \[5.2\] in Subsubsection \[5.3.1\] Let $nb'_{\supset}, \text{out}'_{\supset}, \text{etc.}$ denote the configuration of $S$ right after Lemma \[5.7\] in Subsubsection \[5.3.1\]. Since Algorithm \[1\] has not modified out = out$_{C_{i_2}^{x}}$, we had $m = 0$ in down($C_{i_1}^{x}$). Thus we had $\text{nb}_{C_{i_2}^{x}}(q_1) + n(q_1) = 0$ (because if $m = 0$ then $[\text{nb}_{C_{i_2}^{x}}]_{i_2}^x(q_1) = \min(\text{nb}_{C_{i_2}^{x}}(q_1) + n(q_1), 2)$).

Therefore $\text{nb}_{C_{i_2}^{x}}(q_1) = 0$. Furthermore $n(q_1) = 0$ and so last$(q_1) = [\text{last}(q_1)]_{i_1}^x$. Furthermore, lag $(q_1) = [\text{lag}(q_1)]_{i_1}^x$.

Previous operation: beginning of Subsubsection \[5.3.1\] Now let $nb''_{\supset}, \text{out''}_{\supset}, \text{etc.}$ denote the configuration of $S$ before applying the whole Subsection \[5.3.1\]. Since by construction $C = \text{pre}_{C_{i_2}^{x}, C_{i_1}^{x}}^{(x_{i_1}, x_{i_2})}(C_{i_1}^{x})$, then $nb'_{C_{i_1}^{x}} = \text{nb}''_{C_{i_1}^{x}} \circ \text{pre}_{C_{i_1}^{x}, C_{i_2}^{x}}^{(x_{i_1}, x_{i_2})}$. Thus $\text{nb}_{C_{i_1}^{x}}(q_0) = 0$ if $q_0 := \text{pre}_{C_{i_1}^{x}, C_{i_2}^{x}}^{(x_{i_1}, x_{i_2})}(q_1)$.

Finally, we note that $c = \varepsilon$ since there is no output. As a consequence, it is quite easy to see that $|\prod_{C_{i_1}^{x}, C_{i_2}^{x}}^{(x_{i_1}, x_{i_2})}(q_1)| = |\prod_{C_{i_1}^{x}, C_{i_2}^{x}}^{(x_{i_1}, x_{i_2})}(q_1)| = |\text{last}(q_1)| - |\text{last''}(q_0)| + |\text{lag'}(q_1)| - |\text{lag''}(q_0)|$.

Previous operation: Subsubsection \[5.3.2\] If Subsubsection \[5.3.2\] was not used, the proof is completed. Otherwise, let $nb''_{\supset}, \text{out''}_{\supset}, \text{etc.}$ denote the information of $S$ right after Lemma \[5.7\]. By an analysis of Algorithm \[1\] (similar to what we did above), we see that $\text{nb}_{C_{i_1}^{x}}(q_0) = 0$, last''$(q_0) = \text{last'}(q_0)$ and lag''$(q_0) = \text{lag''}(q_0)$. Furthermore $n(q_0) = 0$ thus |last''$(q_0)$| < $\Omega!$.

Let us finally consider the rest of Subsubsection \[5.3.2\] in the “non-close” case. Since there is no output, $c = \varepsilon$ thus $[\text{lag''}]_{i_0}^x(q_0) = \text{lag''}(q_0)$. Since |last''$(q_0)$| < $\Omega!$, then $\theta^{[\text{nb}_{C_{i_2}^{x}}]_{i_0}^x(q_0)} = \varepsilon$ thus $[\text{nb}_{C_{i_2}^{x}}]_{i_0}^x(q_0) = 0$. Furthermore $[\text{last'}]_{i_0}^x(q_0) = \text{last'}(q_0)$.