

Continuous rational functions are deterministic regular

Olivier Carton ✉ 

IRIF, Université Paris Cité

Gaëtan Douéneau-Tabot ✉

IRIF, Université Paris Cité & Direction générale de l'armement - Ingénierie de projets

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...

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, i.e. which 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 easy to see that any rational function is regular, but the converse does not hold.

Infinite words. The class of *regular functions over infinite words* was defined in [2] using monadic second-order transductions. It coincides with the class of functions realized by 2-dT with ω -regular lookahead, or by copyless dSST with some Müller conditions. However, the use of ω -regular lookaheads (or Müller conditions for dSST) is necessary to capture the expressive power of monadic second-order logic on infinite words, in order to check properties such as infinitely many occurrences of some pattern. Similarly, the model of 1-nT with Büchi acceptance conditions defines the subclass of *rational functions over infinite words*.



© Olivier Carton and Gaëtan Douéneau-Tabot;
licensed under Creative Commons License CC-BY 4.0

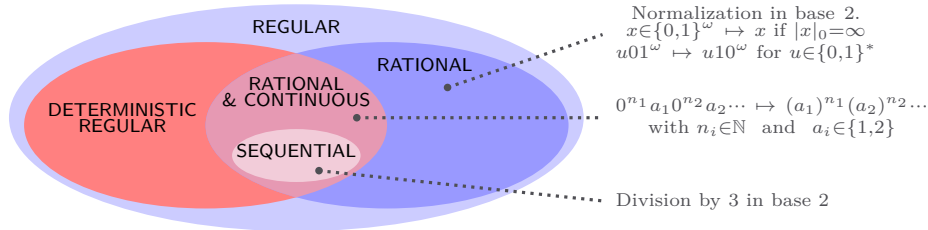
Even if regular and rational functions give very natural frameworks for specification (due to their connections with logic), not all these functions can effectively be computed by a deterministic machine without lookaheads. It turns out that the regular functions which can be computed by a deterministic 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 was already 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 computed by a 2-dT (without lookaheads). Since the converse also holds, this result completely characterizes rational functions which can be computed by 2-dTs. 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 of partial functions over infinite words studied in this paper

Then we introduce various new concepts in order to transform a 1-nT computing a continuous function into a dSST. This determinization procedure is rather involved. The main difficulty is that even if the 1-nT is unambiguous, it might not check its guesses after reading only a finite number of letters. In other words, a given input can label several infinite runs, even if only one of them is accepting. However, a deterministic machine can never determine which run is the accepting one, since it requires to check whether a property occurs infinitely often. This intuition motivates our key definition of *compatible sets* among the states of a 1-nT. Such sets are the sets of states from 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^ω) denotes the set of finite words (resp. non-empty finite words, infinite words) over the alphabet A . If $u \in A^* \cup A^\omega$, we let $|u| \in \mathbb{N} \cup \{\infty\}$ be its length. For $a \in A$, $|u|_a$ denotes the number of a in u . For $1 \leq i \leq |u|$, $u[i] \in A$ is the i -th letter of u . If $1 \leq i \leq j \leq |u|$, $u[i:j]$ stands for $u[i]u[i+1] \cdots$ until j . We write $u[i:]$ for $u[i:|u|]$. If $j > |u|$ we let $u[i:j] := u[i:|u|]$. If $j < i$ we let $u[i:j] := \varepsilon$. We write $u \sqsubseteq v$ (resp. $u \subset v$) 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 $f : S \rightarrow T$. 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) $\mathcal{T} = (A, B, Q, I, F, \Delta, \lambda)$ is:

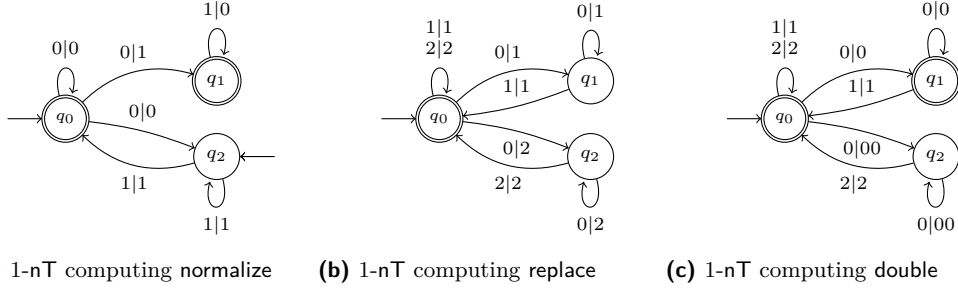
- a finite input (respectively output) alphabet A (respectively B);
- a finite set of states Q with $I \subseteq Q$ initial and $F \subseteq Q$ final;
- a transition relation $\Delta \subseteq Q \times A \times Q$;
- an output function $\lambda : \Delta \rightarrow B^*$ (defined for each transition).

We write $q \xrightarrow{a|\alpha} q'$ whenever $(q, a, q') \in \Delta$ and $\lambda(q, a, q') = \alpha$. A *run* labelled by some $x \in A^* \cup A^\omega$ is a sequence of consecutive transitions $\rho := q_0 \xrightarrow{x[1]|\alpha_1} q_1 \xrightarrow{x[2]|\alpha_2} q_2 \cdots$. The *output* of ρ is the word $\alpha_1\alpha_2 \cdots \in A^* \cup A^\omega$. If $x \in A^\omega$, we also write $q_0 \xrightarrow{x|\alpha_1\alpha_2 \cdots} \infty$ to denote an infinite run starting in q_0 . The run ρ is *initial* if $q_0 \in I$, *final* if $x \in A^\omega$ and $q_i \in F$ infinitely often (Büchi condition), and *accepting* if both initial and final. \mathcal{T} computes the *relation* $\{(x, y) : y \in B^\omega \text{ is output along an accepting run on } x\}$. It is *functional* if this relation is a (partial) function. In this case, \mathcal{T} can be transformed in an equivalent *unambiguous* 1-nT (a transducer which has at most one accepting run on each $x \in A^\omega$) [3, Corollary 3]. A function $f : A^\omega \rightarrow B^\omega$ 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** : $\{0, 1\}^\omega \rightarrow \{0, 1\}^\omega$ mapping $x \mapsto x$ if $|x|_0 = \infty$ and $u01^\omega \mapsto u10^\omega$ if $u \in \{0, 1\}^*$;
- **replace** : $\{0, 1, 2\}^\omega \rightarrow \{1, 2\}^\omega$ with $\text{Dom}(\text{replace}) = \{x : |x|_1 = \infty \text{ or } |x|_2 = \infty\}$ and mapping $0^{n_1}a_10^{n_2}a_2 \cdots \mapsto a_1^{n_1+1}a_2^{n_2+1} \cdots$ if $a_i \in \{1, 2\}$, $n_i \in \mathbb{N}$;
- **double** : $\{0, 1, 2\}^\omega \rightarrow \{0, 1, 2\}^\omega$ mapping $0^{n_1}a_10^{n_2}a_2 \cdots \mapsto 0^{a_1n_1}a_10^{a_2n_2}a_2 \cdots$ and $0^{n_1}a_1 \cdots 0^{n_m}a_m0^\omega \mapsto 0^{a_1n_1}a_1 \cdots 0^{a_mn_m}a_m0^\omega$ (if finitely many 1 or 2).

XX:4 Continuous rational functions are deterministic regular



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

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

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

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

3 Deterministic regular functions

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

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

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

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

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

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

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

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

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

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

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

- a finite input (resp. output) alphabet A (resp. B);
- a finite set of states Q with $q_0 \in Q$ initial;
- a transition function $\delta : Q \times A \rightarrow Q$;
- a finite set of registers \mathfrak{R} with a distinguished output register **out** $\in \mathfrak{R}$;
- an update function $\lambda : Q \times A \rightarrow \mathcal{S}_{\mathfrak{R}}^B$ 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 **out** in $\{\lambda(q, a)(\mathfrak{r}) : \mathfrak{r} \in \mathfrak{R}\}$.

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

This machine defines a function $f : A^\omega \rightarrow B^\omega$ as follows. For $i \geq 0$ let $q_i^x := \delta(q_0, x[1:i])$ (when defined). For $i \geq 1$, we let $\lambda_i^x := \lambda(q_{i-1}^x, x[i])$ (when defined) and $\lambda_0^x(\mathfrak{r}) = \varepsilon$ for all $\mathfrak{r} \in \mathfrak{R}$. For $i \geq 0$, define the substitution $\llbracket \cdot \rrbracket_i^x := \lambda_0^x \circ \cdots \circ \lambda_i^x$. By construction we get $\llbracket \text{out} \rrbracket_i^x \subseteq \llbracket \text{out} \rrbracket_{i+1}^x$ (when defined). We thus let $f(x) := \bigvee_i \llbracket \text{out} \rrbracket_i^x$, if $\llbracket \text{out} \rrbracket_i^x$ is defined for all $i \geq 0$ and $\|\llbracket \text{out} \rrbracket_i^x\| \rightarrow +\infty$. Otherwise $f(x)$ is undefined.

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

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

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

The proof of the next result is quite involved, but it is largely inspired by the techniques used for regular functions over finite or infinite words (see e.g. [5, 7]).

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

1. two-way deterministic 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.

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 3.8.** *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 3.9.** *A restricted 1-nT is a 1-nT whose states all are final.*

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

► **Theorem 3.10.** *Given a restricted 1-nT computing a function $f : A^\omega \rightarrow B^\omega$ and a deterministic regular function $g : B^\omega \rightarrow C^\omega$, $g \circ f$ is (effectively) deterministic regular.*

4 Continuous rational functions are deterministic regular

We now state the main result of this paper, which shows that a rational function can be extended to a deterministic regular function. Using an extension of the original function is necessary since not all ω -regular languages are Büchi deterministic (see Proposition 3.8). 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 $f : A^\omega \rightarrow B^\omega$ is *continuous* if and only if for all $x \in \text{Dom}(f)$ and $n \geq 0$, there exists $p \geq 0$ such that $\forall y \in \text{Dom}(f)$, $|x \wedge y| \geq p \Rightarrow |f(x) \wedge f(y)| \geq n$.

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

► **Theorem 4.2.** *Given a continuous rational function $f : A^\omega \rightarrow B^\omega$, 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 3.10 to show that f' can be computed as a composition of a restricted 1-nT and a 1-bounded dSST (see Subsection 4.2).

4.1 Properties of continuous rational functions

We first describe some structural properties of 1-nT computing continuous functions. In this subsection, we let $\mathcal{T} = (A, B, Q, I, F, \Delta, \lambda)$ be an unambiguous, clean and trim 1-nT computing a continuous function $f : A^\omega \rightarrow B^\omega$. It is well known that \mathcal{T} verifies Lemma 4.3. This property is in fact equivalent to the continuity of f (see e.g. [11] or [6, Theorem 3]).

► **Lemma 4.3.** *For all $q_1, q_2 \in I$, $q'_1 \in F$, $q'_2 \in Q$, $u \in A^*$, $u' \in A^+$, $\alpha_1, \alpha'_1, \alpha_2, \alpha'_2 \in B^*$ such that $q_i \xrightarrow{u|\alpha_i} q'_i \xrightarrow{u'|\alpha'_i} q'_i$ for $i \in \{1, 2\}$ we have (note that $\alpha'_1 \neq \varepsilon$ since \mathcal{T} is clean):*

- if $\alpha'_2 \neq \varepsilon$, then $\alpha_1 \alpha'_1{}^\omega = \alpha_2 \alpha'_2{}^\omega$;
- if $\alpha'_2 = \varepsilon$, $x \in A^\omega$, $\beta \in B^\omega$ and $q'_2 \xrightarrow{x|\beta} \infty$ is final, then $\alpha_1 \alpha'_1{}^\omega = \alpha_2 \beta$;

Empty cycles $q \xrightarrow{u|\varepsilon} q$ for $q \notin F$ cannot be avoided in a 1-nT. However, we shall see in Lemma 4.4 that such cycles can be avoided in the context of Lemma 4.3. Formally, we say that the clean \mathcal{T} is *productive* if the hypotheses of Lemma 4.3 imply $\alpha'_2 \neq \varepsilon$.

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

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

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

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

Let Comp be the set of compatible sets. If $S \subseteq Q$, let $\text{Comp}(S)$ 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 $\text{pre}_{C,D}^u(q) \in C$ such that $\text{pre}_{C,D}^u(q) \xrightarrow{u} q$.*

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

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

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

► **Lemma 4.8.** *Let J, u, C be an initial step. Then $\text{prod}_{J,C}^u(q)$ 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 $\text{com}_{J,C}^u \in B^*$ as the longest common prefix $\bigwedge_{q \in C} \text{prod}_{J,C}^u(q)$;
- for all $q \in C$, its advance $\text{adv}_{J,C}^u(q) \in B^*$ as $(\text{com}_{J,C}^u)^{-1} \text{prod}_{J,C}^u(q)$;
- the maximal advance $\text{max-adv}_{J,C}^u$ as the longest advance, i.e. $\bigvee_{q \in C} \text{adv}_{J,C}^u(q)$.

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

► **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 $|\text{adv}_{J,C}^u(p)| - |\text{adv}_{J,C}^u(q)| > \Omega$.*

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

► **Lemma 4.12** (Looping futures). *Let $C \in \text{Comp}$ be separable and J, u, C be an initial step (not necessarily the one which makes C separable). There exists $\tau, \theta \in B^*$ with $|\tau| \leq 3\Omega$, and $|\theta| = \Omega!$ which can be uniquely determined from C and $\text{adv}_{J,C}^u(q)$ for $q \in C$, such that:*

- $\tau \sqsubseteq \text{max-adv}_{J,C}^u \sqsubseteq \tau\theta^\omega$;

■ for all step C, v, D and $q \in D$, $\text{prod}_{C,D}^v(q) \sqsubseteq (\text{adv}_{J,C}^u(p))^{-1} \tau \theta^\omega$ with $p := \text{pre}_{C,D}^v(q)$.

► Remark 4.13. Since $\text{adv}_{J,C}^u(p) \sqsubseteq \text{max-adv}_{J,C}^u \sqsubseteq \tau \theta^\omega$, the second item makes sense.

► Example 4.14. In Figure 2c, the compatible set $C := \{q_1, q_2\}$ is separable. For all step C, v, D we have $D = C$ thus $v = 0^n$, $\text{prod}_{C,D}^{0^n}(q_1) = 0^n$ and $\text{prod}_{C,D}^{0^n}(q_2) = 0^{2n}$.

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

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

► Lemma 4.15. One can build a restricted 1-nT \mathcal{N} computing $g : A^\omega \rightarrow (\text{Comp} \uplus A)^\omega$ such that $\text{Dom}(f) \subseteq \text{Dom}(g)$, and for all $x \in \text{Dom}(g)$, $g(x) = C_0 x[1] C_1 x[2] C_2 \dots$ where:

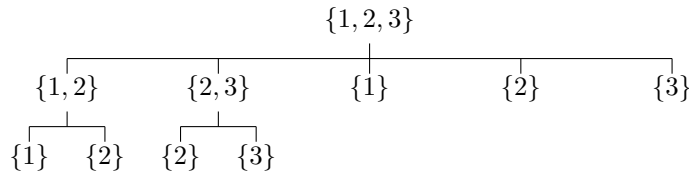
- $C_0 \subseteq I$ and for all $i \geq 0$, $C_i, x[i+1], C_{i+1}$ is a pre-step;
- if $x \in \text{Dom}(f)$ then $\forall i \geq 0$, $q_i^x \in C_i$, where $q_0^x \xrightarrow{x[1]} q_1^x \xrightarrow{x[2]} \dots$ is the accepting run of \mathcal{T} .

Given $x \in \text{Dom}(f)$, we denote by C_0^x, C_1^x, \dots the sequence of compatibles produced by \mathcal{N} in Lemma 4.15. We now describe a 1-bounded dSST \mathcal{S} which, when given as input $g(x) \in (\text{Comp} \uplus A)^\omega$ 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 \subset . 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 4.17.

► Definition 4.16 (Tree of compatibles). Given $C \in \text{Comp}$, we denote by $\text{tree}(C)$ the set of words $\pi = C_1 \dots C_n \in (\text{Comp}(C))^+$ such that $C_1 = C$ and for all $1 \leq i \leq n-1$, $C_i \supset C_{i+1}$.

► Example 4.17. If $C = \{1, 2, 3\}$ and $\text{Comp}(C) = \{\{1, 2, 3\}, \{1, 2\}, \{2, 3\}, \{1\}, \{2\}, \{3\}\}$, then we have $\text{tree}(C) = \{\{1, 2, 3\}\{1, 2\}\{1\}, \{1, 2, 3\}\{1, 2\}\{2\}, \{1, 2, 3\}\{2, 3\}\{2\}, \{1, 2, 3\}\{2, 3\}\{3\}, \{1, 2, 3\}\{1\}, \{1, 2, 3\}\{2\}, \{1, 2, 3\}\{3\}\}$. Its view as a tree is depicted in Figure 3.



■ Figure 3 The tree of compatibles obtained from Example 4.17.

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

- the content of a register **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 $|\text{lag}(q)| \leq 3\Omega$ for all $q \in C$ (stored in the state).

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

- a value $\theta \in B^*$ with $|\theta| = \Omega!$ (stored in the state);

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

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

Invariants. The main idea for building \mathcal{S} is the following. If C_i^x is a non-separable set, then the productions along the initial runs which end in C_i^x are mutual prefixes (by Lemma 4.8) which only differ from a bounded information. Hence the common part com of these runs is stored to out , and the adv are stored in the lag . If C_i^x becomes separable, then these runs still produce mutual prefixes, but two of them can differ from a large information. However by Lemma 4.12, they iterate some value θ . Hence the only relevant information is the number of θ which were produced along these runs. Formally, our construction ensures that the following invariants hold when \mathcal{S} has just read $C_0^x x[1]C_1^x \cdots x[i]C_i^x$ for $i \geq 0$:

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

5 Description of the 1-bounded dSST for Subsection 4.2

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

Let us first deal with the initialization of \mathcal{S} . When reading the first letter C_0^x of $g(x)$, \mathcal{S} stores $J \leftarrow C_0^x$, $\mathcal{C} \leftarrow C_0^x$ and $\text{lag}(q) \leftarrow \varepsilon$ for all $q \in C_0^x$. This is enough if C_0^x is not separable. Otherwise, we let θ be given by Lemma 4.12 (applied to the initial simulation $C_0^x, \varepsilon, C_0^x$), $\text{nb}_\pi(q) \leftarrow 0$ and $\text{out}_\pi \leftarrow \varepsilon$ for all $\pi = C_1 \cdots C_n \in \text{tree}(C_0^x)$ and all $q \in C_n$.

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

XX:10 Continuous rational functions are deterministic regular

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

5.1 If C_i^x was not separable

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

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

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

- if C is not separable, then \mathcal{S} stays in non-separable mode and it updates $\text{lag}(q) \leftarrow \alpha_q$ for all $q \in C$ (note that $|\alpha_q| \leq \Omega \leq 3\Omega$). We easily see that invariants 1, 2 and 3 hold.
- if C is separable, \mathcal{S} goes to separable mode. By applying Lemma 4.12 to $J, x[1:i+1], C$ we get $\tau \in B^*$ with $k := |\tau| \leq 3\Omega$. We update $\text{lag}(q) \leftarrow \alpha_q[1:k]$ and $\text{last}(q) \leftarrow \alpha_q[k+1:]$ for all $q \in C$. The θ is given by Lemma 4.12, and we 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 $\text{out}_\pi = \text{out}$ when $\pi = C = C_{i+1}^x$) and all $q \in C_n$.

► **Lemma 5.3.** *Invariants 1, 2 and 4 hold in $i+1$ after this operation. Furthermore $|\theta| = \Omega!$, $|\text{lag}(q)| \leq 3\Omega$ for all $q \in C$, and for all $\pi = C_1 \cdots C_n \in \text{tree}(C)$, $\text{nb}_\pi = 0$.*

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 $\text{last}(q)$ into the $\text{nb}_\pi(q)$ and out_π).

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

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

From invariant 4c, there exists $n : C \rightarrow \mathbb{N}$ such that $\text{last}(q) = \theta^{n(q)} \delta_q$ with $\delta_q \sqsubset \theta$ 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 $|\text{last}(q)| < \Omega!$ and $\text{nb}_\pi(q) \leq 4$ when $\pi \neq C$.

In order to reduce the value nb_C , we apply the function **down**(C) of Algorithm 1 which adds some θ 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 $\text{max-lag} = \varepsilon$ by invariant 4a. With the notations of invariant 4e (note that $\pi_1 = C$), we get $\text{prod}_{x[1:i]}^{J, C}(q) = \text{out} \theta^{\text{nb}_{\pi_1}(q)} \left(\prod_{i=2}^n \text{out}_{\pi_i} \theta^{\text{nb}_{\pi_i}(q)} \right) \text{last}(q)$ for all $q \in C$. Thus we can produce in out the value $\theta^m := \bigwedge_{q \in C} \theta^{\text{nb}_C(q)}$ (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 2 and 4 hold, and furthermore we have $|\theta| = \Omega!$, $|\text{lag}(q)| \leq 3\Omega$ and $|\text{last}(q)| < \Omega!$ for all $q \in C$, and for all $\pi = C_1 \cdots C_n \in \text{tree}(C)$, $\text{nb}_\pi : C_n \rightarrow [0:4]$.*

5.3 If C_i^x was separable

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

■ **Algorithm 1** Sending down values in $\text{tree}(\mathbf{C})$

```

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

```

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

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

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

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

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

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

Now let $c := \bigwedge_{q \in \mathbf{C}} \text{lag}(q)$. We update $\text{lag}(q) \leftarrow c^{-1} \text{lag}(q)$ for all $q \in \mathbf{C}$ (therefore max-lag becomes $c^{-1} \text{max-lag}$), $\text{out} \leftarrow \text{out} \cdot c$, $\mathbf{C} \leftarrow C_{i+1}^x$ and finally $\text{pre} \leftarrow \text{pre} \circ \text{pre}_{\mathbf{C}, C_{i+1}^x}^a$.

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

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

5.3.2 Preprocessing when \mathbf{C}, a, C_{i+1}^x is not a step

In the current subsection we assume that invariants 1, 2 and 4 hold in $i \geq 0$, that $\mathbf{C} = C_i^x$ is separable, and that C_i^x, a, C_{i+1}^x is *not* a step. Then let $C' := \text{pre}_{C_i^x, C_{i+1}^x}^a(C_{i+1}^x) \subset \mathbf{C}$ (an equality would give a step) and $\pi := \mathbf{C} C' \in \text{tree}(\mathbf{C})$. Two cases can occur.

XX:12 Continuous rational functions are deterministic regular

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

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

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

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

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

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

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

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

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

► **Remark 5.8.** Contrary to the former cases, the main difficulty here is to show the preservation of invariant 4f. For this we essentially rely on invariant 4g and show that θ is still suitable. Again, we may have $|\text{last}(q)| \geq \Omega!$. Thus we finally apply Subsection 5.2 once more.

5.4 Boundedness and productivity of the construction

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

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

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

► **Lemma 5.10.** *If $x \in \text{Dom}(f)$, then $|\text{out}| \rightarrow \infty$ when \mathcal{S} reads $g(x)$.*

6 Outlook

This paper provides a solution to an open problem. From a practical point of view, it allows to build a copyless streaming algorithm from a rational specification whenever it is possible (it is impossible when the rational function is not continuous). We conjecture that the techniques introduced in this paper can be extended to show that any continuous *regular* function is deterministic regular. Furthermore, they may also be used to study the rational or regular functions which are uniformly continuous for the Cantor topology, and capture them with a specific transducer model (another open problem of [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 Jákl. 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 Gaëtan Douéneau-Tabot, Emmanuel Filiot, and Paul Gastin. Register transducers are marble transducers. In *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24–28, 2020, Prague, Czech Republic, 2020*.
- 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.
- 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.

A Proof of Remark 2.3

We show in this section that the function `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 `double` is computed by a deterministic one way transducer. There exists $N, M \geq 0$, $u_1, u_2, u_3, u_4, u'_3, u'_4 \in \{0, 1, 2\}^*$ such that for all $n \geq 0$, $\text{double}(0^{M+Nn}10^\omega) = u_1u_2^n u_3u_4^\omega = 0^{M+Nn}10^\omega$ and $\text{double}(0^{M+Nn}20^\omega) = u_1u_2^n u'_3u'_4^\omega = 0^{2(M+Nn)}20^\omega$. A contradiction can easily be deduced.

B Proof of Lemma 2.4

Let $\mathcal{T} = (A, B, Q, I, F, \Delta, \lambda)$ be a trim 1-nT computing $f : A^\omega \rightarrow B^\omega$. By a pumping argument, it is easy to see that \mathcal{T} is clean if and only if it has no $q \xrightarrow{u|\alpha} q$ for some $q \in F$. Assume furthermore that \mathcal{T} is unambiguous. We build a clean and unambiguous (trimming can be done later) 1-nT $\mathcal{T}' = (A, B, Q', I', F', \Delta', \lambda')$ as follows:

- $Q' := (Q \times \{0\}) \uplus (Q \times \{1\})$, $I' := I \times \{1\}$ and $F' := F \times \{1\}$;
- for all $(q, a, q') \in \Delta$ we add:
 - $((q, 1), a, (q', 1)) \in \Delta'$ if $q \notin F$, and then $\lambda'((q, 1), a, (q', 1)) := \lambda(q, a, q')$;
 - $((q, 1), a, (q', 0)) \in \Delta'$ if $q \in F$, and then $\lambda'((q, 1), a, (q', 0)) := \lambda(q, a, q')$;
 - $((q, 0), a, (q', 0)) \in \Delta'$ if $\lambda'((q, 0), a, (q', 0)) := \lambda(q, a, q') = \varepsilon$;
 - $((q, 0), a, (q', 1)) \in \Delta'$ if $\lambda'((q, 0), a, (q', 1)) := \lambda(q, a, q') \neq \varepsilon$.

It is easy to see that \mathcal{T}' is unambiguous, clean, and computes f .

C Proof of Theorem 3.7

C.1 From 2-dTs to 1-bounded dSSTs

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 $Q_\perp := Q \uplus \{\perp\}$. When reaching a position $i \geq 0$ of the input $\vdash x$, the dSST will keep track the following information (see Figure 4):

- the state $\text{first}_i^x \in Q_\perp$ that is “the state of \mathcal{T} the first time it goes to position $i+1$ ”. More formally, first_i^x 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 \perp if such a run does not exist, which implies that $x \notin \text{Dom}(f)$). This information is coded in the state of the dSST;
- the concatenation of the outputs by λ along the run φ_i^x , stored in the register `out`;
- a function $\text{next}_i^x(q) : Q \rightarrow Q_\perp$, 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) (\perp if it does not exist). This information is coded in the state of the dSST;
- the concatenation of the outputs by λ along the run $\rho_i^x(q)$ for all $q \in Q$. This information is stored in a register `outq` (it is empty if $\text{next}_i^x(q) = \perp$).

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

Now assume that $x \notin \text{Dom}(f)$. Then either $\text{first}_i^x = \perp$ 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 $\text{first}_i^x \neq \perp$ for all $i \geq 0$, but the output labels of the φ_i^x do not tend to an infinite word, and therefore we get $\llbracket \text{out} \rrbracket_i^x \not\rightarrow \infty$.

Copyless? One could ask if the resulting dSST is copyless. It is not the case, since a state $p \in Q$ may occur both in $\sigma_i^x(q) \in Q^*$ and $\sigma_i^x(q') \in Q^*$ (which implies that out_p is used both for out_q and $\text{out}_{q'}$). However, in this case it means that either out_q or $\text{out}_{q'}$ stores information which is never used in out (since otherwise, it would induce a looping behavior in the \mathcal{T}).

Bounded copies. Let $(A, B, S, s_0, \delta', \mathfrak{R}, \text{out}, \lambda')$ be the dSST built along the previous paragraphs (this does not modify its semantics). Let us show by Claim C.4 that it is 1-bounded. Recall that λ_i^x is the substitution applied (when defined) when reading $x[i]$ on input $x \in A^\omega$.

▷ **Claim C.4.** Let $x \in A^\omega$, $1 \leq j \leq i$ be such that λ_j^x is defined, then:

1. for all $\mathfrak{r}, \mathfrak{s} \in \mathfrak{R}$, \mathfrak{r} occurs at most once in $\lambda_j^x \circ \dots \circ \lambda_i^x(\mathfrak{s})$;
2. if out_p occurs in $\lambda_j^x \circ \dots \circ \lambda_i^x(\text{out}_q)$, then $\text{next}_i(q) \neq \perp$ and (p, j) occurs in the run $\rho_i^x(q)$;
3. if out_p occurs in $\lambda_j^x \circ \dots \circ \lambda_i^x(\text{out})$, then $\text{first}_i(q) \neq \perp$ and (p, j) occurs in the run φ_i^x .

Proof. We show the three items by induction on $i \geq 1$. The base case for $i = j$ follows from the definition of σ and of the updates. Now consider the induction step from $i \geq 1$ to $i+1$. Item 1 holds for $\mathfrak{r} = \text{out}$ by definition of the updates λ' . Assume that out_p occurs in $\lambda_j^x \circ \dots \circ \lambda_{i+1}^x(\text{out}_q)$, we show simultaneously that items 1 and 2 hold. First note that $\text{next}_{i+1}^x(q) \neq \perp$, since otherwise the update in $x[i+1]$ is $\text{out}_q \leftarrow \varepsilon$. Let $p_1 \dots p_n = \sigma_{i+1}^x(q)$, then there exists $\alpha_0, \dots, \alpha_n \in B^*$ such that $\lambda_{i+1}^x(\text{out}_q) = \alpha_0 \text{out}_{p_1} \dots \text{out}_{p_n} \alpha_n$. Then by Claim C.1 the run $\rho_{i+1}^x(q)$ is obtained by concatenating the $\rho_i^x(p_k)$ for $1 \leq k \leq n$ in a disjoint way. Since $j \leq i+1$, then out_p occurs $\lambda_j^x \dots \lambda_i^x(\text{out}_{p_k})$ for some $1 \leq k \leq n$. then by induction hypothesis (item 2), (p, j) occurs in $\rho_{i+1}^x(q)$. But since there is no loop in $\rho_{i+1}^x(q)$, then out_p only occurs in $\lambda_j^x \dots \lambda_i^x(\text{out}_{p_k})$. By induction hypothesis (item 1) out_p occurs only once in this $\lambda_j^x \dots \lambda_i^x(\text{out}_{p_k})$, and finally out_p occurs only once in $\lambda_j^x \dots \lambda_{i+1}^x(\text{out}_q)$. The proof of item 3 is similar. ◀

C.2 From copyless dSSTs to 2-dTs

Let $\mathcal{T} = (A, B, Q, q_0, \delta, \mathfrak{R}, \text{out}, \lambda)$ be a copyless dSST computing a function $f : A^\omega \rightarrow B^\omega$. 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 $\rho := q_0^x \rightarrow q_1^x \rightarrow \dots$ be the initial run of \mathcal{T} on x (with the convention that $q_i^x = \perp$ if it is undefined). For $i \geq 1$, we also define $\lambda_i^x := \lambda(q_{i-1}^x, x[i])$ the substitution applied when reading $x[i]$. We give in Algorithm 2 a function **substitute** $(\alpha, i, \mathfrak{r})$, producing $\llbracket \alpha \rrbracket_i^x$ when $i \geq 0$ and $\alpha \in (B \uplus \mathfrak{R})^*$ as input (\mathfrak{r} 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 α .

■ **Algorithm 2** Computing recursively the values of a register

```

Function substitute( $u, i, \mathfrak{r}$ )
  /*  $\alpha \in (B \uplus \mathfrak{R})^*$  to be computed,  $i \geq 0$  current position */
  for  $a$  in  $\alpha[1], \dots, \alpha[|\alpha|]$  do
    if  $a \in B$  then
      Output  $a$ ; /* Letter  $a \in B$  is output */
    else
      if  $i > 0$  then
        substitute( $\lambda_i^x(a), i-1, a$ ); /* Recursion on  $\lambda_i^x(a)$  */
      end
    end
  end

Function simulation
  for  $i$  in  $\{1, \dots, \infty\}$  do
    out  $\alpha \leftarrow \lambda_i^x(\text{out})$ ;
    substitute( $\alpha, i, \text{out}$ );
  end

```

▷ **Claim C.5** (Algorithm 2 is correct). Let $x \in A^\omega$ and $i \geq 0$ be such that $q_i^x \neq \perp$. Then for all $\alpha \in (\mathfrak{R} \uplus B)^*$ and $\mathfrak{r} \in \mathfrak{R}$, **substitute**(α, i, \mathfrak{r}) terminates and outputs $\llbracket \alpha \rrbracket_i^x$.

Proof. Immediate by induction on $i \geq 0$. ◀

► **Remark C.6.** We shall assume that Algorithm 2 blocks if $q_i^x = \perp$ (since it happens if and only if λ_i^x is undefined).

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

▷ **Claim C.7.** If $x \in \text{Dom}(f)$, then **simulation** loops infinitely and outputs $f(x)$.

▷ **Claim C.8.** If $x \notin \text{Dom}(f)$, then either **simulation** gets blocked at some point, or it produces a finite output.

Proof. Two cases are possible if $x \notin \text{Dom}(f)$. Either $q_i^x = \perp$ for some $i \geq 0$, and then **simulation** gets blocked before position i . Or $q_i^x \neq \perp$ for all $i \geq 0$, but $\llbracket \text{out} \rrbracket_i^x$ tends to a finite word $w \in B^*$. 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_i^x , hence the state q_{i-1}^x (possibly \perp) when in position $i \geq 1$. 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 [4]). 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 $i \geq 1$ is executed by moving right on the input. At each position $i \geq 1$ the 2-dT determines the value $\alpha \in (\mathfrak{R} \uplus B)^*$ such that $\lambda_i^x(\text{out}) = \text{out } \alpha$ (if it is defined) by using its lookbehind. Then it

processes each character $a \in \alpha[1] \cdots \alpha[|\alpha|]$ (since α is a bounded information, this loop is hardcoded in the states without moving). If $a \in B$, it is output. Otherwise, the machine determines $\beta := \lambda_i^x(a)$, moves left and executes **substitute**($\beta, i-1, a$) by doing recursive calls. The case when $i = 0$ is detected by reading the initial letter \vdash .

However, a 2-dT (which has a bounded memory) cannot keep track of the “call stack” for **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-dT has just finished executing **substitute**($\beta, i-1, a$), then it moves right:

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

Let $\mathcal{T} = (A, B, Q, q_0, \delta, \mathfrak{R}, \text{out}, \lambda)$ 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 [5, 7] or ω -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 ω -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 \mathcal{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 \mathfrak{r} is used to update both \mathfrak{r}_1 and \mathfrak{r}_2 . If we have K copies of \mathfrak{r} , we cannot produce, in a copyless way, K copies of \mathfrak{r}_1 and K copies of \mathfrak{r}_2 .

This issue is solved as follows. Recall that q_i^x (resp. λ_i^x) is the state reached by \mathcal{T} after reading $x[1:i]$ (resp. the transition applied when reading $x[i]$) on input $x \in A^\omega$. Let $\mathfrak{R}' := \mathfrak{R} \setminus \{\text{out}\}$ and $\mathfrak{r} \in \mathfrak{R}'$. If $i \geq 0$ and $x \in A^\omega$ are such that q_i^x is defined, we want to maintain $\text{copies}_i^x(\mathfrak{r})$ copies of the value $\llbracket \mathfrak{r} \rrbracket_i^x$, where $\text{copies}_i^x(\mathfrak{r}) \leq K$ is the number of times that \mathfrak{r} will be used in **out** after position i . Formally we define the following:

► **Definition C.9.** Let $x \in A^\omega$ and $i \geq 0$ such that q_i^x is defined. Given $\mathfrak{r} \in \mathfrak{R}'$, we let:

$$\text{copies}_i^x(\mathfrak{r}) := \max\{ |\lambda_{i+1}^x \circ \cdots \circ \lambda_j^x(\text{out})|_{\mathfrak{r}} : j \geq i \text{ and } \lambda_j^x \text{ is defined} \}.$$

▷ **Claim C.10.** $\text{copies}_i^x(\mathfrak{r}) \leq K$ since \mathcal{T} is K -bounded.

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

► **Lemma C.11.** Let $x \in A^\omega$ and $i \geq 0$ such that q_{i+1}^x is defined. Then for all $\mathfrak{r} \in \mathfrak{R}'$:

$$\text{copies}_i^x(\mathfrak{r}) = |\lambda_{i+1}^x(\text{out})|_{\mathfrak{r}} + \sum_{\mathfrak{s} \in \mathfrak{R}'} |\lambda_{i+1}^x(\mathfrak{s})|_{\mathfrak{r}} \times \text{copies}_{i+1}^x(\mathfrak{s}).$$

However, this number $c_i^x(\mathbf{r})$ 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 $\text{copies}_i^x(\mathbf{r})$. 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 \mathcal{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 \mathcal{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 $[\![\cdot]\!]_i^x$ 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 $[\![\cdot]\!]_i^x$ is defined. A decomposition of $[\![\cdot]\!]_i^x$ is:*

1. *a sequence $0 = i_0 < \dots < i_m = i$ of positions;*
2. *a sequence $\sigma_1, \dots, \sigma_m \in S_{\mathfrak{R}'}^B$ of K -bounded substitutions such that for all $1 \leq \ell \leq m$, $\sigma_\ell = \lambda_{i_{\ell-1}+1}^x \circ \dots \circ \lambda_{i_\ell}^x$ restricted to \mathfrak{R}' ;*
3. *a finite forest F whose nodes are labelled by functions $g : \mathfrak{R}' \rightarrow [0:K]$ such that:*
 - a. *all the leaves have same depth m and distinct labels;*
 - b. *there exists a leaf whose label is copies_i^x ;*
 - c. *if h labels a node of depth $1 \leq \ell \leq m$ and g labels its parent, then for all $\mathbf{r} \in \mathfrak{R}'$:*

$$g(\mathbf{r}) = \sum_{\mathbf{s} \in \mathfrak{R}'} |\sigma_\ell(\mathbf{s})|_{\mathbf{r}} \times h(\mathbf{s}). \quad (1)$$

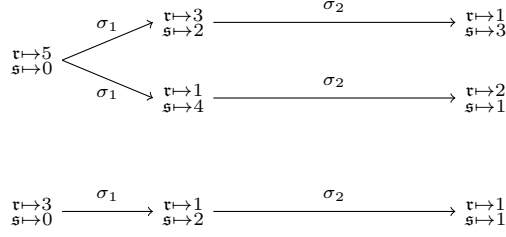
► **Example C.13.** Assume that $\mathfrak{R}' = \{\mathbf{r}, \mathbf{s}\}$, $K = 5$ and $\lambda_1^x(\mathbf{r}) = a\mathbf{r}$, $\lambda_1^x(\mathbf{s}) = b\mathbf{s}$; $\lambda_2^x(\mathbf{r}) = \mathbf{s}a\mathbf{s}$, $\lambda_2^x(\mathbf{s}) = \mathbf{r}b$; $\lambda_3^x(\mathbf{r}) = \mathbf{r}$, $\lambda_3^x(\mathbf{s}) = \mathbf{s}$. We describe a possible decomposition of $[\![\cdot]\!]_3^x$:

1. the sequence $i_0 = 0, i_1 = 1, i_2 = 3$;
2. the substitutions σ_1, σ_2 defined by $\sigma_1(\mathbf{r}) = a\mathbf{r}$, $\sigma_1(\mathbf{s}) = \mathbf{r}b$ and $\sigma_2(\mathbf{r}) = \mathbf{s}a\mathbf{s}$, $\sigma_2(\mathbf{s}) = \mathbf{r}b$;
3. the forest depicted in Figure 5 (note that Equation 1 holds).

Information stored by \mathcal{T}' . The configuration of \mathcal{T}' when in position $i \geq 0$ of $x \in A^\omega$ stores some decomposition D_i^x of $[\![\cdot]\!]_i^x$ as follows, with $m \leq L := (K+1)^{|\mathfrak{R}'|}$:

1. the sequence of positions $i_0 < \dots < i_m$ is not stored by \mathcal{T}' (its existence is an invariant which will be preserved along the computation);
2. the sequence of substitutions $\sigma_1, \dots, \sigma_m$ is stored depending on the forest F , see below;
3. the forest F_i^x associated to D_i^x . F_i^x 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

XX:20 Continuous rational functions are deterministic regular



■ **Figure 5** The forest of the decomposition described in Example C.13 (roots are on the left).

substitutions. If g labels a node of depth $1 \leq \ell \leq m$ in F_i^x , then \mathcal{T}' “stores $g(\mathbf{r})$ virtual copies of $\sigma_\ell(\mathbf{r})$ ” for all $\mathbf{r} \in \mathfrak{R}'$. What we mean by “storing virtual copies” is explained in Subsubsection C.3.2 (the ideas are those of [5, 7]).

Furthermore, the register `out` of \mathcal{T}' will contain $\llbracket \text{out} \rrbracket_i^x$.

Initialization of the decomposition. When $i = 0$, the configuration of \mathcal{T}' describes the following decomposition D_0^x of $\llbracket \cdot \rrbracket_0^x$ with $m = 0 \leq L$:

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 $L = (K + 1)^{|\mathfrak{R}'|}$ nodes labelled by all possible functions $\mathfrak{R}' \rightarrow [0:K]$. 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}^x is defined (otherwise \mathcal{T}' gets blocked), and that \mathcal{T}' stores a decomposition D_i^x of $\llbracket \cdot \rrbracket_i^x$. Then \mathcal{T}' transforms D_i^x into a decomposition of $\llbracket \cdot \rrbracket_{i+1}^x$ as follows:

1. the new sequence of positions is $i_0 = 0 < \dots < i_m = i < i_{m+1} = i+1$. 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 $\sigma_1, \dots, \sigma_m, \sigma_{m+1}$ where $\sigma_{m+1} := \lambda_{i+1}^x$. Note that the substitution σ_{m+1} is K -bounded since \mathcal{T} is so;
3. before describing the new forest built from F_i^x , let us give some intuitions. Recall that $\sigma_{m+1}(\text{out}) = \text{out} \alpha$ for some $\alpha \in (B \uplus \mathfrak{R}')^*$. We want \mathcal{T}' to add $\llbracket \alpha \rrbracket_i^x$ to `out` when reading $x[i+1]$. For this purpose, it needs to determine the value $\lambda_0^x \circ \sigma_1 \circ \dots \circ \sigma_m(\mathbf{r}) = \llbracket \mathbf{r} \rrbracket_i^x$ for each $\mathbf{r} \in \mathfrak{R}'$ which occurs in α . We thus define the functions $\text{used}_\ell : \mathfrak{R}' \rightarrow [0:K]$ for $0 \leq \ell \leq m$, which describe “how many virtual copies” of the $\sigma_\ell(\mathbf{r})$ will be consumed to compute $\llbracket \alpha \rrbracket_{i+1}^x$. They are built by a decreasing induction:

- $\text{used}_m(\mathbf{r}) := |\sigma_{m+1}(\text{out})|_{\mathbf{r}}$ for all $\mathbf{r} \in \mathfrak{R}'$;
- $\text{used}_\ell(\mathbf{r}) := \sum_{\mathbf{s} \in \mathfrak{R}'} |\sigma_{\ell+1}(\mathbf{s})|_{\mathbf{r}} \times \text{used}_{\ell+1}(\mathbf{s})$ for all $\mathbf{r} \in \mathfrak{R}'$, if $0 \leq \ell \leq m-1$.

We now want to subtract the used_ℓ to the labels of the nodes, since they describe the number of copies that we need to “consume” to compute $\lambda_0^x \circ \sigma_1 \circ \dots \circ \sigma_m(\alpha)$.

▷ **Claim C.14.** Assume that h labels a node of depth $1 \leq \ell \leq m$ in F_i^x and let g label its parent. Define $\bar{h} := h - \text{used}_\ell$ and $\bar{g} := g - \text{used}_{\ell-1}$, then:

- if $\bar{h} \geq 0$, then $\bar{g} \geq 0$;
- \bar{g} and \bar{h} verify Equation 1 in Condition 3c.

Proof. By Condition 3c on F we have $g(\mathbf{r}) = \sum_{\mathbf{s} \in \mathfrak{R}'} |s_{\ell+1}(\mathbf{s})|_{\mathbf{r}} \times h(\mathbf{s})$, therefore we get $g(\mathbf{r}) - \text{used}_\ell(\mathbf{r}) = \sum_{\mathbf{s} \in \mathfrak{R}'} |s_{\ell+1}(\mathbf{s})|_{\mathbf{r}} \times (h(\mathbf{s}) - \text{used}_{\ell+1}(\mathbf{s}))$ by definition of used . ◀

We now describe in three steps how to build the new forest from F_i^x :

Step 1: consuming the used $_\ell(\mathfrak{r})$. We replace each label g in F_i^x by \bar{g} from Claim C.14, which may create negative labels. But since one leaf is labelled by copies_i^x , then by Lemma C.11 we see that $\overline{\text{copies}_i^x} \geq 0$. Hence by Claim C.14, there is a branch whose labels are nonnegative. The copyless dSST shall use the “virtual copies of the $\sigma_\ell(\mathfrak{r})$ ” stored along this branch to output $\lambda_i^x \circ \sigma_1 \circ \dots \circ \sigma_m(\alpha)$ 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 $h : \mathfrak{R}' \rightarrow [0:K]$ such that for all $\mathfrak{r} \in \mathfrak{R}'$ we have:

$$\bar{g}(\mathfrak{r}) = \sum_{\mathfrak{s} \in \mathfrak{R}'} |\sigma_{m+1}(\mathfrak{s})|_{\mathfrak{r}} \times h(\mathfrak{s}).$$

For all $\mathfrak{r} \in \mathfrak{R}$ and all created leaf labelled by h , the dSST creates $h(\mathfrak{r})$ virtual copies of $\sigma_{m+1}(\mathfrak{r}) \in (\mathfrak{R}' \uplus B)^*$ (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 $\overline{\text{copies}_i^x}$ has a leaf labelled by copies_{i+1}^x ;

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 ℓ have exactly one children (in the forest build in the previous paragraph).

Proof. Assume that for all $1 \leq \ell \leq m$, some node of depth ℓ 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” σ_ℓ 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_{i+1}^x

1. the positions become $i_0 < i_1 < \dots < i_{\ell-1} < i_{\ell+1} < \dots < i_{m+1}$
2. the substitutions become $\sigma_1, \dots, \sigma_{\ell-1}, \sigma_\ell \circ \sigma_{\ell+1}, \sigma_{\ell+2}, \dots, \sigma_{m+1}$ (note that $\sigma_\ell \circ \sigma_{\ell+1}$ is K -bounded and corresponds to the restriction of $\lambda_{i_{\ell-1}+1}^x \circ \dots \circ \lambda_{i_{\ell+1}}^x$);
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 $\mathfrak{r} \in \mathfrak{R}'$:

$$g(\mathfrak{r}) = \sum_{\mathfrak{s} \in \mathfrak{R}'} |\sigma_\ell \circ \sigma_{\ell+1}(\mathfrak{s})|_{\mathfrak{r}} \times h(\mathfrak{s}).$$

Proof. By Claim C.19, we have:

$$\sum_{\mathfrak{s} \in \mathfrak{R}'} |\sigma_\ell \circ \sigma_{\ell+1}(\mathfrak{s})|_{\mathfrak{r}} \times h(\mathfrak{s}) = \sum_{\mathfrak{s} \in \mathfrak{R}'} \sum_{\mathfrak{t} \in \mathfrak{R}'} |\sigma_\ell(\mathfrak{t})|_{\mathfrak{r}} \times |\sigma_{\ell+1}(\mathfrak{s})|_{\mathfrak{t}} \times h(\mathfrak{s})$$

The result follows by permuting the sums and using condition 3c in ℓ and $\ell+1$. ◀

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

Finally \mathcal{T}' has the same domain than \mathcal{T} , since it gets blocked at position $i \geq 0$ if λ_i^x is undefined, and otherwise it stores $\llbracket \text{out} \rrbracket_i^x$ in its output **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 $1 \leq \ell \leq m$ and $\mathbf{r} \in \mathfrak{R}'$, this machine could “store $g(\mathbf{r})$ virtual copies of $\sigma_\ell(\mathbf{r}) \in (\mathfrak{R}' \uplus B)^*$ ”. 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 $\sigma \in \mathcal{S}_{\mathfrak{R}'}^B$ be a K -bounded substitution, then for all $\mathbf{r} \in \mathfrak{R}'$ there exists $n \leq K|\mathfrak{R}'|$ such that $\sigma(\mathbf{r}) = \alpha_0 \mathbf{s}_1 \alpha_1 \cdots \mathbf{s}_n \alpha_n$ with $\alpha_i \in B^*$, $\mathbf{t}_i \in \mathfrak{R}'$. We mainly have two informations in this expression:

- the sequence $\mathbf{s}_1 \mathbf{s}_2 \cdots \mathbf{s}_n$ 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 α_i , which is a bounded information.

We can now explain what we mean by “storing $g(\mathbf{r})$ virtual copies of $\sigma(\mathbf{r}) \in (\mathfrak{R}' \uplus B)^*$ ”: it means that we store $g(\mathbf{r})$ copies (in $g(\mathbf{r})$ distinct registers) of each α_i . Note that if $g(\mathbf{r}) \leq K$, we need at most $K^2|\mathfrak{R}'|$ registers. The sequence $\mathbf{s}_1 \mathbf{s}_2 \cdots \mathbf{s}_n$ is stored the state.

Composing K -bounded substitutions. With this representation, \mathcal{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 $\sigma, \sigma' \in \mathcal{S}_{\mathfrak{R}'}^B$ and $\sigma \circ \sigma'$ are K -bounded and that \mathcal{T}' stores:

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

such that $g(\mathbf{r}) = \sum_{\mathbf{s} \in \mathfrak{R}'} |\sigma'(\mathbf{s})|_{\mathbf{r}} \times g'(\mathbf{s})$. Then there exists a copyless update of \mathcal{T}' which allows to store $g'(\mathbf{s})$ copies of $\sigma \circ \sigma'(\mathbf{s})$ for $\mathbf{s} \in \mathfrak{R}'$.

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

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

► **Remark C.18.** To produce $\llbracket \alpha \rrbracket_i^x$, we only need to consume the copies along one branch of F_i^x . However, to maintain a forest which is consistent with our decomposition D_i^x , 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 $\sigma, \sigma' \in \mathcal{S}_{\mathfrak{R}}^B$, then for all $\mathfrak{r}, \mathfrak{s} \in \mathfrak{R}$, $|\sigma \circ \sigma'(\mathfrak{r})|_{\mathfrak{s}} = \sum_{\mathfrak{t} \in \mathfrak{R}} |\sigma(\mathfrak{t})|_{\mathfrak{s}} \times |\sigma'(\mathfrak{r})|_{\mathfrak{t}}$.

Proof. We have $s'(\mathfrak{r}) = \alpha_0 \mathfrak{t}_1 \alpha_1 \mathfrak{t}_2 \dots \mathfrak{t}_n \alpha_n$ for some words $\alpha_i \in B^*$ and registers $\mathfrak{t}_i \in \mathfrak{R}$. Therefore $|\sigma \circ \sigma'(\mathfrak{r})|_{\mathfrak{s}} = \sum_{i=1}^n |\sigma(\mathfrak{t}_i)|_{\mathfrak{s}} = \sum_{\mathfrak{t} \in \mathfrak{R}} |\sigma'(\mathfrak{r})|_{\mathfrak{t}} \times |\sigma(\mathfrak{t})|_{\mathfrak{s}}$. ◀

We then note that since **out** is always updated under the form **out** α , the number of copies of a given register in **out** can only grow.

▷ **Claim C.20.** Given $\mathfrak{r} \in \mathfrak{R}'$ and $i \geq 1$ such that λ_i^x is defined, the function which maps $j \geq i-1 \mapsto |\lambda_i^x \circ \dots \circ \lambda_j^x(\mathbf{out})|_{\mathfrak{r}}$ is increasing (on its domain).

Proof. By Claim C.19 we have $|\lambda_i^x \circ \dots \circ \lambda_{j+1}^x(\mathbf{out})|_{\mathfrak{r}} \geq |\lambda_i^x \circ \dots \circ \lambda_j^x(\mathbf{out})|_{\mathfrak{r}} \times |\lambda_{j+1}^x(\mathbf{out})|_{\mathbf{out}}$. ◀

We are now ready to show Lemma C.11. If $j \geq i+1$ is such that $[\![\cdot]\!]_j^x$ is defined, we have the following for all $\mathfrak{r} \in \mathfrak{R}'$ by Claim C.19:

$$|\lambda_{i+1}^x \circ \dots \circ \lambda_j^x(\mathbf{out})|_{\mathfrak{r}} = |\lambda_{i+1}^x(\mathbf{out})|_{\mathfrak{r}} \times 1 + \sum_{\mathfrak{s} \in \mathfrak{R}'} |\lambda_{i+1}^x(\mathfrak{s})|_{\mathfrak{r}} \times |\lambda_{i+2}^x \circ \dots \circ \lambda_j^x(\mathbf{out})|_{\mathfrak{s}}.$$

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

$$\text{copies}_i^x(\mathfrak{r}) = |\lambda_{i+1}^x \circ \dots \circ \lambda_j^x(\mathbf{out})|_{\mathfrak{r}} = |\lambda_{i+1}^x(\mathbf{out})|_{\mathfrak{r}} + \sum_{\mathfrak{s} \in \mathfrak{R}'} |\lambda_{i+1}^x(\mathfrak{s})|_{\mathfrak{r}} \times |\lambda_{i+2}^x \circ \dots \circ \lambda_j^x(\mathbf{out})|_{\mathfrak{s}}.$$

And by the same argument of Claim C.19 applied to the $|\lambda_{i+2}^x \circ \dots \circ \lambda_j^x(\mathbf{out})|_{\mathfrak{s}}$, we conclude:

$$\text{copies}_i^x(\mathfrak{r}) = |\lambda_{i+1}^x(\mathbf{out})|_{\mathfrak{r}} + \sum_{\mathfrak{s} \in \mathfrak{R}'} |\lambda_{i+1}^x(\mathfrak{s})|_{\mathfrak{r}} \times \text{copies}_{i+1}^x(\mathfrak{s}).$$

D Proof of Proposition 3.8

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

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

E Proof of Theorem 3.10

Let us consider a restricted 1-nT computing a function $f : A^\omega \rightarrow B^\omega$ and a copyless dSST computing a deterministic regular function $g : B^\omega \rightarrow C^\omega$. By making their wreath 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) $\mathcal{N} = (A, C, Q, I, \Delta, \mathfrak{R}, \text{out}, \lambda)$ 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 \mathfrak{R} with a distinguished output register $\text{out} \in \mathfrak{R}$;
- an update function $\lambda : \Delta \rightarrow \mathcal{S}_{\mathfrak{R}}^C$ such that for all $(q, a, q') \in \Delta$:
 - $\lambda(q, a, q')(\text{out}) = \text{out} \cdots$;
 - there is no other occurrence of out in $\{\lambda(q, a, q')(\mathfrak{r}) : \mathfrak{r} \in \mathfrak{R}\}$.

Given $x \in A^\omega \cup A^*$, a (resp. initial, final, accepting) run of \mathcal{N} labelled by x is a (resp. initial, final, accepting) run of the underlying one-way automaton (A, Q, I, F, Δ) where $F = Q$ (all states are accepting). Given an initial run $\rho = q_0 \rightarrow q_1 \rightarrow \cdots$ composed of $n \in \mathbb{N} \uplus \{\infty\}$ transitions, we define for $0 \leq i \leq n$ the copyless substitution $\lambda_i^\rho \in \mathcal{S}_{\mathfrak{R}}^C$ by $\lambda_i^\rho(\mathfrak{r}) = \varepsilon$ for all $\mathfrak{r} \in \mathfrak{R}$, and $\lambda_i^\rho := \lambda(q_i, x[i])$ for $i \geq 1$. We define the substitution $[\![\cdot]\!]_i^\rho \in \mathcal{S}_{\mathfrak{R}}^B : \mathfrak{R} \rightarrow B^*$ by $[\![\cdot]\!]_i^\rho = \lambda_0^\rho \circ \cdots \circ \lambda_i^\rho$. By construction one has that $[\![\text{out}]\!]_i^\rho \subseteq [\![\text{out}]\!]_{i+1}^\rho$ (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 ρ labelled by x , and such that $|\![\![\text{out}]\!]\!|_i^\rho| \rightarrow \infty$. Then $h(x) := \bigvee_i [\![\text{out}]\!]_i^\rho$. Otherwise $h(x)$ is undefined. We say that the restricted nSST is *copyless* if all the $\lambda_i^\rho \circ \cdots \circ \lambda_j^\rho$ are copyless when ρ 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 \mathcal{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 \mathcal{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 \mathcal{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_i^x as a tree whose nodes are labelled by tuples (i_1, ρ, i_2) with $0 \leq i_1 \leq i_2$ and ρ is a run labelled by $x[i_1+1:i_2]$. It is built by induction as follows:

- T_0^x consists of a single node labelled by $(0, \rho, 0)$ where $\rho = q_0$ is a run labelled by ε ;
- if T_i^x is built, we obtain T_{i+1}^x by doing the following operations where $a := x[i+1]$:
 - New transition:** below each leaf labelled by (i_1, ρ, i) with $\rho = q_{i_1} \rightarrow \cdots \rightarrow q_i$, and for all transition $(q_i, a, q_{i+1}) \in \Delta$, we add a leaf labelled by $(i, q_i \rightarrow q_{i+1}, i+1)$;
 - 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,

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_{i+1}^x which are not an ancestor of a new leaf;

Merging single nodes: for all node (labelled by $(i_1, q_{i_1} \rightarrow \dots \rightarrow q_{i_2}, i_2)$) which has only one child (labelled by $(i_2, q_{i_2} \rightarrow \dots \rightarrow q_{i_3}, i_3)$) we merge these two nodes together with common label $(i_1, q_{i_1} \rightarrow \dots \rightarrow q_{i_2} \rightarrow \dots \rightarrow q_{i_3}, i_2)$.

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_i^x has at most L nodes.

▷ **Claim E.4.** If $x \in \text{Dom}(f)$, there exists a root-to-leaf branch of T_i^x 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_i^x becomes longer and longer.

▷ **Claim E.5.** If $x \in \text{Dom}(f)$ and $(0, \rho_i, r_i)$ is the label of the root of T_i^x , then $r_i \rightarrow \infty$.

Proof. Assume that r_i is ultimately constant. Then the root of T_i^x always has at least two children, which implies that there exists two distinct infinite runs labelled by x . ◀

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

- the structure of T_i^x , i.e. the tree without its labels (stored in the state);
- for each leaf of T_i^x labelled by $(i_1, q_{i_1} \rightarrow \dots \rightarrow q_i, i)$, the last state q_i (stored in the state);
- for each node of T_i^x labelled by (j, ρ_n, j') , let $\rho := \rho_1 \dots \rho_n$ be the run labelling the branch of T_i^x starting in the root and ending in (j, ρ_n, j') . Let $\sigma_n := \lambda_{j+1}^\rho \circ \dots \circ \lambda_{j'}^\rho$ be the substitution applied along ρ_n . Let $\alpha \in ((\mathfrak{R} \setminus \{\text{out}\}) \cup C)^*$ be such that $\sigma_n(\text{out}) = \text{out} \alpha$, then we store $\llbracket \alpha \rrbracket_j^{\rho_1 \dots \rho_{n-1}}$ (it is the value which is added in **out** when following ρ_n). Furthermore we do the following for the root and the leaves:
 - if (j, ρ_n, j') is the root of T_i^x , then $\llbracket \alpha \rrbracket_j^{\rho_1 \dots \rho_{n-1}}$ is stored in the register **out** of \mathcal{T} ;
 - if (j, ρ_n, j') is a leaf, we also store for all $\mathfrak{r} \in \mathfrak{R} \setminus \{\text{out}\}$, the value $\llbracket \mathfrak{r} \rrbracket_{j'}^\rho$ (it is the value of \mathfrak{r} after executing ρ).

It is clear that for $x \in \text{Dom}(h)$, the value of **out** in \mathcal{T} is 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_i^x 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_i^x , 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_i^x as \mathcal{T} does. However, checking that $r_i \rightarrow \infty$ (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 $x \in A^\omega$ and $\beta \in B^\omega$ be such that $q'_2 \xrightarrow{x|\beta}$ is final (such a run exists since the transducer is trim and clean). Therefore, for all $n \geq 0$ we have $f(uu'^n x) = \alpha_2 \alpha'_2{}^n \beta$. On the other hand $f(uu'^\omega) = \alpha_1 \alpha'_1{}^\omega$ because $q_1 \in F$. By continuity in $uu'^\omega \in \text{Dom}(f)$, for all $p \geq 0$ we have $|f(uu'^n x) \wedge f(uu'^\omega)| \geq p$ for n large enough. The result follows directly.

G Proof of Lemma 4.4

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

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

Since \mathcal{T} is clean, the existence of constant states is clearly equivalent to the non-productivity of \mathcal{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 $\alpha_q \in B^*$, $\alpha'_q \in B^+$ such that for all final run $q \xrightarrow{x|\beta}$, $\beta = \alpha_q \alpha'_q{}^\omega$.

Proof. If q'_2 is constant, we can enforce $|u|, |u'| \leq |Q|^{|\mathcal{Q}|}$ (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 $\beta = (\alpha_2)^{-1} \alpha_1 \alpha'_1{}^\omega$ for all $q \xrightarrow{x|\beta}$ final. ◀

Given q constant, we describe a transducer without ε -loops, which computes the function $x \mapsto \alpha_q \alpha'_q$ when x labels a final run starting in q . Let $\mathcal{T}_q = (A, B, Q_q, I_q, F_q, \Delta_q, \lambda_q)$ be:

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

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

Proof. It is clear that \mathcal{T}_q has no loop which produces ε since $\alpha'_q \neq \varepsilon$. Furthermore, an accepting run of \mathcal{T}_q labelled by x is necessarily of the form $q \xrightarrow{x[1]|\alpha_q} (q_1, 1) \xrightarrow{x[2]|\alpha'_q} (q_2, 1) \dots$, where $q \xrightarrow{x[1]} q_1 \xrightarrow{x[2]} q_2 \dots$ is final in \mathcal{T} . The converse also holds. Hence \mathcal{T}_q is unambiguous since \mathcal{T} is unambiguous and trim, and furthermore it computes the function f_q . ◀

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

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

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

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 $q_1, q_2 \in I'$, $q'_1 \in F'$, $q'_2 \in Q \setminus F$, $u \in A^*$, $u' \in A^+$, $\alpha_1, \alpha'_1, \alpha_2, \alpha'_2 \in B^*$ with $\alpha'_2 = \varepsilon$ such that $q_i \xrightarrow{u|\alpha_i} q'_i \xrightarrow{u'|\alpha'_i} q'_i$. Then q'_2 is not in one of the \mathcal{T}_q , because these machines have no ε -loops by Claim G.3. Thus we had a run $q_2 \xrightarrow{u|\alpha_2} q'_2 \xrightarrow{u'|\alpha'_2} q'_2$ in \mathcal{T} and so q'_2 was not constant in \mathcal{T} (because otherwise it would be the initial state of $\mathcal{T}_{q'_2}$). Now:

- either $q_1 \xrightarrow{u|\alpha_1} q'_1 \xrightarrow{u'|\alpha'_1} q'_1$ 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 $q_1 \xrightarrow{u|\alpha_1} q'_1 \xrightarrow{u'|\alpha'_1} q'_1$ goes in \mathcal{T}_q at some point, for some constant state q . Assume without loss of generality that $q_1 \notin \mathcal{T}_q$, then $q'_1 = (q'_1, 1)$ and $q_1 \xrightarrow{u|\beta_1} q'_1 \xrightarrow{u'|\beta'_1} q'_1$ is a run in \mathcal{T} . Then $q'_1 \in F$ since $q'_1 \in F_q$, hence $\beta'_1 \neq \varepsilon$ because \mathcal{T} was clean. This contradicts the fact that q'_2 is not constant in \mathcal{T} .

H Proofs of lemmas 4.8 and 4.12

The goal of this section is to show lemmas 4.8 and 4.12. 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.5.

H.1 Compatible sets and continuity

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 $d : C \rightarrow Q$, and $u, u' \in A^*$ such that the following holds:*

- for all $q \in C$, $q \xrightarrow{u} d(q) \xrightarrow{u'} d(q)$;
- 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 ρ_q for $q \in C$ labelled by x such that $\forall q \in C$, ρ_q begins in q , and furthermore ρ_p is final for some $p \in Q$. Therefore we have $\rho_p(i) \in F$ 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 $\rho_p(i) \in F$. We define $d(q) := \rho_q(i)$ for $q \in C$. Finally we get $q \xrightarrow{x[1:i]} d(q) \xrightarrow{x[i+1:j]} d(q)$ for all $q \in C$.

Now if $|x[i+1:j]| \geq |Q|^{|Q|} + 1$, by a similar pumping argument we factor $x[1:i] = vv'v''$ such that $v, v' \neq \varepsilon$ and $q \xrightarrow{vv''} d(q) \xrightarrow{x[i+1:j]} d(q)$ 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 $\text{end}_C : C \rightarrow B^\omega$ such that for all initial step J, u, C and $p, q \in C$ we have $\text{prod}_{J,C}^u(p) \text{end}_C(p) = \text{prod}_{J,C}^u(q) \text{end}_C(q)$.*

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 $q \in C$, $q \xrightarrow{v|\alpha(q)} d(q) \xrightarrow{v'|\alpha'(q)} d(q)$ with $\alpha(q), \alpha'(q) \in B^*$. Since \mathcal{T} is trim and clean, for all $q \in C$ there exists $x(q) \in A^\omega$, $\beta(q) \in B^\omega$ such that $d(q) \xrightarrow{x(q)|\beta(q)} \infty$ is accepting. We define $\text{end}_C(q) := \alpha(q)\alpha'(q)^\omega$ if $\alpha'(q) \neq \varepsilon$ and $\text{end}_C(q) := \alpha(q)\beta(q)$ otherwise. Let us now justify that 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 $\text{prod}_{J,C}^u(p) \text{end}_C(p) = \text{prod}_{J,C}^u(q) \text{end}_C(q)$, which is a direct consequence of Lemma 4.3. ◀

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

► **Lemma H.3.** *Let J, u, C be an initial step and C, v, D be a step. Then for all $p, q \in D$:*

$$\text{adv}_{J,C}^u(\text{pre}_{C,D}^v(q)) \text{prod}_{C,D}^v(q) \text{end}_D(q) = \text{adv}_{J,C}^u(\text{pre}_{C,D}^v(p)) \text{prod}_{C,D}^v(p) \text{end}_D(p).$$

Proof. By Lemma H.2 we have $\text{prod}_{J,D}^{uv}(p) \text{end}_D(p) = \text{prod}_{J,D}^{uv}(q) \text{end}_D(q)$ which by splitting gives $\text{prod}_{J,C}^u(\text{pre}_{C,D}^v(p)) \text{prod}_{C,D}^v(p) \text{end}_D(p) = \text{prod}_{J,C}^u(\text{pre}_{C,D}^v(q)) \text{prod}_{C,D}^v(q) \text{end}_D(q)$. ◀

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).

► **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:*

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

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 $||\text{prod}_{J,C}^v(p) - \text{prod}_{J,C}^v(q)|| > M|Q|^{|Q|}$. Suppose by symmetry that $|\text{prod}_{J,C}^v(p)| > |\text{prod}_{J,C}^v(q)| + M|Q|^{|Q|}$. Thus $|v| > |Q|^{|Q|}$. By pumping we can factor $v = uu'u''$ with $0 < |u'| \leq |Q|^{|Q|}$ such that $i(q) \xrightarrow{u|\alpha(r)} \ell(r) \xrightarrow{u'|\alpha'(r)} \ell(r) \xrightarrow{u''|\alpha''(r)} r$ 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. ◀

We now state the strong version of Lemma 4.12. Its proof is given in Subsection H.3.

► **Lemma H.5** (Looping futures - strong version). *Let $C \in \text{Comp}$ be separable and J, w, C be an initial step. There exists $\tau, \theta \in B^*$ with $|\theta| = \Omega!$ and $|\text{lag}| \leq 3\Omega$, which can be uniquely determined from C and $\text{adv}_{J,C}^w(q)$ for $q \in C$, such that:*

- $\text{lag} \sqsubseteq \text{max-adv}_{J,C}^w \sqsubseteq \tau\theta^\omega$;
- for all step C, v, D and $q \in D$, $\text{prod}_{C,D}^w(q) \text{end}_D(q) = (\text{adv}_{J,C}^w(\text{pre}_{C,D}^v(q)))^{-1}\tau\theta^\omega$.

H.3 Proof of Lemma H.5

Since C is separable, we get $p, q \in C$ verifying the conditions of Lemma H.4. Assume by symmetry that $0 \leq |\alpha'(q)| < |\alpha'(p)| \leq M|Q|^{|Q|} = \Omega$. Note that $i(C), uu''u'', C$ is an initial

step for all $n \geq 0$. Furthermore for $n := |\alpha'(p)|k$ and $k \geq 0$ large enough, we get:

$$\begin{aligned}
 \left(\text{prod}_{i(C),C}^{uu'u''}(q) \right)^{-1} \text{prod}_{i(C),C}^{uu'u''}(p) &= (\alpha(q)\alpha'(a)^n\alpha''(q))^{-1} \alpha(p)\alpha'(p)^n\alpha''(p) \\
 &= (\alpha(p)\alpha'(p)^n\alpha''(p)) [|\alpha(q)\alpha''(q)| + k|\alpha'(p)\alpha'(q)| :] \\
 &= \left(\alpha'(p)^{k(|\alpha'(p)|-|\alpha'(q)|)} \alpha''(p) \right) [K :] \\
 &\quad \text{with } K := |\alpha(q)\alpha''(q)| - |\alpha(p)| \\
 &= \varphi^{k-K} \delta(k)
 \end{aligned}$$

where $\varphi := (\alpha'(p)^\omega)[K : K + |\alpha'(p)|]$ is a conjugate of $\alpha'(p)$ which is computable from C , and $\delta(k) := (\varphi^{k-K})^{-1} \left((\alpha'(p)^{k(|\alpha'(p)|-|\alpha'(q)|)} \alpha''(p)) [K :] \right)$.

Let $\psi := \varphi^{\Omega! / |\varphi|}$, note that $|\psi| = \Omega!$ and it can be computed from C .

► **Sublemma H.6.** *For all step C, v, D and for all $\bar{r} \in D$, if $r := \text{pre}_{C,D}^v(\bar{r})$ then we have $\text{prod}_{C,D}^v(\bar{r}) \text{end}_D(\bar{r}) = (\text{adv}_{i(C),C}^{uu'u''}(r))^{-1} (\text{adv}_{i(C),C}^{uu'u''}(q)) \psi^\omega$.*

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

$$\begin{aligned}
 \text{prod}_{C,D}^v(\bar{q}) \text{end}_D(\bar{q}) &= \left(\text{prod}_{i(C),C}^{uu'u''}(q) \right)^{-1} \text{prod}_{i(C),C}^{uu'u''}(p) \text{prod}_{C,D}^v(\bar{p}) \text{end}_D(\bar{p}) \\
 &= \varphi^{k-K} \delta(k) \text{prod}_{C,D}^v(\bar{p}) \text{end}_D(\bar{p}) \quad \text{for } k \geq 0 \text{ large enough}
 \end{aligned}$$

By using arbitrarily large k , we see that $\text{prod}_{C,D}^v(\bar{q}) \text{end}_D(\bar{q}) = \varphi^\omega = \psi^\omega$. Now if \bar{r} is such that $\text{pre}_{C,D}^v(\bar{r}) = r$, then by Lemma H.3 applied to $i(C), uu'u'', C$ and C, v, D we get:

$$\text{prod}_{C,D}^v(\bar{r}) \text{end}_D(\bar{r}) = (\text{adv}_{i(C),C}^{uu'u''}(r))^{-1} \underbrace{\text{adv}_{i(C),C}^{uu'u''}(q) \text{prod}_{C,D}^v(\bar{q}) \text{end}_D(\bar{q})}_{\psi^\omega}.$$

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 $\text{adv}_{J,C}^w(r) = \varepsilon$ (resp. $\text{adv}_{J,C}^w(s) = \text{max-adv}_{J,C}^w$), i.e. r is the furthest behind. We conclude the proof by making a case disjunction on the value of $\text{max-adv}_{J,C}^w$:

■ either $|\text{max-adv}_{J,C}^w| \leq 3\Omega$. Then we define:

$$\begin{cases} \tau := \text{max-adv}_{J,C}^w \\ \theta := \psi^\omega [|(\text{adv}_{J,C}^w(q))^{-1} \tau| : |(\text{adv}_{J,C}^w(q))^{-1} \tau| + |\psi|] \end{cases}$$

▷ **Claim H.7.** For all step C, v, D and $\bar{s} \in D$ be such that $s = \text{pre}_{C,D}^v(\bar{s})$, we have: $\tau \text{prod}_{C,D}^v(\bar{s}) \text{end}_D(\bar{s}) = \tau \theta^\omega$.

Proof. By Lemma H.3, $\text{max-adv}_{J,C}^w \text{prod}_{C,D}^v(\bar{s}) \text{end}_D(\bar{s}) = \text{adv}_{J,C}^w(q) \underbrace{\text{prod}_{C,D}^v(\bar{q}) \text{end}_D(\bar{q})}_{= \psi^\omega \text{ by Sublemma H.6}}$

and so $\text{prod}_{C,D}^v(\bar{s}) \text{end}_D(\bar{s}) = (\psi^\omega) [|(\text{adv}_{J,C}^w(q))^{-1} \tau| :] = \theta^\omega$. ◀

■ or $|\text{max-adv}_{J,C}^w| > 3\Omega$. Let $m := |(\text{adv}_{i(C),C}^{uu'u''}(q)) - |\text{adv}_{i(C),C}^{uu'u''}(r)|$. Since $|uu'u''| \leq 3|Q|^{|\mathcal{Q}|}$, we have $m \leq M \times 3|Q|^{|\mathcal{Q}|} = 3\Omega \leq |\text{max-adv}_{J,C}^w|$. In this case we define:

- if $m \leq 0$, $\tau := \varepsilon$ and $\theta := \psi^\omega [-m : -m + |\psi|]$;
- if $m > 0$, $\tau := \text{max-adv}_{J,C}^w[1:m]$ and $\theta := \psi$.

XX:30 Continuous rational functions are deterministic regular

▷ Claim H.8. For all step C, v, D and $\bar{s} \in D$ be such that $s = \text{pre}_{C,D}^v(\bar{s})$, we have:
 $\text{max-adv}_{J,C}^w \text{prod}_{C,D}^v(\bar{s}) \text{end}_D(\bar{s}) = \tau\theta^\omega$.

Proof. Let $\bar{r} \in D$ be such that $r = \text{pre}_{C,D}^v(\bar{r})$. We get from Lemma H.3:

$$\begin{aligned} \text{max-adv}_{J,C}^w \text{prod}_{C,D}^v(\bar{s}) \text{end}_D(\bar{s}) &= \varepsilon \text{prod}_{C,D}^v(\bar{r}) \text{end}_D(\bar{r}) \\ &= (\text{adv}_{i(C),C}^{uu'u''}(r))^{-1} (\text{adv}_{i(C),C}^{uu'u''}(q)) \psi^\omega \quad \text{by Sublemma H.6.} \end{aligned}$$

If $m \leq 0$, then $|\text{adv}_{i(C),C}^{uu'u''}(q)| \leq |\text{adv}_{i(C),C}^{uu'u''}(r)|$ and $\text{max-adv}_{J,C}^w \text{prod}_{C,D}^v(\bar{s}) \text{end}_D(\bar{s}) = \psi^\omega[-m:]$.

If $m > 0$, then $\text{max-adv}_{J,C}^w \text{prod}_{C,D}^v(\bar{s}) \text{end}_D(\bar{s}) = (\text{adv}_{i(C),C}^{uu'u''}(q))[1:m] \psi^\omega$. ◀

Note that claims H.7 and H.8 give the same result (in two cases). Now let C, v, D and apply Lemma H.3 to J, w, C and C, v, D , we get for all $\bar{t} \in D$ with $t := \text{pre}_{C,D}^v(\bar{t})$:

$$\begin{aligned} \text{prod}_{C,D}^w(\bar{t}) \text{end}_D(\bar{t}) &= (\text{adv}_{J,C}^w(t))^{-1} \text{max-adv}_{J,C}^w \text{prod}_{C,D}^v(\bar{s}) \text{end}_D(\bar{s}) \\ &= (\text{adv}_{J,C}^w(t))^{-1} \tau\theta^\omega. \end{aligned}$$

I Proof of Lemma 4.15

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

I.1 Properties of compatible sets

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

► **Definition I.1.** If $u \in A^*$ and $S \subseteq Q$, we let $S \cdot u := \{q' : \exists q \in S, q \xrightarrow{u|\alpha} q'\} \subseteq Q$.

► **Lemma I.2** (Compatible sets cover the future). Let $x \in \text{Dom}(f)$, $i \geq 0$ and $q_i^x \in S \subseteq Q$. 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 I.2 does not hold. Let P be the set of subsets $C \subseteq S$ such that $q_i^x \in C$, and for all $p \in C$ there exists an infinite run starting in p and labelled by $x[i+1:]$. Then $P \subseteq \text{Comp}(S)$ and $\{q_i^x\} \in P$.

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 a infinite branch, i.e. there exists a state $p \in S \setminus C$ and a infinite run starting in p and labelled by x . Finally $C \uplus \{p\} \in P$, which contradicts the maximality of $|C|$. ◀

Using this result, one can define $\text{cover}_i^x(S)$ to be the smallest set for $<$ among the elements of $\text{Comp}(S)$ whose future completely covers the future of S as quickly as possible. Formally we have Definition I.3 (which makes sense by Lemma I.2).

► **Definition I.3** (Time and cover). Let $x \in \text{Dom}(f)$, $i \geq 0$ and $q_i^x \in S \subseteq Q$, we define:

- $\text{time}_i^x(S) := \min\{j \geq i : \exists C \in \text{Comp}(S) \text{ such that } C \cdot x[i+1:j] = S \cdot x[i+1:j]\}$
- $\text{cover}_i^x(S)$ the minimal element for $<$ in the (non-empty) set:
 $\{C \in \text{Comp}(S) : C \cdot x[i+1:\text{time}_i^x(S)] = S \cdot x[i+1:\text{time}_i^x(S)]\}$.

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

► **Definition I.4** (Good). Let $x \in \text{Dom}(f)$, the sequence $(\text{Good}_i^x)_i \in \text{Comp}^{\mathbb{N}}$ is defined by:

- $\text{Good}_0 = \text{cover}_0^x(I)$;
- for $i \geq 0$, $\text{Good}_{i+1}^x = \text{cover}_{i+1}^x(\text{Good}_i^x \cdot x[i+1])$.

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

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 $q_{i+1}^x \in \text{Good}_i^x \cdot x[i+1]$ since $q_i^x \in \text{Good}_i^x$. Therefore Good_{i+1}^x is well defined by Definition I.3. Furthermore, by induction hypothesis and definition of cover_{i+1}^x we have $\text{Good}_{i+1}^x \subseteq \text{Good}_i^x \cdot x[i+1] \subseteq I \cdot x[1:i] \cdot x[i+1] = I \cdot x[1:i+1]$.

It remains to show that $q_{i+1}^x \in \text{Good}_{i+1}^x$. Indeed, by definition of cover_{i+1}^x there exists $j \geq i+1$ such that $\text{Good}_{i+1}^x \cdot x[i+2:j] = \text{Good}_i^x \cdot x[i+1] \cdot x[i+2:j]$. But we necessarily have $q_j^x \in \text{Good}_i^x \cdot x[i+1] \cdot x[i+2:j]$ because $q_{i+1}^x \in \text{Good}_i^x \cdot x[i+1]$. Therefore we can find a run of \mathcal{T} of the form $p_{i+1} \rightarrow p_{i+2} \rightarrow \dots \rightarrow p_j$ 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 \mathcal{T} is unambiguous, one has $p_{i+1} = q_{i+1}^x$. ◀

► **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 \mathcal{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 4.15, we prove that the sequence $(\text{Good}_i^x)_i$ can be computed by a restricted 1-nT.

► **Proposition I.7.** One can build a restricted 1-nT \mathcal{A} computing some $f' : A^\omega \rightarrow (A \uplus \text{Comp})^\omega$ such that if $x \in \text{Dom}(f)$, then $f'(x) = \text{Good}_0^x x[1] \text{Good}_1^x x[2] \dots$.

Proof of Lemma 4.15. Immediate by definition of $(\text{Good}_i^x)_i$ and Lemma I.5. ◀

The rest of this section is devoted to showing Proposition I.7.

1.2 Description of the restricted 1-nT

We now describe how the restricted 1-nT \mathcal{A} of Proposition I.7 is built.

States. A state of \mathcal{A} is a tuple (S, C, \mathfrak{H}) where:

- $S \in \text{Parts}$ and $C \in \text{Comp}(S)$ (hint: we want C to be Good_i^x after reading $x[1:i]$);
- \mathfrak{H} 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).

► **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 \mathcal{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 $(S_1, C_1, \mathfrak{H}_1)$ to $(S_2, C_2, \mathfrak{H}_2)$ if the following conditions hold:

1. $S_2 = C_1 \cdot a$;
2. $\mathfrak{H}_2 = \{(S^h, C^h, T \cdot a, g \cdot a) : (S^h, C^h, T, g) \in \mathfrak{H}_1\} \cup \{(S_2, C_2, S_2, \chi_{S_2})\}$ 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 Good_i^x :
 - a. if $S_1 \in \text{Comp}$ then $C_1 = S_1$;
 - b. if $(S^h, C^h, T, g) \in \mathfrak{H}_1$ with $g^{-1}(T) = \emptyset$ but $(g \cdot a)^{-1}(T \cdot a) \neq \emptyset$, then C^h is the minimal element for \prec in $(g \cdot a)^{-1}(T \cdot a) \neq \emptyset$.

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

I.3 Correctness of the construction

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

► **Lemma I.9.** *Let $x \in \text{Dom}(f)$, then \mathcal{A} has only one final run labelled by x . It is of the form $(S_0, \text{Good}_0^x, \mathfrak{H}_0) \rightarrow (S_1, \text{Good}_1^x, \mathfrak{H}_1) \rightarrow \dots$ for some S_0, S_1, \dots and $\mathfrak{H}_0, \mathfrak{H}_1, \dots$.*

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 \mathcal{A} .

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

▷ **Claim I.11.** If $u \in A^*$ and $a \in A$ then $g \cdot u \cdot a = g \cdot ua$. Furthermore $g \cdot \varepsilon = g$.

We now show that the \mathfrak{H}_i along an accepting run store some “history” about the S_i and C_i .

► **Sublemma I.12.** *Let $(S_0, C_0, \mathfrak{H}_0) \rightarrow (S_1, C_1, \mathfrak{H}_1) \rightarrow \dots$ be a accepting run of \mathcal{A} labelled by $x \in A^\omega$, then for all $n \geq 0$:*

- $S_0 = I$ and $S_{n+1} = C_n \cdot x[n+1]$;
- $\forall n \geq 0, C_n \in \text{Comp}(S_n)$;
- $\forall n \geq 0, \mathfrak{H}_n = \{(S_i, C_i, S_i \cdot x[i+1:n], \chi_{S_i} \cdot x[i+1:n]) : 0 \leq i \leq n\}$.

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

$$\begin{aligned} \mathfrak{H}_{n+1} = & \{(S_i, C_i, S_i \cdot x[i+1:n+1], \chi_{S_i} \cdot x[i+1:n+1]) : 0 \leq i \leq n\} \\ & \cup \{(S_{n+1}, C_{n+1}, S_{n+1} \cdot \varepsilon, \chi_{S_{n+1}} \cdot \varepsilon)\}. \end{aligned}$$

by condition 2 on transitions and Claim I.11. ◀

Given $x \in \text{Dom}(f)$, we now completely describe the unique accepting run of \mathcal{A} labelled by x . Let $(S_n^x)_n \in \text{Parts}^\mathbb{N}$ defined by $S_0^x := I$ and for $n \geq 0$ let $S_{n+1}^x := \text{Good}_n^x \cdot x[n+1]$. Furthermore for $n \geq 0$ we define $\mathfrak{H}_n^x := \{(S_i^x, \text{Good}_i^x, S_i^x \cdot x[i+1:n], \chi_{S_i^x} \cdot x[i+1:n]) : 0 \leq i \leq n\}$.

► **Sublemma I.13.** *Given $x \in \text{Dom}(f)$, then $\rho_x^x = (S_0^x, \text{Good}_0^x, \mathfrak{H}_0^x) \rightarrow (S_1^x, \text{Good}_1^x, \mathfrak{H}_1^x) \rightarrow \dots$ is an accepting run of \mathcal{A} labelled by x .*

Proof. Each $(S_n^x, \text{Good}_n^x, \mathfrak{H}_n^x)$ is a state since $\text{Good}_n^x \in \text{Comp}(S_n^x)$. The first state is initial since $S_0^x = I$ and $\text{Good}_0^x \in \text{Comp}(I)$, $\mathfrak{H}_0^x = \{(I, \text{Good}_0^x, I, \chi_I)\}$. Condition 1 holds since $S_{n+1}^x = \text{Good}_n^x \cdot x[n+1]$ for $n \geq 0$. Condition 2 holds, since by definition and Claim I.11:

$$\begin{aligned} \mathfrak{H}_{n+1}^x = & \{(S_i^x, \text{Good}_i^x, S_i^x \cdot x[i+1:n] \cdot x[n+1], \chi_{S_i^x} \cdot x[i+1:n] \cdot x[n+1]) : 0 \leq i \leq n\} \\ & \cup \{(S_{n+1}^x, \text{Good}_{n+1}^x, S_{n+1}^x \cdot \varepsilon, \chi_{S_{n+1}^x} \cdot \varepsilon)\}. \end{aligned}$$

Now let us show that given $n \geq 0$, conditions 3a and 3b for transitions hold between $(S_n^x, \text{Good}_n^x, \mathfrak{H}_n^x)$ and $(S_{n+1}^x, \text{Good}_{n+1}^x, \mathfrak{H}_{n+1}^x)$. Indeed:

- if $S_n^x \in \text{Comp}$, then S_n^x is the unique $C \in \text{Comp}(S_n^x)$ such that $C \cdot \varepsilon = S_n^x \cdot \varepsilon$. Hence $\text{time}_x^n = n$ and $\text{Good}_n^x = \text{cover}_n^x(S_n^x) = S_n^x$. Thus condition 3a holds;

- assume that $\exists n \geq i$ such that $(\chi_{S_i^x} \cdot x[i+1:n])^{-1}(S_i^x \cdot x[i+1:n]) = \emptyset$ and such that:

$$\underbrace{(\chi_{S_i^x} \cdot x[i+1:n] \cdot x[n+1])^{-1}(S_i^x \cdot x[i+1:n] \cdot x[n+1])}_{=(\chi_{S_i^x} \cdot x[i+1:n+1])^{-1}(S_i^x \cdot x[i+1:n+1])} \neq \emptyset$$

By definition of $g \cdot u$ and χ we get for $j \geq i$:

$$(\chi_{S_i^x} \cdot x[i+1:j])^{-1}(Y) = \{Y \subseteq \text{Comp}(S_i^x) : X \cdot x[i+1:j] = Y\} \quad (2)$$

therefore for $Y = S_i^x \cdot x[i+1:j]$ we have:

$$(\chi_{S_i^x} \cdot x[i+1:j])^{-1}(S_i^x \cdot x[i+1:j]) = \{C \in \text{Comp}(S_i^x) : C \cdot x[i+1:j] = S_i^x \cdot x[i+1:j]\}$$

Hence our hypotheses yield $\text{time}_i^x(S_i^x) = n+1$ (see Definition I.3) and furthermore Good_{n+1}^x is the minimal element for \prec in the set $(\chi_{S_i^x} \cdot x[i+1:n+1])^{-1}(S_i^x \cdot x[i+1:n+1])$.

Therefore the run is accepting. \blacktriangleleft

► **Sublemma I.14.** *Given $x \in \text{Dom}(f)$, the run ρ_x^x is the unique accepting run labelled by x .*

Proof. Let us consider an accepting run $\rho := (S_0, C_0, \mathfrak{H}_0) \rightarrow (S_1, C_1, \mathfrak{H}_1) \rightarrow \dots$ and suppose that it coincides with ρ_x^x until the state $(S_{i-1}, C_{i-1}, \mathfrak{H}_{i-1})$. Then one has $S_i = S_i^x$ since:

- either $i = 0$ and $(S_0, C_0, \mathfrak{H}_0)$ is initial, thus $S_0 = I = S_0^x$;
- or $i \geq 1$ and by Sublemma I.12 we have $S_i = C_{i-1} \cdot x[i] = \text{Good}_{i-1}^x \cdot x[i] = S_i^x$.

Furthermore $C_i \in \text{Comp}(S_i^x)$. Now let $n := \text{time}_i^x(S_i^x)$, two cases may occur:

- either $n = i$ which means that $S_i^x \in \text{Comp}(S_i^x)$ hence $\text{Good}_i^x = \text{cover}_i^x(S_i^x) = S_i^x$. By condition 3a on transitions (and since there is an outing transition from the state $(S_i, C_i, \mathfrak{H}_i)$) we conclude that $C_i = S_i^x = \text{Good}_i^x$;
- or $n > i$. Hence by Equation 2, $(\chi_{S_i^x} \cdot x[i+1:n-1])^{-1}(S_i^x \cdot x[i+1:n-1]) = \emptyset$ and Good_i^x is the smallest element for \prec of $(\chi_{S_i^x} \cdot x[i+1:n])^{-1}(S_i^x \cdot x[i+1:n]) \neq \emptyset$. But by Sublemma I.12 we have $(S_i^x, C_i, S_i^x \cdot x[i+1:n-1]), \chi_{S_i^x} \cdot x[i+1:n-1] \in \mathfrak{H}_{n-1}$. Considering the transition from $n-1$ to n , we must have by condition 3b that C_i is the smallest element in $(\chi_{S_i^x} \cdot x[i+1:n])^{-1}(S_i^x \cdot x[i+1:n])$, that is Good_i^x .

We conclude that $C_i = \text{Good}_i^x$ and finally $\mathfrak{H}_i = \mathfrak{H}_i^x$ by Sublemma I.12. \blacktriangleleft

J Invariant preservation in Section 5: correctness of \mathcal{S}

J.1 Proof of Lemma 5.3

Invariants 1 and 2 are obvious. Invariant 4a follows since the $\text{adv}_{J,C}^{x[1:i+1]}(q)$ were mutual prefixes and one of them is empty. Invariants 4b and 4d are obvious due to emptiness of the nb_π and out_π . For Invariant 4c we use Remark 4.13. Invariant 4e holds by definition of $\text{lag}(q)$ and $\text{last}(q)$ and emptiness of the nb_π and out_π . For invariant 4f we use Lemma 4.12 which gives for all step C, u, D and $q \in D$, $\text{prod}_{C,D}^u(q) \subseteq (\text{adv}_{J,C}^{x[1:i+1]}(\text{pre}_{C,D}^u(q)))^{-1} \tau \theta^\omega$. Therefore by adding $\text{prod}_{J,C}^{x[1:i+1]}(\text{pre}_{C,D}^u(q))$ on both sides we get $\text{prod}_{J,D}^{x[1:i+1]u}(q) \subseteq \text{com}_{J,C}^{x[1:i+1]} \tau \theta^\omega$. We conclude because $\text{max-lag} = \tau$. Finally invariant 4g follows since all paths $\pi \neq C$ are close.

J.2 Proof of Lemma 5.4

Invariant 2 is preserved along the operation, since we never modify J nor C . It is clear that invariant 4 holds before using the function $\text{down}(C)$. Indeed, we do not modify the $\text{lag}(q)$, 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 $\text{tree}(C)$ (in a strictly decreasing way). For all $\pi \in \text{tree}(C)$, let n_π (resp. o_π) be the value of nb_π (resp. o_π) before launching the function $\text{down}(C)$.

► **Sublemma J.1.** *Let $\pi = C_1 \cdots C_n \in \text{tree}(C)$. During the execution of $\text{down}(C)$, the following invariants hold just before $\text{down}(\pi)$ makes its recursive calls:*

- i. for all $\pi' = C_1 \cdots C_{n'} \sqsubseteq \pi$, $\text{nb}_{\pi'} : C_{n'} \rightarrow [0:4]$;
- ii. for all $C' \in \text{Comp}(C_n)$, $C' \neq C_n$ and all $\pi C' \sqsubset \pi'$, we have $\text{nb}_{\pi'} = n_{\pi'}$ and $\text{out}_{\pi'} = o_{\pi'}$;
- iii. if $\text{down}(\pi)$ has performed the update $\text{nb}_{\pi C'}(q) \leftarrow \text{nb}_{\pi C'}(q) + (\text{nb}_\pi(q) - 4)$, for some C' and $q \in C_n$, then for all $\pi' \sqsubseteq \pi$, we have $\text{nb}_{\pi'}(q) = 4$.
- iv. if $q \in C$ was lagging before launching $\text{down}(C)$, then for all $\pi' = C'_1 \cdots C'_{n'} \in \text{tree}(C)$ such that $q \in C'_{n'}$, we have $\text{nb}_{\pi'}(q) = 0$ and, if $\pi' \neq C$, $\text{out}_{\pi'} = \varepsilon$;
- v. for all $\pi \sqsubseteq \pi' = C_1 \cdots C_{n'} \in \text{tree}(C)$ such that $C_{n'} = \{q\}$, if π_i denotes $C_1 \cdots C_i$ then:

$$\theta^{\text{nb}_{\pi_1}} \left(\prod_{i=2}^{n'} \text{out}_{\pi_i} \theta^{\text{nb}_{\pi_i}(q)} \right) = \theta^{n_{\pi_1}} \left(\prod_{i=2}^{n'} o_{\pi_i} \theta^{n_{\pi_i}(q)} \right);$$

- vi. for all $\pi' = C_1 \cdots C_{n'} \sqsubseteq \pi$ which is not close and was close before launching $\text{down}(C)$, let $J' := \text{pre}_{J,C}^{x[1:i]}(C_{n'}) \subseteq J$. Then $J', x[1:i], C_{n'}$ is an initial step and $|\text{max-adv}_{J',C_{n'}}^{x[1:i]}| \geq 4\Omega!$.

Proof. Invariant i. is clear since step 2 forces $\text{nb}_\pi = 4$ if $\text{nb}_\pi > 4$. Invariants ii., iv. and v. result from a simple but bureaucratic verification. For invariant iii., note that if $\text{nb}_{\pi C'}(q) \leftarrow \text{nb}_{\pi C'}(q) + (\text{nb}_\pi(q) - 4)$ is performed, then we had $\text{nb}_\pi(q) > 4$ (so now $\text{nb}_\pi(q) = 4$) before executing $\text{down}(C)$. Therefore using invariants ii. and iii., we conclude that we had $\text{nb}_{\pi'}(q) = 4$ for all $\pi' \sqsubseteq \pi$ before launching $\text{down}(\pi)$.

Let us show that invariant vi. is preserved. First, if $\pi' \sqsubset \pi$ is not close before $\text{down}(\pi)$ makes its recursive calls, then it was not close before launching $\text{down}(\pi)$. Assume now that π is not close, but was close before launching $\text{down}(C)$. By invariant ii. we conclude that π was close before launching $\text{down}(\pi)$. Hence it means that $\text{down}(\pi)$ performed a $\text{nb}_{\pi C'}(q) \leftarrow \text{nb}_{\pi C'}(q) + (\text{nb}_\pi(q) - 4)$. Hence for all $\pi' \sqsubseteq \pi$ we have $\text{nb}_{\pi'}(q) = 4$ by invariant iii.. Before executing the first “for” loop of $\text{down}(\pi)$, we had $\text{nb}_\pi(q') = 0$ for some $q' \in C'$, and thus $\text{nb}_{\pi\{q'\}}(q') = \text{nb}_\pi(q') = 0$ after this loop (by construction and since π was close). Let $\pi_i := C_1 \cdots C_i$ for $1 \leq i \leq n$. From invariant 4e of \mathcal{S} , and invariants iv. and v., we get:

$$\text{prod}_{J,C}^{x[1:i]}(q) = \text{out max-lag } \theta^{\text{nb}_{\pi_1}(q)} \left(\prod_{i=2}^n \text{out}_{\pi_i} \theta^{\text{nb}_{\pi_i}(q)} \right) \theta^{\text{nb}_{\pi\{q\}}(q)} \text{last}(q)$$

and similarly for q' (which may be lagging):

$$\text{prod}_{J,C}^{x[1:i]}(q') = \text{out lag}(q) \theta^{\text{nb}_{\pi_1}(q')} \left(\prod_{i=2}^n \text{out}_{\pi_i} \theta^{\text{nb}_{\pi_i}(q')} \right) \theta^{\text{nb}_{\pi\{q'\}}(q')} \text{last}(q')$$

Therefore we get:

$$\begin{aligned} |\text{max-adv}_{J,C}^{x[1:i]}| &\geq \left| |\text{prod}_{J,C}^{x[1:i]}(q)| - |\text{prod}_{J,C}^{x[1:i]}(q')| \right| \\ &= \left| \underbrace{|\text{max-lag}| - |\text{lag}(q')|}_{\geq 0} + \underbrace{|\text{last}(q)| - |\text{last}(q')| + \Omega! |\text{nb}_{\pi\{q\}}(q)|}_{\geq 0} \right| \\ &\quad + \Omega! \times \sum_{i=1}^n (4 - \text{nb}_{\pi_i}(q')) \geq \Omega!(4 - \text{nb}_\pi(q')) \geq 4\Omega! \end{aligned}$$

which concludes the proof. ◀

Let us conclude that the invariants of \mathcal{S} hold after applying this algorithm. Invariants 4a and 4c hold because we did not modify the $\text{lag}(q)$ and $\text{last}(q)$. Invariant 4b is clearly preserved along Algorithm 1. Invariant 4d follows from invariant iv.. Invariant 4e follows from invariant v. and invariant 4d. Invariant 4f is obvious since $J, x[1:i], C$ is unchanged. Finally, for invariant 4g, we use invariant vi. to deal with the paths $\pi \in \text{tree}(C)$ which became close when applying Algorithm 1.

Furthermore, $\text{nb}_\pi : C_n \rightarrow [0:4]$ for all $\pi = C_1 \cdots C_n \in \text{tree}(C)$ which conclude the proof.

J.3 Proof of Lemma 5.5

Since C was separable, then so is C_{i+1}^x (indeed, if the lengths of the runs which end in C can differ significantly, so are those obtained by adding an a -transition to C_{i+1}^x). Furthermore, it is clear that invariants 1 and 2 hold after this step. We now use J , lag , etc. to denote the values stored before the operation, and \bar{J} , $\bar{\text{lag}}$, etc. (with a tilde)

Invariants 4a, 4b and 4c. To show invariant 4a, we note that $k_q \neq 0$ if and only if $\text{pre}_{C, C_{i+1}^x}^a(q)$ was lagging. Thus we use invariants 4a, 4e and Lemma 4.8 to show that the lag are still mutual prefixes, and furthermore $\overline{\text{max-lag}} = c^{-1} \text{max-lag}$. For invariant 4b, it is sufficient to see that the $\overline{\text{out}_\pi}$ are obtained from the out_π . We just have to note that out_C is not used anywhere except for $\overline{\text{out}_{\bar{C}}}$. Indeed (with the notations of Subsubsection 5.3.1), if $\rho = \bar{C}$ and $i_m = n$, then $i_m = i_1 = 1 = n$ and by definition of $\text{tree}(C)$ we have $\pi = C$. For invariant 4c, we use invariant 4e and 4f which imply that for all $q \in \bar{C}$:

- if $\text{pre}_{C, C_{i+1}^x}^a(q)$ was lagging then $\text{prod}_{C, \bar{C}}^a(q) \sqsubseteq (\text{lag}(\text{pre}_{C, C_{i+1}^x}^a(q)))^{-1} \text{max-lag } \theta^\omega$;
- otherwise, $\text{prod}_{C, \bar{C}}^a(q) \sqsubseteq (\text{last}(\text{pre}_{C, C_{i+1}^x}^a(q)))^{-1} \theta^\omega$.

Thus $\text{prod}_{C, C_{i+1}^x}^a(q)[k_q+1:] \sqsubseteq (\text{last}(\text{pre}_{C, C_{i+1}^x}^a(q)))^{-1} \theta^\omega$ by invariant 4d, and so $\overline{\text{last}(q)} \sqsubseteq \theta^\omega$.

Invariant 4d. For invariant 4d, let us consider a lagging state $q \in \bar{C}$. Then $\text{pre}_{C, C_{i+1}^x}^a(q)$ was also lagging and $|\text{prod}_{C, C_{i+1}^x}^a(q)| < k_q$. Thus $\overline{\text{last}(q)} = \varepsilon$. Now, let $\pi = D_1 \cdots D_n \in \text{tree}(C_{i+1}^x)$ be such that $q \in D_n$, with the notations of Subsubsection 5.3.1 we have:

- either $i_m < n$, then $\overline{\text{nb}_\pi} = 0$ and $\overline{\text{out}_\pi} = \varepsilon$;
- or $i_m = n$. Since $C_{i_m} = C_n = \text{pre}_{C, C_{i+1}^x}^a(D_n)$, we get $\text{pre}_{C, C_{i+1}^x}^a(q) \in C_{i_m}$ and so $\overline{\text{nb}_\pi} = 0$.

Furthermore, if $D_n \neq \bar{C}$, then (see before) $C_{i_m} = C_n \neq C$ thus $\overline{\text{out}_\pi} = \varepsilon$.

Invariant 4e. The proof is similar to that of invariant 4c. 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 $\rho_j := C_{i_1} \cdots C_{i_j}$ for $1 \leq j \leq m$, we get:

- $\overline{\text{out}_{\pi_i}} = \varepsilon$ and $\overline{\text{nb}_{\pi_i}} = 0$ if $i \notin \{i_1, \dots, i_m\}$;
- $\overline{\text{out}_{\pi_{i_j}}} = \text{out}_{\rho_j}$ for $1 < j \leq m$ and $\overline{\text{nb}_{\pi_{i_j}}} = \text{nb}_{\rho_j} \circ \text{pre}_{C, \bar{C}}^a$ if $1 \leq j \leq m$

Hence $\theta^{\overline{\text{nb}_{\pi_1}(q)}} \prod_{i=2}^n \overline{\text{out}_{\pi_i}} \theta^{\overline{\text{nb}_{\pi_i}(q)}} = \theta^{\text{nb}_{\rho_1}(\text{pre}_{C, \bar{C}}^a(q))} \prod_{j=2}^m \text{out}_{\rho_j} \theta^{\text{nb}_{\rho_j}(\text{pre}_{C, \bar{C}}^a(q))}$.

Invariant 4f. Let \bar{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 $\text{out max-lag } \theta^\omega = \overline{\text{outmax-lag}} \theta^\omega$.

Invariant 4g. Let $\pi = D_1 \cdots D_n \in \text{tree}(\bar{C})$ be not close. With the notations of Subsubsection 5.3.1, we get $\rho = C_{i_1} \cdots C_{i_m} \in \text{tree}(C)$. Since C_{i_m}, a, C_n is a step, it is sufficient to show that ρ was not close. Let $\pi' = \pi D_{n+1} \cdots D_{n'}$ be such that $n' > n$ and $\overline{\text{out}_{\pi'}} \neq \varepsilon$ or

$\overline{\text{nb}_{\pi'}} \neq 0$. Then we have (with the notations of Subsubsection 5.3.1) $\rho' = \rho C_{i_{m+1}} \cdots C_{i_{m'}}$ and necessarily $i_{m'} = n'$ (which implies $m' > m$). Hence $\text{out}_{\rho'} = \overline{\text{out}_{\pi'}}$ and $\text{nb}_{\rho'} = \overline{\text{nb}_{\pi'}}$, showing that ρ was not close.

J.4 Proof of Lemma 5.7

Let us show that invariants 2 and 4 hold after this operation. We use \mathbf{J} , out , etc. to denote the values stored before the operation, and $\bar{\mathbf{J}}$, $\overline{\text{out}}$, etc. (with a bar) to denote them after. The case of invariant 2 is trivial because $\bar{\mathbf{C}} \in \text{Comp}$ and $\bar{\mathbf{J}} = \text{pre}_{x[1:i]}^{\mathbf{J}, \mathbf{C}}(\bar{\mathbf{C}})$. Furthermore, $\bar{\mathbf{C}}$ is separable by invariant 4g and Lemma H.4.

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

Since $C' = \bar{\mathbf{C}}$ was not close, there was some $p \in \bar{\mathbf{C}}$ which was not lagging, and so $\overline{\text{max-lag}} = c^{-1} \text{max-lag}$. Therefore some $q \in \bar{\mathbf{C}}$ is now lagging if and only if it was lagging before the operation. Let us consider such a $q \in \bar{\mathbf{C}}$ if it exists, and let $\pi = C_1 \cdots C_n \in \text{tree}(\bar{\mathbf{C}})$ be such that $q \in C_n$. The update gives $\overline{\text{nb}_{\pi}} = \text{nb}_{\mathbf{C}\pi}$ which must be null since q was lagging. The cases of out_{π} (for $\pi \neq \mathbf{C}$) and $\text{last}(q)$ are similar. Thus invariant 4d holds.

Invariant 4e. Let us now show invariant 4e. Two cases occur:

- if $q \in \bar{\mathbf{C}}$ is lagging, then it was lagging before and so we had $\text{prod}_{\mathbf{J}, \mathbf{C}}^{x[1:i]}(q) = \text{out } \text{lag}(q)$. Furthermore, since $q \in \bar{\mathbf{C}}$ was lagging we necessarily had $\text{out}_{\bar{\mathbf{C}}\bar{\mathbf{C}}} = \varepsilon$ by invariant 4d. Since the update gives $\overline{\text{out}} = \text{out } c \text{out}_{\bar{\mathbf{C}}\bar{\mathbf{C}}}$ and $\overline{\text{lag}(q)} = c^{-1} \text{lag}(q)$, we conclude that $\overline{\text{out } \text{lag}(q)} = \overline{\text{out}} \overline{\text{lag}(q)}$;
- now if $q \in \bar{\mathbf{C}}$ is not lagging, let $\bar{\pi} = C_1 \cdots C_n \in \text{tree}(\bar{\mathbf{C}})$ be such that $C_n = \{q\}$. Then let $\pi := \mathbf{C} C_1 \cdots C_n \in \text{tree}(\mathbf{C})$, if $\pi_i := \mathbf{C} C_1 \cdots C_i$ we had:

$$\text{prod}_{x[1:i]}^{\mathbf{J}, \mathbf{C}}(q) = \text{out } \text{max-lag } \theta^{\text{nb}_{\mathbf{C}}(q)} \left(\prod_{i=1}^n \text{out}_{\pi_i} \theta^{\text{nb}_{\pi_i}(q)} \right) \text{last}(q).$$

Now if $\bar{\pi}_i := C_1 \cdots C_i$, the update gives $\overline{\text{out}_{\bar{\pi}_i}} = \text{out}_{\pi_i}$ and $\overline{\text{nb}_{\bar{\pi}_i}} = \text{nb}_{\pi_i}$ if $i > 1$; and $\overline{\text{out}} \leftarrow \text{out } \text{out}_{\pi_1}$ and $\overline{\text{nb}_{\bar{\pi}_i}} \leftarrow \text{nb}_{\pi_i}$ for $i = 1$. Therefore we have:

$$\text{prod}_{x[1:i]}^{\mathbf{J}, \mathbf{C}}(q) = \text{out } \text{max-lag } \left(\prod_{i=2}^n \overline{\text{out}_{\bar{\pi}_i}} \theta^{\overline{\text{nb}_{\bar{\pi}_i}}(q)} \right) \theta^{\text{nb}_{\mathbf{C}}(q)} \text{last}(q).$$

The result follows by update of $\overline{\text{last}(q)}$ and since $\overline{\text{max-lag}} = c^{-1} \text{max-lag}$.

Invariant 4f. Let $\bar{\mathbf{C}}, u, D$ be a step, we show that for all $q \in D$: $\text{prod}_{\bar{\mathbf{J}}, D}^{x[1:i]u}(q) \subseteq \overline{\text{out } \text{max-lag}} \theta^{\omega}$.

By invariant 4g, we can decompose the step $\bar{\mathbf{J}}, x[1:i], \bar{\mathbf{C}}$ in an initial step $\bar{\mathbf{J}}, x[1:j], E$ and a step $E, x[j+1:i], \bar{\mathbf{C}}$ such that $|\text{max-adv}_{\bar{\mathbf{J}}, E}^{x[1:j]}| \geq 4\Omega!$. By applying Lemma 4.12 to $\bar{\mathbf{J}}, x[1:j], E$, it follows that there exists φ, τ, γ with $|\varphi| = \Omega!$ and $|\tau| \leq 3\Omega$ such that $\text{max-adv}_{\bar{\mathbf{J}}, E}^{x[1:j]} = \tau\gamma$ (thus $|\gamma| \geq \Omega!$), $\gamma \subseteq \varphi^{\omega}$ and for all step $\bar{\mathbf{C}}, u, D$ and $q \in D$:

$$\text{prod}_{E, D}^{x[j+1:i]u}(q) \subseteq (\text{adv}_{\bar{\mathbf{J}}, E}^{x[1:j]}(\text{pre}_{E, D}^{x[j+1:i]u}(q)))^{-1} \tau \varphi^{\omega}.$$

and therefore by adding $\text{prod}_{\bar{\mathbf{J}}, E}^{x[1:j]}(\text{pre}_{E, D}^{x[j+1:i]u}(q))$ on both sides:

$$\text{prod}_{\bar{\mathbf{J}}, D}^{x[1:i]u}(q) \subseteq \text{com}_{\bar{\mathbf{J}}, D}^{x[1:j]} \tau \varphi^{\omega}. \quad (3)$$

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

► **Sublemma J.2.** $\text{com}_{J,D}^{x[1:j]} \tau \varphi^\omega = \overline{\text{out}} \overline{\text{max-lag}} \theta^\omega$.

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

$$\text{prod}_{J,C}^{x[1:i]}(q) = \text{prod}_{J,\bar{C}}^{x[1:i]}(q) = \overline{\text{out}} \overline{\text{max-lag}} \theta^{n(q)} \overline{\text{last}(q)}.$$

for some $n(q) > 0$. Let us consider the $q \in \bar{C}$ such that $(n(q), |\text{last}(q)|)$ is maximal. Then for all $p \in \bar{C}$ we get $\text{prod}_{J,\bar{C}}^{x[1:i]}(p) \subseteq \text{prod}_{J,\bar{C}}^{x[1:i]}(q)$, thus (use Equation 3 for the right handside):

$$\text{com}_{J',E}^{x[1:j]} \tau \gamma = \text{com}_{J',E}^{x[1:j]} \text{max-adv}_{J',E}^{x[1:j]} \subseteq \overline{\text{out}} \overline{\text{max-lag}} \theta^{n(q)} \overline{\text{last}(q)} \subseteq \text{com}_{J',D}^{x[1:j]} \tau \varphi^\omega. \quad (4)$$

Two cases can occur depending on $k := |\overline{\text{out}} \overline{\text{max-lag}}| - |\text{com}_{J',D}^{x[1:j]} \tau|$

- either $k \geq 0$, then $\overline{\text{out}} \overline{\text{max-lag}} = \text{com}_{J',D}^{x[1:j]} \tau(\varphi^\omega[1:k])$ and $\theta = \varphi^\omega[k+1 : k+1+\Omega!]$;
- or $k < 0$, then $\overline{\text{out}} \overline{\text{max-lag}}((\theta^{n(q)} \overline{\text{last}(q)})[k:]) = \text{com}_{J',D}^{x[1:j]} \tau$ and $\gamma \subseteq \theta^{n(q)} \overline{\text{last}(q)}[k+1:]$. But since $|\gamma| \geq \Omega!$ and $\gamma \subseteq \varphi^\omega$, we conclude that $\varphi = \theta^\omega[k+1 : k+1+\Omega!]$.

In both cases, we conclude that $\text{com}_{J,D}^{x[1:j]} \tau \varphi^\omega = \overline{\text{out}} \overline{\text{max-lag}} \theta^\omega$. ◀

Invariant 4g. Let us consider $\bar{\pi} = C_1 \cdots C_n \in \text{tree}(\bar{C})$, $\bar{\pi}$ not close after the operation. Then there exists $\bar{\pi} \sqsubset \bar{\pi}' = C_1 \cdots C_{n'} \in \text{tree}(\bar{C})$ such that $\text{nb}_{\bar{\pi}'} \neq 0$ or $\text{out}_{\bar{\pi}'} \neq \varepsilon$. Let $\pi' := C C_1 \cdots C_{n'}$, then $C\bar{C} \sqsubset \pi'$, and by the updates we get $\text{nb}_{\pi'} = \text{nb}_{\bar{\pi}'}$ and $\text{out}_{\pi'} = \text{out}_{\bar{\pi}'}$. Finally, since invariant 4g held before the operation, it still holds.

K Proofs of section 5.4: boundedness and productivity of \mathcal{S}

For $x \in A^\omega$ and $i \geq 0$, we denote by $\llbracket \text{out}_\pi \rrbracket_i^x$, etc. the values of the registers of \mathcal{S} after reading $C_0^x \cdots x[i]C_i^x$ (when defined).

K.1 Proof of Lemma 5.9: 1-boundedness of \mathcal{S}

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

► **Sublemma K.1.** For all $\pi \in \text{tree}(C_i^x)$, $\llbracket \text{out}_\pi \rrbracket_i^x$ is 1-bounded. Furthermore, if $\rho \in \text{tree}(C_i^x)$ is such that $\pi \sqsubset \rho$, then $\llbracket \text{out}_\pi \rrbracket_i^x$ and $\llbracket \text{out}_\rho \rrbracket_i^x$ 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 $\llbracket \text{out}_\pi \rrbracket_{i+1}^x$ is obtained using $\llbracket \text{out}_\pi \rrbracket_i^x$ once, plus constant values. Furthermore, if C_{i+1}^x is not separable, then each value $\llbracket \text{out}_\pi \rrbracket_{i+1}^x$ 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 π is close, then the situation is similar to that of Subsection 5.1, except that we may use $\text{out}_{C'}$

to update **out**. However, since $\llbracket \text{out}_{C'} \rrbracket_i^x$ and $\llbracket \text{out} \rrbracket_i^x = \llbracket \text{out}_C \rrbracket_i^x$ are 1-bounded and have no shared memory by induction hypothesis, then the resulting value of **out** is 1-bounded. Now if π is not close, the argument for **out** is similar. Furthermore, we update $\text{out}_{C'\pi} \leftarrow \text{out}_{C C'\pi}$ for $\pi \neq \varepsilon$, which clearly preserves the fact that these registers are 1-bounded and have no shared memory. Furthermore, they also have no shared memory with $\llbracket \text{out} \rrbracket_{i+1}^x$.

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 $n' < n$, then $\llbracket \text{out}_{\pi_n} \rrbracket_{i+1}^x$ and $\llbracket \text{out}_{\pi_{n'}} \rrbracket_{i+1}^x$ have no shared memory. Let $1 = i_1 < \cdots < i_m \leq n$ be given by Subsubsection 5.3.1 for π_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 $g(x) = C_0^x x[1] \cdots$, we have $|\llbracket \text{out} \rrbracket_i^x| \rightarrow \infty$. 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 $\llbracket \text{out} \rrbracket_i^x = \text{com}_{J,C}^{x[1:i]}$, and the $\alpha_q = \text{adv}_{J,C}^{x[1:i]}(q)$ have a size bounded by $\Omega + 2 \times 4\Omega!$. Since $\text{prod}_{J,C}^{x[1:i]}(q_i^x)$ tends to $f(x)$, we conclude that **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 **out** when performing these 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 $u' \neq \varepsilon$, $\text{pre}_{S,S}^{u'} : S \rightarrow S$ is the identity function, and some $q'_1 \in S$ is accepting. Then $\text{prod}_{S,S}^{u'}(q) \neq \varepsilon$ for all $q \in S$.*

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 $q_i \xrightarrow{u|\alpha_i} q'_i \xrightarrow{u'|\text{prod}_{S,S}^{u'}(q'_i)} q'_i$ for $i \in \{1, 2\}$. These are the conditions of Lemma 4.3, hence since \mathcal{T} is productive we get $\text{prod}_{S,S}^{u'}(q'_2) \neq \varepsilon$. ◀

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

► **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 $q_i^x = \text{pre}_{C_i^x, C_{i'}^x}^{x[i+1:i']}(q_{i'}^x)$ 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 $p_{i'} = \text{pre}_{C_{i'}^x, C_{i'+1}^x}^{x[i'+1:i'+1]}(p_{i'+1})$ 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 $p_{i'} = \text{pre}_{C_{i'}^x, C_{i'+1}^x}^{x[i'+1:i'+1]}(p_{i'+1})$. By König’s lemma (see **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 $\text{pre}_{C_i^x, C_{i+1}^x}^{x[i+1:i+1]}(S_{i+1}^x) = S_i^x$, 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 $\text{pre}_{C_i^x, C_{i'}^x}^{x[i+1:i']}(C_{i'}^x) = S_i^x$.*

Proof. Since $C_{i'}^x, x[i'+1]$, $C_{i'+1}^x$ is a pre-step, $\text{pre}_{C_i^x, C_{i'}^x}^{x[i+1:i']}(C_{i'}^x) \supseteq \text{pre}_{C_i^x, C_{i'}^x}^{x[i+1:i'+1]}(C_{i'+1}^x)$. Hence $(\text{pre}_{C_i^x, C_{i'}^x}^{x[i+1:i']}(C_{i'}^x))_{i' \geq i}$ 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 $N \leq \ell_1 < \ell_2 < \dots$ such that $S_{\ell_1}^x = S_{\ell_2}^x = \dots =: S$, $q_{\ell_1}^x = q_{\ell_2}^x = \dots \in F$ and $\text{pre}_{S, S}^{x[\ell_j+1:\ell_{j'}]}$ is the identity function for all $j \leq j'$.*

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

By sublemmas K.2 and K.6, we get $\text{prod}_{C_{\ell_j}^x, C_{\ell_{j+1}}^x}^{x[\ell_j+1:\ell_{j+1}]}(q) = \text{prod}_{S, S}^{x[\ell_j+1:\ell_{j+1}]}(q) \neq \varepsilon$ for all $j \geq 1$ and all $q \in S$. Therefore $|\text{prod}_{C_{\ell_1}^x, C_{\ell_{1+K}}^x}^{x[\ell_1+1:\ell_{1+K}]}(q)| \geq K$ for all $K \geq 1$ and $q \in S$.

Let us now fix $K := 4\Omega!$, $i := \ell_1$ and $i' := \ell_{1+K}$. By Sublemma K.5, there exists $i'' \geq i$ such that $\text{pre}_{C_{i'}^x, C_{i''}^x}^{x[i'+1:i'']}(C_{i''}^x) = S_{i'}^x = S$. Hence for all $q \in C_{i''}^x$, if $q' := \text{pre}_{C_{i'}^x, C_{i''}^x}^{x[i'+1:i'']}(C_{i''}^x)$ we get $|\text{prod}_{C_{i'}^x, C_{i''}^x}^{x[i'+1:i'']}(q)| = |\text{prod}_{S, S}^{x[i'+1:i'']}(q') \text{prod}_{C_{i'}^x, C_{i''}^x}^{x[i'+1:i'']}(q)| \geq 4\Omega!$.

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

► **Sublemma K.7.** *Let $N \leq i_1 \leq i_2$. Let $q_2 \in C_{i_2}^x$ be such that $\llbracket \text{nb}_{C_{i_2}^x} \rrbracket_{i_2}^x(q_2) = 0$. Let us define $q_1 := \text{pre}_{C_{i_1}^x, C_{i_2}^x}^{x[i_1+1:i_2]}(q_2)$, then $\llbracket \text{nb}_{C_{i_1}^x} \rrbracket_{i_1}^x(q_1) = 0$ and:*

$$|\text{prod}_{C_{i_1}^x, C_{i_2}^x}^{x[i_1+1:i_2]}(q_2)| = |\llbracket \text{last} \rrbracket_{i_2}^x(q_2)| - |\llbracket \text{last} \rrbracket_{i_1}^x(q_1)| + |\llbracket \text{lag} \rrbracket_{i_2}^x(q_2)| - |\llbracket \text{lag} \rrbracket_{i_1}^x(q_1)|.$$

► **Remark K.8.** In particular, we get $|\text{prod}_{C_{i_1}^x, C_{i_2}^x}^{x[i_1+1:i_2]}(q_2)| < 4\Omega!$.

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 $\llbracket \text{nb}_{C_{i_2}^x} \rrbracket_{i_2}^x(q_2) = 0$ and $q_1 := \text{pre}_{C_{i_1}^x, C_{i_2}^x}^{x[i_1+1:i_2]}(q_2)$. By induction hypothesis, Sublemma K.7 holds. Now let us consider the transition of \mathcal{S} from $C_{i_0}^x$ 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 $\text{nb}'_{\pi}, \text{out}'_{\pi}$, etc. denote the configuration of \mathcal{S} right after Lemma 5.7 in Subsubsection 5.3.1. Since Algorithm 1 has not modified $\text{out} = \text{out}_{C_{i_1}^x}$, we had $m = 0$ in $\text{down}(C_{i_1}^x)$. Thus we had $\text{nb}'_{C_{i_1}^x}(q_1) + n(q_1) = 0$ (because if $m = 0$ then $\llbracket \text{nb}_{C_{i_1}^x} \rrbracket_{i_1}^x(q_1) = \min(\text{nb}'_{C_{i_1}^x}(q_1) + n(q_1), 4)$). Therefore $\text{nb}'_{C_{i_1}^x}(q_1) = 0$. Furthermore $n(q_1) = 0$ and so $\text{last}'(q_1) = \llbracket \text{last}(q_1) \rrbracket_{i_1}^x$. Furthermore, $\text{lag}'(q_1) = \llbracket \text{lag}(q_1) \rrbracket_{i_1}^x$.

Previous operation: beginning of Subsubsection 5.3.1. Now let $\text{nb}''_\pi, \text{out}''_\pi$, etc. denote the configuration of \mathcal{S} before applying the whole Subsubsection 5.3.1. Since by construction $\mathbf{C} = \text{pre}_{C_{i_0}^x, C_{i_1}^x}^{x[i_1]}(C_{i_1}^x)$, then $\text{nb}'_{C_{i_1}^x} = \text{nb}''_{\mathbf{C}} \circ \text{pre}_{\mathbf{C}, C_{i_1}^x}^{x[i_1]}$. Thus $\text{nb}''_{\mathbf{C}}(q_0) = 0$ if $q_0 := \text{pre}_{\mathbf{C}, C_{i_1}^x}^{x[i_1]}(q_1)$.

Finally, we note that $c = \varepsilon$ since there is no output. As a consequence, it is quite easy to see that $|\text{prod}_{C_{i_0}^x, C_{i_1}^x}^{x[i_1]}(q_1)| = |\text{prod}_{\mathbf{C}, C_{i_1}^x}^{x[i_1]}(q_1)| = |\text{last}'(q_1)| - |\text{last}''(q_0)| + |\text{lag}'(q_1)| - |\text{lag}''(q_0)|$.

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

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