



HAL
open science

Reversible Transducers over Infinite Words

Luc Dartois, Paul Gastin, Germerie Loïc, R. Govind, Shankaranarayanan
Krishna

► **To cite this version:**

Luc Dartois, Paul Gastin, Germerie Loïc, R. Govind, Shankaranarayanan Krishna. Reversible Transducers over Infinite Words. CONCUR 24, Sep 2024, Calgary, Canada. 10.4230/LIPIcs.CONCUR.2024.21 . hal-04708807

HAL Id: hal-04708807

<https://hal.science/hal-04708807v1>

Submitted on 25 Sep 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Reversible Transducers over Infinite Words

Luc Dartois ✉ 

Université Paris Est Creteil, LACL, F-94010 Créteil, France

Paul Gastin ✉ 

Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, 91190, Gif-sur-Yvette, France
CNRS, ReLaX, IRL 2000, Siruseri, India

Loïc Germerie Guizouarn ✉ 

Université Paris Est Creteil, LACL, F-94010 Créteil, France

R. Govind ✉ 

Uppsala University, Sweden

Shankaranarayanan Krishna ✉ 

Indian Institute of Technology Bombay, Mumbai, India

Abstract

Deterministic two-way transducers capture the class of regular functions. The efficiency of composing two-way transducers has a direct implication in algorithmic problems related to reactive synthesis, where transformation specifications are converted into equivalent transducers. These specifications are presented in a modular way, and composing the resultant machines simulates the full specification. An important result by Dartois et al. [8] shows that composition of two-way transducers enjoy a polynomial composition when the underlying transducer is reversible, that is, if they are both deterministic and co-deterministic. This is a major improvement over general deterministic two-way transducers, for which composition causes a doubly exponential blow-up in the size of the inputs in general. Moreover, they show that reversible two-way transducers have the same expressiveness as deterministic two-way transducers. However, the question of expressiveness of reversible transducers over infinite words is still open.

In this article, we introduce the class of reversible two-way transducers over infinite words and show that they enjoy the same expressive power as deterministic two-way transducers over infinite words. This is done through a non-trivial, effective construction inducing a single exponential blow-up in the set of states. Further, we also prove that composing two reversible two-way transducers over infinite words incurs only a polynomial complexity, thereby providing foundations for efficient procedure for composition of transducers over infinite words.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases Transducers, Regular functions, Reversibility, Composition, SSTs

Introduction

Transducers extend finite state automata with outputs. While finite state automata are computational models for regular languages, transducers are computational models for transformations between languages. Finite state automata remain robust in their expressiveness accepting regular languages across various descriptions like allowing two-way-ness, non-determinism and otherwise. However, this is not the case with transducers. Non-deterministic transducers realize relations while deterministic transducers realize functions. Likewise, two-way transducers are strictly more expressive than one-way transducers: for instance, the function reverse which computes the reverse of all input words in its domain is realizable by deterministic two-way transducers, but not by one-way transducers.

One of the cornerstone results of formal language theory is the beautiful connection which establishes that the class of regular languages corresponds to those recognized by

finite state automata, to the class of languages definable in MSO logic, and to the class of languages whose syntactic monoid is finite. Engelfriet and Hoogeboom [14] generalized this correspondence between machines, logics and algebra in the case of regular languages to regular transformations. They showed that regular transformations are those which are captured by two-way transducers and by Monadic second-order (MSO) transductions a la Courcelle [7]. Inspired by this seminal work of Engelfriet and Hoogeboom, there has been an increasing interest over recent years in characterizing the class of functions defined by deterministic two-way transducers [2, 3, 13].

One such characterization is that of *reversible* two-way transducers [8] over finite words. Reversible transducers are those which are deterministic and also co-deterministic. While determinism says that any given state, on any given input symbol, does not transition to two distinct states, co-determinism says that no two distinct states can transition to the same state on any input symbol. Reversibility makes the composition operation in two-way transducers very efficient : the composition of reversible transducers has polynomial state complexity. This makes reversible transducers a very attractive formalism in synthesis where specifications are given as relations of input-output pairs. [8] showed that reversible two-way transducers capture the class of regular transformations. However, reversible transducers over infinite words have not been studied.

In another line of work, [1] initiated the study of transformations on infinite words. They considered functional, copy-less streaming string transducers (SST) with a Müller acceptance condition. An SST is a one-way automaton with registers; the outputs of each transition are stored in registers as words over the register names and the output alphabet. In a run, the contents of the registers are composed. The Müller acceptance condition is defined as follows: in any accepting run which settles down in a Müller set, the output is defined as a concatenation x_1, x_2, \dots, x_n of registers where only x_n is updated by appending words to x_n . [1] proved the equivalence of this class of SST to deterministic two-way transducers with Müller acceptance and having an ω -regular look-ahead. They also showed that these are equivalent to MSO transductions over infinite words. The ω -regular look-aheads were necessary to obtain the expressiveness of MSO transductions on infinite words.

In this paper, we continue the study of two-way transducers over infinite words. We introduce two-way transducers with the parity acceptance condition (2DPT). The main result of our paper is a non-trivial generalization of [8], where we show that 2DPT's can be made reversible, obtaining 2RPT (two-way reversible transducers with parity acceptance). Our conversion of 2DPT to 2RPT incurs a single exponential blow-up, and goes via a new kind of SSTs that we introduce, namely, copyless SST with a parity acceptance condition (cPSST). The parity condition used in both machines employs a finite set of coloring functions, c_1, \dots, c_k , where each c_i assigns to the transitions of the underlying machine, a natural number. An infinite run ρ is accepting if the minimum number which appears infinitely often is even in all the c_i 's.

1. We first show that starting from a 2DPT, we can obtain an equivalent cPSST where the number of states of the cPSST is exponential in the number of states and coloring functions of the 2DPT. The proof of this is a fairly non-trivial generalization of the classical Shepherdson construction [16] which goes from two-way automata to one-way automata.
2. Then we show that, starting from a cPSST \mathcal{A} , we can obtain an equivalent 2RPT \mathcal{B} which is polynomial in the number of states and registers of \mathcal{A} . This construction is a bit technical : we show that \mathcal{B} is obtained as the composition of a deterministic one-way parity transducer \mathcal{D} and an 2RPT \mathcal{F} . To complete the proof, we show that (i) \mathcal{D} can be

converted to an equivalent 2RPT with polynomial blow-up, and (ii) 2RPTs are closed under composition with a polynomial complexity.

Thus, our results extend [8] to the setting of infinite words, retaining the expressivity of deterministic machines and a polynomial complexity for composition. The main challenges when going to the infinite word setting is in dealing with the acceptance conditions. Unlike the finite word setting where acceptance is something to take care of at the end, here we need to deal with it throughout the run. The difficulty in doing this comes from the fact that we cannot compute the set of co-accessible states at a given input position. It must be noted that in the proof [8] for finite words, the equivalent reversible transducer was constructed by computing the set of accessible and co-accessible states at each position of the input word. Indeed, computing the co-accessible states at each input position requires an infinite computation or an oracle, and hence, the proof of [8] fails for infinite words. Instead, we introduce the intermediate model of cPSST where we employ a dedicated “out” register that serves as the output tape.

Our result of extending [8] can be seen as a positive contribution to reactive synthesis : transforming specifications over infinite word transformations to an equivalent 2RPT can give rise to efficient solutions to algorithmic problems on transformation specifications. To the best of our knowledge, there is no such translation for infinite words; the closest result in this direction, but for finite words, is [9], which gives an efficient procedure for converting specifications given as RTE (regular transducer expressions) to reversible transducers.

Continuing with transformations on infinite words, [12] investigated a practical question on functions over infinite words, namely, “given a function over infinite words, is it computable?”. They established that the decidability of this question boils down to checking the continuity of these functions. Further, they conjectured that any continuous regular function can be computed by a deterministic two-way transducer over infinite words without ω -regular look-ahead. [4] took up this conjecture and showed that any continuous rational function over infinite words can be extended to a function which is computable by deterministic two-way transducers over infinite words without ω -regular look-ahead. Most recently, [5] conjectured that deterministic two-way transducers with the Büchi acceptance condition capture the class of continuous, regular functions.

Apart from its application to synthesis, 2DPT also realize continuous functions. This implies that the conjecture of [5] fails, since 2DPT are more expressive than the class of deterministic two-way transducers with Büchi acceptance. A simple example illustrating this is the function $f : \{a, b\}^\omega \rightarrow \{a, b\}^\omega$ such that $f(u) = u$ if the number of a 's in u is finite, and is undefined otherwise. f is continuous since it is continuous on its domain; f cannot be realized by a deterministic transducer with Büchi acceptance, but it can be realized by a 2DPT. Note however that the extension is only able to refine the domain, and not the production. In particular, by simply dropping the accepting condition of an 2RPT, we obtain a function realized by a deterministic two-way transducer with a Büchi condition. Moreover, our constructions (going from deterministic two-way to reversible) for this class become simpler. And conversely, we show that two-way reversible transducers with no acceptance condition have the same expressiveness as those with the Büchi acceptance condition, which in turn have the same expressiveness as two-way deterministic transducers with Büchi acceptance.

Organization of the Paper. Section 2 defines the two models we introduce in the paper, namely, cPSST and 2DPT. Section 3 states our main result : starting from a 2DPT, we can obtain an equivalent 2RPT. Most of the remaining sections are devoted to the proof

of this result. In sections 4 and 5 respectively, we prove the closure under composition of 2RPTs with a polynomial complexity and the polynomial conversion from one-way parity transducers to 2RPT. Section 6 uses both these results, where we describe the conversion from cPSST to 2RPT with a polynomial complexity. Section 7 contains one of the most non-trivial constructions of the paper, namely, going from 2DPT to cPSST with a single exponential blow-up. Finally, Section 8 wraps up by discussing the connection between continuity and the topological closure of 2DPTs.

2 Preliminaries

Let A be an *alphabet*, i.e., a finite set of letters. A finite or infinite word w over A is a (possibly empty) sequence $w = a_0a_1a_2 \cdots$ of letters $a_i \in A$. The set of all finite (resp. infinite) words is denoted by A^* (resp. A^ω), with ε denoting the empty word. We let $A^\infty = A^* \cup A^\omega$. A *language* is a subset of the set of all words.

Two-way Parity Automata and Transducers

Let A be a finite alphabet and let $\vdash \notin A$ be a left delimiter symbol. We write $A_\vdash = A \cup \{\vdash\}$.

A two-way parity automaton (2PA) is a tuple $\mathcal{A} = (Q, A, \Delta, q_0, \chi)$, where the finite set of states Q is partitioned into a set of forward states Q^+ and a set of backward states Q^- . The initial states is $q_0 \in Q^+$, $\Delta \subseteq Q \times A_\vdash \times Q$ is the transition relation and χ is a finite set of coloring functions $c: \Delta \rightarrow \mathbb{N}$ which are used to define the acceptance condition. We assume that if $(p, \vdash, q) \in \Delta$, then $p \in Q^-$ and $q \in Q^+$: on reading \vdash , the reading head does not move.

A configuration of a 2PA over an input word $w \in A^\omega$ is some $\vdash u p v$ where $p \in Q$ is the current state and $u \in A^*$, $v \in A^\omega$ with $w = uv$. The configuration admits several successor configurations as defined below.

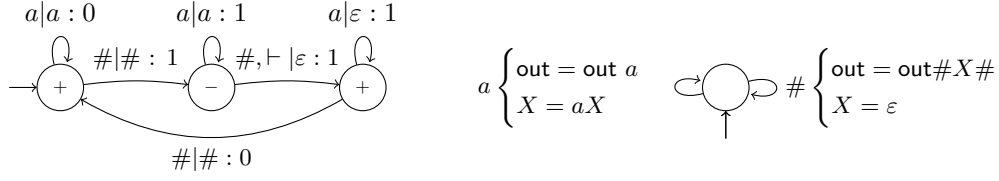
1. If $p \in Q^+$, then the input head reads the first symbol $a \in A$ of the suffix $v = av' \in A^\omega$. Let $(p, a, q) \in \Delta$ be a transition. If $q \in Q^+$, then the successor configuration is $\vdash ua q v'$. Likewise, if $q \in Q^-$, then the successor configuration is $\vdash u q av'$. Thus, if the current and target states are both in Q^+ , then the reading head moves right. If the current state is forward and the target state is backward, then the reading head does not move.
2. If $p \in Q^-$, then the input head reads the last symbol $a \in A_\vdash$ of the prefix $\vdash u$. Let $(p, a, q) \in \Delta$ be a transition. If $q \in Q^+$, the successor configuration is $\vdash u q v$. If $q \in Q^-$ then $a \neq \vdash$, we write $u = u'a$ with $u' \in A^*$ and the successor configuration is $\vdash u' q av$. Thus, if the current state is backward and the target state is forward, the reading head does not move. If both states are backward, then the reading head moves left.

A run ρ of \mathcal{A} is a finite or infinite sequence of configurations starting from an initial configuration $\vdash \varepsilon q_0 w$ where $w \in A^\omega$ is the input word:

$$\vdash q_0 w = \vdash u_0 q_0 v_0 \rightarrow \vdash u_1 q_1 v_1 \rightarrow \vdash u_2 q_2 v_2 \rightarrow \vdash u_3 q_3 v_3 \rightarrow \vdash u_4 q_4 v_4 \cdots$$

We say that ρ reads the whole word $w \in A^\omega$ if $\sup\{|u_n| \mid n > 0\} = \infty$. The set of transitions used by ρ infinitely often is denoted $\text{inf}(\rho) \subseteq \Delta$. The word w is accepted by \mathcal{A} , i.e., $w \in \text{dom}(\mathcal{A})$ if ρ reads the whole word w and $\min(c(\text{inf}(\rho)))$ is even for all $c \in \chi$. Note that, even though we call our machine parity, we in fact consider a conjunction of parity conditions. This allows to easily describe intersection of automata or composition of transducers.

The parity automaton \mathcal{A} is called



■ **Figure 1** An example of 2RPT (left) and cPSST (right) defining the function *map-copy-reverse* (*mcr*) defined on $(A \uplus \{\#\})^\omega \rightarrow (A \uplus \{\#\})^\omega$ by: $mcr(u_1\#u_2\#\dots) = u_1\#\tilde{u}_1\#u_2\#\tilde{u}_2\#\dots$ for words with an infinite number of letter #, and $mcr(u_1\#\dots\#u_n\#u) = u_1\#\tilde{u}_1\#\dots\#u_n\#\tilde{u}_n\#u$ if $u \in A^\omega$, where \tilde{v} denotes the mirror image of v . There is only one coloring function, denoted on the transitions after the colon. The color of all transitions of the cPSST is 0.

- *one-way* if $Q^- = \emptyset$,
- *deterministic* if for all pairs $(p, a) \in Q \times A_+$, there is at most one state $q = \delta(p, a)$ such that $(p, a, q) \in \Delta$, in this case we identify the transition relation Δ with the partial function $\delta: Q \times A \rightarrow Q$,
- *co-deterministic* if for all pairs $(q, a) \in Q \times A_+$, there is at most one state p such that $(p, a, q) \in \Delta$,
- *reversible* if it is both deterministic and co-deterministic.

A two-way parity transducer 2PT is a tuple $\mathcal{T} = (Q, A, \Delta, q_0, \chi, B, \lambda)$ where $\mathcal{A} = (Q, A, \Delta, q_0, \chi)$ is a *deterministic* 2PA, called the underlying parity automaton of \mathcal{T} , B is a finite *output* alphabet, and $\lambda: \Delta \rightarrow B^*$ is the output function. As in the case of 2PA, a 2PT is one-way/co-deterministic/reversible if so is the underlying parity automaton. Let 2DPT (resp. 2RPT) denote two-way (deterministic) (resp. reversible) parity transducers. The notion of run and accepting run is inherited from the underlying 2PA. For $w \in A^\omega$ such that $w \in \text{dom}(\mathcal{A})$, let the accepting run ρ of w be

$$\vdash q_0 w = \vdash u_0 q_0 v_0 \xrightarrow{t_1} \vdash u_1 q_1 v_1 \xrightarrow{t_2} \vdash u_2 q_2 v_2 \xrightarrow{t_3} \vdash u_3 q_3 v_3 \xrightarrow{t_4} \vdash u_4 q_4 v_4 \dots$$

where $t_i \in \Delta$ is the i -th transition taken during the run, i.e., from $\vdash u_{i-1} q_{i-1} v_{i-1}$ to $\vdash u_i q_i v_i$. For $i > 0$, let $\gamma_i = \lambda(t_i)$ be the output produced by the i -th transition of ρ . If $\gamma_1 \gamma_2 \gamma_3 \gamma_4 \dots \in B^\omega$, then $w \in \text{dom}(\mathcal{T})$ and we let $\llbracket \mathcal{T} \rrbracket(w) = \gamma_1 \gamma_2 \gamma_3 \gamma_4 \dots$ be the output word computed by \mathcal{T} . Hence, the semantics of a 2PT is a partial function $\llbracket \mathcal{T} \rrbracket: A^\omega \rightarrow B^\omega$ with $\text{dom}(\mathcal{T}) \subseteq \text{dom}(\mathcal{A})$.

Parity Streaming String Transducers

Let R be a finite set of variables called *registers*. A *substitution* of R into an alphabet B is a mapping $\sigma: R \rightarrow (R \uplus B)^*$. It is called *copyless* if for all $r \in R$, r appears at most once in the concatenation of all the $\sigma(r')$ for $r' \in R$. We denote by Λ_R^B the set of all copyless substitutions of R into B .

A copyless parity Streaming String Transducers (cPSST) is given by a tuple $\mathcal{T} = (Q, A, \Delta, q_0, \chi, B, R, \text{out}, \lambda)$ where $\mathcal{A} = (Q, A, \Delta, q_0, \chi)$ is a deterministic one-way parity automaton called the underlying parity automaton of \mathcal{T} , R is a finite set of registers, $\text{out} \in R$ is a distinguished register, called the output register, $\lambda: \Delta \rightarrow \Lambda_R^B$ is the update function satisfying additionally $\lambda(t)(\text{out}) \in \text{out} \cdot (R \uplus B)^*$ for all $t \in \Delta$.

A configuration of a copyless parity SST \mathcal{T} is a tuple (q, ν) where $q \in Q$ and $\nu: R \rightarrow B^*$ is an assignment. The initial configuration is (q_0, ν_0) where $\nu_0(r) = \varepsilon$ for all $r \in R$. Since the automaton \mathcal{A} is *deterministic*, we simply describe a run on an input word $w = a_0 a_1 a_2 \dots$

as a sequence a sequence of transitions applying the corresponding substitutions to the assignments:

$$(q_0, \nu_0) \xrightarrow{a_0} (q_1, \nu_1) \xrightarrow{a_1} (q_2, \nu_2) \xrightarrow{a_2} (q_3, \nu_3) \cdots$$

where (q_0, ν_0) is the initial configuration and for all $i \geq 0$ we have $t_i = (q_i, a_i, q_{i+1}) \in \Delta$ and $\nu_{i+1} = \nu_i \circ \lambda(t_i)$ ¹. Notice that, from the restriction of the update function, we deduce that $\nu_0(\text{out}), \nu_1(\text{out}), \nu_2(\text{out}), \dots$ is a (weakly) increasing sequence of output words in B^* . If this sequence is unbounded then $w \in \text{dom}(\mathcal{T})$ and we let $\llbracket \mathcal{T} \rrbracket(w) = \bigsqcup_{i \geq 0} \nu_i(\text{out}) \in B^\omega$ be the limit (least upper-bound) of this sequence. Hence, the semantics of a cPSST is a partial function $\llbracket \mathcal{T} \rrbracket: A^\omega \rightarrow B^\omega$ with $\text{dom}(\mathcal{T}) \subseteq \text{dom}(\mathcal{A})$.

3 Main Result

We are now ready to state our main result, which is an effective procedure to construct a reversible two-way transducer for a deterministic machine. Our result is stated using a conjunction of parity conditions.

The proof relies on constructions that go through cPSST, and are presented in the subsequent sections.

► **Theorem 1.** *Given a deterministic 2DPT T with n states, k color conditions and ℓ colors, we can construct a 2RPT S with $O(\ell^{2kn}(2n)^{4n+1})$ states, k color conditions and ℓ colors such that $\llbracket T \rrbracket = \llbracket S \rrbracket$.*

Proof. Let T be a 2DPT with n states, k color conditions and ℓ colors. Using Theorem 7, we can construct an equivalent cPSST T' with $O(n(\ell^k)^n(2n+1)^{2n-1})$ states, $2n$ variables, k color conditions and ℓ colors. Then by Theorem 5, we can construct a 2RPT S equivalent to T' whose size is quadratic in the number of states and linear in the number of variables. More precisely, S has $O((n(\ell^k)^n(2n+1)^{2n-1})^2(2n)) = O(\ell^{2kn}(2n)^{4n+1})$ states, k color conditions and ℓ colors, concluding the proof. ◀

4 Composition of 2RPT

The main reason to use reversible two-way machines is that they are easily composable. Given two composable reversible transducers, we can construct one whose size is linear in both machines, and whose transition function is rather straight-forward. It is explicated in the following theorem and proof.

► **Theorem 2.** *Given two 2RPT \mathcal{S} and \mathcal{T} , of size n and m respectively, and such that the output alphabet of \mathcal{S} is the input alphabet of \mathcal{T} , we can construct a 2RPT \mathcal{U} , also denoted by $\mathcal{T} \circ \mathcal{S}$, of size $O(nm)$ such that $\llbracket \mathcal{U} \rrbracket = \llbracket \mathcal{T} \rrbracket \circ \llbracket \mathcal{S} \rrbracket$.*

Sketch of proof. The set of states of the machine \mathcal{U} is the cartesian product of the sets of states of \mathcal{S} and \mathcal{T} . Given an input word u of \mathcal{S} , \mathcal{U} simulates \mathcal{S} until some transition produces a nonempty output word $v \in B^+$. Then, it stops the simulation of \mathcal{S} to simulate the run $\rho_{\mathcal{T}}$ of \mathcal{T} over v . If $\rho_{\mathcal{T}}$ exits v on the right, then \mathcal{U} resumes the simulation of \mathcal{S} up to the next transition producing a nonempty word. Otherwise, it rewinds the run of \mathcal{S} to get its previous production, and simulates \mathcal{T} on it, starting from the right.

¹ An assignment $\nu: R \rightarrow B^*$ is extended to a morphism $\nu: (R \uplus B)^* \rightarrow B^*$ by $\nu(b) = b$ for all $b \in B$. Hence, if $\sigma \in \Lambda_R^B$ is a substitution then $\nu' = \nu \circ \sigma$ is an assignment defined by $\nu'(r) = \nu(\sigma(r))$ for all $r \in R$. For instance, if $\sigma(r) = br'cbr$ then $\nu'(r) = b\nu(r')cb\nu(r)$.

The conjunction of parity conditions allows to an easy construction for intersection, which is similar to what is expected here. A word u should be accepted if u belongs to the domain of \mathcal{S} and $\llbracket \mathcal{S} \rrbracket(u)$ belongs to the domain of \mathcal{T} . By doing the conjunction of both acceptance, we are able to recognize the domain of $\mathcal{U} = \mathcal{T} \circ \mathcal{S}$. ◀

Proof. Let $\mathcal{S} = (Q, A, \delta, q_0, \chi, B, \lambda)$ and $\mathcal{T} = (P, B, \alpha, p_0, \chi', C, \beta)$. We define the composition $\mathcal{U} = \mathcal{T} \circ \mathcal{S} = (R, A, \mu, r_0, \chi'', C, \nu)$ where $R = Q \times P$ is splitted as

$$R^+ = Q^+ \times P^+ \cup Q^- \times P^- \quad R^- = Q^- \times P^+ \cup Q^+ \times P^-.$$

The initial state is $r_0 = (q_0, p_0)$ and μ, ν and χ'' are defined below.

To properly define μ and ν , we extend α and β to finite words, and more precisely to the productions of \mathcal{S} . Given a word $v = \lambda(q, a) \in B^*$ for some $(q, a) \in Q \times A$, and a state p of \mathcal{T} , we define $\rho_p(v)$ to be the maximal run of \mathcal{T} over v starting in state p on the left (resp. right) of v if $p \in P^+$ (resp. $p \in P^-$). Then we define $\alpha^*(p, v)$ as the state reached by $\rho_p(v)$ when exiting v . It is undefined if $\rho_p(v)$ loops within v . Note that if $\alpha^*(p, v)$ belongs to P^+ (resp. P^-), then \mathcal{T} exits v on the right (resp. on the left). We define $\beta^*(p, v)$ as the concatenation of the productions of $\rho_p(v)$. If $\alpha^*(p, v)$ is defined, then $\beta^*(p, v)$ is finite. Note that $\rho_p(\varepsilon)$ is an empty run, so we have $\alpha^*(p, \varepsilon) = p$ and $\beta^*(p, \varepsilon) = \varepsilon$.

For the parity conditions, we let $\chi'' = \{\bar{c} \mid c \in \chi \cup \chi'\}$ and we extend the functions $c \in \chi'$ to finite runs $\rho_p(v)$. More precisely, we let $c^*(p, v)$ be the minimum c -value taken by the transitions of $\rho_p(v)$. When $v = \varepsilon$ then $\rho_p(v)$ is an empty run and we set $c^*(p, v)$ to the largest odd value in all values taken by c on transitions of \mathcal{T} . Then, given a state (q, p) of \mathcal{U} ,

- If $p \in P^+$ then we let $v = \lambda(q, a)$. We set $\nu((q, p), a) = \beta^*(p, v)$ and, with $q' = \delta(q, a)$ and $p' = \alpha^*(p, v)$ we define

$$\mu((q, p), a) = \begin{cases} (q', p') & \text{if } p' \in P^+, \\ (q, p') & \text{if } p' \in P^- \end{cases} \quad \text{and} \quad \bar{c}((q, p), a) = \begin{cases} c(q, a) & \text{if } c \in \chi, \\ c^*(p, v) & \text{if } c \in \chi'. \end{cases}$$

- If $p \in P^-$ then we let q' be such that $q = \delta(q', a)$ and $v = \lambda(q', a)$. Note that q' is unique by co-determinism of \mathcal{S} . We set $\nu((q, p), a) = \beta^*(p, v)$ and, with $p' = \alpha^*(p, v)$ we define

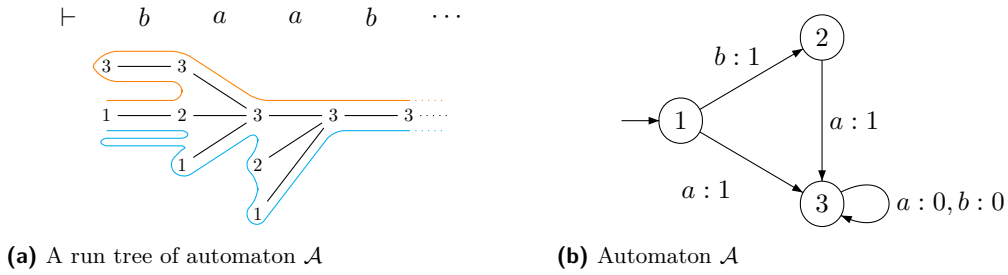
$$\mu((q, p), a) = \begin{cases} (q, p') & \text{if } p' \in P^+, \\ (q', p') & \text{if } p' \in P^-. \end{cases} \quad \text{and} \quad \bar{c}((q, p), a) = \begin{cases} c(q', a) & \text{if } c \in \chi, \\ c^*(p, v) & \text{if } c \in \chi'. \end{cases}$$

The intuition behind the transition function is that \mathcal{U} simulates \mathcal{S} to feed a simulation of \mathcal{T} . If \mathcal{T} moves forward on its input, then \mathcal{U} simulates \mathcal{S} forward. If \mathcal{T} moves backward on its input, then \mathcal{U} backtracks the computation of \mathcal{S} . During a switch of direction, \mathcal{S} stays put.

The acceptance condition of conjunctive parity was chosen specifically to allow for smooth composition. By using both sets of parities, the transducer \mathcal{U} ensures that the input word is accepted by \mathcal{S} , and that its production is accepted by \mathcal{T} . It is worth noting that since \mathcal{U} can rewind the run of \mathcal{S} , it can take a transition (and consequently its colors) multiple times on a given input position. This increases the multiplicity of the transitions taken during a non-looping run by a constant factor since a deterministic transducer never visits twice a given position in the same state. Hence, the set of colors of \mathcal{S} that \mathcal{U} sees infinitely often on a non-looping run is the same as the ones seen by \mathcal{S} . ◀

5 1DPT to 2RPT

Similarly to the finite words, given a deterministic one-way machine, one can construct a reversible one realizing the same function.



■ **Figure 2** The automaton \mathcal{A} depicted in Figure 2b recognizes all infinite words over alphabet $\{a, b\}$ having an a in first or second position (it has only one coloring function, represented after the colons in the transitions). Figure 2a is the part of the tree corresponding to the run of \mathcal{A} on the prefix $baab$ of an accepted word. Each node of the tree is a configuration of \mathcal{A} , represented here by a control state. Its horizontal position allows to deduce the position in the input word, depicted above the tree. The horizontal straight path represents the accepting run. Notice that when the top reading head needs to go backward to go around a branch, the bottom ones follows and goes backward on the accepting run.

► **Theorem 3.** *Let \mathcal{T} be a 1DPT with n states, we can construct a 2RPT \mathcal{T}' of size $O(n^2)$ such that $\llbracket \mathcal{T} \rrbracket = \llbracket \mathcal{T}' \rrbracket$.*

Proof. The construction is reminiscent of the tree-outline construction for co-deterministic transducers of [8]. The difference is that here, we begin with a deterministic transducer with an infinite input word, so instead of starting from the root of the tree, which is at the end of a finite input word, our outline has to start from a leaf at the beginning of the input word, corresponding to the initial configuration. We also generalize the construction by allowing any degree of non (co-)determinism: while in [8], at most two branches could merge on any vertex, here we allow any number.

We begin with the underlying automaton: from a one-way deterministic parity automaton (1DPA) \mathcal{A} , we build a two-way reversible parity automaton (2RPA) \mathcal{A}' simulating the behavior of \mathcal{A} . For any accepted input word w , we consider the infinite acyclic graph (simply called a tree) representing all the partial runs of \mathcal{A} merging with the accepting run of \mathcal{A} on w (note that because \mathcal{A} is deterministic, there is only one accepting run for a given word). Automaton \mathcal{A}' will simulate two synchronized reading heads going along the outline of this tree, as illustrated in Figure 2.

The two heads are required to make \mathcal{A}' equivalent to \mathcal{A} : we need to be able to discriminate configurations of a run of \mathcal{A}' occurring in the accepting run of \mathcal{A} from the ones added to account for the non-initial runs. One reading head follows the outline of the run tree from above, and the other one from below. The configurations where the two reading heads point to the same state of \mathcal{A} correspond to those occurring in the accepting run of this automaton.

The reading heads are placed above and below the initial state, and they move together to the right, until one of them encounters a branching in the tree. When this happens, the 2RPA moves backwards to go around the branch. When the branch dies (which necessarily happens because from each position, the prefix of a word is finite), the exploration continues to the right. As the two heads are synchronized, and because branches may not all be of the same length, when one head needs going left the other one may impose right moves in order to reach another branch, on which it will be able to go left far enough to follow the first head.

A run of \mathcal{A}' can be seen as a straightforward journey along the flattened outline of the tree, hence the reversibility.

Once \mathcal{A}' is defined, we define \mathcal{T}' : it is the 2RPT having \mathcal{A}' as underlying automaton, and whose output function is that of \mathcal{T} from states where the two reading heads point to the same state, and ε otherwise.

Formal construction of \mathcal{A}' . Let $\mathcal{T} = (Q, A, \delta, q_0, \chi, B, \lambda)$ be a 1DPT.

Fix an arbitrary total order \preceq over Q . For $q \in Q$ and $a \in A$ we let $\delta_a^{-1}(q) = \{p \in Q \mid \delta(p, a) = q\}$, and we also set $\delta_a^{-1}(q) = \emptyset$, except if $q \neq q_0$ (it is undefined otherwise). Let $\mathcal{S}_a(q, q')$ be a predicate, true if q' is minimal with respect to \prec such that $q \prec q'$ and $\delta(q, a) = \delta(q', a)$. Let $\overline{Q} = \{\overline{q} \mid q \in Q\}$ and $\underline{Q} = \{\underline{q} \mid q \in Q\}$ be two copies of Q . Define $\mathcal{A}' = (Q', A, \delta, q'_0, \chi')$ by

- $Q' = Q'^+ \uplus Q'^- = ((Q \cap \overline{Q}) \times (\underline{Q} \cap \overline{Q})) \setminus \{(q, q), (\overline{q}, \overline{q}) \mid q \in Q\}$ with $q'_0 = (q_0, \overline{q_0})$ and
 - $Q'^+ = \underline{Q} \times \overline{Q} \cup \overline{Q} \times \underline{Q}$,
 - $Q'^- = (\overline{Q} \times \underline{Q} \cup \underline{Q} \times \overline{Q}) \setminus \{(\overline{q}, \overline{q}), (q, q) \mid q \in Q\}$.

In a state $(r, s) \in Q'$, the first (resp. second) component is for the head which is “above” (orange line) (resp. “below” (blue line)) the accepting run (black straight line) in Figure 2a. In both cases, a state $\underline{q} \in \underline{Q}$ (resp. $\overline{q} \in \overline{Q}$) means that the corresponding head (colored line) is above (resp. below) the state.

- Transitions: first two cases for Q'^+ states and then for Q'^- states

$$1. \delta'((\underline{p}, \overline{q}), a) = \begin{cases} (\overline{p'}, \overline{q}) & \text{if } \mathcal{S}_a(p, p') \text{ for some } p' \in Q \\ (\underline{p}, \underline{q'}) & \text{elseif } \mathcal{S}_a(q', q) \text{ for some } q' \in Q \\ (\delta(\underline{p}, a), \overline{\delta(q, a)}) & \text{otherwise.} \end{cases}$$

$$2. \delta'((\overline{p}, \underline{q}), a) = \begin{cases} (\underline{p'}, \underline{q}) & \text{if } \mathcal{S}_a(p', p) \text{ for some } p' \in Q \\ (\overline{p}, \overline{q'}) & \text{elseif } \mathcal{S}_a(q, q') \text{ for some } q' \in Q \\ (\overline{\delta(p, a)}, \underline{\delta(q, a)}) & \text{otherwise.} \end{cases}$$

$$3. \delta'((\overline{p}, \overline{q}), a) = \begin{cases} (\underline{p}, \overline{q}) & \text{if } \delta_a^{-1}(p) = \emptyset \\ (\overline{p}, \underline{q}) & \text{elseif } \delta_a^{-1}(q) = \emptyset \\ (\overline{\min \delta_a^{-1}(p)}, \underline{\min \delta_a^{-1}(q)}) & \text{otherwise.} \end{cases}$$

$$4. \delta'((\underline{p}, \underline{q}), a) = \begin{cases} (\overline{p}, \underline{q}) & \text{if } \delta_a^{-1}(p) = \emptyset \\ (\underline{p}, \overline{q}) & \text{elseif } \delta_a^{-1}(q) = \emptyset \\ (\underline{\max \delta_a^{-1}(p)}, \overline{\max \delta_a^{-1}(q)}) & \text{otherwise.} \end{cases}$$

- $\chi' = \{c' \mid c \in \chi\}$ with $c'((r, s), a) = \begin{cases} c(q, a) & \text{if } (r, s) = (\underline{q}, \overline{q}) \\ \max\{c(p, a) \mid p \in Q, a \in A\} & \text{otherwise.} \end{cases}$

Reversibility of \mathcal{A}' . From the definition of δ' , \mathcal{A}' is clearly deterministic.

We show that \mathcal{A}' is also codeterministic. There are three potential transitions leading to a given state in Q' by reading a given letter, only one of which can be part of δ' .

The following case analysis shows this:

$$\delta_a'^{-1}((\underline{p}, \overline{q})) = \begin{cases} (\overline{p}, \overline{q}) & \text{if } \delta_a^{-1}(p) = \emptyset \\ (\underline{p}, \underline{q}) & \text{elseif } \delta_a^{-1}(q) = \emptyset \\ (\underline{\max \delta_a^{-1}(p)}, \overline{\min \delta_a^{-1}(q)}) & \text{otherwise} \end{cases}$$

$$\delta_a'^{-1}((\overline{p}, \underline{q})) = \begin{cases} (\underline{p}, \underline{q}) & \text{if } \delta_a^{-1}(p) = \emptyset \text{ (so } p \neq q_0) \\ (\overline{p}, \overline{q}) & \text{elseif } \delta_a^{-1}(q) = \emptyset \\ (\overline{\min \delta_a^{-1}(p')}, \underline{\max \delta_a^{-1}(q')}) & \text{otherwise} \end{cases}$$

$$\delta'_a{}^{-1}(\overline{(p', q')}) = \begin{cases} (\underline{p}, \overline{q'}) & \text{if } \mathcal{S}_a(p, p') \text{ for some } p \in Q \\ (\overline{p'}, \underline{q}) & \text{elseif } \mathcal{S}_a(q, q') \text{ for some } q \in Q \\ (\overline{\delta(p', a)}, \overline{\delta(q', a)}) & \text{otherwise.} \end{cases}$$

$$\delta'_a{}^{-1}(\underline{(p', q')}) = \begin{cases} (\overline{p}, \underline{q'}) & \text{if } \mathcal{S}_a(p', p) \text{ for some } p \in Q \\ (\underline{p'}, \overline{q}) & \text{elseif } \mathcal{S}_a(q', q) \\ (\underline{\delta(p', a)}, \underline{\delta(q', a)}) & \text{otherwise.} \end{cases}$$

We conclude that \mathcal{A}' is codeterministic, and therefore reversible.

Intuitively, automaton \mathcal{A}' will follow the run of \mathcal{A} , adding extra steps to deal with the states that are co-reachable from states of this run. States of \mathcal{A}' of the form $(\underline{q}, \overline{q})$ correspond to states q in the run of \mathcal{A} . The key idea behind the construction of \mathcal{A}' is that in a run of this automaton, configurations of the form $\vdash u(\underline{q}, \overline{q})v$ will occur in the same order as the configurations $\vdash uqv$ in the run of \mathcal{A} . This is the point of the following claim.

▷ **Claim 4.** Let $\rho = \vdash u_0q_0v_0 \rightarrow \vdash u_1q_1v_1 \rightarrow \vdash u_2q_2v_2 \rightarrow \dots$ be an accepting run of \mathcal{A} on $w \in A^\omega$ (we have $w = u_i v_i$ for all $i \geq 0$ where u_i is the prefix of length i of w). There is an accepting run ρ' of \mathcal{A}' on w such that the projection of ρ' on the configurations with states of the form $(\underline{p}, \overline{p})$ is $\vdash (q_0, \overline{q_0})w \xrightarrow{+} \vdash u_1(\underline{q_1}, \overline{q_1})v_1 \xrightarrow{+} \vdash u_2(\underline{q_2}, \overline{q_2})v_2 \xrightarrow{+} \dots$.

Proof of claim. Let G be the configuration graph of \mathcal{A} on the input word $w \in A^\omega$. The vertices of G are all configurations $\vdash uqv$ with $w = uv$, $u \in A^*$ and $q \in Q$. Edges correspond to transitions: we have an edge $\vdash upav \rightarrow \vdash uaqv$ if $\delta(p, a) = q$. Since \mathcal{A} is deterministic, there is at most one outgoing edge from each configuration and since \mathcal{A} is one-way, thus the graph G is acyclic.

Let T be the connected component of G that contains the initial configuration $\vdash q_0w$. Note that the run ρ corresponds to the only infinite path in G starting from $\vdash q_0w$. Any configuration $\vdash u_i p_i v_i$ of T which is not on ρ ($p_i \neq q_i$) will eventually merge with ρ : $\vdash u_i p_i v_i \xrightarrow{*} \vdash u_{j-1} p_{j-1} v_{j-1} \rightarrow \vdash u_j q_j v_j$ with $i < j$ and $p_{j-1} \neq q_{j-1}$. We say that $\vdash u_i p_i v_i$ is below (resp. above) ρ if $p_{j-1} \prec q_{j-1}$ (resp. $q_{j-1} \prec p_{j-1}$).

Let us consider the run ρ' of \mathcal{A}' on w . Due to the definition of the transition function of \mathcal{A}' , the run ρ' only moves along T . Indeed a configuration $\vdash u(r, s)v$ of ρ' encodes the position of two tokens, each placed either above or below a configuration of T . Moreover, the first token is always above the branch ρ while the second token is always below. This can be shown by case analysis of δ' . The two transitions where the upper token goes from above a branch to below are the following:

- $\delta'(\underline{(p, \overline{q})}, a) = (\overline{(p', \overline{q})})$ if $(\mathcal{S}_a(p, p'))$: there, we know that $p \prec p'$, so the token ends up on a branch that is above where it was;
- $\delta'(\underline{(p, \underline{q})}, a) = (\overline{p}, \overline{q})$ if $\delta_a^{-1}(p) = \emptyset$ and $a \neq \vdash$, so p must have reached the end of the branch it was placed on, and we know it was not placed on ρ , because when this branch ends, $a = \vdash$.

A similar observation can be made for transitions where the lower token goes from below a branch to above.

We denote by T_i the subtree of T containing all configurations having a path to $\vdash u_i q_i v_i$. We aim to prove that ρ' reaches the position i , and the first time it does is in state $(\underline{q_i}, \overline{q_i})$.

We remark that for every cases 1 to 4 of the transition function, as long as the transition function is defined the run ρ' can continue. As we only visit configurations of T , as long as ρ is infinite, as assumed by Claim 4, so is ρ' .

Next, as \mathcal{A}' is reversible, it cannot loop, as it would require two different configuration to go to the same one to enter the loop, which would break codeterminism. So ρ' starts in T_i , does not stop nor loops, so since T_i is finite, ρ' has to end up leaving T_i . So ρ' reaches position i , while only moving along T_i . As $(\underline{q}_i, \overline{q}_i)$ is the only possible state where ρ' follows T_i while having its first token above ρ and the second below ρ , ρ' first reaches i in state $(\underline{q}_i, \overline{q}_i)$.

As this is true for any position i , and since T_i contains T_{i-1} , we can conclude the proof of Claim 4. \blacktriangleleft

Based on this claim, we show that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. Let w be an infinite word accepted by \mathcal{A} , and ρ be the accepting run of \mathcal{A} on w . We showed that the configurations $\vdash uqv$ happen in the same order in ρ as configurations of the form $\vdash u(\underline{q}, \overline{q})v$ in ρ' , the run on w of \mathcal{A}' . Moreover $c'((\underline{p}, \overline{p}), a) < c'(s, a)$ for all $a \in A$ and for all s of another form, and as $|\text{inf}(\rho)| > 0$ (because w is infinite and \mathcal{A} has a finite number of states), $\min\{t | t \in \text{inf}(\rho)\} = \min\{t | t \in \text{inf}(\rho')\}$. So $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.

Construction of \mathcal{T}' . Let $\mathcal{T}' = (Q', A, \delta', q'_0, \chi', B, \lambda')$ be the 2RPT having \mathcal{A}' as underlying automaton, with

$$\lambda'(s, a) = \begin{cases} \lambda(q, a) & \text{if } s = (\underline{q}, \overline{q}) \\ \varepsilon & \text{if } s \text{ is of another form.} \end{cases}$$

Because we showed that states of the form $(\underline{q}, \overline{q})$ are met in the run of \mathcal{A}' on a given word in the same order as states q in the run of \mathcal{A} on the same word, the output of \mathcal{T}' is the same as the output of \mathcal{T} . So we have that $\llbracket \mathcal{T} \rrbracket = \llbracket \mathcal{T}' \rrbracket$.

Finally, $|Q'| = 4|Q|^2$, justifying the complexity. \blacktriangleleft

6 From cPSST to 2RPT

We extend another result from [8] about constructing a reversible two-way transducer from a Streaming String Transducer. The procedure and the complexity are similar. The main difference is that the procedure only works thanks to the distinguished register `out` of cPSST. Without it, production could depend on an infinite property of the input word, which is not realizable by a deterministic (and hence reversible) machine.

► Theorem 5. *Let \mathcal{T} be a cPSST with n states and m registers. Then we can construct a 2RPT \mathcal{S} with $O(n^2m)$ states such that $\llbracket \mathcal{T} \rrbracket = \llbracket \mathcal{S} \rrbracket$.*

Proof. We prove that $\llbracket \mathcal{T} \rrbracket = \llbracket \mathcal{F} \rrbracket \circ \llbracket \mathcal{D} \rrbracket$ where \mathcal{D} is a 1DPT and \mathcal{F} is a 2RPT. The transducer \mathcal{D} has the same underlying automaton as \mathcal{T} , but instead of applying a substitution σ to the registers, \mathcal{D} enriches the input letter with σ . Then, the transducer \mathcal{F} uses the flow of registers output by \mathcal{D} to output the contents of the relevant registers in a reversible fashion. Finally, we construct a 2RPT \mathcal{D}' equivalent to \mathcal{D} by Theorem 3 and we obtain the desired 2RPT $\mathcal{S} = \mathcal{F} \circ \mathcal{D}'$ by Theorem 2.

Formal Construction. Let $\mathcal{T} = (Q, A, \delta, q_0, \chi, B, R, \text{out}, \lambda)$ be a cPSST.

We define the 1DPT by $\mathcal{D} = (Q, A, \delta, q_0, \chi, \Lambda_R^B, \gamma)$ where $\gamma(q, a) = \lambda(q, a)$. The reversible transducer is $\mathcal{F} = (Q', \Lambda_R^B, \alpha, q'_0, \emptyset, B, \beta)$ where:

- $Q' = R \times \{i, o\}$ with $Q^+ = R \times \{o\}$ and $Q^- = R \times \{i\}$. We will denote by r_i (resp. r_o) the state (r, i) (resp. (r, o)). Informally, being in state r_i means that we need to compute the content of r , while state r_o means we have just finished computing it.
- The initial state is $q'_0 = \text{out}_o$.
- There is no accepting condition;
- α and β both read a state in Q' and a substitution $\sigma \in \Lambda_{\mathbb{R}}^B$, or the leftmarker \vdash , which is treated as a substitution σ_{\vdash} associating ε to every register. We define α and β as follow:
 - If the state is r_i for some $r \in R$.
 - * If $\sigma(r) = v \in B^*$ contains no register, then $\alpha(r_i, \sigma) = r_o$ and $\beta(r_i, \sigma) = v$.
 - * If $\sigma(r) = vs\gamma$ with $v \in B^*$ and $s \in R$ is the first register appearing in $\sigma(r)$, then $\alpha(r_i, \sigma) = s_i$ and $\beta(r_i, \sigma) = v$.
 - If the state is r_o for some $r \in R$. Recall that from the definition of copyless SSTs, for any register r , there exists at most one register t , such that r occurs in $\sigma(t)$, and in this case r occurs exactly once in $\sigma(t)$.
 - * Suppose that for some register s we have $\sigma(s) = \gamma r v$ with $v \in B^*$. Then $\alpha(r_o, \sigma) = s_o$ and $\beta(r_o, \sigma) = v$.
 - * Suppose that for some register t we have $\sigma(t) = \gamma r v s \gamma'$ with $v \in B^*$ and $s \in R$. Then $\alpha(r_o, \sigma) = s_i$ and $\beta(r_o, \sigma) = v$.
 - * If r does not appear in any $\sigma(s)$, then the computation stops and rejects. This somehow means that we are computing the contents of a register that is dropped in the original SST. This will not happen if what is fed to \mathcal{F} is produced by \mathcal{D} .

Reversibility of \mathcal{F} . The transducer \mathcal{F} is clearly deterministic by construction. Let us prove that it is codeterministic. To this end, let s_i be a state and σ a substitution. Looking at the transition function, its antecedent $\alpha^{-1}(s_i, \sigma)$ is either r_i if $\sigma(r)$ starts with vs for some word $v \in B^*$, or r_o if there is some register t that contains $r v s$ for some word $v \in B^*$. Since we only consider copyless substitutions, there is at most one register that contains s . The two options are then mutually exclusive, as one requires that s be the first register to appear, and the second requires that there is a register before s .

The proof for a state s_o is similar. The antecedent $\alpha^{-1}(s_o, \sigma)$ is either s_i if $\sigma(s)$ contains no register, or r_o if r is the last register appearing in $\sigma(s)$. Since these two are mutually exclusive, we get that \mathcal{F} is reversible.

Correctness of the construction. First, let us remark that the domain of \mathcal{D} is the set of input words u such that \mathcal{T} has an infinite accepting run over u since they share the same underlying automaton. So the domain of \mathcal{T} is the set of words in the domain of \mathcal{D} on which \mathcal{T} produces an infinite word.

Let $(q_0, \nu_0) \xrightarrow{a_0} (q_1, \nu_1) \xrightarrow{a_1} (q_2, \nu_2) \xrightarrow{a_2} (q_3, \nu_3) \cdots$ be the accepting run of some input word $u = a_0 a_1 a_2 \cdots \in A^\omega$ in the domain of the cPSSST \mathcal{T} . For $j \geq 0$, let $\sigma_j = \lambda(q_j, a_j)$ so that $\llbracket \mathcal{D} \rrbracket(u) = \sigma_0 \sigma_1 \sigma_2 \cdots$.

We prove by induction that for every position $j \geq 0$ of u , the run of the transducer \mathcal{F} on $\llbracket \mathcal{D} \rrbracket(u)$ reaches the state out_o in position j having produced the content $\nu_j(\text{out})$ of the run of \mathcal{T} on u up to position j . For $j = 0$, there is nothing to prove as the registers are initially empty and out_o is the initial state of \mathcal{F} .

Now suppose that the run of \mathcal{F} on $\llbracket \mathcal{D} \rrbracket(u)$ reaches some position j in state out_o , having produced $\nu_j(\text{out})$. Recall that, by definition of λ , the substitution $\sigma_j = \lambda(q_j, a_j)$ used by \mathcal{T} at position j is such that $\sigma_j(\text{out}) = \text{out} \cdot \gamma$. Then if there is no other register, i.e., if $\gamma = v \in B^*$, by definition of α , \mathcal{F} moves to $j + 1$ in state out_o and produces v , so that its cumulated production is $\nu_j(\text{out}) \cdot v = \nu_{j+1}(\text{out})$.

The interesting case is of course when some registers are flown to out. Let r be the second register of $\sigma_j(\text{out})$, i.e., $\sigma_j(\text{out})$ starts with $\text{out} \cdot vr$ with $v \in B^*$. Then, by definition of α and β , \mathcal{F} stays at position j switching to state r_i and producing v . Then, using Claim 6, \mathcal{F} reaches r_o at position j producing the content of $\nu_j(r)$. We repeat this process to exhaust all registers appearing in $\sigma(\text{out})$, reaching finally state out_o at position $j + 1$ with cumulated production $\nu_j(\sigma_j(\text{out})) = \nu_{j+1}(\text{out})$, proving the induction.

▷ **Claim 6.** For all positions $j \geq 0$ and registers $r \in R$, there exists a right-to-right run (r_i, r_o) of \mathcal{F} starting and ending at position j and which produces the content of $\nu_j(r)$.

Proof of Claim 6. The proof is by induction on j . If $j = 0$ then $\nu_0(r) = \varepsilon$ and the run of \mathcal{F} starting at position 0 in state r_i reads σ_- . By definition of α and β the run produces $\sigma_-(r) = \varepsilon$ and switches from r_i to r_o , proving the claim for $j = 0$.

Now assume that the claim is true for j . Consider the run ρ of \mathcal{F} starting in state r_i at position $j + 1$. The run ρ starts by reading σ_j . If $\sigma_j(r) = v \in B^*$ then the run produces $v = \nu_{j+1}(r)$ and switches from r_i to r_o , proving the claim. The second case is when $\sigma_j(r)$ starts with some vs with $v \in B^*$ and $s \in R$. Then, the first transition of ρ produces v and moves to position j in state s_j . By induction hypothesis, there is a right-right (s_i, s_o) -run starting at j and producing $\nu_j(s)$. Then, the run reads σ_j in state s_o and, either goes r_o in position $j + 1$ producing $v' \in B^*$ if $\sigma_j(r)$ ends with sv' (s is the last register flown to r), or goes to t_i in position j producing $v' \in B^*$ if $\sigma_j(r)$ contains the factor $sv't$ (t is the next register flown to r). By iterating this process again, we exhaust the registers flown to r , produces their content meanwhile. Finally, the run ρ ends in position $j + 1$ with state r_o and has produced $\nu_{j+1}(r) = \nu_j(\sigma_j)(r)$, proving the claim. ◀

Coming back to the proof of correctness, we have shown that for all positions $j \geq 0$, \mathcal{F} has an initial run on $\llbracket \mathcal{D} \rrbracket(u)$ reaching position j in state out_o and producing $\nu_j(\text{out})$. This proves that $\llbracket \mathcal{D} \rrbracket(u)$ is in the domain of \mathcal{F} (the maximal initial run of \mathcal{F} on $\llbracket \mathcal{D} \rrbracket(u)$ reads the whole word and \mathcal{F} accepts only if \mathcal{T} produces infinitely often) and $\llbracket \mathcal{F} \rrbracket(\llbracket \mathcal{D} \rrbracket(u)) = \bigsqcup_{j \geq 0} \nu_j(\text{out}) = \llbracket \mathcal{T} \rrbracket(u)$. Therefore, $\llbracket \mathcal{T} \rrbracket = \llbracket \mathcal{F} \rrbracket \circ \llbracket \mathcal{D} \rrbracket$.

Finally, we construct a 2RPT \mathcal{D}' equivalent to \mathcal{D} by Theorem 3 and we obtain the desired 2RPT $\mathcal{S} = \mathcal{F} \circ \mathcal{D}'$ by Theorem 2. ◀

7 From 2DPT to cPSST

The construction presented in this section is the most involved of the paper. It is adapted from [11]. Given a deterministic two-way transducer, we construct a cPSST that realizes the same function. Here again, the main complications from infinite words are dealing with the acceptance condition and the impossibility to get the final configuration of the run.

► **Theorem 7.** *Given a 2DPT \mathcal{T} with $n > 0$ states and k coloring functions over ℓ colors, we can construct a cPSST \mathcal{S} with $O(\ell^{kn}(2n)^{2n})$ states, $2n - 1$ registers and k coloring functions over ℓ colors such that $\llbracket \mathcal{T} \rrbracket = \llbracket \mathcal{S} \rrbracket$.*

Sketch of proof. We improve on the classical Shepherdson construction [16] from two-way machines to one-way. In this construction, the one-way machine computes information about the runs of the two-way machine on the prefix read up to the current position. More precisely, it stores the state reached on reading the prefix starting at the initial state, as well as a succinct representation of the information about all right-right runs. Further, by associating a register to each run in this representation, we can construct an SST equivalent to the two-way machine.

Upon reading some letter $a \in A$, the prefix w we are interested in grows to wa , and consequently, we have to update the information about the right-right runs. While some right-right runs on the prefix w may be extended to right-right runs on wa , some runs (which cannot be extended) may die, and hence needs to be removed. A third possibility is that, upon reading a letter a , some right-right runs may merge. This implies that the construction would not be copyless, as if two right-right runs over ua are the extension of a same right-right run over u , the register storing the production of the run over u needs to be copied in both runs over ua .

In order to compute a copyless SST, we improve this construction by refining the information stored by the one-way machine: it stores not only the set of right-right runs, but also whether they merge and the respective order of the merges. Essentially, the latter representation keeps track of the *structure* of the right-right runs on the prefix read up to the current position, as well as the *output* generated by these runs. The resulting information can be represented as a forest, which is a (possibly empty) set of trees. Then we associate a register to each edge of the forest, so that the update function can be made copyless. The number of registers required is still linear in the number of states. ◀

Formally, we call the structure used to model the right-right runs *merging forests*, which we define as follow:

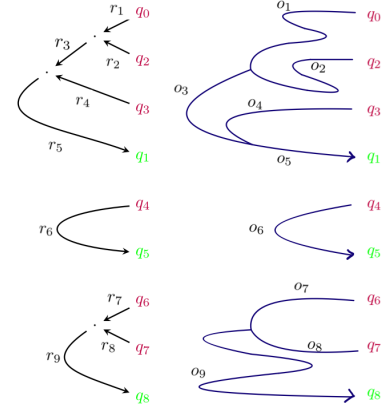
► **Definition 8.** Given a set of states $Q = Q^+ \uplus Q^-$, a set of coloring functions χ and an integer $\ell > 0$, we define the merging forests on (Q, χ, ℓ) , denoted \mathcal{MF} , as the set of forests F such that:

- the leaves of all trees of F are labeled by distinct elements of Q^- ,
- the roots of all trees of F are labeled by distinct elements of Q^+ ,
- all unary nodes (exactly one child) are roots.

The leaves are also labeled by a χ -tuple of integers less than ℓ .

Informally, an element $F \in \mathcal{MF}$ describes a set of right-right runs, such that if q is the root of a tree and p is one of its leaves, then (p, q) is a right-right run. Notice that, if two leaves x and y belong to the same tree, then the right-right runs starting in x and y will merge. The structure of the tree reflects the order in which the runs sharing the same root merge. The integer labels of leaves serve as coloring for the parity acceptance condition. An example of a merging forest is depicted in Figure 3.

Note that in Figure 3, the states in Q^- are depicted in purple, while the states in Q^+ are depicted in green. The forest comprises of 3 trees - one with root q_1 and leaves q_0, q_2 and q_3 , the second with root q_5 and leaf q_4 , and the third with root q_8 and leaves q_6 and q_7 . For each tree, there are right-right runs from its leaves to the root. For instance, from the merging forest in Figure 3, we can infer that there are three right-right runs entering u on the right in states q_0, q_2 and q_3 respectively, and emerging out of u in state q_1 , moreover the run starting from q_0 first merges with the run starting from q_2 and these two runs then merge with the one starting from q_3 . The figure also depicts the register corresponding to the edges of the forest. Further, the output generated by the part of the run labeled o_1 is stored in the register r_1 , the output for the part labeled o_2 is stored in r_2, o_3 in r_3 and o_5 in r_5 .



■ **Figure 3** Example of a merging forest (on the left) corresponding to right-right runs (on the right) of a two-way machine for some prefix u of an input word.

We are now ready to give the formal construction of Theorem 7. We defer the proof of correctness to later in this section.

Proof of Theorem 7. Given a 2DPT $\mathcal{T} = (Q, A, \delta, q_0, \chi, B, \lambda)$, we construct a cPSST $\mathcal{S} = (Q', A, \alpha, q'_0, \chi', B, R, \text{out}, \beta)$ such that:

- $Q' = Q \times \mathcal{MF}$,
- $q'_0 = (q_0, F_0)$ where F_0 is the forest having only leaves and roots and edges from a leaf u labeled $p \in Q^-$ to a root v labeled $q \in Q^+$ if $\delta(p, \vdash) = q$.
- R is a set of registers of size $2|Q| - 1$ with a distinguished register out .
- the coloring functions $\chi' = \{c' \mid c \in \chi\}$ are described below.
- the definitions of α and β are more involved and given below.

For each merging forest $F \in \mathcal{MF}$, we fix a map ξ_F associating distinct registers from $R \setminus \{\text{out}\}$ to edges of F . This is possible since the number of edges in F is at most $2|Q| - 2$ (see Lemma 9).

For simplicity sake, we assume that for each edge (u, v) of F_0 corresponding to transition $\delta(p, \vdash) = q$, the register $\xi_{F_0}(u, v)$ is initialized with the production $\lambda(p, \vdash)$. Note that considering initialized registers does not add expressiveness, as it could be simulated using a new unreachable initial state. Other registers are initially empty.

The definitions of the transition function α and the update function β are intertwined. Let $(q, F) \in Q \times \mathcal{MF}$ be a state of \mathcal{S} and a a letter of A . We describe the state $(p, F') = \alpha((q, F), a)$. First we construct an intermediate *graph* G that does not satisfy the criteria of the merging forests, then explain how G is transformed into a merging forest $F' \in \mathcal{MF}$. The steps of the construction are depicted in Figure 4.

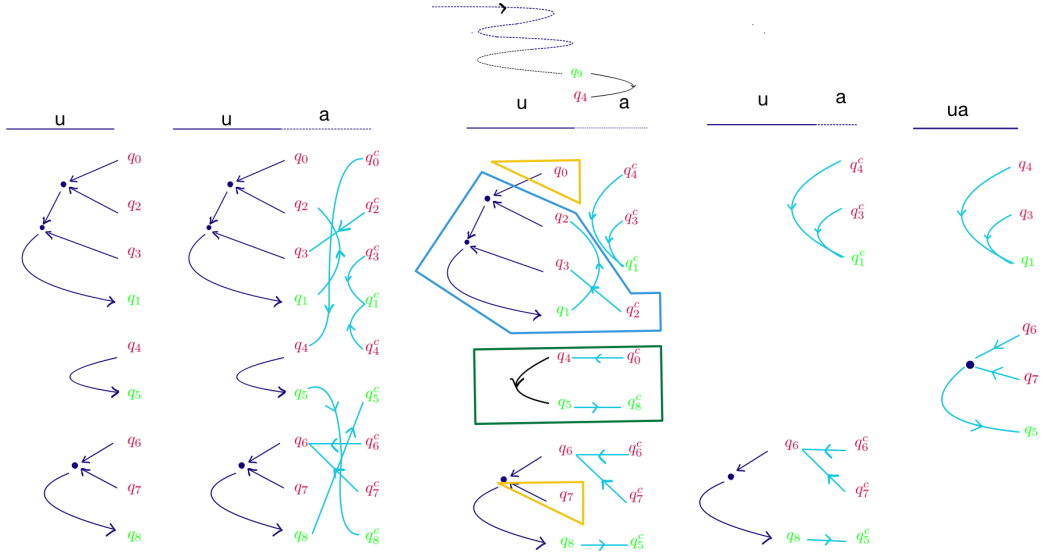
Construction of G . The graph G is built from F as follows. First, we add new isolated nodes $Q_c = \{q_c \mid q \in Q\}$ to the forest F . These nodes will serve as the roots and leaves of F' . For each transition $\delta(p, a) = q$, we will add an edge connecting these new nodes and the roots and leaves of F . We also extend the labelling ξ_F to a labelling ξ_G by adding productions $\lambda(p, a)$ to the new edges of G .

Let $p \in Q$ and let $q = \delta(p, a)$. If $p \in Q^+$ is the label of the root u of some tree in F , we add an edge from u to either q_c if $q \in Q^+$, or to v if $q \in Q^-$ and there exists a leaf v labeled q in F . If $p \in Q^-$, we add an edge from p_c to either q_c if $q \in Q^+$, or to v if $q \in Q^-$ and there exists a leaf v labeled q in F . The added edge is labeled $\lambda(p, a)$ by ξ_G . The construction is illustrated in the first two steps of Figure 4. Note that G may now have cycles.

The new state p . To compute the first component of the new state of \mathcal{S} , let $r = \delta(q, a)$. If r belongs to Q^+ , then $p = r$ and $\lambda(q, a)$ is appended to the output register: $\beta((q, F), a)(\text{out}) = \text{out} \cdot \lambda(q, a)$. Moreover, the colors are directly inherited: $\bar{c}((q, F), a) = c(q, a)$ for all $c \in \chi$. Otherwise, assume that there is a leaf u in F labeled $r \in Q^-$ (if not, the transition is undefined). We consider the maximal path in G starting from u . If this path is looping or if it ends in a root of F then the transition is undefined. Otherwise, it ends in a new node v of G . Let $p \in Q^+$ be the state such that $v = p_c$. We append to the out register first $\lambda(q, a)$ and then the ξ_G labels of the edges of the path from u to v in G , in the order of the path. These ξ_G labels are either registers given by ξ_F for edges of F , or local outputs of the form $\lambda(x, a)$ for the new edges, i.e., those in $G \setminus F$. Moreover, for each $c \in \chi$, we let $c'((q, F), a)$ be the minimum of (1) the c -values of the labels of leaves of F appearing in the path from u to v in G , and (2) the c -values of the transitions used to create the new edges of G in this path.

The third figure of Figure 4 illustrates the case where $\delta(q, a) = q_4 \in Q^-$. We then look at the path starting in q_4 , which leads to the state $(q_8)_c$ after reading a .

Construction of F' and the update of registers other than out. This is illustrated by



■ **Figure 4** Illustration of the procedure to calculate F' from F and a letter a . The first figure is F , the second is the graph G built from F and the transition occurring at a . The third figure illustrates the trimming done from G to obtain a forest depicted in the fourth figure. Finally, we merge unary internal nodes to obtain the merging forest of the last figure.

the third and fourth figures of Figure 4. We erase all nodes (and adjacent edges) that are on a cycle of G since such cycles cannot be part of an accepting run of \mathcal{T} (see the part of G boxed in blue). The resulting graph is now acyclic. We also erase all nodes and edges which are not on a path from a leaf in Q_c^- to a root in Q_c^+ since they cannot be part of a right-right run on ua (see the parts boxed in yellow). Finally, we erase the tree with root $v = p_c$ in the second case of the definition of p above. Indeed, should any right-right run simulated by this tree appear later, the resulting run would loop on a finite prefix of the input (see the part boxed in green).

The resulting forest has leaves in Q_c^- and roots in Q_c^+ . Each remaining new leaf or root $q_c \in Q_c$ is labeled q , i.e., by dropping the c index. A new leaf $q_c \in Q_c^-$ is labeled by a χ -tuple $(m_c)_{c \in \chi}$ of integers less than ℓ where m_c is the minimum of (1) the c -component of the labels of leaves of F appearing in the branch in G from q_c to its root, and (2) the values $c(p, a)$ of the transitions used to create the new edges of G in this branch. We forget the initial labeling of leaves and roots of F .

To obtain a merging forest, it remains to remove unary internal nodes. This is shown between the fourth and fifth figures of Figure 4. We replace each *maximal* path $\pi = u_0, u_1, u_2, \dots, u_{n-1}, u_n$ ($n \geq 1$) with u_1, \dots, u_{n-1} unary nodes by a single edge $e = (u_0, u_n)$. We obtain the merging forest F' . The update function is simultaneously defined by

$$\beta((q, F), a)(\xi_{F'}(e)) = \xi_G(u_0, u_1)\xi_G(u_1, u_2) \cdots \xi_G(u_{n-1}, u_n).$$

Notice that for each remaining edge f of G we have either $f \in F$ and $\beta((q, F), a)(\xi_{F'}(f)) = \xi_G(f) = \xi_F(f)$ or f is a new edge and $\beta((q, F), a)(\xi_{F'}(f))$ is set to some $\lambda(s, a)$. Since each edge f of F contributes to at most one edge e of F' , or to the path from u to $v = p_c$ which flows into the out register (but not both), it implies that the substitution $\beta((q, F), a)$ is copyless. ◀

To prove correctness of Theorem 5, we need the following lemma, relying on Cayley's Formula to count the number of merging forests.

► **Lemma 9.** *Let $Q = Q^+ \uplus Q^-$ be of size $n > 0$ with $Q^+ \neq \emptyset$, χ of size $k \geq 0$ and $\ell > 0$. Then each element F of \mathcal{MF} has at most $2n - 2$ nodes, $2n - 2$ edges; and \mathcal{MF} itself is of size at most $\ell^{k(n-1)}(2n-1)^{2n-3}$.*

Proof. We first compute the maximal number of nodes and edges in a nonempty forest F of \mathcal{MF} . Note that, since F is nonempty, both Q^+ and Q^- must be nonempty. Let $\text{nbe}(t)$ and $\text{nbl}(t)$ be the number of edges and leaves of the tree t . If t has no unary nodes, then $\text{nbe}(t) \leq 2\text{nbl}(t) - 2$. Since in a forest $F \in \mathcal{MF}$, all unary nodes are roots, it follows that $\text{nbe}(t) \leq 2\text{nbl}(t) - 1$ for all trees $t \in F$. Also, the number of nodes of a tree is $\text{nb}(t) = 1 + \text{nbe}(t)$. We deduce that

$$\begin{aligned} \text{nbe}(F) &= \sum_{t \in F} \text{nbe}(t) \leq \sum_{t \in F} 2\text{nbl}(t) - 1 \leq 2|Q^-| - 1 \leq 2n - 3 \\ \text{nb}(F) &= \sum_{t \in F} \text{nb}(t) \leq \sum_{t \in F} 2\text{nbl}(t) \leq 2|Q^-| \leq 2n - 2. \end{aligned}$$

Now we compute the size of the set \mathcal{MF} . Cayley's formula [6] states that the number of non oriented trees with m differently labeled nodes is m^{m-2} . The difference here is that first we deal with forests with at most $2n - 2$ nodes, and secondly only the leaves and roots are labeled. The first point can be dealt with by adding a new node as the root of all trees of the forest, and new nodes if needed to get exactly $m = 2n - 1$ nodes in the tree. For the second point, we can label arbitrarily the remaining nodes. Finally, as each leaf is labeled by a χ -tuple of integers less than ℓ , each tree can appear in \mathcal{MF} up to $(\ell^k)^{|Q^-|}$ many times. The size of \mathcal{MF} is then smaller than $\ell^{k(n-1)}(2n-1)^{2n-3}$. ◀

We can now prove correctness for Theorem 7.

Proof of Correctness for Theorem 7. We begin with the size of \mathcal{S} . Using Lemma 9, we get that $|Q'| \leq n(\ell^{k(n-1)}(2n-1)^{2n-3}) = \mathcal{O}(\ell^{kn}(2n)^{2n})$. Using carefully the registers, we only ever need at most $2n - 1$ registers.

Proof of correctness. First given a finite word w , we say that a right-right run $(x, y) \in Q^- \times Q^+$ of the 2DPT \mathcal{T} on w is useful if there exists an infinite word w' such that the run of \mathcal{T} on ww' is accepting and reaches x on position $|w|$.

We first prove that the state of the cPSST contains all the needed information, then prove that the registers can be used to produce the output. We prove by induction on the size of a word w that the state (q, F) of the constructed cPSST reached after reading w is such that (q_0, q) is a left-right run on w and F contains information about all useful runs on w . Moreover, the out register contains the production of the left-right run (q_0, q) on w and, given a path $\pi = u_0, \dots, u_n$ in F from a leaf u_0 labeled by x to a root u_n labeled by y , the production of the right-right run (x, y) is given by the concatenation of the registers $\xi_F((u_0, u_1)) \dots \xi_F((u_{n-1}, u_n))$.

First, if w is empty, then the initial state is (q_0, F_0) where F_0 describes the set of all right-right runs on \vdash . The register out is empty and each tree in the forest F_0 is reduced to a single edge containing the associated production, hence proving the initial case.

Now suppose that the statement holds for some word w and some state (q, F) and let $a \in A$ be a letter. We prove the statement for wa . Let $(p, F') = \alpha((q, F), a)$. If $p = \delta(q, a) \in Q^+$, then (q_0, p) is a left-right run on wa and $\beta((q, F), a)(\text{out}) = \text{out} \cdot \lambda(q, a)$, corresponding to the claim for the left-right run. Otherwise, let $r = \delta(q, a) \in Q^-$. The state p is then described in

G as the state reached by the maximal path in G from the leaf u of F labeled by r , to the new node p_c . Using the induction hypothesis, F describes the useful right-right runs on w . Then, following the maximal path from u in G , we see the sequence of right-right runs on w and left-left runs on a , up to the last left-right transition on a leading to state p . This means that we have computed p such that (q_0, p) is the left-right run on wa . We also append to the register out all $\xi_G(e)$ for edges e in the path. By induction hypothesis, the registers contain the production of the useful right-right runs on w , and the added edges contains the local production, proving the claim for the left-right run.

We now prove that all useful runs of wa are in F' . Let $(x, y) \in Q^- \times Q^+$ be such a run. Then either $\delta(x, a) = y$ and this edge is added in G and remains in F' , or (x, y) is a sequence starting with a right-left transition over a , then useful right-right runs over w and left-left transitions over a , and finally a left-right transition over a . In the first case, the register $\xi_{F'}((x, y))$ takes the label of G , i.e. the local production $\lambda(x, a)$, satisfying the claim as the path is reduced to a single edge. In the second case, let u_0, \dots, u_n be the path from x to y in G . Each edge (u_i, u_{i+1}) is either a new edge whose label is a local production, or a single edge of F . The associated sequence of registers contains then all the output information of the (x, y) run. When reducing G to F' , as only non branching paths of G can be reduced to a single edge, there is no loss of information. Finally, notice that the edges deleted from G to obtain F' are the ones that are not part of a useful right-right run on wa , or are merging with the left-right run. These latter right-right runs are not useful for wa : they cannot occur anymore in an accepting run of \mathcal{T} since they would induce a loop on a finite prefix of the input. Hence all edges required for the path from x to y appear in F' . Consequently, there is no loss of run nor information, the path from x to y in F' exists and the associated sequence of registers contains the production of the run.

Finally, to prove that both transducers have the same domain, we remark that given the previous induction, if (q, F) is the state reached by \mathcal{S} after reading an input w , then upon reading a letter a , the color of the transition $\alpha((q, F), a) = (p, F')$ is the minimum of the color of all transitions used when extending the left-right run (q_0, q) of \mathcal{T} on w to the left-right run (q_0, p) on wa . Then given an infinite word u , \mathcal{S} has an infinite run on u if and only if \mathcal{T} does, and the minimum of all colors appearing infinitely often is the same on both runs. \blacktriangleleft

8 Continuity and topological closure of a 2DPT

The classical topology on infinite words (see e.g. [15]) defines the distance between two infinite words u and v as $d(u, v) = 2^{-|u \wedge v|}$, where $u \wedge v$ is the longest common prefix of u and v . Then a function $f: A^\omega \rightarrow B^\omega$ is continuous at $x \in \text{dom}(f)$ if

$$\forall i \geq 0, \exists j \geq 0 \forall y \in \text{dom}(f), |x \wedge y| \geq j \implies |f(x) \wedge f(y)| \geq i$$

A function f is continuous if it is continuous at every $x \in \text{dom}(f)$. We refer to [12] for more details.

Since a 2DPT is in particular deterministic, it realizes a continuous function. Indeed, the longer two input words share a common prefix, the longer their output will also do.

By comparison, a non-deterministic transducer can make choices depending on an infinite property of the input, e.g. whether there is an infinite number of as , and thus realize a noncontinuous function.

The question of characterizing the continuous functions realizable by transducers was studied in [4, 5]. In [4], it was proved that for any continuous function realized by a non-

deterministic one-way transducer T , there exists a deterministic two-way transducer S such that for any $u \in \text{dom}(T)$, $\llbracket T \rrbracket = \llbracket S \rrbracket$. This means that if a non-deterministic one-way transducer realizes a continuous function, although it can use the non-determinism to refine its domain, the continuity property forbids it from producing non-deterministically.

In [5], it was conjectured that the class of deterministic regular functions, i.e. functions defined by deterministic two-way transducers with a Büchi acceptance condition, corresponds to the class of functions realized by a non-deterministic two-way transducer, called regular functions, that are continuous.

Since the class of 2DPT is strictly more expressive than the class of deterministic regular functions of [5], but still realize continuous functions, the conjecture of [5] fails. One can for example consider the function f such that $f(u) = u$ if u has a finite number of a s and is undefined otherwise. Then f is continuous, as it is continuous on its domain, but it cannot be realized by a deterministic two-way transducer with a Büchi condition.

However, the refinement here only acts on the domain of the function, and not the production. Indeed, let T be a 2DPT which realizes a function f . We can define the topological closure of $\text{dom}(T)$, which we denote $\widehat{\text{dom}(T)}$ as the words u such that there exists a sequence $(u_i)_{i \geq 1}$ where for all i , $u_i \in \text{dom}(T)$ and

$$\forall n, \exists i \forall j \geq i, |u \wedge u_j| \geq n$$

We say that the sequence $(u_i)_{i \geq 1}$ converges to u . Note that if two sequences $(u_i)_{i \geq 1}$ and $(v_j)_{j \geq 1}$ belong to $\text{dom}(T)$ and converge to the same word w , then the elements will share longer and longer prefixes. Since T is deterministic, both sequences will then also produce words that share longer and longer prefixes. Thus we can define \hat{f} , whose domain is $\widehat{\text{dom}(T)}$, where $\hat{f}(u)$ is the limit of the images of any sequence $(u_i)_{i \geq 1}$ that belong to $\text{dom}(T)$ and which converges to u . The function \hat{f} is in fact realized by the transducer T where the accepting condition is dropped. The domain of its underlying automaton is then a closed set in the classical topology over infinite words, as it is recognized by a deterministic Büchi automaton where all transitions are final (see [15, Proposition 3.7, p. 147]). Note however that since the semantics of our transducers requires an input to produce an infinite word to be in the domain of the transducer, the domain of \hat{f} might not be a closed set.

Reversible two-way transducers with no accepting condition.

Following this, let us consider the class of reversible transducers with no accepting condition (2RT), which is equivalent to saying all transitions are final within a Büchi condition.

The constructions presented in this article get simpler, with a better complexity:

► **Theorem 10.** *Given a deterministic two-way transducer T with n states and no accepting condition, we can construct a 2RT S with $O(n^{4n+1})$ states such that $\llbracket T \rrbracket = \llbracket S \rrbracket$.*

Proof. Let T be a deterministic two-way transducer with n states. Since there is no accepting condition, the construction from the proof of Theorem 7 drops the arrays of integers from the merging forest. Then the number of merging forests is in $O(n^{2n})$, and applying Theorem 7 results in an SST T' with $O(n^{2n})$ states and $2n$ variables. Hence, by applying Theorem 5 to T' , we get an equivalent 2RT S with $O(n^{4n+1})$ states. ◀

Surprisingly, the class of 2RT has the expressive power of deterministic Büchi two-way transducers. This is due to the fact that for an input to be accepted, it requires the corresponding output to be infinite. We can *hide* the Büchi acceptance condition in this restriction.

The following theorem proves that starting from a reversible Büchi transducer, we can construct one which does not use any acceptance condition. To notice that reversible and deterministic Büchi transducers have the same expressive power, one can rely on our main theorem, specializing it to a Büchi condition. Indeed, a Büchi acceptance condition corresponds to a unique coloring function associating 0 to accepting transitions and 1 otherwise. Then if we are given a deterministic Büchi two-way transducer, we are able to produce a reversible one using a coloring function on the same domain, hence a reversible Büchi two-way transducer.

► **Theorem 11.** *Let T be a reversible Büchi two-way transducer with n states. We can construct an equivalent 2RT S with $3n$ states such that $\llbracket T \rrbracket = \llbracket S \rrbracket$.*

Proof. Let T be a reversible Büchi two-way transducer with n states. We define the 2RT S similarly to T , but with three *modes* of operation (and hence thrice the states): *simulation*, *rewind* and *production*. The transducer S starts by simulating T without producing anything. Upon reaching an accepting transition t , it switches to rewind mode to and unfolds the run of T back to the previous accepting transition (or the start of the run). It finally follows the run of T and produces the corresponding output up to seeing the accepting transition t , at which point it restarts the simulation.

Then for a given word u , if u belongs to the domain of T , the run of T over u will see accepting transitions infinitely often, and hence S will go to production mode infinitely often too. Conversely, if u does not belong to the domain of T , it means that either the run of T over u only sees a finite number of accepting transitions, or does not produce an infinite word. In the latter case, S will also produce a non finite word. In the former case, S will remain in simulation mode and never produce anything, and thus u will not be in the domain of S .

We can remark that S is reversible if T is since:

- within modes, transitions of S are transitions of T ,
- The transitions from one mode to another happen on every accepting transition. Hence transitions that come to a given mode from another are exactly the ones that exit it, so two transitions cannot go to a same state upon reading the same letter.

◀

9 Conclusion

The main contribution of this paper is the result which shows that deterministic two-way transducers over infinite words with a generalized parity acceptance condition are reversible. We also show that reversible two-way transducers over infinite words are closed under composition. Our results can help in an efficient construction of two-way reversible transducers from specifications presented as RTE [13] or SDRTE [10] over infinite words. Earlier work [9] in this direction on finite words relied on an efficient translation from non-deterministic transducers used in parsing specifications to reversible ones; our results can hopefully help extend these to infinite words.

References

- 1 R. Alur, E. Filiot, and A. Trivedi. Regular transformations of infinite strings. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 65–74. IEEE Computer Society, 2012.
- 2 R. Alur, A. Freilich, and M. Raghothaman. Regular combinators for string transformations. In T. A. Henzinger and D. Miller, editors, *Joint Meeting of the 23rd EACSL Annual Conference*

- on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014, pages 9:1–9:10. ACM, 2014.
- 3 N. Baudru and P.-A. Reynier. From two-way transducers to regular function expressions. In M. Hoshi and S. Seki, editors, *22nd International Conference on Developments in Language Theory, DLT 2018*, volume 11088 of *Lecture Notes in Computer Science*, pages 96–108. Springer, 2018.
 - 4 O. Carton and G. Douéneau-Tabot. Continuous rational functions are deterministic regular. In S. Szeider, R. Ganian, and A. Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*, volume 241 of *LIPICs*, pages 28:1–28:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
 - 5 O. Carton, G. Douéneau-Tabot, E. Filiot, and S. Winter. Deterministic regular functions of infinite words. In K. Etessami, U. Feige, and G. Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPICs*, pages 121:1–121:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
 - 6 A. Cayley. A theorem of trees. 23:376–378, 1889.
 - 7 B. Courcelle. Monadic second-order definable graph transductions: A survey. *Theor. Comput. Sci.*, 126(1):53–75, 1994.
 - 8 L. Dartois, P. Fournier, I. Jecker, and N. Lhote. On reversible transducers. In I. Chatzigiannakis, P. Indyk, F. Kuhn, and A. Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 113:1–113:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
 - 9 L. Dartois, P. Gastin, R. Govind, and S. N. Krishna. Efficient construction of reversible transducers from regular transducer expressions. In C. Baier and D. Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 50:1–50:13. ACM, 2022.
 - 10 L. Dartois, P. Gastin, and S. N. Krishna. Sd-regular transducer expressions for aperiodic transformations. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021.
 - 11 L. Dartois, I. Jecker, and P. Reynier. Aperiodic string transducers. In *Developments in Language Theory - 20th International Conference, DLT 2016, Montréal, Canada, July 25-28, 2016, Proceedings*, pages 125–137, 2016.
 - 12 V. Dave, E. Filiot, S. N. Krishna, and N. Lhote. Synthesis of computable regular functions of infinite words. *Log. Methods Comput. Sci.*, 18(2), 2022.
 - 13 V. Dave, P. Gastin, and S. N. Krishna. Regular Transducer Expressions for Regular Transformations. In M. Hofmann, A. Dawar, and E. Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic In Computer Science (LICS'18)*, pages 315–324, Oxford, UK, July 2018. ACM Press.
 - 14 J. Engelfriet and H. J. Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic*, 2(2):216–254, 2001.
 - 15 D. Perrin and J.-E. Pin. *Infinite Words: Automata, Semigroups, Logic and Games*, volume 141. Elsevier, 2004.
 - 16 J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM J. Res. Dev.*, 3(2):198–200, 1959.