From Monadic Second-Order Definable String Transformations to Transducers

Rajeev Alur
University of Pennsylvania
alur@seas.upenn.edu

Antoine Durand-Gasselin
LIAFA, Université Paris Diderot
adg@liafa.univ-paris-diderot.fr

Ashutosh Trivedi
Indian Institute of Technology Bombay
trivedi@cse.iitb.ac.in

***Abstract*—Courcelle (1992) proposed the idea of using logic, in particular Monadic second-order logic (MSO), to define graph to graph transformations. Transducers, on the other hand, are executable machine models to define transformations, and are typically studied in the context of string-to-string transformations. Engelfriet and Hoogeboom (2001) studied two-way finite state string-to-string transducers and showed that their expressiveness matches MSO-definable transformations (MSOT). Alur and Černý (2011) presented streaming transducers—one-way transducers equipped with multiple registers that can store output strings, as an equi-expressive model. Natural generalizations of streaming transducers to string-to-tree (Alur and D'Antoni, 2012) and infinite-string-to-string (Alur, Filiot, and Trivedi, 2012) cases preserve MSO-expressiveness. While earlier reductions from MSOT to streaming transducers used two-way transducers as the intermediate model, we revisit the earlier reductions in a more general, and previously unexplored, setting of infinite-string-to-tree transformations, and provide a direct reduction. Proof techniques used for this new reduction exploit the conceptual tools (composition theorem and finite additive coloring theorem) presented by Shelah (1975) in his alternative proof of Büchi's theorem. Using such streaming string-to-tree transducers we show the decidability of functional equivalence for MSO-definable infinite-string-to-tree transducers.**

***Index Terms*—Streaming string transducers, monadic second-order logic, ω-regular transformations, tree transducers.**

I. Introduction

The class of regular languages of finite strings is a well-established concept [1] in formal language theory. A number of widely different and equi-expressive formalisms—based on, for instance, logic (monadic second-order logic, MSO), executable machine models (deterministic finite state automata [2], [3], [4]), concise expressions (regular expressions), and algebra (finite semigroups [5])—to recognize languages solidify the status of this class as "regular". Büchi [6] and McNaughton [7] generalized the notion of regularity to languages of infinite strings by showing equi-expressiveness of MSO and deterministic Muller automata, while Wilke [8] established the connection with ω-semigroups. Apart from showing robustness of its class of languages, regularity (in particular, the logic and automata connection) has been quite influential as specification-formalisms in varied contexts like text-editors, programming languages, and verification, since it provides computational algorithms for an expressive logic.

A natural extension to the theory of regular languages is to consider transformations of strings, i.e. functions from strings to strings, which has applications in image processing, machine translation, natural language processing, and program verification. A robust class of transformations can be defined using Courcelle's monadic second-order logic definable graph transformations [9] where MSO formulas are used to interpret the output graphs in a finite number of copies of the input graph. These transformations enjoy certain nice properties including closure under function composition and ease of defining restricted transformations such as (finite and infinite) string-to-string, string-to-tree, or tree-to-tree transformations.

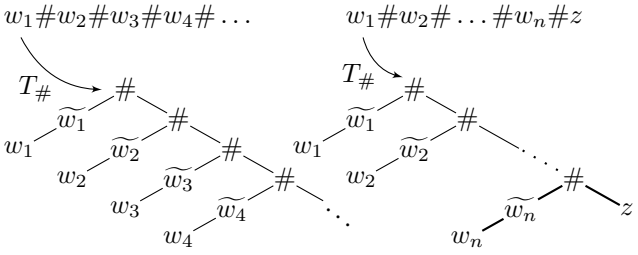

Fig. 1. An infinite string-to-tree transformation $T_\#$. Here the w's are finite $\#$-free strings, while z is a $\#$-free ω-string. The string $\widetilde{w}$ denotes the reverse of string w as a left-branch tree-string.

Engelfriet and Hoogeboom [10] showed the regularity of MSO-definable finite string-to-string transformations by proving that the two-way extension of generalized sequential machines (2GSM) express the same class of transformations. Alur and Černý [11] re-emphasized the regularity of this class by defining a one-way model, called streaming string transducers, capturing the same class of transformations. Streaming string transducers (SSTs) read their input in one left-to-right pass and construct the output by manipulating multiple write-only registers using copyless updates. Streaming string transducers is a promising computational model to capture MSO-definable transformations as its natural extensions to infinite-string-to-string [12] and finite tree-to-tree [13] transformations precisely capture corresponding MSO-definable classes.

Recently, finite string-to-tree transformations were used to define more general ways of associating costs with strings [14]. If we want to associate costs with infinite executions with similar generality, we first need to develop the theory of infinite-string-to-tree transformations, and the resulting notion of regular cost functions can be useful for quantitative analysis. In this paper we study a previously unexplored extension of SSTs to infinite string-to-tree transformations, and show

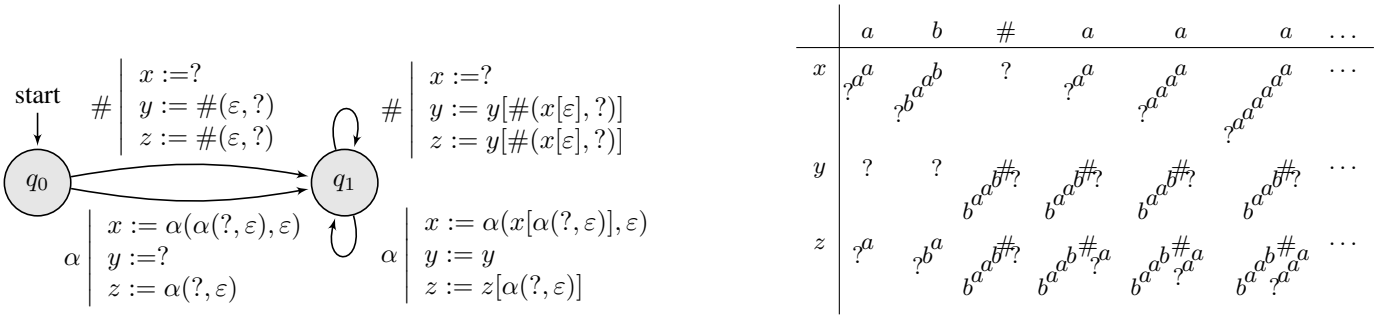

Fig. 2. Streaming string-to-term transducer implementing $T_{\#}$ from Fig. 1. Here α stands for any letter except $\#$.

its equi-expressiveness to MSO-definable transformations. We call this model streaming string-to-tree transducers (SSTTs).

A streaming string-to-tree transducer reads its input tape in one left-to-right pass. Hence, it stores output trees corresponding to various eventualities in its registers, and the exact output tree is given as the limit of a register determined by a Muller acceptance condition on its transitions. Registers of SSTTs store trees with a special symbol ? (called a hole) appearing in the leaves where further trees or register values can be substituted. Registers are updated with transition using the following set of expressions:

$$e ::= \varepsilon \mid ? \mid x \mid a(e_1, e_2) \mid x[e_1, \dots, e_n] \tag{1}$$

where x stands for a register, while a stands for an output letter. The expression ε stands for an empty tree, ? stands for an empty tree with a hole, $a(e_1, e_2)$ stands for an a-labeled tree whose left and right successors are trees corresponding to expressions e_1 and e_2, while the expression $x[e_1, \dots, e_n]$ specifies the tree stored in register x whose holes are substituted by trees corresponding to expressions $e_1, \dots, e_n$ in a left-to-right traversal of the tree.

Example 1. The infinite string-to-tree transformation shown in Figure 1 can be implemented with an SSTT (see, Fig. 2) with two states q_0 and q_1 and three registers x, y and z such that the limit of the register y is the output if $\#$ appears infinitely often, and the register z is the output, otherwise. Right side of Fig. 2 shows a computation on string $ab\#a^\omega$ and one can easily verify that the limit of the register z is the correct output.

We say that a register update is copyless if each register appears at most once in the right-hand-side. For instance, in Fig. 2 the register update in the transition $(q_1, \#)$ is not copyless since the value of register y is copied in both register y and register z, while all other updates are copyless. Unrestricted copy in register updates, e.g. $x := \alpha(x, x)$, may lead to non-linear size increase in the output, and such transformations are not expressible using MSO transducers. To disallow such behavior, traditionally streaming string transducers allowed only copyless register updates. In this paper we permit a more general update rule called *restricted copy*. Under restricted copy updates a register is allowed to be copied in multiple registers, however these registers cannot later be combined together. For instance, registers y and z are never combined in Figure 2.

We extend the theory of regular transformations to infinite string-to-tree transformations by showing the following result.

Theorem 1. *An infinite string-to-tree transformation is MSO-definable if and only if it is* SSTT*-definable, and the reduction from* SSTT *to MSO-transducers and vice-versa is effective.*

The reduction from an SSTT to an MSO transducer is a straightforward extension of similar proofs for previous SST models [11], [12], [13]. However, it is not possible to extend previous proofs in the other direction, since all previous reductions exploited the existence of an already known MSO-equivalent transducer model (e.g., 2GSM for string-to-string and macro-tree transducers for tree-to-tree transformations) and used automata-theoretic results to make a connection between such computational model and streaming string transducers. However, we do not know of any computational model that captures MSO-definable infinite string-to-tree transducers. This required us to explore further properties of MSO-definable transducers to yield a direct reduction to streaming string transducers which is also relevant for previous models. Our proof has its roots in the conceptual tools (composition theorem and finite additive coloring theorem) presented by Shelah [15] in his celebrated alternative proof of Büchi's theorem. We observed that the output infinite trees of MSO-transducers belong to an interesting class of infinite trees that seems to have not been studied earlier:

Theorem 2. *For every infinite string-to-tree MSO transducer there is a finite bound (related to the quantifier depth of the MSO formulas) on the number of infinite branches in the output trees.*

We also study the *equivalence problem* for MSO-definable infinite string-to-tree transducers that asks whether two MSO-transducers implement the same function. Figure 3 shows a difficulty in proving equivalence of string-to-tree transducers: although both transducers $T_{\neg 1}$ and $T_{\ll}$ implement equivalent transformations, they differ in their logical characterization. Transducer $T_{\neg 1}$ does not modify the labels of the nodes but excludes the first one from the output, while $T_{\ll}$ shifts all the labels to the left. The SSTT for $T_{\ll}$ will output at each step one string which will always "lag" one symbol

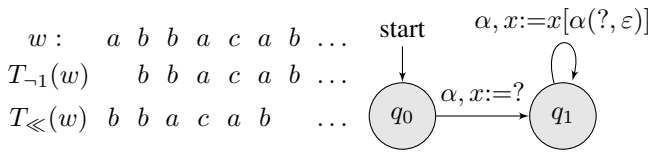

Fig. 3. Transducers $T_{\neg 1}$ and $T_{\ll}$ implementing the same transformation.

from the image up to the current position. This lag is not necessarily bounded, and it is challenging to ensure that, despite such delay, the successive outputs converge towards the image by the transducer. In Section VI we show that the equivalence problem for SSTTs can be decided using appropriate generalization of the corresponding procedure for SSTs on infinite strings [12]. This result when combined with Theorem 1 yields the following result.

Theorem 3. *Equivalence problem for* MSOT *is decidable.*

II. Preliminaries

Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{N}_k$ be the set $\{i \in \mathbb{N} : i \leq k\}$. An alphabet Σ is a finite set of letters.

A. Graphs, Strings, and Trees

A *labeled graph* G is a tuple $(V, (E_b)_{b\in\Gamma}, (L_a)_{a\in\Sigma})$ where V is the set of vertices, $(L_a)_{a\in\Sigma}$ are disjoint subsets of V whose union is V, and $(E_b)_{b\in\Gamma}$ are disjoint binary relations over V. We write $E = \cup_{b\in\Gamma} E_b$. We say that a vertex $v \in V$ is labeled with letter $a \in \Sigma$ if $v \in L_a$, and an edge $(v, v') \in E$ is labeled with letter $b \in \Gamma$ if $(v, v') \in E_b$. Let $GR(\Sigma, \Gamma)$ be the set of graphs with node labels in Σ and edge labels in Γ. We define strings and trees as special cases of labeled graphs.

- (Strings). A string s of length $n \geq 0$ over an alphabet Σ is a labeled graph $s = (\{1, 2, \ldots, n\}, S, (L_a)_{a\in\Sigma})$ where $1, 2, \ldots, n$ are letter positions in the string, S is the successor relation on $\{1, 2, \ldots, n\}$, and L_a is the set of positions of s that carry the letter a. We denote by ε the empty string i.e. the set of vertices is the empty set. We call a string ω-string if its set of vertices is the set of natural numbers.

  For a string s we write $|s|$ for its length; note that for an ω-string s we have that $|s| = \infty$. For a string s and for positions i in s we write $s[i]$ for the letter at the i-th position of the string s. For any j, $i < j \leq |s|$, we write $s[i{:}j]$, $s(i{:}j)$, $s[i{:}j)$, and $s(i{:}j]$, to denote substrings of s respectively starting at i and ending at j, starting just after i, ending just before j, and so on. For instance, $s[1{:}x)$ denotes the prefix ending just before x (it is ε if $x = 1$), while $s(x{:}|w|]$ denotes the suffix starting just after x.
- (Trees). For convenience, we only consider *binary trees* where each vertex has at most two successors. This allows us to represent nodes of a tree t as strings over $\{1, 2\}$. A tree t is a labeled graph $(\mathrm{Pos}(t), (S_i)_{i\in\{1,2\}}, (L_a)_{a\in\Sigma})$ such that $\mathrm{Pos}(t)$ is a prefix-closed subset of $\{1, 2\}^*$ and where S_1 and S_2 are 1-successor and 2-successor relations naturally defined on $\mathrm{Pos}(t)$. We say that the tree is infinite if $|\mathrm{Pos}(t)| = \infty$. The root of the tree is the node $\varepsilon \in \mathrm{Pos}(t)$, while a *leaf* of a tree t is defined as a node whose both successors are absent in $\mathrm{Pos}(t)$, i.e. $p \in \mathrm{Pos}(t)$ is a leaf if $S_1(p), S_2(p) \notin \mathrm{Pos}(t)$.

We write Σ^* and Σ^ω for the set of finite and ω-strings over Σ; and $\mathcal{T}^*_\Sigma$ and $\mathcal{T}^\omega_\Sigma$ for the set of finite and infinite trees over Σ. We write Σ^∞ and $\mathcal{T}^\infty_\Sigma$ for $\Sigma^* \cup \Sigma^\omega$ and $\mathcal{T}^*_\Sigma \cup \mathcal{T}^\omega_\Sigma$, respectively.

B. Monadic Second-Order Logic over Labeled Graphs

Regular properties of labeled graphs [16] can be formalized by monadic second order logic (MSO) denoted by $\mathrm{MSO}(\Sigma, \Gamma)$. The formulas for $\mathrm{MSO}(\Sigma, \Gamma)$ have first-order variables $x, y, \ldots$ ranging over nodes, and second-order variables $X, Y, Z, \ldots$ ranging over sets of nodes of labeled graphs. A formula is built up from *atomic formulas* of the form

$$x = y, x \in X, L_a(x), \text{ and } E_b(x, y)$$

where the node-label formula $L_a(x)$ states that the node x has the label $a \in \Sigma$, while the edge-relation $E_b(x, y)$ states that there is an edge between the node x and the node y labeled with $b \in \Gamma$. Atomic formulas are combined with *propositional connectives* $\neg$, $\wedge$, $\vee$, $\rightarrow$, and *quantifiers* $\forall$ and $\exists$ that range over both node variables and node-set variables. We say that a variable is *free* in a formula if it does not occur in the scope of some quantifier. A *sentence* is a formula without any free variable. We write $\phi(X_1, \ldots, X_k, x_1, \ldots, x_{k'})$ to denote that at most the second-order variables $X_1, \ldots, X_k$ and the first-order variables $x_1, \ldots, x_{k'}$ occur free in ϕ. For a graph G (with set of vertices V) and for valuations $N_1, \ldots, N_k \in 2^V$ and $n_1, \ldots, n_{k'} \in V$ we say that the graph G with valuation $\nu = (N_1, \ldots, N_k, n_1, \ldots, n_{k'})$ satisfies the formula $\phi(X_1, \ldots, X_k, x_1, \ldots, x_{k'})$ and we write $(G, \nu) \models \phi(X_1, \ldots, X_k, x_1, \ldots, x_{k'})$ or $G \models \phi(X_1/N_1, \ldots, X_k/N_k, x_1/n_1, \ldots, x_{k'}/n_{k'})$ if ϕ with N_i (resp., n_i) as interpretations of X_i (resp., x_i) is satisfied in the graph G.

MSO can be restricted appropriately to express regular properties of strings and trees. Next, we present some well-known results on MSO over strings that are used in this paper.

C. k-type: Elementary Equivalence of MSO over Strings

Quantifier depth of an MSO formula is defined as the maximal number of nested quantifiers appearing in the formula. For a string $s \in \Sigma^\infty$ and $k \in \mathbb{N}$ we define its k-type as the set of MSO sentences with quantifier depth at most k which hold for this string. For a given $k \in \mathbb{N}$ we write Θ_k for the set of k-types. We write $s \cong_k s'$ when two strings s, s' have the same k-type, or $[s]_{\cong_k} = \tau$ when s has k-type $\tau \in \Theta_k$.

Proposition 4 ([17])**.** *For each k the set Θ_k of k-types is finite. Moreover, every k-type $\tau \in \Theta_k$ can be represented by an MSO sentence with quantifier depth k.*

Proof: (Sketch). The finiteness of Θ_k can be shown with the following remark: there are finitely many non-equivalent

MSO formulas with r free variables and quantifier depth at most k. By induction over k: there are only finitely many non-equivalent quantifier free formulas with r free variables, as it is a boolean algebra generated by a finite basis (the set of atomic formulas); to show the finiteness of formulas with quantifier depth at most $k+1$, we rely on the finiteness of formulas with quantifier depth at most k and $r+1$ free variables: abstracting any one of these $r+1$ variables on all those formulas gives us a (finite) boolean basis to generate any formula of quantifier depth at most $k+1$. Now, notice that each k-type can be represented by an MSO sentence with quantifier depth k, for instance by corresponding Hintikka formulas [18]. ■

The next fundamental result is due to Shelah's extension [15] of Feferman-Vaught composition theorem [19] to the context of monadic second-order logic. It follows from the fact that k-types of any two strings uniquely determine the k-type of their concatenation and there is an effective procedure to compute the resulting k-type.

Proposition 5 (Composition Theorem [15])**.** *Let $\cong_k$ be the k-type equivalence relation for MSO formulas over strings.*

1) *$\cong_k$ is a monoid congruence, i.e. for strings $u, u' \in \Sigma^*$ s.t. $u \cong_k u'$ and strings $v, v' \in \Sigma^\infty$ s.t. $v \cong_k v'$, we have that $uv \cong_k u'v'$.*
2) *For strings $u, u' \in \Sigma^+$ s.t. $u \cong_k u'$, we have $u^\omega \cong_k u'^\omega$.*

III. Regular String-To-Tree Transformations

In this section we formally present MSO-definable string-to-tree transformations and give the definition of our streaming string-to-tree transducer model with matching expressiveness.

A. MSO-Definable String-to-Tree Transformations

Courcelle [9] proposed a way to use MSO to define a graph transformation $R \subseteq GR(\Sigma,\Sigma') \times GR(\Gamma,\Gamma')$. The main idea is to define a transformation $(G,G') \in R$ by defining the graph G' using a finite set of copies of the graph G. The existence of nodes, edges, and node-labels in G' is then given as $\mathrm{MSO}(\Sigma,\Sigma')$ formulas. Formally, an *MSO graph transducer* is a tuple $T{=}(\Sigma,\Gamma,\phi_{\mathrm{dom}},C,\phi_{\mathrm{nodes}},\phi_{\mathrm{edges}})$ where:

- Σ and Γ are finite sets of input and output alphabets;
- ϕ_{dom} is a closed $\mathrm{MSO}(\Sigma,\Sigma')$ formula characterizing the domain of the transformation;
- $C{=}\{1,2,\ldots,n\}$ is a finite set of copies of the nodes of the input graph;
- $\phi_{\mathrm{nodes}}{=}\left\{\phi^c_\gamma(x) : c \in C \text{ and } \gamma \in \Gamma\right\}$ is a finite set of $\mathrm{MSO}(\Sigma,\Sigma')$ formulas with a free node variable x;
- $\phi_{\mathrm{edges}}{=}\left\{\phi^{c,d}_{\gamma'}(x,y) : c,d{\in}C \text{ and } \gamma'{\in}\Gamma'\right\}$ is a finite set of $\mathrm{MSO}(\Sigma,\Sigma')$ formulas with two free node variables.

The graph transformation $[\![T]\!]$ characterized by T is defined as follows. A graph $G{=}(V,(E_b)_{b\in\Sigma'},(L_a)_{a\in\Sigma}) \in GR(\Sigma,\Sigma')$ is in the domain of $[\![T]\!]$ if $G \models \phi_{\mathrm{dom}}$ and the output is the graph $G'{=}(V',(E'_b)_{b\in\Gamma'},(L'_a)_{a\in\Gamma}) \in GR(\Gamma,\Gamma')$ such that

- V' is the set of nodes v^c such that $v \in V$, $c \in C$ and there is a unique $a \in \Gamma$ such that $G \models \phi^c_a(v)$; notice that we follow the convention that a node v^c is absent if $G \models \neg\phi^c(v)$ where $\phi^c(v) \stackrel{\text{def}}{=} \vee_{a\in\Gamma}\phi^c_a(v)$;
- $(E'_b)_{b\in\Gamma'}$ is the set of b-labeled edges such that for $v,u \in V$ and $c,d \in C$ we have that $(v^c,u^d) \in E'_b$ if $G \models \phi^{c,d}_b(v,u)$;
- $(L'_a)_{a\in\Gamma}$ is the set of a-labeled nodes such that $v^c \in L'_a$ if $G \models \phi^c_a(v)$.

Note that as the output is unique, MSO graph transducers implement functions. It is well-known [9] fact that such MSO graph transducers are closed under function composition.

An MSO string-to-tree transducer is an MSO graph transducer such that its domain is restricted to strings, while the output is restricted to trees. Such restriction can be imposed by composing two graph transducers where the first one defines the required transformation, while the second one verifies whether the output is a tree. We write MSOT for the set of string-to-tree transformations expressible by MSO transducers.

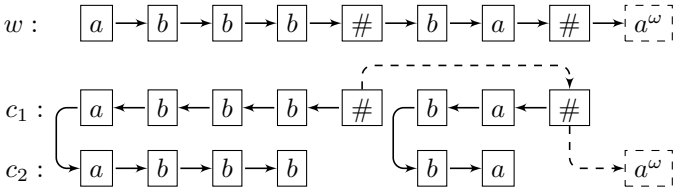

Fig. 4. The result of MSO transformation $T_\#$ on the string $abbb\#ba\#a^\omega$.

Example 2. Let us consider the following MSO formulas with their intuitive meaning: $\mathrm{reach}_\#(x)$ (holds if from x one can reach a node labeled $\#$), $\mathrm{first}(x)$ (holds if x is the first position of the string) and $\mathrm{path}(x,y)$ (holds if there is a path from x to y). Transformation $T_\#$ from Figure 1 is implemented (see Fig.4) by MSOT $T{=}(\Sigma,\Gamma,\phi_{\mathrm{dom}},C,\phi_{\mathrm{nodes}},\phi_{\mathrm{edges}})$ where:

- $\Sigma{=}\Gamma{=}\{a,b,\#\}$, $C=\{1,2\}$, and
- $\phi_{\mathrm{dom}} = true$,
- $\phi^1_\gamma(x) = L_\gamma(x) \wedge (L_\#(x) \vee \mathrm{reach}_\#(x))$
- $\phi^2_\gamma(x) = L_\gamma(x) \wedge (\neg L_\#(x))$
- $\phi^{1,1}_1(x,y){=}\phi^1(x){\wedge}\phi^1(y){\wedge}\mathrm{reach}_\#(x){\wedge}\neg L_\#(y){\wedge}E(y,x)$
- $\phi^{1,1}_2(x,y){=}\forall z((\mathrm{path}(x,z) \wedge \mathrm{path}(z,y)) \rightarrow \neg L_\#(z)) \wedge L_\#(x) \wedge L_\#(y) \wedge \mathrm{path}_\#(x,y)$
- Other formulas $\phi^{1,2}_1,\phi^{1,1}_2,\phi^{2,1}_1,\phi^{2,1}_2,\phi^{2,2}_1$ and $\phi^{2,2}_2$ can also be expressed in MSO according to Figure 4.

B. Streaming String-to-Tree Transducers

Streaming string-to-tree transducers are finite state machines that read the input string once in a left-to-right pass and manipulate a finite set of registers containing trees with marked positions (called holes) where new trees can be substituted. The set of infinitely fired transitions determines the output register that defines the output as the limit of the successive values of this register.

Before we formally introduce streaming string-to-tree transducer we discuss the set of allowable register updates. Let X be a finite set of registers and Γ be an alphabet. Let $\mathcal{T}^\infty_{\Gamma\cup\{?\}}$ be the set of trees over Γ with a special symbol $? \notin \Gamma$ (called

the *hole*) that appears only at leaf positions. Streaming string-to-tree transducers use registers to store and manipulate trees with holes using the following register expressions:

$$e ::= \epsilon \mid ? \mid x \mid a(e_1, e_2) \mid x[e_1, \ldots, e_n]$$

where $x \in X$ and $a \in \Gamma$. We write $E(\Gamma, X)$ for the set of *register expressions* over the alphabet Γ and register set X. A *register update* s of the set of registers X is defined as a mapping $s \in [X \to E(\Gamma, X)]$.

Given a register valuation $\nu : X \to \mathcal{T}_{\Gamma \cup \{?\}}$ we say that the update s is *compatible* with ν if and only if, each $x \in X$ appearing in an image of s we have that if $\nu(x)$ has n holes (?-labeled nodes) then x can only appear in the form x or $x[e_1, \ldots, e_n]$ in the images of s. If s is compatible with ν, we define the new valuation post update as $x \mapsto [\![s(x)]\!]_\nu$. This evaluation $[\![.]\!]_\nu$ is a standard conversion from terms to trees with the following difference: x is evaluated as the tree $\nu(x)$ (which must be defined because of compatibility), and $x[e_1, \ldots, e_n]$ is evaluated as the tree $\nu(x)$ in which each of the n ?-labeled leaves (in order of occurrence in the left-to-right traversal of the tree $\nu(x)$) has been replaced (in that order) by $[\![e_1]\!]_\nu, \ldots, [\![e_n]\!]_\nu$.

As we have seen in the previous subsection, MSO transducers implement transformations with linear size increase (proportional to the size of copy set). We need to add further restriction on the register updates for SSTTs so that they do not allow more general transformations. We say that a register expression $u \in E(\Sigma, X)$ is *copyless* (or linear) if each $x \in X$ occurs at most once in u. Similarly, we call an update of registers s copyless if each expression in the image is copyless and each $x \in X$ appears in at most one image. Traditionally streaming string transducers are required to have copyless register updates since it implies linear updates of contents of registers. However, we can achieve this purpose with a more general update rule called *restricted copy*. Under restricted copy update rule a register is allowed to be copied in multiple registers, however these registers can not later be combined together. Restricted copy rule is imposed by defining a symmetric and reflexive (but not transitive) *conflict relation* [13] over the set of registers. The content of one register can be duplicated in two conflicting registers, but any two conflicting registers can not be combined together, neither directly, nor indirectly by ensuring that two non-conflicting registers do not receive values from conflicting registers.

Now we are in a position to define our transducer model.

Definition 1 (SSTT)**.** A streaming string-to-tree transducer T is a tuple $(\Sigma, \Gamma, Q, q_0, \delta, X, \kappa, \rho, F)$ where

- Σ and Γ are finite input and output alphabets,
- Q is a finite set of states whose initial state is q_0,
- $\delta : Q \times \Sigma \to Q$ is a transition function,
- X is a finite set of registers,
- κ is the conflict relation over X,
- $\rho : Q \times \Sigma \times X \to E(\Gamma, X)$ is the restricted-copy (w.r.t. κ) registers update function,
- $F : 2^{Q \times \Sigma} \rightharpoonup X$ is the Muller (over transitions) output (partial) function for infinite input such that for all S such that $F(S)$ is defined we have that the update of register $F(S)$ by transitions in S are of the form $F(S)[e_1, \ldots, e_n]$ where $e_1, \ldots, e_n \in E(\Gamma, X)$.

We now detail the semantics of this model: the unique run $\varrho(T, w)$ of SSTT T over the infinite string w is the sequence $(q_i, \nu_i)_{i \leq |w|}$ where $q_i \in Q$, and ν_i is a valuation of the set of registers X, such that $q_{i+1} = \delta(q_i, w[i])$, and ν_{i+1} is defined as $x \mapsto [\![\rho(q_i, w[i], x)]\!]_{\nu_i}$ (initially ν_0 is assumed to be the empty valuation, that maps each register to ε). Let Δ be the set of infinitely often fired transitions (that is a set of tuples of $Q \times \Sigma$) in the run $\varrho(T, w)$. The output of T for the string w is defined as the limit of the sequence of values appearing in the register $F(\Delta)$ if it converges towards a tree of $\mathcal{T}_\Gamma^\infty$. The syntactic restriction on the update of the output register enforces that if the Muller set is defined then the output always converges towards a tree.

The central result (Theorem 1) of this paper is that SSTTs capture precisely the same class of transformations as MSOTs. The proof of this fact follows form the following two lemmas.

Lemma 6 (SSTT $\subseteq$ MSOT)**.** *Every* SSTT*-definable transformation is* MSOT*-definable.*

Lemma 7 (MSOT $\subseteq$ SST)**.** *Every* MSOT*-definable transformation is* SSTT*-definable.*

Lemma 6 follows from a straightforward extension of the reduction [12] from SST to MSO transducers. The proof of Lemma 7 is covered in the following two sections. In the next section we first introduce an intermediate model, SSTT with regular lookaround, and show a reduction from MSOT to this model. In Section V we complete the proof of Theorem 1 by showing that SSTTs are closed under regular lookaround.

IV. MSOT and SSTT with Regular Look-Around

We consider an extension of SSTT where the transducer can make transitions based on regular properties of the string read so far (look-behind) as well as ω-regular properties of the remaining ω-string (look-ahead). We call this extension SSTT with look-around. Notice that due to the aforementioned look-around capabilities such a transducer can be state-free since the state information can be encoded in the look-around queries. Formally, we define SSTT with look-around as the following.

Definition 2 (SSTT_{rla})**.** A streaming string-to-tree transducer with look-around is a tuple $(\Sigma, \Gamma, X, \Lambda_b, \Lambda_a, \rho, F)$ where

- Σ, Γ, and X are similar to Definition 1,
- Λ_b and Λ_a are finite sets of (pairwise disjoint) regular and ω-regular languages over Σ called regular look-behind and look-ahead, respectively.
- $\rho : \Lambda_b \times \Sigma \times \Lambda_a \times X \to E(\Gamma, X)$ is the (look-around based) copyless register-update function.
- $F : \Lambda_b \times \Sigma \times \Lambda_a \to X$ is the output function.

The semantics of SSTT_{rla} is defined as follows. A run over a string w is a finite or infinite sequence of partial

valuation of registers X, $(\nu(i))_{1\leq i\leq |w|}$ that is compatible with the update function, that is for any $i \leq |w|$, we have $\lambda_b^i \in \Lambda_b$, $\lambda_a^i \in \Lambda_a$, such that $w[1{:}i) \in \lambda_b^i$, $w(i{:}|w|] \in \lambda_a^i$ and the update of registers $x \mapsto \rho(\lambda_a^i, w[i], \lambda_b^i, x)$ is compatible with the valuation ν_{i-1} (when $i = 1$, we consider ν_0 to be the empty valuation), and $\nu_i(x) = [\![\rho(\lambda_b^i, w[i], \lambda_a^i, x)]\!]_{\nu_{i-1}}$ when it is defined. The output of the string w is then defined as the limit of the sequence $\nu_i(F(\lambda_a^i, w[i], \lambda_b^i))$ if it exists.

In the remainder of the section we prove the following Theorem:

Theorem 8 (MSOT $\subseteq$ SSTT$_{\text{rla}}$)**.** *Every* MSOT*-definable transformation is* SSTT$_{\text{rla}}$*-definable.*

A. Bounded-Crossing Property of MSOT

To show the reduction from MSOT to SSTT$_{\text{rla}}$ we first establish the so-called bounded-crossing property of MSOT that allows us to capture this class by a computational model that manipulates only a bounded number of registers. Informally, this property states that at any position in the input the number of crossings, i.e. the number of edges from one side of the position to the other side of the position, is bounded by a number that depends only on the number of copies and the quantifier depth of the formulas used in MSOT.

Given a (possibly infinite) string w, a position x of w and MSOT $T = (\Sigma, \Gamma, \phi_{\text{dom}}, C, \phi_{\text{nodes}}, \phi_{\text{edges}})$, a *crossing* at position x is a pair $(x_1, c_1), (x_2, c_2) \in \mathbb{N}_{|s|} \times C$ such that $x_1 < x < x_2$, and there is an edge in $[\![T]\!](s)$ between the nodes $x_1^{c_1}$ and $x_2^{c_2}$.

Theorem 9 (Bounded-Crossing Property)**.** *For every MSO-definable (infinite) string-to-tree (or string-to-string) transducer T, the number of crossings at any position is bounded by $2|C||\Sigma||\Theta_k|^2$, where k is the maximal quantifier depth of MSO formulas appearing in T and C is the copy set of T.*

The proof of this theorem relies on the following lemma:

Lemma 10. *Given two strings w, w' each of those with two marked positions (namely $(x_1, x_2), (x_1', x_2')$, which are not necessarily distinct and need not appear in that order in w, w'), they satisfy the same MSO formulas with 2 free first-order variables and quantifier depth at most k if and only if:*

- $w[x_1] = w'[x_1']$, $w[x_2] = w'[x_2']$
- $w[1{:}x_1) \cong_k w'[1{:}x_1')$, $w[1{:}x_2) \cong_k w'[1{:}x_2')$
- $w(x_1{:}|w|] \cong_k w'(x_1'{:}|w|]$, $w(x_2{:}|w|] \cong_k w'(x_2'{:}|w|]$
- $x_1 = x_2$ *and* $x_1' = x_2'$,
  or $x_1 < x_2$, *and* $x_1' < x_2'$ *and* $w(x_1{:}x_2) \cong_k w'(x_1'{:}x_2')$,
  or $x_1 > x_2$, *and* $x_1' > x_2'$ *and* $w(x_2{:}x_1) \cong_k w'(x_2'{:}x_1')$.

Proof of Theorem 9: Given a transducer T and a string w, let us first consider the case (see Fig. 5) of an edge from a position $y < x$ and a copy c to a position $y' > x$ and a copy d. For a given copy c, k-type of the string $w[1, y)$ (substring u in Fig 5), letter $w[y]$, and k-type of the string $w(y, x)$ (substring v is Fig. 5), we show that only one such edge may exist.

For the sake of contradiction assume that there exist two distinct such edges, and denote them by $((y_a, c), (y_a', c_a))$ and $((y_b, c), (y_b', c_b))$. We first show that this assumption implies that $((y_a, y_a'), w)$ is indistinguishable from $((y_b, y_a'), w)$ by MSO formulas with 2 free first-order variables and quantifier depth k (that, we show in the next paragraph, yields a contradiction). Let us apply lemma 10, see Fig 5, to this case: notice that $y_a < y_a'$ and $y_b < y_a'$; by assumption $w[1{:}y_a) \cong_k w[1{:}y_b)$. Now indistinguishability can be shown by exploiting monoid congruence of k-types: $w(y_a{:}x) \cong_k w(y_b{:}x)$ so $w(y_a{:}y_a') = w(y_a{:}x) \cdot w[x] \cdot w(x{:}y_a') \cong_k w(y_b{:}x) \cdot w[x] \cdot w(x{:}y_a') = w(y_b, y_a')$, thus all the hypotheses of lemma 10 are satisfied hence the indistinguishability.

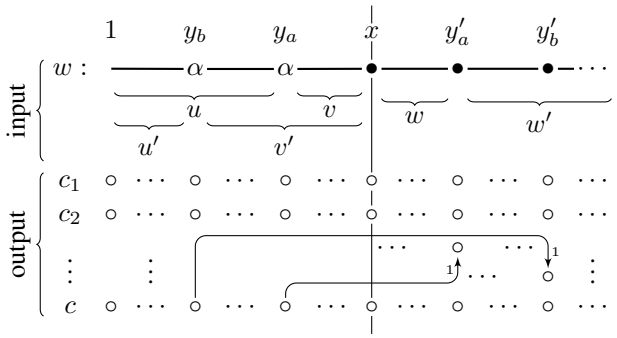

Fig. 5. If $w[y_a] = w[y_a]$, $u \cong_k u'$, $v \cong_k v'$, and y_a and y_b are left-hand of two crossing edges, then if y_a' is a right-hand of the first crossing-edge, there is also an edge from y_b to y_a': the output is not a tree.

We have an edge from y_a^c to $y_a'^{c_a}$, so $(w, (y_a, y_a')) \vDash \varphi_i^{c,c_a}$ (for some $i \in \{1, 2\}$), and since it was shown indistinguishable, we also have $(w, (y_b, y_a')) \vDash \varphi_i^{c,c_a}$, this means that there are two disjoint nodes in the output mapping to the same node (with an edge with the same label), which is not possible as the output is by definition a tree (or a string). This contradiction ensures the uniqueness of crossing edges at some position once we fixed the copy of the originating node, the k-type of the prefix from its position, the label of the position in the input string w and the k-type of the string between that position and the position at which the crossing occurs. Since there are $|C||\Sigma||\Theta_k|^2$ such choices for this case, and an equal number of choice for the other symmetric case when $y > x$ and $y' < x$, the theorem follows. ∎

Corollary 1. The image of an infinite string by a string-to-tree MSO transducer with quantifier depth at most k, and $|C|$ copies, has at most $|C||\Sigma||\Theta_k|^2$ infinite branches.

B. MSOT $\subseteq$ SSTT$_{\text{rla}}$

In this section we exploit Theorem 9 to build an equivalent SSTT$_{\text{rla}}$ from an MSOT. Given an MSOT T, an input string w (for which $T(w)$ is defined), and a position x in w, we define $T(w)\downarrow_{<x}$ as a subgraph of $T(w)$, whose set of nodes is the set of all nodes y^c (corresponding to position y and copy $c \in C$) of $T(w)$, such that $y < x$. We define $T(w)\downarrow^?_{<x}$ from $T(w)\downarrow_{<x}$ as follows: for each node in $T(w)\downarrow_{<x}$, if that node had a 1-successor (resp. 2-successor) in $T(w)$ (namely y^c), but no such successor in $T(w)\downarrow_{<x}$, (which means $y \geq x$) then we add to this node a 1-successor (resp. 2 successor) that is a

?-labeled node. We say that this ?-labeled node corresponds to y^c. Intuitively, while processing a string at position x the SSTT needs to store each $T(w)\downarrow^?_{<x}$ tree in a register. To access these trees and their holes so as to perform necessary updates, we need a way to refer to them in a unique way. The following lemma gives us exactly that.

Lemma 11. *Let T be an MSO transducer with quantifier depth at most k, w be an input string in the domain of T, and x be a position in w. The following observations hold.*

- *For $\tau_1, \tau_2 \in \Theta_k$, $a \in \Sigma$, $c \in C$, there exists at most one tree in $T(w)\downarrow^?_{<x}$ that is rooted on a node y^c, $y < x$, and $w[y] = a$, $[w[1{:}y)]_{\cong_k} = \tau_1$, $[w(y{:}x)]_{\cong_k} = \tau_2$.*
- *For $\tau_1, \tau_2 \in \Theta_k$, $a \in \Sigma$, $c \in C$, there exists at most one ?-labeled node in $T(w)\downarrow^?_{<x}$ that corresponds to a node y^c, such that $y{>}x$, and $w[y]{=}a$, $[w(x{:}y)]_{\cong_k}{=}\tau_1$, $[w(y{:}|w|]]_{\cong_k}{=}\tau_2$.*
- *For a copy $c \in C$ there exists at most one ?-labeled node in $T(w)\downarrow^?_{<x}$ that corresponds to a node x^c.*

The trees $T(w)\downarrow^?_{<x}$ and their holes can thus be characterized uniquely by an element of $\Theta_k \times \Sigma \times \Theta_k \times C$.

We will next define registers, look-around, updates and the output function of the resulting SSTT_{rla}.

a) Registers: The set of registers will be $\Theta_k{\times}\Sigma{\times}\Theta_k{\times}C$. During the processing of a string w, (for which $T(w)$ is defined), the register will satisfy the following invariant:

Invariant 1. While processing position x the register (τ_1, a, τ_2, c) contains the tree in $T(w)\downarrow^?_{<x}$ (if it exists, otherwise the register is empty) that is rooted at some node y^c such that $w[y] = a$, $[w[1{:}y)]_{\cong_k} = \tau_1$, $[w(y{:}x)]_{\cong_k} = \tau_2$.

b) Regular look-around: The set of regular look-ahead and look-behind will be K-types where $K = k + |C| + 4$ (more accurately the set, for all K-type, of the language of strings that has this K-type). So given a letter $a \in \Sigma$ and two K-types λ_b (regular look-behind) and λ_a (regular look-ahead), we need to describe the update of registers corresponding to $(\lambda_b, a, \lambda_a)$.

c) Update of registers: Given a string w for which $T(w)$ is defined, and a position x in that string, such that $w[x] = a$, $[w[1{:}x)]_{\cong_K}\lambda_b$ and $[w(x{:}|w|]]_{\cong_K} = \lambda_a$, provided the registers satisfy Invariant 1, we will now describe how to compute $T(w)\downarrow^?_{<x+1}$ using these registers, and store such trees in the appropriate registers such that the invariant remains preserved.

Lemma 12. *There exists MSO formulas with one free-variable of quantifier depth at most K that, given a string w and a position x, characterize that while processing position x*

- *whether a register is non-empty,*
- *the holes (characterized according to lemma 11) of the tree stored in the register, and*
- *the order these holes appear in an in-order traversal of the content of that register.*

The regular look-around determines the validity of these K-depth formulas and thus describes the content of registers according to lemma 12. It is now straightforward to build $T(w)\downarrow^?_{<x+1}$ and to store these trees in the register while still ensuring the invariant on the content of registers. Observe that $T(w)\downarrow_{<x}$ is a subgraph of $T(w)\downarrow_{<x+1}$, where $T(w)\downarrow_{<x+1}$ has as additional nodes the x^c that are in $T(w)$ (which are determined, together with their label, by a quantifier depth $k+1$ formula, hence by $(\lambda_b, a, \lambda_a)$). The trees in $T(w)\downarrow^?_{<x+1}$ are obtained by adding a Γ-labeled node for each of those x^c, and combining those nodes with the trees in $T(w)\downarrow^?_{<x}$.

We describe the outgoing edges for these x^c nodes. The look-around determines which of those nodes have a successor occurring before, at x or after x. For those who have a successor after x, we will give a fresh ?-labeled node as corresponding successor, for those who have as successor a x^c, we will connect it to x^c accordingly, and for those who have successor before x, we will give as corresponding successor the tree of $T(w)\downarrow^?_{<x}$ whose root is that successor. A $k+1$ depth formula determines that register (as the characterization of the root is the name of the register).

Now for the incoming edges, notice that once again the regular look-around determines where the ancestor occurs w.r.t. x. The case where the predecessor occurs after x leads to no incoming transition (hence we have a tree rooted at some x^c), the case where the predecessor is some $x^{c'}$ has been taken care of already, and for the case where the predecessor occurs before x, we insert this tree rooted at x^c instead of the ?-labeled node corresponding to x^c.

Finally we now detail how to store those trees appropriately in the registers. For those trees which are rooted at some x^c, we put them in the corresponding register $(\tau_b, w[x], [\varepsilon]_{\cong_k}, c)$ (where τ_b is the uniquely determined k-type of λ_b). A tree which was in register (τ_1, a', τ_2, c) whose root (namely y^c) was not connected to its parent, is relocated in register $(\tau_1, a', \tau_2 \cdot [w[x]]_{\cong_k}, c)$. Notice that $\tau_2 \cdot [w[x]]_{\cong_k}$ indeed corresponds to the k-type of $w(y{:}x] = w(y{:}x{+}1)$. This update of registers we described here can be encoded as a substitution as described in the previous section, and this substitution is copyless.

d) The output function: At each step during the processing, the regular look-around tells us whether the root of the image is in the prefix of the position we are processing (this property being expressible as a quantifier depth $k+1$ MSO formula). If it is, we just output the corresponding register. Thus at each step we output the tree of $T(w)\downarrow^?_{<x}$ that contains the root of $T(w)$, this sequence of trees necessarily converges towards the image $T(w)$.

The construction is now complete.

V. Closure Under Regular Look-Around

In this section we show that SSTT are closed under regular look-around by showing the following theorem. The other direction is trivial.

Theorem 13 ($\text{SSTT}_{\text{rla}} \subseteq$ SSTT)**.** *Every* SSTT_{rla}*-definable transformations is* SSTT*-definable.*

We sketch the proof of this theorem in two parts over the next two subsections. In the first subsection we show how to remove the regular look-ahead from the update function

at the cost of introducing states and relaxing copylessness to restricted copy. Note that this step is enough for SSTTs on finite strings. For infinite strings we need to also remove the regular look-ahead in the output function. In the second subsection we show how to shift from output guarded by regular look-ahead to output determined by the set of infinitely occurring transitions.

A. Removing Look-around from Register Updates

Regular look-behind can be easily removed by introducing states in the SSTT_{rla}. In the state-free SSTT_{rla} the set of regular look-behind was the set of K-types. Notice that the set of K-types covers the set of finite strings and is a finite monoid. This allows to build a finite state-transition structure, each state of which corresponds to a K-type and can be reached from the initial state by strings with the same K-type. In any such finite state-transition structure, the current state stores all the necessary information about the regular look-behind.

We now show how to remove regular look-ahead from the registers updates. We synchronize registers with all possible regular look-aheads: the set of registers is the Cartesian product of registers in the one state automaton with the set of K-types of ω-strings.

Given a configuration (q, ν) of a run over a string w, the register (r, λ) (r denoting some register in the one-state automaton, λ some K-type) is meant to contain what r would contain should the regular look-ahead be λ. When processing some letter a (in some state q), it is updated similarly as r would (with regular look-behind corresponding to q and regular look-ahead λ) from the registers with the second component being $a{\cdot}\lambda$. Formalizing this syntactic trick, if the update function in the one-state transducer with regular look-around is ρ, and we define ρ' as: $\rho'(q, a, (r, \lambda)) = \rho(\lambda_b, a, \lambda, r)[r_i/(r_i, a{\cdot}\lambda)]$ (we substitute each register r_i by $(r_i, a{\cdot}\lambda)$) where λ_b denotes the K-type of strings reaching state q.

Removing look-ahead in this fashion introduces copies in the register updates, as a K-type can be obtained by prepending the same letter to more than one K-type. However such register update follows restricted copy rule as any two registers with different K-type component conflict: given a K-type, registers with this K-type will be updated in a copyless manner with the values of registers all with the same K-type.

B. Removing Look-around from Output function

In this section we sketch how to shift from look-ahead based output to an output based on Muller condition. Removing look-ahead based output in case of finite string setting—where the regular look-ahead are K-types of finite strings—is trivial. In this case while processing the last letter of the input string, we precisely know that the regular look-ahead is the K-type of the empty string, and hence we can (for each state, that is each possible regular look-behind) output the corresponding register $(r, [\varepsilon]_{\cong_K})$. In the case of infinite strings, we do not have such a privileged position where we know the regular look-ahead, furthermore we need to output an infinite sequence of trees converging toward the image. So we need to correctly guess infinitely often the regular look-ahead, that needs to be correct eventually always, so that we output a sequence of trees that converges towards the image.

The rest of this section is dedicated to show that, relying on the concept of merging relation [15], by introducing more states and registers, we can guess the regular look-ahead from the set of infinitely occurring transitions.

Definition 3 (Merging Relation)**.** Let $w{\in}\Sigma^\infty$ and $K{\in}\mathbb{N}$, we say that two positions x and y *merge* at some position z (written $x{\sim}^w_K y(z)$, we thereafter omit K and w) if $w(x{:}z] \cong_K w(y{:}z]$. We write $x \sim y$ if x and y merge at some position.

Proposition 14. *Let $\sim$ be the merging relation.*

- *$(\sim\ (z))_{z\in w}$ and $\sim$ are equivalence relations of index bounded by the number of K-types (of finite strings).*
- *If w is an ω-string and $x \sim y$ then $w(x{:}\infty] \cong_K w(y{:}\infty]$.*

Given an infinite string, there is an equivalence class that has infinitely many representatives, and at all these positions the (K-type of the) regular look-ahead is the same. We need to determine those positions so that we can output the value of the register associated to this regular look-ahead K-type.

1) Tracking these merging equivalence classes: We will define a (Σ-labeled) transition system which tracks the following properties: the number of equivalence classes of merging at current position, and for each of those equivalence classes, the K-type of the string between the first occurring representative of any two of those equivalence classes. We can store this information in a triangular matrix with values in the set of K-types (of finite strings), of size bounded by the number of K-types (of finite strings). We choose thereafter to formally represent this "triangular matrix" by a square matrix with $|\Theta_K|$ rows and columns, and with values in the set of (finite-string) K-types with an additional dummy symbol, say $\perp$.

Given an infinite string w, a position x, let n denote the number of equivalence classes of $\sim (x)$, let $i < j \leq n$, let p_i denote the first occurrence in w of the i-th appearing equivalence class of $\sim (x)$, and p_j the first occurrence of the j-th appearing equivalence class. The element at position (i, j) in this matrix, is the K-type of $w(p_i{:}p_j]$. The rest of the matrix is filled with $\perp$. This construction is shown in Figure 6.

We now detail how to update this matrix (namely M) by processing the next position, i.e. $x{+}1$. It is done in two steps: first we "add" a new column, then we shrink the matrix as some equivalence classes can merge at this new position.

We first put values in the $n+1$-th column: $M(i, n+1) = M(i, n){\cdot}w[x+1]$, for any $i < n$; and $M(n, n+1) = [\varepsilon]_{\cong_K}$. For the latter case, we rely on the fact that as $K > 1$, ε is a K-type on its own, hence position x is alone in its $\sim (x)$ equivalence class, so it is the last occurring $\sim (x)$ equivalence class, and $x+1$ is obviously related to x by ε. For the other cases, we just built $[w(p_i{:}x+1]]_{\cong_K}$ from $[w(p_i{:}x]]_{\cong_K}$.

We then "shrink" our matrix, the idea is as follows: if two distinct $\sim (x)$ equivalence classes (i-th and i'-th) merged into a single $\sim\ (x+1)$ equivalence class, we want to remove the track of the newest (i.e. the i'-th if $i' > i$), we do so by

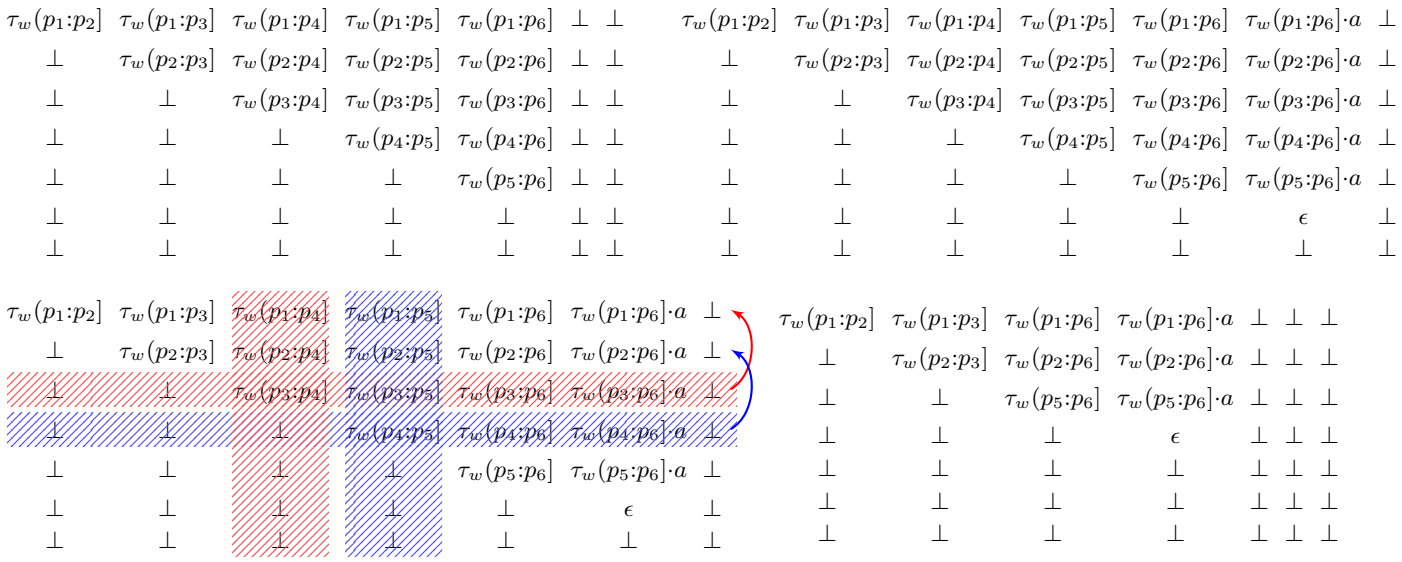

Fig. 6. Update of the matrix, from left-to-right top-to-bottom manner: (1) before processing position x; $\tau_w[p_i{:}p_j)$ denotes the K-type of $w[p_i{:}p_j)$ (2) adding an extra column $(w[x] = a)$, (3) as $[w(p2{:}p6] \cdot a]_{\cong_K} = [w(p4{:}p6] \cdot a]_{\cong_K}$ and $[w(p1{:}p6] \cdot a]_{\cong_K} = [w(p3{:}p6] \cdot a]_{\cong_K}$, we cross out the third and fourth rows and columns, (4) after processing position x.

"crossing out" in the matrix the i'-th row and column; formally the entry (x, y) is now entry (x', y') with $x' = x$ (resp. $y' = y$) if $x < i'$ (resp $y < i'$), $x' = x + 1$ (resp. $y' = y + 1$) if $x \geq i'$ (resp. $y \geq i'$). By definition two $\sim (x)$ equivalence classes (i and i') have merge into one $\sim (x + 1)$ equivalence class if and only if $M(i, n + 1) = M(i', n + 1)$. We argue that, thanks to additivity of K-types, all the information held by the matrix is contained by the $M(i, i + 1)$, for $i < |\Theta_K|$, as $M(i, j) = M(i, i + 1) + \ldots + M(j - 1, j)$. The following lemma states that look-ahead guessed using transitions in our matrix state-transition system are correct.

Lemma 15. *The set of infinitely fired transitions when running an infinite string in this transition system gives us the number of $\sim$-equivalence classes in this string. Furthermore, for each $\sim$-equivalence class, it gives us the K-type of the look-ahead at positions in this equivalence class.*

From the set of infinitely occurring transitions, we therefore deduce the K-type of the regular look-ahead from the equivalence classes, but we only know the equivalence class once it has merged, so we will need to somehow delay the output of our transducer: at each step we "pre-output" some tree according to a guess of the regular look-ahead, and effectively output it when the corresponding position has merged with the equivalence class. We achieve that by delaying output.

2) Delaying output: We present the details of outputting for a given set S of infinitely occurring transitions. This introduces finitely many new registers, among which a distinguished output register for this set S. The other cases of infinitely occurring transitions can be handled simultaneously. As S is a set of infinitely occurring transitions, it forms a strongly connected component, so it contains a merging transition and therefore there is an integer m such that m is the lowest index of column that get erased by merging. Let us denote $t \in S$ such a transition that merges column m into some column $m' < m$), this gives us the K-type (namely e^ω) of the regular look-ahead from any position in the m' $\sim$-equivalence class.

We introduce a set of $(|\Theta_K| - m)$ registers $(r_{S,m}, \ldots, r_{S,|\Theta_K|})$ that will track output candidates, and the output register r_S for the set S. When processing position x in a string w, that fired a transition $t' \in S$, denote n the number of columns of the state just after t'. This transition will have the following behavior on the $r_{S,i}$:

- In $r_{S,n}$ we put the tree that should be output just after processing x, would the regular look-ahead be e^ω: we introduce a new output candidate, and we will show that for this candidate, we will be able to say whether the guess for the regular look-ahead was correct or not.
- In $r_{S,i}$ for $m \leq i < n$ we put the older content of $r_{S,j}$ where j corresponds to the index of the column that was mapped to the i-th by the transition (or the greatest index of the column that was merged if some columns were merged into the i-th).

Hence each register $r_{S,i}$ (for any i smaller than the number of columns in current state) contains the tree that would be output by the one state automaton if the regular look-ahead was e^ω at position y where y is the last position in the i-th $\sim (x)$-equivalence class.

We finally detail how we can distinguish among our guesses which ones were correct, and should therefore be placed in the output register $r_S = F(S)$. We obtain this information when processing transition t, we know that the content of $r_{S,m}$ contains the value of the tree output by the one state automaton (would the regular look-ahead be e^ω) at the last position in the m-th $\sim (x - 1)$ equivalence class. As the m-th $\sim (x - 1)$ equivalence class has merged into the $m' \sim (x)$-equivalence

class with this transition t, if the m'-th equivalence class of merging up to current position equivalence class is already the m'-th $\sim$-equivalence class (which happens eventually) then the guess was correct, though delayed.

VI. Equivalence Problem

Two non equivalent logical definitions can define the same function (see Figure 3), but the computational model of SSTTs allows to check for functional equivalence. That is given two SSTTs M_1 and M_2, one can decide whether for every word w $M_1(w) = M_2(w)$. The case of finite input leads to a finite output, and it has been shown that for these finite cases this function equivalence problem can be solved in NExpTime [13], essentially by guessing non deterministically the index (in the in-order traversal) where two output trees differ. This problem is reduced to checking in a non-deterministic two-counter system (where counters can only be incremented) the existence of a reachable configuration where the two counters hold the same value.

As in the infinite case, the output trees can have infinite branches the positions in the trees can not be indexed as easily as their index of occurrence in the in-order traversal. We will rely on the fact that the number of infinite paths is bounded (Theorem 2) to characterize a conflicting position. We will non-deterministically guess not only a conflicting position in the two outputs but also the places where to chop the infinite branches so that we can characterize this conflicting position in the in-order traversal of those two identically trimmed trees.

Definition 4. Given a tree t, and an integer p, such that $p \leq |\mathrm{dom}(t)|$, let us denote $\#_p(t)$ the p-th occurring node in t in its preorder traversal, we denote $\sigma\chi_p(t)$ the subtree of t such that exactly every successor of $\#_p(t)$ has been removed.

We denote $\sigma\chi_{p_1,\ldots,p_k}(t)$ for $\sigma\chi_{p_k}(\sigma\chi_{p_{k-1}}(\ldots\sigma\chi_{p_1}(t)))$.

Lemma 16. *Given two trees t_1 and t_2, with at most k infinite paths, $t_1 \neq t_2$ if and only if there exists $p_1, \ldots, p_k, p_c \in \mathbb{N}$ such that $t'_1 = \sigma\chi_{p_1,\ldots,p_k}(t_1)$ and $t'_2 = \sigma\chi_{p_1,\ldots,p_k}(t_2)$ differ at position p_c, that is either their shape differs around p_c, $\#_{p_c}(t'_1).a \in \mathrm{dom}(t'_1)$ and $\#_{p_c}(t'_2).a \notin \mathrm{dom}(t'_2)$ (for some $a \in \{1, 2\}$) or vice-versa, or the label of $\#_{p_c}$ differ $t'_1(\#_{p_c}(t'_1)) \neq t'_2(\#_{p_c}(t'_2))$.*

Exploiting Lemma 16, the proof [13] of functional equivalence problem for finite string-to-tree case yield the proof for the following result.

Theorem 17. *Given two SSTTs M_1 and M_2, the problem of checking whether $[\![M_1]\!] \neq [\![M_2]\!]$ is decidable.*

VII. Conclusion

In this paper we detailed a direct reduction from MSO transducers to streaming string transducers relying on the logical concept of k-types. We also initiated the study of MSO-definable transformations from infinite strings to trees, and presented a transducer model with matching expressiveness. Using this model we established that the equivalence problem is decidable for MSO-definable string-to-tree transformations.

A key feature of our reduction is that it only exploits logical properties of MSO, such as the finiteness of k-types and the composition theorem, to yield a computational model. Thanks to this feature, it is immediate that this proof can be adapted with some efforts to establish a computational model (an appropriate star-free restriction of SSTTs) for first-order definable transformations. More ambitiously perhaps, our reduction can provide a framework for showing reductions from more expressive logic based transducers (symbolic transducers) to corresponding computational models, given an analog for the finiteness of k-types and the composition theorem. This reduction also paves the way for extending the theory of regular cost functions [14] to associate costs to infinite strings.

Acknowledgment

This research was partially supported by the NSF Expeditions in Computing award 1138996.

References

[1] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
[2] J. R. Büchi, "Weak second-order arithmetic and finite automata," *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, vol. 6, no. 1–6, pp. 66–92, 1960.
[3] C. C. Elgot, "Decision problems of finite automata design and related arithmetics," *In Transactions of the American Mathematical Society*, vol. 98, no. 1, pp. 21–51, 1961.
[4] B. A. Trakhtenbrot, "Finite automata and monadic second order logic," *Siberian Mathematical Journal*, vol. 3, pp. 101–131, 1962.
[5] M. O. Rabin and D. Scott, "Finite automata and their decision problems," *IBM Journal of Res. and Dvlp.*, vol. 3, no. 2, pp. 114–125, 1959.
[6] J. R. Büchi, "On a decision method in restricted second-order arithmetic," in *Int. Congr. for Logic Methodology and Philosophy of Science*. Standford University Press, Stanford, 1962, pp. 1–11.
[7] R. McNaughton, "Testing and generating infinite sequences by a finite automaton," *Inform. Contr.*, vol. 9, pp. 521–530, 1966.
[8] T. Wilke, "An algebraic theory for regular languages of finite and infinite words," *International Journal of Algebra and Computation*, vol. 03, no. 04, pp. 447–489, 1993.
[9] B. Courcelle, "Monadic second-order definable graph transductions: a survey," *Theoretical Computer Science*, vol. 126(1), pp. 53–75, 1994.
[10] J. Engelfriet and H. J. Hoogeboom, "MSO definable string transductions and two-way finite-state transducers," *ACM Trans. Comput. Logic*, vol. 2, pp. 216–254, 2001.
[11] R. Alur and P. Černý, "Streaming transducers for algorithmic verification of single-pass list-processing programs," in *POPL*, 2011, pp. 599–610.
[12] R. Alur, E. Filiot, and A. Trivedi, "Regular transformations of infinite strings," in *LICS*, 2012, pp. 65–74.
[13] R. Alur and L. D'Antoni, "Streaming tree transducers," in *ICALP (2)*, 2012, pp. 42–53.
[14] R. Alur, L. D'Antoni, J. V. Deshmukh, M. Raghothaman, and Y. Yuan, "Regular functions and cost register automata," in *LICS*, 2013.
[15] S. Shelah, "The monadic theory of order," *Annals of Mathematics*, vol. 102, no. 3, pp. 379–419, 1975.
[16] B. Courcelle and J. Engelfriet, *Graph Structure and Monadic Second-Order Logic: a Language Theoretic Approach*, ser. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2012.
[17] W. Thomas, "Ehrenfeucht games, the composition method, and the monadic theory of ordinal words," in *In Structures in Logic and Computer Science: A Selection of Essays in Honor of A. Ehrenfeucht, Lecture*. Springer-Verlag, 1997, pp. 118–143.
[18] J. Hintikka, *Distributive normal forms in the calculus of predicates*, ser. Acta philosophica Fennica. Edidit Societas Philosophica, 1953, no. 6.
[19] S. Feferman and R. L. Vaught, "The first order properties of products of algebraic systems," *Funda. Mathematicae*, vol. 47, pp. 57–103, 1959.

APPENDIX

A. Proofs from Section IV

Proof of Theorem 10: The "if" part is a direct consequence of the composition theorem, the hypotheses imply that the two strings are built as a sum of some (finite) string of some k-type, then a letter (which can be seen as some rather trivial structure), then another (finite) string of some k-type, another letter, and a string of some k-type. We can also use an Ehrenfeucht-Frass k-round game to show this result, the strategy of duplicator is built by composing the strategies in each of the substrings (for which he has a winning strategy for they have the same k-type).

The "only if" part is achieved by giving an explicit formula to distinguish the two strings if they are decomposed differently: the order of appearance of positions can be expressed by a quantifier free formula: they are necessarily the same; formulas holding for the substrings (and hence the k-types of the substrings) can be expressed by restricting quantifications in the formulas, which does not involve more quantifications, hence the k-types of the substrings are deduced from the formulas (with quantifier-depth k, and exactly 2 free first-order variable) validated by $(w,(x_1,x_2))$, $(w',(x'_1,x'_2))$. ∎

Proof of Lemma 11: This lemma is shown similarly as Theorem 9, assume there are to trees in $T(w)\downarrow^{?}_{<x}$ rooted at two positions y^c and y'^c such that $w[y]=w[y']$, $w[1{:}y)\cong_k w[1{:}y')$ and $w(y{:}x)\cong_k w(y'{:}x)$. One of these two trees has a root that has an ancestor in $T(w)$, say a 1-ancestor (namely $z^{c'}$). Then as $(w,(y,z))$ and $(w,(y',z))$ are indistinguishable, according to Theorem 10, we deduce that $z^{c'}$ is also a 1-ancestor which contradicts the fact that $T(w)$ is a tree.

Similarly, if two ?-labeled nodes correspond to the same quadruple (τ_1,a,τ_2,c), then by undistinguishability, the node in $T(w)$ corresponding to one ?-labeled node should also correspond in $T(w)$ to the other ?-labeled node which once again contradicts the fact that $T(w)$ is a tree.

The case where two ?-labeled nodes correspond to the same x^c is also forbidden by the tree structure of $T(w)$.

Any ?-labeled in $T(w)\downarrow^{?}_{<x}$ is thus uniquely characterized by an element of $\Theta_k\times\Sigma\times\Theta_k\times C\cup C$, as by definition of holes, it corresponds to a node occurring in $T(w)$ but not in $T(w)\downarrow_x$ which means to a node y^c for some c and some $y\geq x$. ∎

Proof of Lemma 12: By definition, a register (τ_1,a,τ_2,c) contains a value if there exists in $T(w)\downarrow^{?}_{<x}$ a tree rooted at some node y^c, such that $y<x$, and either y^c is the root of $T(w)$ or the ancestor of y occurs at or after x. The challenge is to show that we can characterize such a y by a formula of depth bounded by K. The main difficulty is to characterize a y that satisfies the specifications that $[w[1{:}y)]_{\cong_k}=\tau_1$ and $[w(y{:}x)]_{\cong_k}=\tau_2$. We rely on the fact that a k-type can be determined by a Hintikka formula which has quantifier-depth k, and if we restrict the quantifications in this formula (that is $\forall p,\ \phi$ is translated in $\forall p,\ p<y\rightarrow\phi$) we can ensure the k-types of subwords of w. Thus the existence of such a y can be written:

$$\exists y,\ y<x\wedge\mathscr{H}^{<y}_{\tau_1}\wedge P_a(y)\wedge\mathscr{H}^{>y,<x}_{\tau_2}$$

Notice this formula has quantifier-depth $k+1\leq K$. We further need to impose that such a y has an image in the copy c, and has no ancestor in $T(w)\downarrow_{<x}$, which leads to formula $E_{(\tau_1,a,\tau_2,c)}$ in Figure 7, which holds at position x iff register (τ_1,a,τ_2,c) holds a value just before processing position x. Hence the regular look around determines uniquely which registers contain a value.

We now address the case to determine which holes a register contains. Say we want to determine whether register (τ_1,a,τ_2,c) contains the hole corresponding to (τ'_1,a',τ'_2,c'), for that we existentially quantify a path (which is a common thing to do in MSO) that we check is in the register (τ_1,a,τ_2,c) (that is, in a tree of $T(w)\downarrow_{<x}$, whose root satisfies the specification imposed by the name of the register), and check that a node satisfying the specification (τ'_1,a',τ'_2,c'), hence appearing after x, is a direct successor in $T(w)$ of a node in the path. This is checked by the formula $H_{(\tau'_1,a',\tau'_2,c')\text{ in }(\tau_1,a,\tau_2,c)}$ in Figure 7. Determining whether the hole corresponding to $x^{c'}$ is in the register (τ_1,a,τ_2,c), is performed similarly by $H_{c'\text{ in }(\tau_1,a,\tau_2,c)}$.

Finally, to determine the relative order of appearance of those holes in the content of some register, one can notice that it just suffices to determine the relative order of appearance of these nodes in $T(w)$, which is rather straightforward to implement with an MSO formula, see for instance $G_{(\tau_1,a,\tau_2,c)\text{ then }(\tau'_1,a',\tau'_2,c')}$ and $G_{(\tau_1,a,\tau_2,c)\text{ then }c'}$ is Figure 7.

Notice that all these formulas have quantifier depth at most $K=k+|C|+4$, hence the regular look-around indeed determines the properties stated in lemma 12, which concludes this proof. ∎

B. Proofs from Section V

Proof of Proposition 14: The reflexivity of these relations is clear by definition, the transitivity can be shown relying on the K-types being a monoid congruence: if $x\sim y(z)$, then $x\sim y(z')$ for any position z' after z.

From this remark, we can easily deduce the bound on the index for finite strings: two positions merge iff they merge at the last position of the string, each equivalence class is characterized by a unique K-type corresponding to the K-type of suffixes from those positions.

For an infinite string, we show that for any finite set of positions, the number of merging equivalence classes is bounded by the number of K-types: take a prefix containing all these positions, merging in that prefix refines (implies) merging in the infinite string, which gives us the bound for any finite set of positions, hence the number of merging equivalence classes in the infinite string is bounded by the number of K-types. ∎

Proof of Lemma 15: There is a transition that merges in this set of infinitely often firing transitions, indeed, each transition that does not merge increases by one the number of tracked representative of equivalence classes; as the number of

$$E_{(\tau_1,a,\tau_2,c)}(x) = \exists y,\ y < x \wedge \mathscr{H}_{\tau_1}^{<y} \wedge P_a(y) \wedge \mathscr{H}_{\tau_2}^{>y,<x} \wedge \left(\bigvee_{\gamma\in\Gamma} \varphi_\gamma^c(y)\right) \wedge \forall z < x,\ \bigwedge_{d\in C} \neg \left(\varphi_1^{d,c}(z,y) \vee \varphi^{d,c}(z,y)\right)$$

$$\begin{aligned} ContainedIn_{(\tau_1,a,\tau_2,c)}((X_d)_{d\in C}, x) = &\bigwedge_{d\in C, d\neq c} \left(\forall y \in X_d,\ y < x \wedge \bigvee_{d'\in C} \exists z \in X_{d'},\ \varphi_1^{d',d}(z,y) \vee \varphi_2^{d',d}(z,y)\right) \\ &\wedge \forall y \in X_c, y < x \wedge \left((\mathscr{H}_{\tau_1}^{<y} \wedge P_a(y) \wedge \mathscr{H}_{\tau_2}^{>y,<x}) \vee \bigvee_{d\in C} \exists z \in X_d,\ \varphi_1^{d,c}(z,y) \vee \varphi_2^{d,c}(z,y)\right) \end{aligned}$$

$$H_{c' \text{ in } (\tau_1,a,\tau_2,c)}(x) = (\exists X_d)_{d\in C}\ ContainedIn_{(\tau_1,a,\tau_2,c)}((X_d)_{d\in C}), x) \wedge \bigvee_{d\in C} \exists y \in X_d,\ \varphi_1^{d,c'}(y,x) \vee \varphi_2^{d,c'}(y,x)$$

$$\begin{aligned} H_{(\tau_1',a',\tau_2',c') \text{ in } (\tau_1,a,\tau_2,c)}(x) = &(\exists X_d)_{d\in C}\ ContainedIn_{(\tau_1,a,\tau_2,c)}((X_d)_{d\in C}), x) \\ &\wedge \exists z,\ z > x \wedge \mathscr{H}_{\tau_1'}^{>x,<z} \wedge P_{a'}(z) \wedge \mathscr{H}_{\tau_2'}^{>z} \wedge \bigvee_{d\in C} \exists y \in X_d,\ \varphi_1^{d,c'}(y,z) \vee \varphi_2^{d,c'}(y,z) \end{aligned}$$

$$\begin{aligned} Subtree_{i,c_s,c_e}(x_s,x_e) = &\varphi_i^{c_s,c_e}(x_s,x_e) \vee (\exists X_d)_{d\in C} \left(\bigvee_{d\in C} \exists y \in X_d,\ \varphi_1^{d,c_e}(y,x_e) \vee \varphi_2^{d,c_e}(y,x_e)\right) \\ &\wedge \bigwedge_{d\in C} \forall y \in X_d,\ \left(\varphi_i^{c_s,d}(x_s,y) \vee \bigvee_{d'\in C} \exists z \in X_{d'},\ \varphi_1^{d',d}(z,y) \vee \varphi_2^{d',d}(z,y)\right) \end{aligned}$$

$$\begin{aligned} G_{(\tau_1,a,\tau_2,c) \text{ then } (\tau_1',a',\tau_2',c')}(x) = &\bigvee_{d\in C} \exists p,\ \left(\exists y > x,\ (\mathscr{H}_{\tau_1}^{>x,<y} \wedge P_a(y) \wedge \mathscr{H}_{\tau_2}^{>y}) \wedge Subtree_{1,d,c}(p,y)\right) \\ &\wedge \left(\exists y > x,\ (\mathscr{H}_{\tau_1'}^{<x,>y} \wedge P_{a'}(y) \wedge \mathscr{H}_{\tau_2'}^{>y}) \wedge Subtree_{2,d,c'}(p,y)\right) \end{aligned}$$

$$G_{(\tau_1,a,\tau_2,c) \text{ then } c'}(x) = \bigvee_{d\in C} \exists p,\ \left(\exists y > x,\ (\mathscr{H}_{\tau_1}^{>x,<y} \wedge P_a(y) \wedge \mathscr{H}_{\tau_2}^{>y}) \wedge Subtree_{1,d,c}(p,y)\right) \wedge Subtree_{2,d,c'}(p,x)$$

Fig. 7. MSO queries to determine how to update the registers. $\mathscr{H}_\tau^{<x}$ denotes the Hintikka formula for τ with restricted quantification.

tracked equivalence classes is bounded, this can not happen. Let us denote m the smallest index of a column that is discarded in this set of transitions: $m-1$ is the number of equivalence classes.

Assume there are m $\sim$-equivalence classes in the infinite string. Let us call e_m the first position of the last appearing $\sim$-equivalence class (the m-th). There is some position (called t) where any two $\sim$-equivalent position before position e_m have merged. Hence at position t, the first m column in the matrix correspond to the first element of each of the m $\sim$-equivalence classes (which are seen as $\sim$ (t)-equivalence classes). After this threshold no two of the m first column can merge. If the m-th column is crossed out (that is merged into another i-th column, $i < m$) infinitely often such a threshold does not exist which means that there are not as many as m $\sim$-equivalence classes. If the m-th column is not crossed out infinitely often (that is no infinitely occurring transition crosses out this column) this means that there are at least m $\sim$-equivalence classes.

Not only does the set of infinitely occurring transitions give us the number of $\sim$-equivalence classes, but it also gives us the K-type of the regular look-ahead from each position in any equivalence class.

Take one infinitely occurring merging transition which has a minimal index of crossed out column (among the set of infinitely occurring transitions). Say it merges column m into column $m' < m$, then we know that the K-type of the regular look-ahead from any position in the m'-th equivalence class is $(M(m,m'))^\omega$.

Let us define inductively a factorization of the suffix from $e_{m'}$: let $x_0 = e_{m'}$ the first element in the m'-th appearing $\sim$-equivalence class. Let us denote t_n the position where x_n has merged in the m'-th column (that is $e_{m'} \sim x_n(t_n)$). x_{n+1} is the first position that appears after t_n and which will merge through the m-th column in the m'-th, it is guaranteed to exist as there are these two columns are merged infinitely often. The matrix told us that $[w[e_{m'}{:}x_{n+1}]]_{\cong_K} = M(m',m)$, as x_{n+1} appeared after $e_{m'}$ and x_n had merged, we deduce $[w(x_n{:}x_{n+1}]]_{\cong_K} = M(m',m)$. The infinite sequence of (x_n) gives us a factorization of $w(e_{m'}{:}\infty]$, which proves its K-type to be $M(m',m)^\omega$.

We can similarly deduce that for any $m'' < m$, ($e_{m''}$ de-

noting the first position in the m''-th occurring $\sim$-equivalence class) $[w(e_{m''}{:}\infty]]_{\cong_K} M(m'', m')(M(m', m))^\omega$. ■

C. Proofs from Section VI

Proof of Theorem 17: Two SSTTs are not equivalent if either for some input has an image through only one of the two SSTTs or there exists an input w such that $M_1(w) \neq M_2(w)$. We find if it exists such a counterexample thanks to lemma 16.

We sketch a reduction of this problem very similarly as in [13], except that we also guess the positions $p_1, \ldots, p_k$ where to trim the tree. Therefore we can reduce this equivalence problem to checking in a nondeterministic $2(k+1)$-counters $(c_1, \ldots, c_k, c_c, c'_1, \ldots, c'_k, c'_c)$ system the reachability of a configuration in which each pair (c_i, c'_i) of counters has the same value.

If the two transducers are inequivalent, the counters (c_i, c'_i) will eventually contain the integers p_i that are exhibited by a counterexample.

The set of states will be a product of the set of states of the two SSTTs and also track the number of holes in each register and what is the relative position of the content of each register w.r.t. to these positions $p_1, \ldots, p_k, p_c$. There are many possible cases some of them are depicted in Figure 8, for starters the content of the register might not appear in the output, or its root can be a successor of a position p_i, or a position p_i can be on the path to a hole (as p_3 in Figure 8), or "between" two holes (as p_2), or be the successor of some hole (as p_4), or appear "before" (as p_1, not "above") or "after" (as p_5, p_c) the content of the register.

We non-deterministically make all possible choices, when a term of the form $a(e_1, e_2)$ (this a can correspond to a position p_i or not, notice that in the case of position p_c one must also ensure the divergence at this position in the two output trees, for example it was a $a(e_1, e_2)$ in M_1 and $b(e_1, e_2)$ in M_2) each corresponding positions in the two trees need not to appear on the same transition. Then we check that the combinations of registers is sound (a register appearing above some position can not be inserted in a hole of a register appearing below that position; a register whose content does not appear in the image can not be inserted into a register whose content is). Also for each term of the form $a(e_1, e_2)$ in the update we also need to increment the value of the corresponding counters (e.g. none if it appears below a p_i or all those corresponding to positions appearing below of after it).

If one can reach a configuration in which one output register for each SSTT, is such that each hole is below a position (meaning counter won't get incremented anymore) , and the values of each counter pairwise agrees with those in the others, then one has found an input word on which the two SSTTs produce different output. This is guaranteed to exist if the two SSTTs produce a different output on a same word. ■

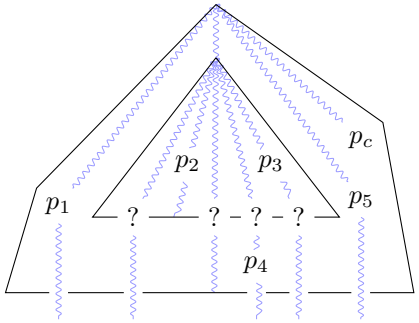

Fig. 8. Tracking the positions of p_i w.r.t. registers.