First-order Definable String Transformations

Emmanuel Filiot[1], Shankara Narayanan Krishna[2], and Ashutosh Trivedi[2]

1 **F.N.R.S. Research Associate, Université Libre de Bruxelles, Belgium**
efiliot@ulb.ac.be
2 **Department of Computer Science and Engineering, IIT Bombay, India**
krishnas,trivedi@cse.iitb.ac.in

Abstract

The connection between languages defined by computational models and logic for languages is well-studied. Monadic second-order logic and finite automata are shown to closely correspond to each-other for the languages of strings, trees, and partial-orders. Similar connections are shown for first-order logic and finite automata with certain aperiodicity restriction. Courcelle in 1994 proposed a way to use logic to define functions over structures where the output structure is defined using logical formulas interpreted over the input structure. Engelfriet and Hoogeboom discovered the corresponding "automata connection" by showing that two-way generalised sequential machines capture the class of monadic-second order definable transformations. Alur and Cerny further refined the result by proposing a one-way deterministic transducer model with string variables – called the streaming string transducers – to capture the same class of transformations. In this paper we establish a transducer-logic correspondence for Courcelle's first-order definable string transformations. We propose a new notion of transition monoid for streaming string transducers that involves structural properties of both underlying input automata and variable dependencies. By putting an aperiodicity restriction on the transition monoids, we define a class of streaming string transducers that captures exactly the class of first-order definable transformations.

1998 ACM Subject Classification F.4 Mathematical Logic and Formal Languages

Keywords and phrases First-order logic, streaming string transducers

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.147

1 Introduction

The class of regular languages is among one of the most well-studied concept in the theory of formal languages. Regular languages have been precisely characterized widely by differing formalisms like monadic second-order logic (MSO), finite state automata, regular expressions, and finite monoids. The connection [8] between finite state automata and monadic second-order logic (MSO) is among the most celebrated results of formal language theory. Over the years, there has been substantial research to establish similar connections for the languages definable using first-order logic (FO) [11]. Aperiodic automata are restrictions of finite automata with certain aperiodicity restrictions defined through aperiodicity of their transition monoid. Recall that the transition monoid of an automaton A is the set of Boolean transition matrices M_s, for all strings s, indexed by states of A: $M_s[p][q] = 1$ if and only if there exists a run from p to q on s. The set of matrices M_s is a finite monoid. It is aperiodic if there exists $m \geq 0$ such that for all $s \in \Sigma^*$, $M_{s^m} = M_{s^{m+1}}$. Aperiodic automata define exactly FO-definable languages [17, 11]. Other formalisms capturing FO-definable languages include counter-free automata, star-free regular expressions, and very weak alternating automata.

34th Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2014).
Editors: Venkatesh Raman and S. P. Suresh; pp. 147–159

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

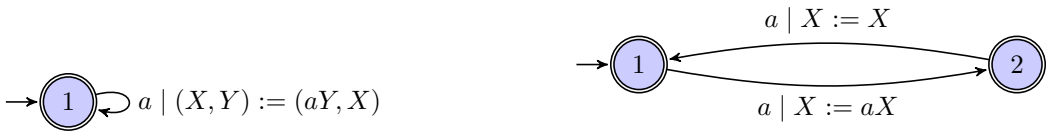


Figure 1 SSTs, T_0 (shown left) and T_1 (shown right), implement the transformation f_{halve}.

Beginning with the work of Courcelle [10], logic and automata connections have also been explored in context of string transformations. The first result in this direction is by Engelfriet and Hoogeboom [12], where MSO-definable transformations have been shown to be equivalent to two-way finite transducers. This result has then been extended to trees and macro-tree transducers [13]. Recently, Alur and Černý [1, 2] introduced *streaming string transducers*, a one-way finite transducer model extended with variables, and showed that they precisely capture MSO-definable transformations not only in finite string-to-string case, but also for infinite strings [6] and tree [3, 5] transformations.

Streaming string transducers (SSTs) manipulate a finite set of string variables to compute their output as they read the input string in one left-to-right pass. Instead of appending symbols to the output tape, SSTs concurrently update all string variables using a concatenation of output symbols and string variables in a *copyless* fashion, i.e. no variable occurs more than once in each concurrent variable update. The transformation of a string is then defined using an output (partial) function F that associates states with a copyless concatenation of string variables. It has been shown that SSTs have good algorithmic properties (such as decidable type-checking, equivalence) [1, 2] and naturally generalize to various settings like trees and nested words [3, 5], infinite strings [6], and quantitative languages [4].

In this paper we study FO-definable string transformation and recover a logic and transducer connection for such transformations. Such FO transformations, although weaker than MSO transformations, still enjoy a lot of expressive power: for instance they can still double, reverse, and swap strings, and are closed under FO look-ahead. We introduce a new concept of transition monoid for SST, used to define the notion of aperiodic SST to capture FO-definable transformations. To appreciate the challenges involved in finding the right definition of aperiodicity for SSTs consider the transformation f_{halve} defined as $a^n \mapsto a^{\lceil \frac{n}{2} \rceil}$ implemented by two SSTs shown in in Figure 1. Intuitively f_{halve} is not FO-definable since it requires to distinguish based on the parity of the input. Consider, the SST T_1 shown in Figure 1 with 2 accepting states and 1 variable.

Readers familiar with aperiodic automata may notice that the automata corresponding to T_1 is not aperiodic, but indeed has period 2. It seems a valid conjecture that SSTs whose transition monoid of underlying automaton is aperiodic characterize first-order definable transformations. However, unfortunately this is not a sufficient condition as shown by the SST T_0 in Figure 1 which also implements f_{halve} (its output is $F(1) = X$). In this example, although the underlying automaton is aperiodic, variables contribute to certain non aperiodicity.

We capture the notion of aperiodicity in SSTs by introducing the notion of variable flow. We say that by reading letter a, variable X flows to Y (if the update of variable Y is based on variable X). The notion of transition monoid is extended to SSTs to take both state and variable flow into account. We define transition matrices M_s indexed by pairs (p, X) where p is a state and X is a variable. Since in general, for copy-full SSTs, a variable X might be copied in more than one variable, it could be that X flows into Y several times. Our notion of transition monoid also takes into account, the number of times a variable flows into another. In particular, $M_s[p, X][q, Y] = i$ means that there exists a run from p to q on s on

which X flows to Y for i number of times. Hence the transition monoid of an SST may not be finite. A key contribution of this paper is that FO string transformations are exactly the transformations definable by SSTs whose transition monoid is aperiodic with matrix values ranging over $\{0, 1\}$ (called 1-bounded transition monoid). In contrast with [1] our proof is not based on the intermediate model of two-way transducers and is more direct. We give a logic-based proof that simplifies that of [5] by restricting it to string-to-string transformations. We also show that checking aperiodicity of an SST is PSPACE-COMPLETE. Finally, simple restrictions on SST transition monoids naturally capture restrictions on variable updates that has been considered in other works. For instance, *bounded copy* of [6] correspond to finiteness of the transition monoid, while *restricted copy* of [3] correspond to its 1-boundedness.

Related work. Diekert and Gastin [11] presented a detailed survey of several automata, logical, and algebraic characterisations of first-order definable languages. As mentioned earlier the connection between MSO and transducers have been investigated in [1, 12]. A connection between two-way transducers and FO-transformations has been mentioned in [9] in an oral communication, where authors left the SST connection as an open question. First-order transformations are considered in [15], but not in the sense of [10]. In particular, they are weaker, as they cannot double strings or mirror them, and are definable by one-way (variable-free) finite state transducers. Finally, [7] considers first-order definable transformations *with origin information*. The semantics is different from ours, because these transformations are not just mapping from string to strings, but they also connect output symbols with input symbols from where they originate. The first-order definability problem for regular languages is known to be decidable. In particular, given a deterministic automaton A, deciding whether A defines a first-order language can be decided in PSPACE. Although we make an important and necessary step in answering this question in the context of regular string transformation, the decidability remains an open problem.

For the lack of space proofs are either sketched or omitted; full proofs can be found in [14].

2 Preliminaries

A string over a finite alphabet Σ is a finite sequence of letters from Σ. We write ϵ for the empty string and by Σ^* for the set of strings over Σ. A language over Σ is a subset of Σ^*. For a string $s \in \Sigma^*$ we write $|s|$ for its length and $\mathrm{dom}(s)$ for the set $\{1, \ldots, |s|\}$ of its positions. For all $i \in \mathrm{dom}(s)$ we write $s[i]$ for the i-th letter of the string s. For any $j \in \mathrm{dom}(s)$, the substring starting at position i and ending at position j is defined as ϵ if $j < i$ and by the sequence of letters $s[i]s[i+1] \ldots s[j]$ otherwise. We write $s[i{:}j]$, $s(i{:}j)$, $s[i{:}j)$, and $s(i{:}j]$, to denote substrings of s respectively starting i and ending at j, starting at $i{+}1$ and ending at $j{-}1$, and so on.

We represent a string $s \in \Sigma^*$ by the relational structure $\Xi_s{=}(\mathrm{dom}(s), \preceq^s, (L^s_a)_{a\in\Sigma})$, called the string model of s, where $\preceq^s$ is a binary relation over the positions in s characterizing the natural order, i.e. $(i, j) \in \preceq^s$ if $i \leq j$; L^s_a, for all $a \in \Sigma$, are the unary predicates that hold for the positions in s labeled with the alphabet a, i.e., $L^s_a(i)$ iff $s[i] = a$, for all $i \in \mathrm{dom}(s)$. When it is clear form context we drop the superscript s from the relations $\preceq^s$ and L^s_a.

2.1 First-order logic for strings

Properties of strings over Σ can be formalized by first-order logic denoted by $\mathrm{FO}(\Sigma)$. Formulas of $\mathrm{FO}(\Sigma)$ are built up from variables $x, y, \ldots$ ranging over positions of string models along

with *atomic formulas* of the form $x{=}y, x{\preceq}y,$ and $L_a(x)$ for all $a \in \Sigma$. Atomic formulas are connected with *propositional connectives* $\neg$, $\wedge$, $\vee$, $\rightarrow$, and *quantifiers* $\forall$ and $\exists$ that range over node 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 with no free variables. We write $\phi(x_1, x_2, \ldots, x_k)$ to denote that at most the variables $x_1, \ldots, x_k$ occur free in ϕ. For a string $s \in \Sigma^*$ and for positions $n_1, n_2, \ldots, n_k \in \mathrm{dom}(s)$ we say that s with valuation $\nu = (n_1, n_2, \ldots, n_k)$ satisfies the formula $\phi(x_1, x_2, \ldots, x_k)$ and we write $(s, \nu) \models \phi(x_1, x_2, \ldots, x_k)$ or $s \models \phi(n_1, n_2, \ldots, n_k)$ if formula ϕ with n_i as the interpretations of x_i satisfies in string model Ξ_s. The language defined by an FO sentence ϕ is $L(\phi) \stackrel{\text{def}}{=} \{s \in \Sigma^* : \Xi_s \models \phi\}$. We say that a language L is FO-definable if there is an FO sentence ϕ such that $L = L(\phi)$.

2.2 Aperiodic Finite Automata

A finite automaton (FA)is a tuple $\mathcal{A} = (Q, q_0, \Sigma, \delta, F)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is an input alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function, and $F \subseteq Q$ is the set of accepting states. (q, a, q') denotes a transition of the automaton $\mathcal{A}$ from q to q' on a; this is written as $q \xrightarrow{a} q'$. We write $q_0 \rightsquigarrow^s_{\mathcal{A}} q_n$ to denote a run from q_0 to q_n on string s; (or $q_0 \rightsquigarrow^s q_n$ if the automaton is clear from the context) s is accepted if $q_n \in F$. The language defined by a finite automaton $\mathcal{A}$ is $L(\mathcal{A}) = \{s : q_0 \rightsquigarrow^s q_n \text{ and } q_n \in F\}$.

Recall that a monoid is an algebraic structure $(M, \cdot, e)$ with a non-empty set M, a binary operation $\cdot$, and an identity element $e \in M$ such that for all $x, y, z \in M$ we have that $(x \cdot (y \cdot z)){=}((x \cdot y) \cdot z)$, and $x \cdot e = e \cdot x$ for all $x \in M$. We say that a monoid $(M, \cdot, e)$ is *finite* if the set M is finite. We say that a monoid $(M, ., e)$ is *aperiodic* [17] if there exists $n \in \mathbb{N}$ such that for all $x \in M$, $x^n = x^{n+1}$. Note that for finite monoids, it is equivalent to require that for all $x \in M$, there exists $n \in \mathbb{N}$ such that $x^n = x^{n+1}$. The following monoids are of special importance in this paper.

1. **Free Monoid.** The set of all strings over Σ, denoted as $(\Sigma^*, ., \epsilon)$ and known as the free monoid, has string concatenation as the operation and the empty string ϵ as the identity.
2. **Transition Monoid.** The set of transition matrices of an automaton $\mathcal{A} = (Q, q_0, \Sigma, \delta, F)$ forms a finite monoid with matrix multiplication as the operation and the unit matrix $\mathbf{1}$ as the identity element. This monoid is denoted as $\mathcal{M}_{\mathcal{A}} = (M_{\mathcal{A}}, \times, \mathbf{1})$ and known as transition monoid of $\mathcal{A}$. Formally, the set $M_{\mathcal{A}}$ is the set of $|Q|$-square Boolean matrices $M_A = \{M_s : s \in \Sigma^*\}$ where for all strings $s \in \Sigma^*$, we have that $M_s[p][q] = 1$ iff $p \rightsquigarrow^s q$.

We say that a FA is aperiodic if its transition monoid is aperiodic. It is well-known [17] that a language $L \subseteq \Sigma^*$ is FO-definable iff it is accepted by some aperiodic FA.

3 Aperiodic String Transducers

For sets A and B, we write $[A \to B]$ for the set of functions $F : A \to B$, and $[A \rightharpoonup B]$ for the set of partial functions $F : A \rightharpoonup B$. A string-to-string transformation from an input alphabet Σ to an output alphabet Γ is a partial function in $[\Sigma^* \rightharpoonup \Gamma^*]$. We have seen some examples of string-to-string transformations in the introduction. For the examples of first-order definable transformations we use the following representative example.

▶ **Example 1.** Let $\Sigma{=}\{a, b\}$. For all strings $s \in \Sigma^*$, we denote by $\overline{s}$ its mirror image, and for all $\sigma \in \Sigma$, by $s\backslash\sigma$ the string obtained by removing all symbols σ from s. The transformation $f_1 : \Sigma^* \rightharpoonup \Sigma^*$ maps any string $s \in \Sigma^*$ to the output string $(s\backslash b)\overline{s}(s\backslash a)$. For example, $f_1(abaa) = aaa.aaba.b$.

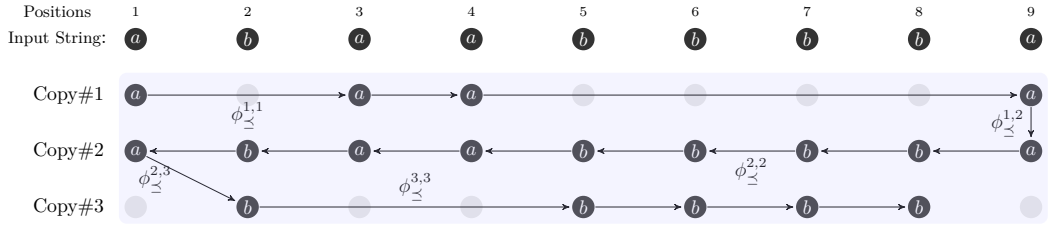


Figure 2 First-Order Transduction $w \mapsto (w\backslash b)\overline{w}(w\backslash a)$.

3.1 First-order logic definable Transformations

Courcelle [10] initiated the study of structure transformations using monadic second-order logic. In this paper, we restrict this logic-based transformation model to FO-definable string transformations. The main idea of Courcelle's transformations is to define a transformation $(w, w') \in R$ by defining the string model of w' using a finite number of copies of positions of the string model of w. The existence of positions, various edges, and position labels are then given as FO(Σ) formulas.

An *FO string transducer* is a tuple $T{=}(\Sigma, \Gamma, \phi_{\text{dom}}, C, \phi_{\text{pos}}, \phi_{\preceq})$ where: Σ and Γ are (finite) input and output alphabets; ϕ_{dom} is a closed FO(Σ) formula characterizing the domain of the transformation; $C{=}\{1, 2, \ldots, n\}$ is a finite index set; $\phi_{\text{pos}}{=}\left\{\phi^c_\gamma(x) : c \in C \text{ and } \gamma \in \Gamma\right\}$ is a finite set of FO(Σ) formulas with a free position variable x; $\phi_{\preceq}{=}\left\{\phi^{c,d}_{\preceq}(x, y) : c, d \in C\right\}$ is a finite set of FO(Σ) formulas with two free position variables x and y. The transformation $[\![T]\!]$ defined by T is as follows. A string s with $\Xi_s = (\text{dom}(s), \preceq, (L_a)_{a\in\Sigma})$ is in the domain of $[\![T]\!]$ if $s \models \phi_{\text{dom}}$ and the output is the relational structure $M = (D, \preceq^M, (L^M_\gamma)_{\gamma\in\Gamma})$ such that $D = \{v^c : c \in \text{dom}(s), c \in C \text{ and } \phi^c(v)\}$ is the set of positions where $\phi^c(v) \stackrel{\text{def}}{=} \vee_{\gamma\in\Gamma}\phi^c_\gamma(v)$; $\preceq^M \subseteq D{\times}D$ is the ordering relation between positions and it is such that for $v, u \in dom(s)$ and $c, d \in C$ we have that $v^c \preceq^M u^d$ if $w \models \phi^{c,d}_{\preceq}(v, u)$; and for all $v^c \in D$ we have that $L^M_\gamma(v^c)$ iff $\phi^c_\gamma(v)$. Observe that the output is unique and therefore FO transducers implement functions. However, note that the output structure may not always be a string. We say that an FO transducer is a *string-to-string* transducer if its domain is restricted to string graphs and the output is also a string graph.

We say that a string-to-string transformation is FO-definable if there exists an FO string-to-string transducer implementing the transformation and write FOT for the set of FO-definable string-to-string transformations. We define the *quantifier rank* $qr(T)$ of an FOT T as the maximal quantifier rank of any formula in T, plus 1. We add 1 for technical reasons, mainly because defining the successor relation requires one quantifier.

▶ **Example 2.** Consider the transformation f_1 of Example 1. It can be defined using an FO transducer that uses three copies of the input domain as shown in Fig. 2.

The domain formula ϕ_{dom}, an FO formula, simply characterizes valid string models. The first copy corresponds to $(w\backslash b)$, therefore the label formula $\phi^1_\gamma(x)$ is defined by false if $\gamma = b$ in order to filter out the input positions labelled b, and by true otherwise. The second copy corresponds to $\overline{w}$, hence all positions of the input are kept and their labels preserved, but the edge direction is complemented; hence the label formula is $\phi^2_\gamma(x) = L_\gamma(x)$. The third copy corresponds to $(w\backslash a)$ and hence $\phi^3_\gamma(x)$ is true if $\gamma = b$ and false otherwise. The transitive closure of the output successor relation is defined by $\phi^{1,1}_{\preceq}(x, y) = x \preceq y$, $\phi^{2,2}_{\preceq}(x, y) = y \preceq x$, $\phi^{3,3}_{\preceq}(x, y) = x \preceq y$, $\phi^{c,c'}_{\preceq}(x, y) =$ true if $c < c'$, and $\phi^{c,c'}_{\preceq}(x, y) =$ false if $c' < c$. Note that the transitive closure is not depicted on the figure, but only the successor relation.

Using first-order logic we define the position successor relation the following way: for all copies c, d, the existence of a direct edge from a position x^c to a position y^d of the output, also called the successor relation $S(x^c, y^d)$, is defined by the formula $\phi^{c,d}_{\text{succ}}(x,y) \stackrel{\text{def}}{=} \phi^{c,d}_{\prec}(x,y) \wedge \neg\exists z. \bigvee_{e\in C} \phi^{c,e}_{\prec}(x,z) \wedge \phi^{e,d}_{\prec}(z,y)$ where $\phi^{c_1,c_2}_{\prec}(x_1,x_2) \stackrel{\text{def}}{=} \phi^{c_1,c_2}_{\preceq}(x_1,x_2) \wedge x_1 \neq x_2$ for all $c_1, c_2 \in C$.

3.2 Streaming String Transducers

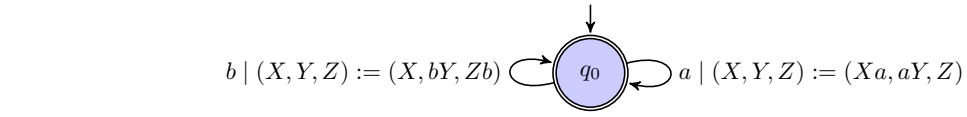
Streaming string transducers [1, 2] are one-way finite-state transducers that manipulates a finite set of string variables to compute its output. Instead of appending symbols to the output tape, SSTs concurrently update all string variables using a concatenation of string variables and output symbols. The transformation of a string is then defined using an output (partial) function F that associates states with a concatenation of string variables, s.t. if the state q is reached after reading the string and $F(q)$=xy, then the output string is the final valuation of x concatenated with that of y. In this section we formally introduce SSTs and introduce restrictions on SSTs that capture FO-definable transformations.

Let $\mathcal{X}$ be a finite set of variables and Γ be a finite alphabet. A substitution σ is defined as a mapping $\sigma : \mathcal{X} \to (\Gamma \cup \mathcal{X})^*$. A valuation is defined as a substitution $\sigma : \mathcal{X} \to \Gamma^*$. Let $\mathcal{S}_{\mathcal{X},\Gamma}$ be the set of all substitutions $[\mathcal{X} \to (\Gamma \cup \mathcal{X})^*]$. Any substitution σ can be extended to $\hat{\sigma} : (\Gamma \cup \mathcal{X})^* \to (\Gamma \cup \mathcal{X})^*$ in a straightforward manner. The composition $\sigma_1\sigma_2$ of two substitutions σ_1 and σ_2 is defined as the standard function composition $\hat{\sigma_1}\sigma_2$, i.e. $\hat{\sigma_1}\sigma_2(X) = \hat{\sigma_1}(\sigma_2(X))$ for all $X \in \mathcal{X}$. We now introduce streaming string transducers.

▶ **Definition 3.** A streaming string transducer is a tuple $(\Sigma, \Gamma, Q, q_0, Q_f, \delta, \mathcal{X}, \rho, F)$ where: (1) Σ and Γ are (finite) input and output alphabets; (2) Q is a finite set of states with initial state q_0; (3) $\delta : Q \times \Sigma \to Q$ is a transition function; (4) $\mathcal{X}$ is a finite set of variables; (5) $\rho : (Q \times \Sigma) \to \mathcal{S}_{\mathcal{X},\Gamma}$ is a variable update function; (6) Q_f is a subset of final states; and (7) $F : Q_f \rightharpoonup \mathcal{X}^*$ is an output function.

The concept of a run of an SST is defined in an analogous manner to that of a finite state automaton. The sequence $\langle\sigma_{r,i}\rangle_{0\le i\le|r|}$ of substitutions induced by a run $r = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \ldots q_{n-1} \xrightarrow{a_n} q_n$ is defined inductively as the following: $\sigma_{r,i}$=$\sigma_{r,i-1}\rho(q_{i-1}, a_i)$ for $1 < i \le |r|$ and $\sigma_{r,1} = \rho(q_0, a_1)$. We denote $\sigma_{r,|r|}$ by σ_r. If the run r is final, i.e. $q_n \in Q_f$, we can extend the output function F to the run r by $F(r) = \sigma_\epsilon\sigma_r F(q_n)$, where σ_ϵ substitutes all variables by their initial value ϵ. For all strings $s \in \Sigma^*$, the output of s by T is defined only if there exists an accepting run r of T on s, and in that case the output is denoted by $T(s) = F(r)$. The transformation $[\![T]\!]$ defined by an SST T is the function $\{(s, T(s)) : T(s) \text{ is defined}\}$.

▶ **Example 4.** Let us consider the SST T_2 with one state q_0 and three variables X, Y, and Z, shown below implementing the transformation f_1 introduced in Example 1. The variable update is shown in the figure and the output function is s.t. $F(q_0) = XYZ$.



Let r be the run of T_2 on $s = abaa$. We have $\sigma_{r,1} : (X,Y,Z) \mapsto (Xa, aY, Z)$, $\sigma_{r,2} : (X,Y,Z) \mapsto \sigma_{r,1}(X, bY, Zb) = (Xa, baY, Zb)$, $\sigma_{r,3} : (X,Y,Z) \mapsto \sigma_{r,2}(Xa, aY, Z) = (Xaa, abaY, Zb)$ and $\sigma_{r,4} : (X,Y,Z) \mapsto \sigma_{r,3}(Xa, aY, Z) = (Xaaa, aabaY, Zb)$. Therefore we have that

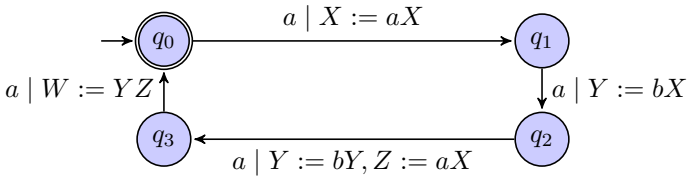
$$T(s) = F(r) = \sigma_\epsilon\sigma_{r,4}F(q_0) = \sigma_\epsilon\sigma_{r,4}(XYZ) = \sigma_\epsilon(XaaaaabaYZb) = aaaaabab.$$

3.3 SSTs: Transition Monoid and Aperiodicity

We define the notion of aperiodic SSTs by introducing an appropriate notion of transition monoid for transducers. The transition monoid of an SST T is based on the effect of a string s on the states and variables. The effect on variables is characterized by, what we call, flow information that is given as a relation that describes the number of copies of the content of a given variable that contribute to another variable after reading a string s.

State and Variable Flow. Let $T = (\Sigma, \Gamma, Q, q_0, Q_f, \delta, \mathcal{X}, \rho, F)$ be an SST. Let s be a string in Σ^* and suppose that there exists a run r of T on s. Recall that this run induces a substitution σ_r that maps each variable $X \in \mathcal{X}$ to a string $u \in (\Gamma \cup \mathcal{X})^*$. For string variables $X, Y \in \mathcal{X}$, states $p, q \in Q$, and $n \in \mathbb{N}$ we say that n copies of Y flow to X from p to q if there exists a run r on s from p to q, and Y occurs n times in $\sigma_r(X)$. We denote the flow with respect to a string s as $(p, Y) \rightsquigarrow_n^s (q, X)$.

▶ **Example 5.** Consider the run r from q_0 to q_0 over the string $aaaa$ in the following SST. To minimize clutter, while drawing SSTs we omit updates of variables that remain unchanged.



On the run r on $aaaa$, $\sigma_{r,4}(W) = \sigma_{r,3}[W := YZ] = \sigma_{r,3}(Y)\sigma_{r,3}(Z)$. However, $\sigma_{r,3}(Y) = b\sigma_{r,2}(Y) = b.b.\sigma_{r,1}(X)$ and $\sigma_{r,3}(Z) = a.\sigma_{r,2}(X) = a.\sigma_{r,1}(X)$, and $\sigma_{r,1}(X) = a$. Now for run r we have $(q_0, X) \rightsquigarrow_2^{aaaa} (q_0, W)$.

In order to define the transition monoid of an SST T, we first extend $\mathbb{N}$ with an extra element $\perp$, and let $\mathbb{N}_\perp = \mathbb{N} \cup \{\perp\}$. This new element behaves as 0, i.e. for all $i \in \mathbb{N}_\perp$, $i.\perp = \perp.i = \perp$, $i + \perp = \perp + i = i$. Moreover, we assume that $\perp < n$ for all $n \in \mathbb{N}$. We assume that pairs $(p, X) \in Q \times \mathcal{X}$ are totally ordered.

▶ **Definition 6** (Transition Monoid of SSTs)**.** The *transition monoid* of a streaming string transducer T is the set of square matrices over $\mathbb{N}_\perp$ indexed (in order) by elements of $Q \times \mathcal{X}$, defined by $M_T = \{M_s \mid s \in \Sigma^*\}$ where for all strings $s \in \Sigma^*$, $M_s[p, Y][q, X] = n \in \mathbb{N}$ if and only if $(p, Y) \rightsquigarrow_n^s (q, X)$, and $M_s[p, Y][q, X] = \perp$ if and only if there is no run from p to q on s. By definition, there is atmost one run r from (p, Y) to (q, X) on any string s.

It is easy to see that $(M_T, \times, \mathbf{1})$ is a monoid, where $\times$ is defined as matrix multiplication and the identity element is the unit matrix $\mathbf{1}$. The mapping $M_\bullet$, which maps any string s to its transition matrix M_s, is a morphism from $(\Sigma^*, ., \epsilon)$ to $(M_T, \times, \mathbf{1})$. We say that the transition monoid M_T of an SST T is n-bounded if all the coefficients of the matrices of M_T are bounded by n. Clearly, any n-bounded transition monoid is finite.

In the original definition [2] of SST, updates were *copyless*, i.e., the content of a variable can never flow into two different variables, and cannot flow more than once into another variable. In [3], this condition was slightly relaxed to the notion of *restricted copy*, where a variable cannot flow more than once into another variable. This allows for a limited form of copy: for instance, X can flow to Y and Z, but Y and Z cannot flow to the same variable. Finally, *bounded copy* SSTs were introduced in [6] as a restriction on the variable dependency graphs. This restriction requires that there exists a bound K such that any variable flows at most K times in another variable. These three restrictions were shown to be equivalent, in the sense that SSTs with copyless, restricted copy, and bounded copy

updates have the same expressive power. Due to our definition of transition monoid, and the results of [6], Theorem 7 is immediate by observing that bounded copy restriction of [6] for SSTs corresponds to finiteness of transition monoid. Also, notice that since the bounded copy assumption generalizes the copyless [2] and restricted copy [3] assumptions, previous definitions in the literature of SSTs correspond to finite transition monoids.

▶ **Theorem 7** ([6]). *[MSO-definable string transformations] A string transformation is MSO-definable iff it is definable by an* SST *with finite transition monoid.*

The main goal of this paper is to present a similar result for FO-definable transformations. For this reason we define aperiodic and 1-bounded SSTs.

▶ **Definition 8** (Aperiodic and 1-bounded SSTs). An SST is aperiodic if its transition monoid is aperiodic. An SST is 1-bounded if its transition monoid is 1-bounded, i.e. for all strings s, and all pairs (p, Y), (q, X), $M_s[p, Y][q, X] \in \{\bot, 0, 1\}$. See [14] for an example.

It can be shown (see [14]) that the domain of an aperiodic SST is FO-definable. We show that an SST is non-aperiodic iff its transition monoid contains a non-trivial cycle. Checking the existence of a non-trivial cycle is in PSPACE for deterministic automata [16].

▶ **Lemma 9.** *Checking aperiodicity and 1-boundedness for SSTs is* PSPACE-COMPLETE.

Now we are in a position to present the main result of this paper. We prove the following key theorem using Lemma 15 (Section 5) and Lemma 11 (Section 4).

▶ **Theorem 10** (FO-definable string transformations). *A string transformation is FO-definable iff it is definable by an aperiodic, 1-bounded* SST.

4 From aperiodic 1-bounded SST to FOT

▶ **Lemma 11.** *A string transformation is FO-definable* ***if it is*** *definable by an aperiodic,1-bounded SST.*

The idea closely follows the SST-to-MSOT construction of [1, 6]. The main challenge here is to show that aperiodicity and 1-boundedness on the SST implies FO-definability of the output string structure (in particular the predicate $\preceq$). We first show that the variable flow of any aperiodic,1-bounded SST is FO-definable. This will be crucial to show that the output predicate $\preceq$ is FO-definable.

Let $X \in \mathcal{X}$, $s \in \mathrm{dom}(T)$, $i \in \mathrm{dom}(s)$, and let $n = |s|$. We say that the pair (X, i) is *useful* if the content of variable X before reading $s[i]$ will be part of the output after reading the whole string s. Formally, if $r = q_0 \dots q_n$ is the accepting run of T on s, then (X, i) is useful for s if $(q_{i-1}, X) \rightsquigarrow_1^{s[i:n]} (q_n, Y)$ for some variable $Y \in F(q_n)$. Thanks to the FO-definability of variable flow this property is FO-definable.

Next, we define the SST-output structure given an input string structure. It is an intermediate representation of the output, and the transformation of any input string into its SST-output structure will be shown to be FO-definable. For any SST T and string $s \in \mathrm{dom}(T)$, the SST-output structure of s is a relational structure $G_T(s)$ obtained by taking, for each variable $X \in \mathcal{X}$, two copies of $\mathrm{dom}(s)$, respectively denoted by X^{in} and X^{out}. For notational convenience we assume that these structures are labeled on the edges. This structure satisfies the following invariants: for all $i \in dom(s)$, (1) the nodes (X^{in}, i) and (X^{out}, i) exist only if (X, i) is useful, and (2) there is a directed path from (X^{in}, i) to (X^{out}, i) whose sequence of labels is equal to the value of the variable X computed by T after reading $s[i]$. The condition on usefulness of nodes implies that SST-output structures consist of a single directed component, and therefore they are edge-labeled string structures.

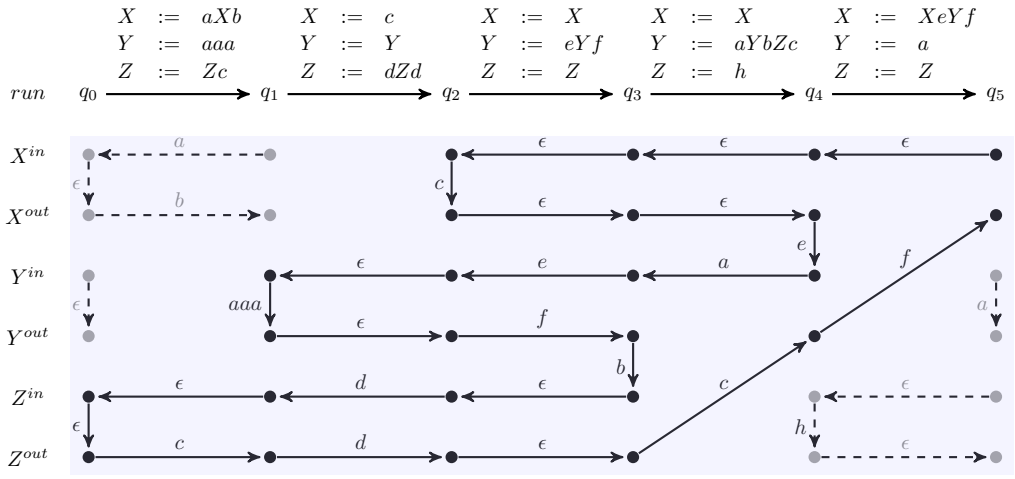


Figure 3 SST-output structure.

▶ **Example 12.** An example of SST-output structure in shown in Figure 3. Here we show only the variable updates. Dashed arrows represent variable updates for useless variables, and do not belong to SST-output structure; solid edges belong to the SST-output structure. Initially the variable content of Z is ϵ, this is represented by the ϵ-edge from $(Z^{in}, 0)$ to $(Z^{out}, 0)$ in the first column. Then, variable Z is updated to Zc. Hence, the new content of Z starts with ϵ (represented by the ϵ-edge from $(Z^{in}, 1)$ to $(Z^{in}, 0)$, which is concatenated with the previous content of Z, and then concatenated with c (it is represented by the c-edge from $(Z^{out}, 0)$ to $(Z^{out}, 1)$). The output is given by the path from $(X^{in}, 5)$ to $(X^{out}, 5)$ and equals $ceaeaaafbdcdcf$. Also note that some edges are labelled by strings with several letters, but there are finitely many possible such strings. In particular, we denote by O_T the set of all strings that appear in right-hand side of variable updates.

What remains for us, is to adapt from [1, 6], the MSO-definability of the transformation that maps a string s to its SST-output structure : we show that it is FO-definable as long as the SST is aperiodic. The main challenge is to define the transitive closure of the edge relation in first-order. Let T be $(Q, q_0, \Sigma, \Gamma, \mathcal{X}, \delta, \rho, Q_f)$. The SST-output structures of T, as node-labeled strings, can be seen as logical structures over the signature $S_{O_T} = \{(E_\gamma)_{\gamma \in O_T}, \preceq\}$ where the symbols E_γ are binary predicates interpreted as edges labeled by O_T. We let E denote the edge relation, disregarding the labels. To prove that transitive closure is FO[Σ]-definable, we use the fact that variable flow is $FO[\Sigma]$-definable. The following property, along with the FO-definability of variable flow, shows that transitive closure is FO-definable.

▶ Proposition 1. Let T be an **aperiodic** SST T. Let $s \in dom(T)$, $G_T(s)$ its SST-output structure and $r = q_0 \dots q_n$ the accepting run of T on s. For all variables $X, Y \in \mathcal{X}$, all positions $i, j \in \text{dom}(s) \cup \{0\}$, all $d, d' \in \{in, out\}$, there exists a path from node (X^d, i) to node $(Y^{d'}, j)$ in $G_T(s)$ iff (X, i) and (Y, j) are both useful and one of the following holds:

1. $Y \rightsquigarrow^{r[j:i]} X$ and $d = in$,
2. $X \rightsquigarrow^{r[i:j]} Y$ and $d' = out$, or
3. there exists $k \geq max(i, j)$ and two variables X', Y' such $X \rightsquigarrow^{r[i:k]} X'$, $Y \rightsquigarrow^{r[j:k]} Y'$ and X' and Y' are concatenated in this order[1] by r when reading $s[k+1]$.

[1] By "concatenated" we mean that there exists a variable update whose rhs is of the form $\dots X' \dots Y' \dots$

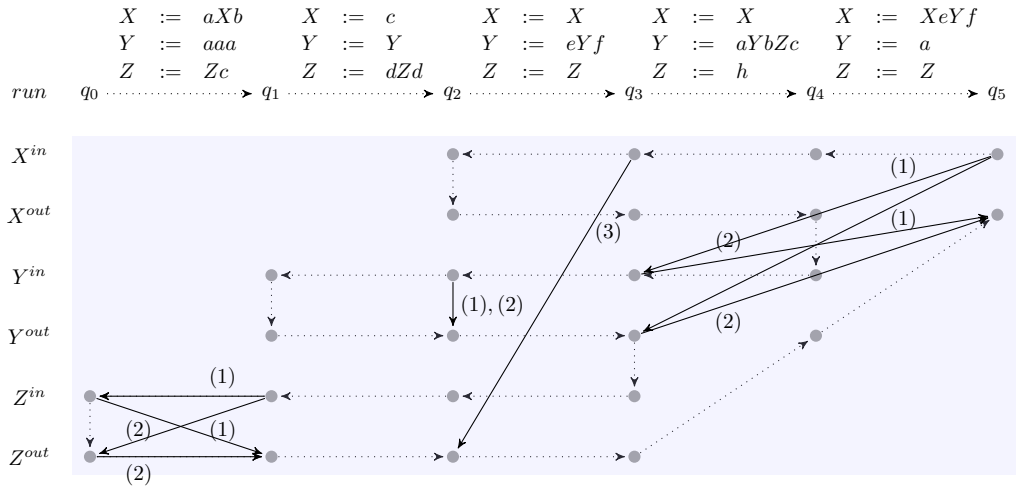


Figure 4 Conditions of Proposition 1.

▶ **Example 13.** We illustrate proposition 1 using example of Fig.4. We have for instance $(q_2, Y) \rightsquigarrow_1^{s[3:2]=\epsilon} (q_2, Y)$, therefore by conditions (1) (and (2)) by taking $X = Y$ and $i = j = 2$, there exists a path from $(Y^{in}, 2)$ to $(Y^{out}, 2)$. Note that none of these conditions imply the existence of an edge from $(Y^{out}, 2)$ to $(Y^{in}, 2)$, but self-loops on $(Y^{in}, 2)$ and $(Y^{out}, 2)$ are implied by conditions (1) and (2) respectively. Now consider positions 0 and 1 and variable Z. It is the case that $(q_0, Z) \rightsquigarrow_1^{s[1:1]} (q_1, Z)$, therefore by condition (1) there is a path from $(Z^{in}, 1)$ to $(Z^{in}, 0)$ and to $(Z^{out}, 0)$. Similarly, by condition (2) there is a path from $(Z^{in}, 0)$ to $(Z^{out}, 1)$ and from $(Z^{out}, 0)$ to $(Z^{out}, 1)$. For positions 3 and 5, note that $(q_3, Y) \rightsquigarrow_1^{s[4:5]} (q_5, X)$, hence there is a path from $(Y^d, 3)$ to $(X^{out}, 5)$ for all $d \in \{in, out\}$. By condition (2) one also gets edges from $(X^{in}, 5)$ to $(Y^d, 3)$. Finally consider nodes $(Z^{out}, 2)$ and $(X^{in}, 3)$. There is no flow relation between variable Z at position 2 and variable X at position 3. However, $(q_3, X) \rightsquigarrow_1^{s[4:4]} (q_4, X)$ and $(q_2, Z) \rightsquigarrow^{s[3:4]} (q_4, Y)$. Then X and Y gets concatenated at position 4 to define X at position 5. Hence, there is a path from $(X^{in}, 3)$ to $(Z^{out}, 2)$ (condition (3)).

▶ **Lemma 14.** *For an* ***aperiodic*** SST T*, variables* $X, Y \in \mathcal{X}$ *and all* $d, d' \in \{in, out\}$*, there exists an FO[*Σ*]-formula* $path_{X,Y,d,d'}(x, y)$ *with two free variables s.t. for all* $s \in \mathrm{dom}(T)$ *and* $i, j \in dom(s)$*,* $s \models path_{X,Y,d,d'}(i, j)$ *iff there exists a path from* (X^d, i) *to* (Y^d, j) *in* $G_T(s)$*.*

We now sketch the proof of Lemma 11. Let Γ be the output alphabet. We adapt the MSO-definability of strings to SST-output structures from [6, 1] and use the FO-definability of transitive closure (Lemma 14) to show that strings to SST-output structure transformations are FO-definable whenever the SST is aperiodic. Since the usefulness of nodes is FO-definable, we filter out useless nodes in the first FO-transformation, unlike [6, 1], where useless nodes in the SST-output structures are later removed by composing with another MSO-definable transformation. We can transform the SST-output structures which are edge-labeled strings over $O_T \subseteq \Gamma^*$ to a node-labeled string over Γ. This transformation is again FO-definable by taking a suitable number of copies of the input domain ($max\{|s| \mid s \in O_T\}$). Now Lemma 11 follows from the closure of FO-transformations under composition [10].

5 From FOT to aperiodic 1-bounded SST

The goal of this section is to prove the following lemma by showing a reduction from FO-definable transformations to aperiodic, 1-bounded SSTs. Due to space limitations, we only sketch the main ideas of the proof of this result.

▶ **Lemma 15.** *A string transformation is FO-definable* ***only if*** *it is definable by an aperiodic, 1-bounded SST.*

FO-types, heads and tails. The FO-K-type (K-type for short) of a string s is the set of FO sentences of quantifier rank at most K that are true in s. The set of K-types is finite (up to logical equivalence) [17]. We start with a key observation. Given an FO-transducer, an input string s and a position i in s, all the maximal paths of the output structure induced by nodes of the form j^c, for all copies c and input positions $j \leq i$ define substrings of the output of s. The starting (resp. ending) nodes of these substrings are respectively called i-head and i-tails. Consider the FO-transduction shown in Figure 2 till position 3. Suppose that we omit the positions and edges of the output graph post position 3. Upto position 3, the output graph consists of two strings: the first string is between the 3-head 1^1 and the 3-tail 3^1 and stores aa, while the second string is between the 3-head 3^2 and the 3-tail 2^3 and stores the string $abab$. The key observation of [5] is that any i-head j^c (resp. i-tail) is uniquely identified by the K-type τ_1 of the string $s[1{:}j)$, the label a of input position j, the copy c, and the K-type τ_2 of the string $s(j{:}i]$, for a bound K that depends only on the FO-transducer.

SST construction. The main lines of the SST construction of [5] is to use as many SST variables X_α as tuples $\alpha = (\tau_1, a, c, \tau_2)$. Since the sets of K-types, labels and copies are finite, then so is the set of variables. At each position i in the input s, the content of X_α computed by the SST is exactly the substring in between the i-head and i-tail identified by the tuple α. To define the variable update when incrementing position i, if the SST knows the K-types of the current prefix and suffix respectively, it can determine how the $(i+1)$-heads and $(i+1)$-tails are connected to the i-heads and i-tails, based on the FO-formulas that define the output edge relation. Let us now explain how the SST can compute the K-types of the prefix up to i and of the suffix from position i. It is known that the K-type of a string $s_1 s_2$ only depends on the K-types of s_1 and s_2 respectively [17]. Therefore, to compute the K-type of the prefix up to $i+1$, the SST only needs to know the K-type of the prefix up to i and the input label at position $i+1$. Therefore, the states of the SST are K-types. To compute the K-type of the suffix, we equip our SST with look-aheads, defined by aperiodic finite state automata. We naturally extend the notion of aperiodic SST to aperiodic SST with look-aheads, and, as an intermediate result, show that removing look-aheads can be done while preserving aperiodicity (as well as 1-boundedness). From an FO-transducer, the construction therefore produces an SST T_{la} with look-ahead. The main difficulty is to show that T_{la} is aperiodic and 1-bounded.

Aperiodicity and 1-boundedness of T_{la}. One of the technical difficulties in showing that T_{la} is aperiodic is to show that it computes the type informations and update the variables in an aperiodic manner. A well-known property [17] we exploit is that for $m \geq 2^K$, for any string s, the strings s^m and s^{m+1} are indistinguishable by FO-sentences of rank at most K.

For the sake of understanding of this sketch and, in order to focus only on aperiodicity of variable updates, we rather assume, in this sketch, that the positions i of the input strings

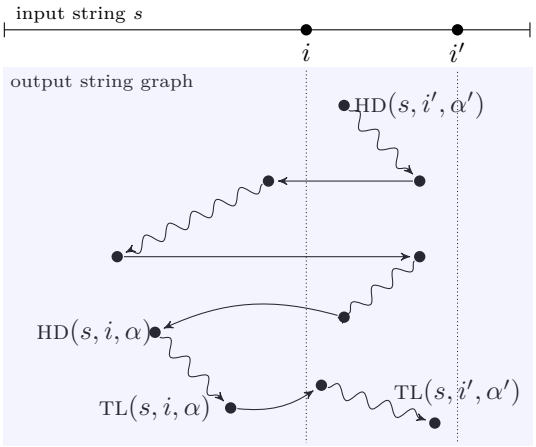


Figure 5 Variable flow is FO-definable.

s have been initially extended with type informations (τ_1, τ_2) where τ_1 is the K-type of $s[1{:}i]$ and τ_2 is the K-type of $s(i{:}|s|]$. Therefore, we can transform T_{la} into a one-state SST T (without look-ahead), assuming it gets as input only strings extended with valid type information. The 1-boundedness of T is a simple consequence of the construction (and was already shown in [5] through the notion of restricted copy). Let us now briefly explain why T is aperiodic, or equivalently, that its variable flow is aperiodic. It is sufficient to show that the variable flow is FO-definable. Given $s \in \Sigma^*$, a position i in s, and a tuple $\alpha = (\tau_1, a, c, \tau_2)$ as defined before, we denote by $\text{HD}(s, i, \alpha)$ the i-head (resp. $\text{TL}(s, i, \alpha)$ the i-tail) defined by α in s. Given another tuple α' and a position $i' > i$ in s, we relate the flow between variable X_α at position i to variable $X_{\alpha'}$ at position i' to the existence of a path from $\text{HD}(s, i', \alpha')$ to $\text{HD}(s, i, \alpha)$ that do not go beyond position i' in the output graph of s.

▶ **Example 16.** Consider the FO-transformation of Fig. 5. As a consequence of the invariant of our construction, the substring s_1 that starts in position $\text{HD}(s, i, \alpha)$ and ends in $\text{TL}(s, i, \alpha)$, at position i, is stored in variable X_α. The substring s_2 from $\text{HD}(s, i', \alpha')$ to $\text{TL}(s, i', \alpha')$ is stored, at position i', in variable $X_{\alpha'}$. Since s_1 is a substring of s_2, the content of variable X_α at position i (i.e. s_1) flows into the content of variable $X_{\alpha'}$ at position i' (i.e. s_2). Based on the fact that the output transition closure of the edge relation is defined in FO, and the fact that types are also FO-definable, we show that the existence of a path from $\text{HD}(s, i', \alpha')$ to $\text{HD}(s, i, \alpha)$ that do not cross position i' is FO-definable, and so is the variable flow.

As mentioned earlier, the complete proof starts directly with an SST with look-ahead that computes the type information, therefore one has to study both state flow and variable flow. An alternative proof could have been to compose two aperiodic SSTs (w/o lookaheads): the first one annotates the string with type information, and the second one is the one-state SST T. Then, it would remain to prove that aperiodic SSTs are closed under composition (which is a consequence of our result and the fact that FO-transducers are closed under composition). However, it is not clear that directly proving that aperiodic SSTs are closed under composition would have been simpler than our proof based on SSTs with look-aheads.

References

1 R. Alur and P. Černý. Expressiveness of streaming string transducers. In *Proc. FSTTCS 2010*, pages 1–12, 2010.

2 R. Alur and P. Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *Proc. POPL 2011*, pages 599–610, 2011.

3 R. Alur and L. D'Antoni. Streaming tree transducers. In *Proc. ICALP 2012*, pages 42–53, 2012.

4 R. Alur, L. D'Antoni, J. V. Deshmukh, M. Raghothaman, and Y. Yuan. Regular functions and cost register automata. In *Proc. LICS 2013*, pages 13–22, 2013.

5 R. Alur, A. Durand-Gasselin, and A. Trivedi. From monadic second-order definable string transformations to transducers. In *Proc. LICS 2013*, pages 458–467, 2013.

6 R. Alur, E. Filiot, and A. Trivedi. Regular transformations of infinite strings. In *Proc. LICS 2012*, pages 65–74, 2012.

7 M. Bojanczyk. Transducers with origin information. In *Proc. ICALP 2014*, pages 26–37, 2014.

8 J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6(1–6):66–92, 1960.

9 O. Carton and L. Dartois. Aperiodic two-way transducers. In *Highlights of Logic, Automata and Games*, 2013. Slides available at `http://highlights-conference.org/pub/3-1-Dartois.pdf`.

10 B. Courcelle. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126(1):53–75, 1994.

11 V. Diekert and P. Gastin. First-order definable languages. In *Logic and Automata: History and Perspectives*, pages 261–306. Amsterdam University Press, 2008.

12 J. Engelfriet and H. J. Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Logic*, 2:216–254, 2001.

13 J. Engelfriet and S. Maneth. Macro tree translations of linear size increase are MSO definable. *SIAM Journal on Computing*, 32:950–1006, 2003.

14 E. Filiot, S. N. Krishna, and A. Trivedi. First-order definable string transformations.

15 P. McKenzie, T. Schwentick, D. Therien, and H. Vollmer. The many faces of a translation. *JCSS*, 72, 2006.

16 J. Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66:163–176, 1985.

17 H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994.