2012 27th Annual ACM/IEEE Symposium on Logic in Computer Science

Regular Transformations of Infinite Strings

Rajeev Alur
Dept. of Computer and Info. Science
University of Pennsylvania
Philadelphia, USA

Emmanuel Filiot
Chargé de recherches du F.R.S.-FNRS.
CS Dept., Université Libre de Bruxelles
CP212, Bvd du Triomphe, 1050 Brussels, Belgium

Ashutosh Trivedi
Dept. of Computer and Info. Science
University of Pennsylvania
Philadelphia, USA

***Abstract*—The theory of regular transformations of finite strings is quite mature with appealing properties. This class can be equivalently defined using both logic (Monadic second-order logic) and finite-state machines (two-way transducers, and more recently, streaming string transducers); is closed under operations such as sequential composition and regular choice; and problems such as functional equivalence and type checking, are decidable for this class. In this paper, we initiate a study of transformations of infinite strings. The MSO-based definition for regular string transformations generalizes naturally to infinite strings. We define an equivalent generalization of the machine model of streaming string transducers to infinite strings. A streaming string transducer is a deterministic machine that makes a single pass over the input string, and computes the output fragments using a finite set of string variables that are updated in a copyless manner at each step. We show how Muller acceptance condition for automata over infinite strings can be generalized to associate an infinite output string with an infinite execution. The proof that our model captures all MSO-definable transformations uses two-way transducers. Unlike the case of finite strings, MSO-equivalent definition of two-way transducers over infinite strings needs to make decisions based on omega-regular look-ahead. Simulating this look-ahead using multiple variables with copyless updates, is the main technical challenge in our constructions. Finally, we show that type checking and functional equivalence are decidable for MSO-definable transformations of infinite strings.**

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

I. Introduction

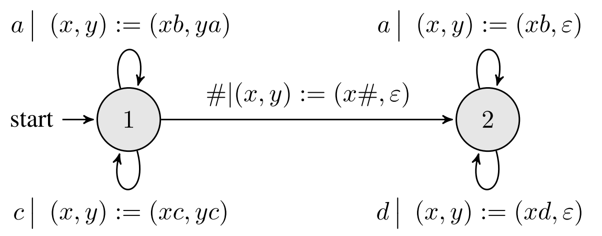
The fundamental theorem of Büchi-Elgot-Trakhtenbrot [1], [2], [3] first established the connection between mathematical logic and automata theory by showing that the deterministic finite state automata accept the same class of languages as monadic second order logic (MSO) interpreted over finite strings. This connection, while providing new tools to solve problems in logic, revolutionized the field of automata theory as Büchi [4] initiated the study of finite automata over infinite strings in search for an analogous connection for MSO interpreted over infinite strings. As a result a number of equally expressive finite state automata over infinite strings, e.g. non-deterministic Büchi automata and deterministic Muller automata [5], were identified characterizing the class of *regular* languages of infinite strings. Since then, MSO equivalence has become a synonym for regularity as the connection between automata and MSO has been extended for languages over different structures such as trees and partial orders.

Courcelle [6] introduced *MSO transducers* by proposing a novel way to use MSO interpreted over graphs to specify graph transformations. MSO transducers work by introducing a fixed number of copies of the vertices of the input graph; and the domain, the labels and the edges of the output graph are defined by MSO formulas with zero, one or two free variables, respectively, interpreted over the input graph. MSO transducers, when restricted appropriately to string-graphs, can be used to specify transformations (functions) of both finite and infinite strings. Engelfriet and Hoogeboom [7] studied such transformations of finite strings, and showed the existence of a finite state model—two-way finite-state transducers—implementing the same class of transformations. In this paper we explore finite state transducer models equivalent to MSO definable transformations of infinite strings.

A. Theory of Regular Transformations of Finite Strings

Classical (deterministic) finite-state transducers, e.g. Mealy machines or generalized sequential machines, extend deterministic finite-state automata as they process the input string in a single pass from left-to-right, and at each step they read the input letter and append zero or more symbols to the output tape. An input string, like in finite-state automata, is accepted if the final state is an accepting state, and the corresponding output string is equal to the contents of the output tape.

One-way transducers, however simple in definition, are less expressive than MSO transducers on finite strings as they can not express certain MSO definable transformations, for instance, *reversing* a string (i.e., $a_1a_2\ldots a_n \mapsto a_na_{n-1}\ldots a_1$), *swapping* two substrings (e.g., $\alpha\#\beta \mapsto \beta\#\alpha$), or involving a *regular choice* (e.g., transformation csub that replaces each occurrence of a with b if the string contains a $\#$, and outputs the input string otherwise). Transformations involving regular choice can be implemented by relaxing the model to non-deterministic finite state transducers, for which the theory is quite rich [8], [9]. However, swapping or reversing is still not expressible by non-deterministic finite state transducers. Engelfriet and Hoogeboom [7] showed that the finite-state transducers when equipped with a two-way input tape have the same expressive power as MSO transducers. A crucial property of two-way finite-state transducers exploited in the proof [7] is the fact that transitions capable of *regular look-ahead* (i.e., transitions that test the whole input string against a regular property) do not increase their expressiveness. Two-way transducers can implement transformations involving reverse, swap, and regular choice. For example, reversing a string can be



$a \mid (x,y) := (xb, ya)$
$a \mid (x,y) := (xb, \varepsilon)$
$\# \mid (x,y) := (x\#, \varepsilon)$
start
1
2
$c \mid (x,y) := (xc, yc)$
$d \mid (x,y) := (xd, \varepsilon)$

Fig. 1. SST implementing csub; Here c and d stand for all letters except $\{a, \#\}$ and a, respectively. Also $F(1) = y$ and $F(2) = x$.

implemented by first moving the input head to the end of the tape, and then outputting the letters sequentially in a right-to-left pass of the input tape. Also, transformation csub can be implemented by first scanning the input string for $\#$, and then moving the head to the first cell again to output accordingly.

Alur and Černý [10], [11] recently proposed a one-way finite-state transducer model, called the *streaming string transducers* (SSTs), that manipulates a finite set of string variables to compute its output, and showed that they have same expressive power as MSO transducers. SSTs, instead of appending symbols to the output tape, concurrently update all string variables using a concatenation of string variables and output symbols 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, 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. SSTs can implement reversing, swap, and regular choice with ease. For example, reversing a string can be implemented by an SST with one state q and one variable x, such that the transition for every letter a loops back to the state with update $x := ax$; and the output function maps q to x. Transformation csub can be defined by the SST in Fig. 1. The correctness of this SST follows from the observation that after reading a string, variable y contains the input string, while variable x contains the input string with occurrences of a replaced with b. Moreover, a run ends in state 2 iff the string contains a $\#$.

The class of regular transformations of finite strings [7] characterized by MSO transducers, two-way finite-state transducers, and SSTs is closed under operations such as sequential composition and regular choice. Moreover, both the equivalence problem (deciding the equivalence of two regular transformations) and the type checking problem (deciding whether the output strings stays in a regular language B for the input strings from a regular language A) are decidable (in PSPACE for SSTs [11]).

B. Defining Regular Transformations of Infinite Strings

To capture the transformations of infinite strings, natural extensions of finite-state transducers with ω-string acceptance conditions like Büchi and Muller have been studied in past [12], [13], [14]; however they do not capture the essence of regularity as they, like their finite string counterparts, are unable to implement typical MSO-definable transformations like reversing substrings (e.g., replace the first finite $\#$-free prefix, when it exists, by its reverse), swapping substrings, or ω-regular choice (e.g., transformation csub that replaces every occurrence of a by b if the ω-string contains a $\#$).

Following the result of Engelfriet and Hoogeboom [7], in this paper we consider two-way (deterministic) finite-state transducers with Muller acceptance condition (2WST), but we find that they are not as expressive as MSO transformations of infinite strings due to the inability of 2WST to perform ω-regular look-ahead. For example, consider the transformation csub: It can easily be implemented using 2WST with ω-regular look-ahead that is used to check whether the input tape contains a $\#$. However, to implement the same transformation using 2WST sans ω-regular look-ahead capability, it needs to first scan the string for $\#$, a procedure that will not terminate for a $\#$-free input string. On a positive note, we show that the proof of [7] can be easily generalized to show the equivalence of 2WST with ω-regular look-around (a regular look-behind combined with an ω-regular look-ahead) and MSO definable transformations of infinite strings (MSOT).

Even though we have identified our first finite-state model implementing regular transformations of infinite strings, it is desirable to have a, preferably one-way, finite-state model equivalent to MSOT that is closed under ω-regular look-around; because ω-regular look-around is not a natural modeling feature, and moreover, one-way models usually allow for a simpler proof of decidability of the functional equivalence.

We generalize streaming string transducers to operate on infinite strings by extending them with Muller acceptance condition. To define the output corresponding to infinite runs we associate the set of infinitely occurring states, à la Muller acceptance condition, with concatenations of string variables using an output function F. For instance, if the set of infinitely occurring states is P and $F(P) = xy$ then the output string is defined as the limit of valuations of x concatenated with valuations of y at each step of the run. The transformation csub can be implemented by the SST shown in Fig. 1 interpreted over infinite strings where the output function is defined as $F(\{1\}) = y$ and $F(\{2\}) = x$. One needs to be cautious while defining the output function to make sure that the limit of the concatenations of string variable valuations always exists as an infinite string. As an example of an ill-formed output function consider $F(\{1\}) = xy$ in Fig. 1 which for the input a^ω outputs bi-infinite string $b^\omega a^\omega$. In our definition of SSTs on infinite strings, we enforce syntactic restrictions to ensure that the output always exists as an infinite string, while capturing the full expressive power of MSOT.

C. Results / Properties

Fig. 2 shows the roadmap of our approach to prove SST=MSOT, i.e. the SSTs and MSOT have the same expressive power. After setting some preliminary results and notations in Section II, we formally introduce our models in Section III. We begin by first observing that 2WST with ω-regular look-around (2WST$_{\mathrm{la}}$) and MSOT have the same expressiveness, and the proof of [7] can be naturally extended

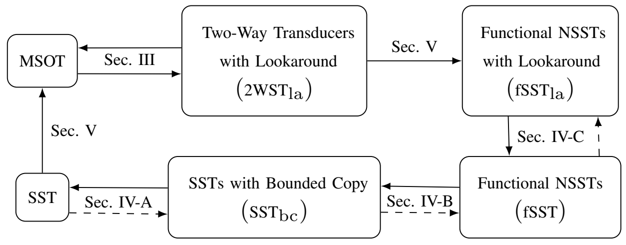

MSOT
Sec. III
Two-Way Transducers with Lookaround ($2WST_{la}$)
Sec. V
Functional NSSTs with Lookaround ($fSST_{la}$)
Sec. V
Sec. IV-C
SST
Sec. IV-A
SSTs with Bounded Copy (SST_{bc})
Sec. IV-B
Functional NSSTs ($fSST$)

Fig. 2. Equivalent models of regular transformations

to show this fact. We exploit this result to show SST=MSOT in several steps. First, we show that every transformation implemented by a $2WST_{la}$ can be expressed by a (functional) non-deterministic version of SSTs with look-around ($NSST_{la}$) by using a construction similar to classical construction that relates two-way to one-way finite automata. Then we show how the look-around can be captured using non-determinism and hence $NSST_{la}$ = NSST. We also observe that the functional non-determinism in SSTs can be removed by introducing copies in variable update in a bounded manner, reminiscent of *finite copying* restriction in [15]. The most challenging part of the proof is to show that SSTs with bounded copy can be simulated by (copyless) SSTs. In Section IV we formally state results on equivalence of SSTs with bounded-copy SSTs, functional nondeterministic SSTs, and SSTs with ω-regular look-around. We also notice that the proof [10] of reduction from SSTs to MSO for finite strings can be extended to infinite strings in a straightforward manner. In Section V, we collect these results to prove the main result of this paper:

Theorem 1. *A transformation of infinite strings is* MSOT-*definable if and only if it is* SST-*definable.*

We discuss, in Section VI, the equivalence and the type checking problems for SSTs, and show that they can be decided in PSPACE using natural extensions of corresponding procedures for SSTs on finite strings [11]. This result when combined with Theorem 1 yields the following key result.

Theorem 2. *Equivalence and type-checking problems for* MSOT *are decidable.*

For the lack of space proofs are either sketched or omitted; full proofs can be found in the technical report [16].

II. Preliminaries

An alphabet Σ is a finite set of letters. A finite string (resp. ω-string) over Σ is defined as a finite sequence (resp. ω-sequence) of letters from Σ. We denote by ε the empty string. We write Σ^* and Σ^ω for the set of finite and ω-strings over Σ. We write Σ^∞ for $\Sigma^* \cup \Sigma^\omega$.

For a string $s \in \Sigma^\infty$ we write $|s|$ for its length; note that for an ω-string s we have that $|s| = \infty$. For a string $s \in \Sigma^\infty$ and for all $0 \leq i < |s|$ we write $s[i]$ for the i-th letter of the string s. For two ω-strings $s, s' \in \Sigma^\omega$ we define their distance $d(s, s')$ as 2^{-j} where $j = \min\{k : s[k] \neq s'[k]\}$. We say that the string $s \in \Sigma^\omega$ is the limit of the sequence of ω-strings $s_1, s_2, \ldots$ s.t. $s_i \in \Sigma^\omega$ if for every $\varepsilon > 0$ there exists an index $N_\varepsilon \in \mathbb{N}$ such that for all $i \geq N_\varepsilon$ we have that $d(s, s_i) \leq \varepsilon$. Such limit, when exists, is unique, and we write $s = \lim_{i\to\infty} s_i$. For instance, $a^\omega = \lim_{i\to\infty} a^i b^\omega$, while the limit of the sequence $a^\omega, ba^\omega, a^\omega, ba^\omega, \ldots$ is undefined.

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 $F : A \rightharpoonup B$. For a partial function $F : A \rightharpoonup B$ we write $\mathrm{dom}(F)$ for its domain. Given a tuple of sets $\langle A_i \rangle_{i=1}^n$ we define the i-th projection operator $\pi_i : A_1 \times A_2 \times \cdots \times A_i \times \cdots \times A_n \to A_i$ for each $1 \leq i \leq n$ as $\pi_i(a_1, a_2, \ldots, a_i, \ldots, a_n) = a_i$.

A. Monadic Second-Order Logic on Graphs

A Σ-labeled graph is a tuple $G = (N, E, L)$ where N is a countable set of nodes, $E \subseteq N \times N$ is a set of edges, and $L : N \to \Sigma$ is a node-labeling function. Let $GR(\Sigma)$ be the set of all Σ-labeled graphs.

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

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

where formula $L_a(x)$ states that node x has the label $a \in \Sigma$, while the formula $E(x, y)$ states that there is an edge from x to y. Atomic formulas are connected 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. 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 graph G and for valuations $n_1, n_2, \ldots, n_k$ (where $n_i \in N$ if x_i is first-order, and $n_i \subseteq N$ if x_i is second-order) we say that graph G with valuation $\nu = (n_1, n_2, \ldots, n_k)$ satisfies the formula $\phi(x_1, x_2, \ldots, x_k)$ and we write $(G, \nu) \models \phi(x_1, x_2, \ldots, x_k)$ or $G \models \phi(n_1, n_2, \ldots, n_k)$ if formula ϕ with n_i as the interpretations of x_i satisfies in graph G.

B. ω-Regular Languages

A language (resp. ω-language) L over an alphabet Σ is defined as a set of finite strings (resp. ω-strings). A string $s \in \Sigma^\infty$ can be represented by its node-graph $\mathcal{G}(s) = (N_s, E_s, L_s)$ where $N_s = \{i : \mathbb{N} \ni i < |s|\}$, $E_s = \{(i, i+1) : \mathbb{N} \ni i < |s|\}$ and $L_s(i) = s[i]$ for all $i \leq |s|$. We also say that a graph $G \in GR(\Sigma)$ is a *string graph* if there is a string $s \in \Sigma^\infty$ such that G is isomorphic to $\mathcal{G}(s)$. MSO formulas over finite and ω-string graphs characterize languages of finite strings and ω-strings, respectively. We call such languages MSO-definable.

A language of finite strings is called *regular* if there is a deterministic finite state automata that accepts it, or equivalently [1] if it is MSO-definable. Muller automata, formally defined below, are finite state models accepting ω-languages.

Definition 1 (Muller Automata)**.** *A Muller automaton is a tuple $A = (Q, q_0, \Sigma, \delta, F)$ where Q is a finite set of states,*

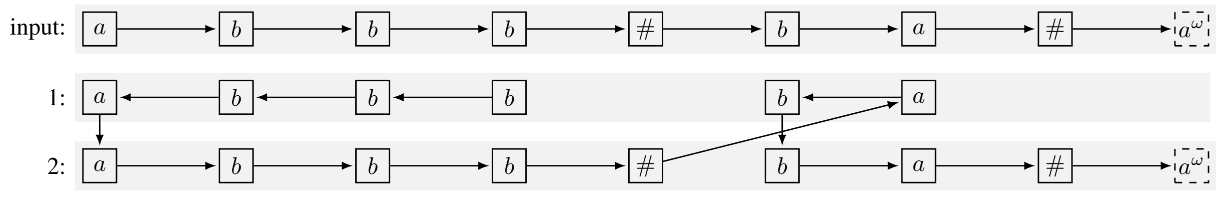


Fig. 3. The result of MSO transformation f_1 on the string $abbb\#ba\#a^\omega$.

$q_0 \in Q$ is the initial state, Σ is an input alphabet, $\delta : Q\times\Sigma \to Q$ is a transition function, and $F \subseteq 2^Q$ are the accepting sets.

For states $q, q' \in Q$ and letter $a \in \Sigma$ we say that (q, a, q') is a transition of a Muller automaton A if $\delta(q, a) = q'$ and we write $q \xrightarrow{a} q'$. A finite run of A over a finite string $a_1a_2 \ldots a_n \in \Sigma^*$ is a finite sequence of transitions $\langle(q_0, a_1, q_1), (q_1, a_2, q_2), \ldots, (q_{n-1}, a_n, q_n)\rangle \in (Q \times A \times Q)^*$ starting from the initial state q_0 and we represent such runs as $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_n$. Infinite runs are similarly defined as infinite sequences of transitions. For a run r of A we define its size $|r| \in \mathbb{N}$ to be equal to its number of transitions. For an infinite run r, we denote by $\Omega(r)$ the set of states that occur infinitely often in r. An ω-string w is accepted by a Muller automaton A if the run r of A on w is such that $\Omega(r) \in F$.

The classical results of Büchi [4] and McNaughton [5] extend the notion of regular languages from finite strings to ω-strings by showing that an ω-language is accepted by a Muller automaton if and only if it is MSO-definable.

III. Regular Transformations

An ω-transformation from an input alphabet Σ to an output alphabet Γ is a partial function in $[\Sigma^\omega \rightharpoonup \Gamma^\omega]$. Transformations of finite strings are special cases of transformations of ω-strings with an appropriate padding of the strings by a special letter $\perp$. We use the following example in our illustrations.

Example 1 (Copy and Reverse). *Let $\Sigma = \{a, b, \#\}$. The ω-transformation $f_1 : \Sigma^\omega \rightharpoonup \Sigma^\omega$ is such that it replaces any maximal $\#$-free finite string u by $\overline{u}u$, where $\overline{u}$ is the reverse of u. Moreover f_1 is defined only for strings in $\Sigma^*\{a, b\}^\omega$. Formally, for all $w = u_1\#u_2\# \ldots u_n\#v$ such that $u_i \in \{a, b\}^*$ and $v \in \{a, b\}^\omega$, we have that $f_1(w) = \overline{u_1}u_1\# \ldots \#\overline{u_n}u_n\#v$.*

In this section we introduce three transducers implementing ω-transformations with equal expressive power: MSO transducers, two-way transducers with ω-regular look-around, and streaming ω-string transducers.

A. MSO Transducers

Courcelle [6] initiated the study of graph transformations $R \subseteq GR(\Sigma) \times GR(\Gamma)$ using monadic second-order logic. The main idea is to define a transformation $(G, G') \in R$ by defining the graph G' using a finite number of copies of G. The existence of nodes, edges, and node-labels in G' is then given as $\mathrm{MSO}(\Sigma)$ 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 $\mathrm{MSO}(\Sigma)$ formula characterizing the domain of the transformation;
- $C{=}\{1, 2, \ldots, n\}$ is a finite index set;
- $\phi_{\mathrm{nodes}}{=}\{\phi^c_\gamma(x) : c \in C \text{ and } \gamma \in \Gamma\}$ is a finite set of $\mathrm{MSO}(\Sigma)$ formulas with a free node variable x;
- $\phi_{\mathrm{edges}}{=}\{\phi^{c,d}(x, y) : c, d \in C\}$ is a finite set of $\mathrm{MSO}(\Sigma)$ formulas with two free node variables x and y.

The graph transformation $[\![T]\!]$ defined by T is as follows. A graph $G{=}(V, E, L) \in GR(\Sigma)$ is in the domain of $[\![T]\!]$ if $G \models \phi_{\mathrm{dom}}$ and the output is the graph $G'{=}(V', E', T')$ s.t.

- V' is the set of nodes v^c such that $v \in V$, $c \in C$ and there is a unique $\gamma \in \Gamma$ such that $G \models \phi^c_\gamma(v)$; notice that a node v^c is absent if $G \models \neg\phi^c(v)$ where $\phi^c(v) \overset{\text{def}}{=} \vee_{\gamma\in\Gamma}\phi^c_\gamma(v)$;
- $E'{\subseteq}V'{\times}V'$ is the set of edges such that for $v, u \in V$ and $c, d \in C$ we have that $(v^c, u^d) \in E'$ if $G \models \phi^{c,d}(v, u)$;
- $L' : V' \to \Gamma$ is such that $L'(v^c) = \gamma$ if $G \models \phi^c_\gamma(v)$.

Note that the output is unique and therefore MSO transducers implement functions. An MSO string transducer is an MSO graph transducer such that its domain is restricted to string graphs, and the output is also a string graph. We write MSOT for the set of MSO-definable ω-transformations.

Example 2. *Let us consider the following MSO formulas with their intuitive meaning: $\phi_{\#<\infty}$ (holds in any string graph that contains finitely many $\#$), $\mathrm{reach}_\#(x)$ (holds if from x one can reach a node labeled $\#$), $\mathrm{first}(x)$ (holds if x is first position of the string) and $\mathrm{path}(x, y)$ (holds if there is a path from x to y). Transformation f_1 from Example 1 is implemented (see Fig.3) by* MSOT *$T{=}(\Sigma, \Gamma, \phi_{\mathrm{dom}}, C, \phi_{\mathrm{nodes}}, \phi_{\mathrm{edges}})$ where:*

- $\Sigma = \Gamma = \{a, b, \#\}$, $C = \{1, 2\}$, *and*
- $\phi_{\mathrm{dom}} = \phi_{\#<\infty}$,
- $\phi^1_\gamma(x) = L_\gamma(x) \wedge \neg L_\#(x) \wedge \mathrm{reach}_\#(x)$
- $\phi^2_\gamma(x) = L_\gamma(x)$
- $\phi^{1,1}(x, y) = \phi^1(x) \wedge \phi^1(y) \wedge E(y, x)$
- $\phi^{2,2}(x, y) = E(x, y){\wedge}(\neg L_\#(x){\vee}(L_\#(x){\wedge}\neg\mathrm{reach}_\#(x)))$
- $\phi^{1,2}(x, y){=}(x{=}y){\wedge}\phi^1(x){\wedge}(\mathrm{first}(x){\vee}\exists z(L_\#(z){\wedge}E(z, x)))$
- $\phi^{2,1}(x, y){=}\phi^1(y) \wedge L_\#(x) \wedge \mathrm{reach}_\#(x) \wedge (\exists z(E(y, z) \wedge L_\#(z))) \wedge (\forall z((\mathrm{path}(x, z) \wedge \mathrm{path}(z, y)) \to \neg L_\#(z)))$

We say that an ω-string transformation is *regular* iff it can be implemented by an MSO string transducer. We next present a two-way finite state machine model of ω-transformations, and show that it can express all regular ω-transformations.

B. Two-Way ω-string Transducers with Regular Look-Around

A tuple $A = (Q_A, \Sigma, \Delta_A, \mathcal{F}_A)$ is a *look-ahead automaton* if for all $p \in Q_A$, $(Q_A, p, \Sigma, \Delta_A, \mathcal{F}_A)$ is a Muller automaton (we write $L(A, p)$ for its recognized language). We say that a tuple $B = (Q_B, \Sigma, \Delta_B, F_B)$ is a *look-behind automaton* if for all $p \in Q_B$, $(Q_B, p, \Sigma, \Delta_B, F_B)$ is a deterministic finite automaton (we write $L(B, p)$ for its recognized language).

We consider 2-way ω-string transducers with regular look-around (with both look-ahead and look-behind capabilities);

however we note that the look-behind feature is introduced for technical convenience, and is not crucial for MSO-equivalence.

Definition 2. *A 2-way ω-string transducer with look-around (*$2\mathrm{WST}_{\mathrm{la}}$*) is a pair (T, A, B) where A and B are look-ahead and look-behind automata, resp., and $T = (\Sigma, \Gamma, Q, q_0, \delta, F)$ is a tuple s.t. Σ and Γ are finite sets of input and output alphabets, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times Q_B \times \Sigma \times Q_A \to Q \times \Gamma \times \{-1, 0, +1\}$ is a transition function, and $F \subseteq 2^S$ is a Muller acceptance condition.*

A 2-way transducer stores the input string on a two-way tape and hence can read each index of input string multiple times. A configuration of a 2-way transducer is thus the tuple $(q, i) \in Q \times \mathbb{N}$ where q is the state of the 2WST, and i is the current position on the input tape. We assume that $2\mathrm{WST}_{\mathrm{la}}$ are *deterministic* i.e. for every string $s \in \Sigma^\omega$ and every input position $i \leq |s|$, there is exactly one state $p \in Q_A$ and one state $r \in Q_B$ such that $s(i)s(i+1)\ldots \in L(A, p)$ and $s(0)s(1)\ldots s(i-1) \in L(B, r)$.

Let $s \in \Sigma^\omega$ be an input string. If the current configuration is (q, i) and $\delta(q, r, s(i), p) = (q', z, d)$ is a transition, such that the string $s(i)s(i+1)\ldots \in L(A, p)$ and $s(0)s(1)\ldots s(i-1) \in L(B, r)$, then $2\mathrm{WST}_{\mathrm{la}}$ writes $z \in \Gamma$ on the output tape and updates its configuration to $(q', i + d)$ where d specify the direction of the movement of input pointer, such that $d \in \{-1, 0, +1\}$ if $i > 0$ and $d \in \{0, +1\}$ if $i = 0$. We denote such a transition as $(q, i) \xrightarrow{r, s(i), p | z} (q', i + d)$. The run of a 2WST T on an input string s is the sequence of transitions $\mathrm{run}(s) = (q_0, i_0{=}0) \xrightarrow{r_1, s(i_0), p_1 | z_1} (q_1, i_1) \xrightarrow{r_2, s(i_1), p_2 | z_2} \cdots$. The output $\mathrm{out}(r)$ of such a run r is defined as $z_1 z_2 \ldots$.

We say that the run r reads the whole string s if $\sup\{i_n : 0 \leq n < |r|\} = \infty$. The output $T(s)$ of a string s is defined only when $\Omega(\mathrm{run}(s)) \in F$ and $\mathrm{run}(s)$ reads the whole string s, and it is equal to $\mathrm{out}(\mathrm{run}(s))$. Without the latter requirement, transductions that are not regular could be defined: e.g. the transduction that maps any ω-string $u\#v$ such that $\#$ occurs only once, to u^ω.

We say that a look-ahead (look-behind) automaton is *blind* if for every starting state $p \in Q_A$ ($r \in Q_B$) we have that $L(A, p) = \Sigma^\omega$ ($L(B, r) = \Sigma^*$). A two-way ω-string transducer 2WST (without a look-around) is defined as a $2\mathrm{WST}_{\mathrm{la}}$ with a blind look-ahead and look-behind. As mentioned in Introduction, $2\mathrm{WST}_{\mathrm{la}}$ are strictly more expressive than 2WST.

The following result can be easily extended from the results in [7] for the transducers over finite strings and classical results of Büchi [4] and McNaughton [5].

Proposition 1 (MSOT $= 2\mathrm{WST}_{\mathrm{la}}$). *An ω-regular transformation is* MSOT*-definable iff it is* $2\mathrm{WST}_{\mathrm{la}}$*-definable.*

C. Streaming ω-String Transducers

Let X be a finite set of variables and Γ be a finite alphabet. A substitution σ is defined as a mapping $\sigma : X \to (\Gamma \cup X)^*$. A valuation is defined as a substitution $\sigma : X \to \Gamma^*$. Let $\mathcal{S}_{X,\Gamma}$ be the set of all substitutions $[X \to (\Gamma \cup X)^*]$. Any substitution σ can be extended to $\hat{\sigma} : (\Gamma \cup X)^* \to (\Gamma \cup X)^*$ 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 $x \in X$. We say that a string $u \in (\Gamma \cup X)^*$ is *copyless* (or linear) if each $x \in X$ occurs at most once in u. A substitution σ is copyless if $\hat{\sigma}(u)$ is copyless, for all $u \in (\Gamma \cup X)^*$.

Definition 3. *A (deterministic) streaming ω-string transducer (*SST*) is a tuple $T = (\Sigma, \Gamma, Q, q_0, \delta, X, \rho, F)$ where:*
- *Σ and Γ are finite sets of input and output alphabets;*
- *Q is a finite set of states with initial state q_0;*
- *$\delta : Q \times \Sigma \to Q$ is a transition function;*
- *X is a finite set of variables;*
- *$\rho : (Q \times \Sigma) \to \mathcal{S}_{X,\Gamma}$ is a variable update function such that $\rho(q, a)$ is a copyless substitution for $(q, a) \in Q \times \Sigma$;*
- *$F : 2^Q \rightharpoonup X^*$ is an output function such that for all $P \in \mathrm{dom}(F)$ the string $F(P)$ is copyless of form $x_1 \ldots x_n$, and for $q, q' \in P$ and $a \in \Sigma$ s.t. $q' = \delta(q, a)$ we have*
  - *$\rho(q, a)(x_i) = x_i$ for all $i < n$ and*
  - *$\rho(q, a)(x_n) = x_n u$ for some $u \in (\Gamma \cup X)^*$.*

The concept of a run of an SST is defined in an analogous manner to that of a Muller automaton. The sequence $\langle \sigma_{r,i} \rangle_{0 \leq i \leq |r|}$ of substitutions induced by a run $r = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \ldots$ is defined inductively as the following: $\sigma_{r,i} {=} \sigma_{r,i-1}\rho(q_{i-1}, a_i)$ for $0 < i \leq |r|$ and $\sigma_{r,0} = x \in X \mapsto \varepsilon$. The output function F can be extended to the infinite runs such that $F(r) = F(\Omega(r))$ for every run r. The output $T(r)$ of an infinite run r of T is defined only if $F(r)$ is defined and equals

$$T(r) \overset{\text{def}}{=} \lim_{i \to \infty} \langle \sigma_{r,i}(F(r)) \bot^\omega \rangle.$$

The assumptions on the output function F, introduced in Definition 3, ensure that this limit always exist whenever $F(r)$ is defined. Indeed, when a run r reaches a point from where it visits only states in P, these assumptions enforce the successive valuations of $F(P)$ to be an increasing sequence of strings by the prefix relation. The padding by unique letter $\bot$ ensures that the output is always an ω-string. The output $T(s)$ of a string s is then defined as the output $T(r)$ of its unique run r. The transformation $[\![T]\!]$ defined by an SST T is the partial function $\{(s, T(s)) : T(s) \text{ is defined}\}$.

Example 3. *The transformation f_1 from Example 1 is definable by the* SST *in Figure 4. Consider the successive valuations of x, y, and z upon reading the string $ab\#a^\omega$.*

		a	b	$\#$	a	a $\ldots$
x	ε	ε	ε	$baab\#$	$baab\#$	$baab\#$
y	ε	aa	$baab$	ε	aa	$aaaa$
z	ε	a	ab	ε	a	aa

Notice that the limit of xz exists and equals $baab\#a^\omega$.

We remark that for every SST $T = (\Sigma, \Gamma, Q, q_0, \delta, X, \rho, F)$, its domain is always an ω-regular language defined by the Muller automaton $(\Sigma, Q, q_0, \delta, \mathrm{dom}(F))$, which can be constructed in linear time. However, the range of an SST may not be ω-regular. For instance, the range of the SST-definable transformation $a^n\#^\omega \mapsto a^n b^n \#^\omega$ $(n \geq 0)$ is not ω-regular.

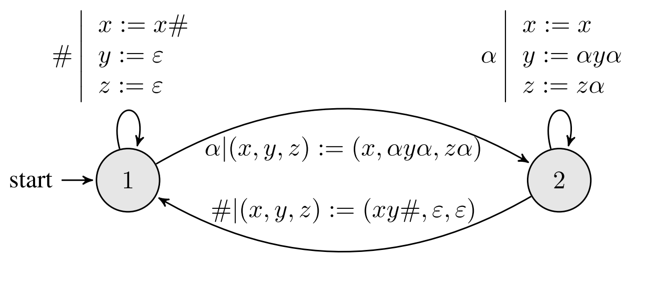


Fig. 4. SST defining the transformation f_1 of Example 1. Here transition labeled with α stands for those with label a or b, and the output function is such that $F(\{2\}) = xz$.

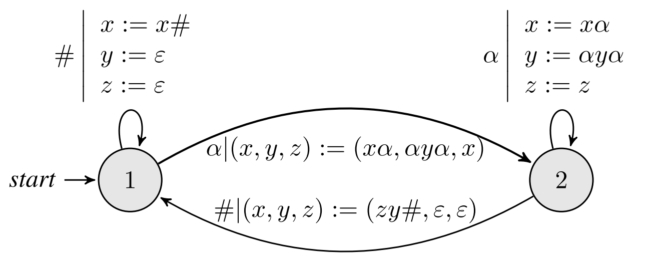
IV. Equivalent Extensions of SST

In this section, we demonstrate the robustness of the class of SST-definable transformations by adding several features that do not increase expressiveness. First, we introduce copying in the variable updates, which in general may lead to non-linear size increase in transformation. We show that the class of SST is closed under a restricted application of such copy—the so-called *bounded copy*—which allows the content of some variable to be copied a bounded number of times. We also introduce non-deterministic SSTs, i.e. SSTs with nondeterministic transitions, and show that any non-deterministic SST which defines a function is equivalent to some (deterministic) SST. Finally, we show that enabling the transitions to employ regular look-around does not increase expressiveness of SST.

A. SST*s with Bounded Copy*

Copyful SST*s* generalize SSTs by dropping the requirement on variable updates to be copyless. Copyful SST are strictly more expressive than SST. For instance, an ω-transformation $a^n b^\omega \mapsto a^{2^n-1} b^\omega$ is not definable by an SST, as SST can only express transformations with linear-size increase [10], but it is definable by a copyful SST.

Example 4. *Consider the following copyful implementation of the transformation f_1.*



Here $\alpha \in \{a, b\}$ and the output function F is defined by $F(\{2\}) = x$.

Also, variable x denotes the output, y denotes the string $\overline{u}u$ where u is the current maximal $\#$-free string, and z denotes the value of x when the last $\#$ was read (if any, otherwise it is the current value of x). Observe that the number of times variable x gets copied (via the thick transition) is unbounded. However, this transformation can be implemented as an SST *(Example 3), as the content of x is never present twice in the same string variable. On the thick transition variable x contributes to both x and z. This copy stays "alive" while looping on state 2, i.e. the content of x before copying is present in two places (x and z). However, the copy "dies" when taking a transition from 2 to 1: x is reset to $zy\#$ and z to ε. Hence, at any given time there are at most 1 alive copy, and we say that this* SST *is 1-bounded.*

In this section, we define a restriction of copyful SSTs, called *bounded copy* SST*s* ($\mathrm{SST_{bc}}$), that stays as expressive as SSTs. Intuitively, it is defined as SST for which the number of alive copies is bounded at any time. This notion is better formalized by introducing *dependency graphs*.

Dependency Graphs. A dependency graph keeps track of the variable dependencies along some run of an SST. The presence of an edge from a node labeled x to a node labeled y indicates that the variable update of y uses variable x. We use multiple edges when some variable is used several times to update the same variable (e.g. $x := yy$). A node labeled x with two successors labeled y and z resp. indicates a copyful variable update, where x has been copied into both variables y and z. A node labeled x with two predecessors indicates a variable update where x has been updated by concatenating the variables of its predecessors. The dependency graphs also remember the order in which the variables are concatenated via a mapping pred from any node to a finite string over its predecessors. E.g., for an assignment $x := zaybz$, we let $\mathrm{pred}(n_x) = n_z n_y n_z$ where the edges (n_y, n_x) and (n_z, n_x) represent the dependencies $y \to x$ and $z \to x$ respectively. We also maintain pointers from variables to terminal nodes.

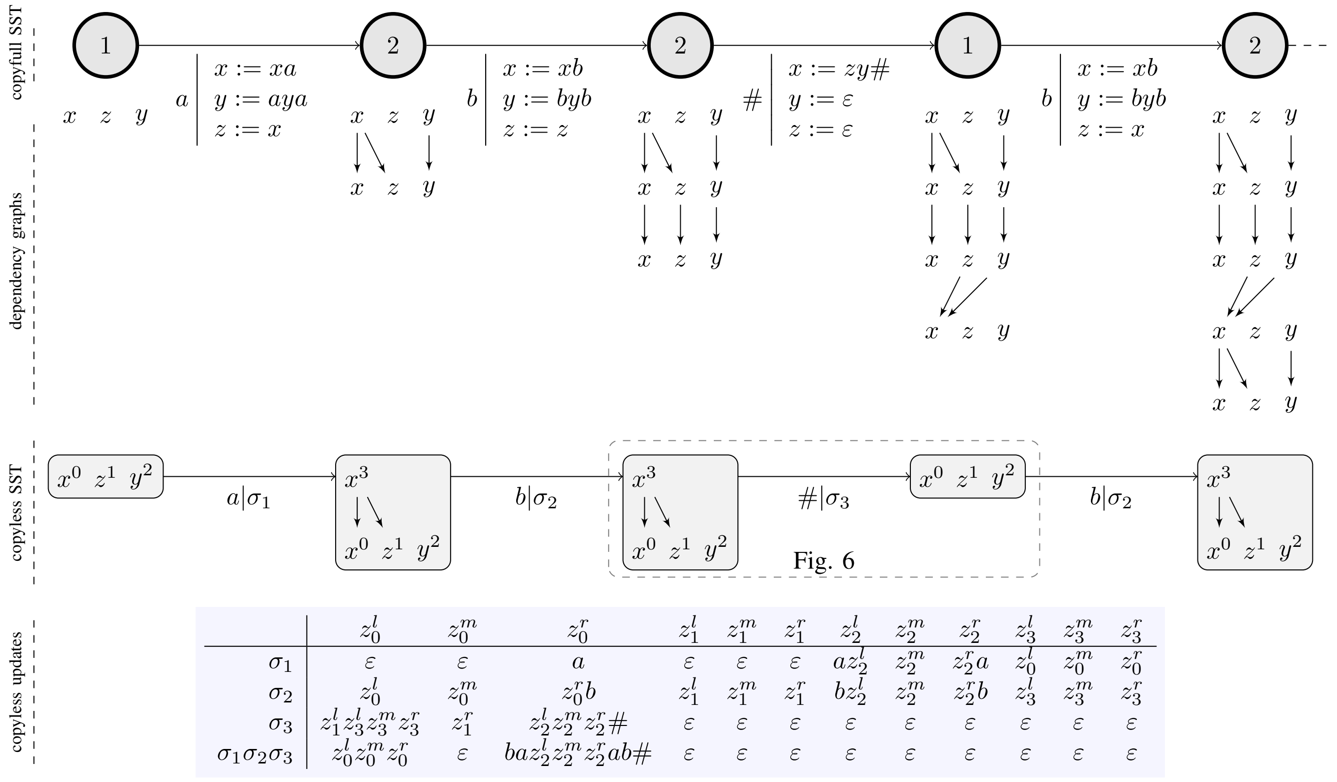
Definition 4 (Dependency Graph). *A* dependency graph *over a set of variables X is a tuple $(V, E, I_0, \lambda, \mathrm{pred})$ such that:*

- *$(V, E, \lambda{:}V{\to}X)$ is (finite) labeled directed acyclic graph, where $E : V^2 \to \mathbb{N}$ is a multiset;*
- *$I_0 : X{\to}V$ is an injective mapping such that $\lambda(I_0(x)){=}x$ and $I_0(x)$ has no successor, for all $x \in X$; and*
- *$\mathrm{pred} : V \to V^*$ associates any $n \in V$ with a finite string over V that represents the predecessors of n and is consistent with E, i.e. for all $n' \in V$, n' occurs exactly $E((n', n))$ times in $\mathrm{pred}(n)$.*

Each finite run of a (copyful) SST can naturally be associated with its dependency graph. Given a copyful SST T and a finite run $r = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots \xrightarrow{a_\ell} q_\ell$ of T, the dependency graph $G_T(r)$ of r is a tuple $(V, E, I_0, \lambda, \mathrm{pred})$ where:

1) $V = X \times \{0, \dots, \ell\}$;
2) for $i < \ell$, $k \geq 0$, and $x, y{\in}X$, $E((x,i),(y,i+1))$ equals k iff x occurs k times in $\rho(q_i, a_{i+1})(y)$;
3) for all $x \in X$, $I_0(x) = (x, \ell)$;
4) for all $(x,i) \in V$, $\lambda(x,i) = x$,
5) for all $i < \ell$, variables $x, x_1, \dots, x_k \in X$, $u_0, \dots, u_k \in \Gamma^*$, if $\rho(q_i, a_{i+1})(x) = u_0 x_1 u_1 \dots x_k u_k$, then $\mathrm{pred}((x, i+1)) = (x_1, i) \dots (x_k, i)$.

Example 5. *Dependency graphs can best be explained via examples. The sequence of dependency graphs of the successive runs of the copyful* SST *of Example 4 on the prefixes of*


copyfull SST
dependency graphs
copyless SST
copyless updates
$x := xa$, $y := aya$, $z := x$
$x := xb$, $y := byb$, $z := z$
$x := zy\#$, $y := \varepsilon$, $z := \varepsilon$
$x := xb$, $y := byb$, $z := x$
$a|\sigma_1$
$b|\sigma_2$
$\#|\sigma_3$
$b|\sigma_2$
Fig. 6

	z_0^l	z_0^m	z_0^r	z_1^l	z_1^m	z_1^r	z_2^l	z_2^m	z_2^r	z_3^l	z_3^m	z_3^r
σ_1	ε	ε	a	ε	ε	ε	az_2^l	z_2^m	$z_2^r a$	z_0^l	z_0^m	z_0^r
σ_2	z_0^l	z_0^m	$z_0^r b$	z_1^l	z_1^m	z_1^r	bz_2^l	z_2^m	$z_2^r b$	z_3^l	z_3^m	z_3^r
σ_3	$z_1^l z_3^l z_3^m z_3^r$	z_1^r	$z_2^l z_2^m z_2^r \#$	ε	ε	ε	ε	ε	ε	ε	ε	ε
$\sigma_1\sigma_2\sigma_3$	$z_0^l z_0^m z_0^r$	ε	$baz_2^l z_2^m z_2^r ab\#$	ε	ε	ε	ε	ε	ε	ε	ε	ε

Fig. 5. A run of the bounded copy SST of Example 4 on $ab\#b$ and its simulation by the construction of Prop.2.

$ab\#b$ is shown in Fig. 5. The mappings I_0 are not represented but they correspond to the bottom levels of the dependency graphs. The order on predecessors (not represented on the figure), are for instance given by $\text{pred}((x,1)) = (x,0)$ *and* $\text{pred}((x,3)) = (z,2)(y,2)$ *for the fourth graph.*

Equivalently, the dependency graph of a run can be defined by successive applications of an *extension* operator $\boxtimes$, defined from the variable updates of the run and the initial dependency graph $(X, \varnothing, id_X, id_X, \text{pred}_0)$ where pred_0 maps any node to the empty sequence. E.g., the extension of some graph by the substitution $(x, y, z) := (zy\#, \varepsilon, \varepsilon)$ is shown in Fig. 6. Let $G{=}(V, E, I_0, \lambda, \text{pred})$ be a dependency graph and β a substitution of X-variables. We assume that $V \cap X = \varnothing$, otherwise we rename the vertices of G. We define $E' = \{(I_o(y), x) \mapsto k \mid y \text{ occurs } k \text{ times in } \beta(x)\}$, $I'_0(x) = x$ and $\lambda'(x) = x$ for all $x{\in}X$. Let $\beta(x) = u_0 x_1 u_1 \dots x_k u_k$ for some $u_i{\in}\Gamma^*$ and $x_i{\in}X$. Then $\text{pred}'(x)$ is defined by $n_1 \dots n_k$ such that $(n_i, x){\in}E'$ and $\lambda(n_i){=}x_i$ for all $1{\le}i{\le}k$. The *extension* $G\boxtimes\beta$ of G by β, is defined as $(V{\cup}X, E{\cup}E', I'_0, \lambda{\cup}\lambda', \text{pred}{\cup}\text{pred}')$.
SSTs with Bounded Copy. We now define the notion of bounded copy. Let $G = (V, E, I_0, \lambda, \text{pred})$ be a dependency graph. We call a node n in G a *copy node* if it has at least two outgoing edges (n, n_1) and (n, n_2) (n_1 may equal n_2 as multiple edges are allowed, in that case we require that $E(n, n_1) \geq 2$). We say that the copy node n is *alive* if there are nodes $n'_1, n'_2 \in I_0(X)$ such that there exist a (directed) path from n_1 to n'_1 and a (directed) path from n_2 to n'_2 ($I_0(X)$ denotes the image of function I_0). For example, in the last dependency graph of Fig. 5, the occurrence of x on the top (level 0) is a copy node but is not alive, while the occurrence of x on the 3rd level is an alive copy node. We are now in the position to define bounded copyful SST:

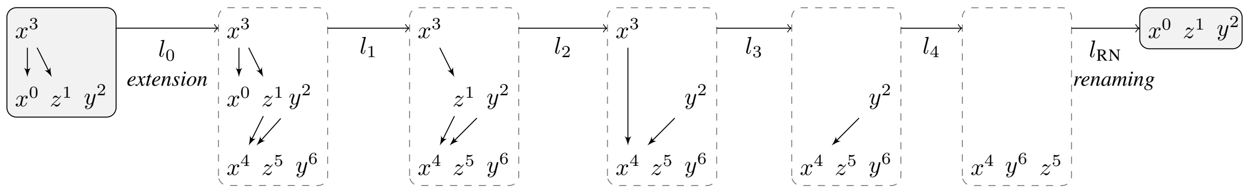
Definition 5 (SST with bounded copy)**.** *A K-bounded copyful* SST *is a copyful* SST *such that for all runs r, the number of alive copy nodes of $G_T(r)$ is bounded by $K \in \mathbb{N}$. The class of K-bounded copy* SST *for all $K \in \mathbb{N}$ is denoted by* SST_{bc}.

For example, the copyful SST of Example 4 is 1-bounded: at most one copy node is alive in any dependency graph. The most technically challenging result of this paper is to show that bounded copy feature does not add expressiveness to SST.

Proposition 2 ($\text{SST}_{\text{bc}} = \text{SST}$)**.** *Any ω-transformation definable by a bounded copyful* SST *is* SST*-definable.*

The rest of this section is devoted to the proof of this proposition. We let $T = (Q, q_0, \Sigma, \Gamma, \delta, X, \rho, F)$ be a K-bounded copyful SST. We show that it can be simulated by a copyless SST T'. For the sake of technical convenience, we make an assumption that ensures any node of dependency graphs to have at most two successors and at most two predecessors. This can be done, while preserving boundedness, by decomposing updates as compositions of simple updates with extra variables, and using epsilon transitions.

Assumption 1. *We assume that any variable update $\rho(q, a)$ satisfies the following conditions, for all $x \in X$: (i) $\rho(q, a)(x)$ concatenates at most two variables, i.e. contains at most two*



	z_0^l	z_0^m	z_0^r	z_1^l	z_1^m	z_1^r	z_2^l	z_2^m	z_2^r	z_3^l	z_3^m	z_3^r	z_4^l	z_4^m	z_4^r	z_5^l	z_5^m	z_5^r	z_6^l	z_6^m	z_6^r
l_0													ε	ε	$\#$	ε	ε	ε	ε	ε	ε
l_1																					
l_2				ε	ε	ε							$z_4^l z_1^l$	$z_1^r z_4^m$							
l_3										ε	ε	ε	$z_4^l z_3^l z_3^m z_3^r$								
l_4							ε	ε	ε						$z_2^l z_2^m z_2^r z_4^r$						
l_{RN}	z_4^l	z_4^m	z_4^r	z_5^l	z_5^m	z_5^r	z_6^l	z_6^m	z_6^r	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε	ε

Fig. 6. Intermediate steps of the construction of $\sigma_3 = l_0 l_1 l_2 l_3 l_4 l_{\mathsf{RN}}$ in the the copyless SST of Fig. 5. Only changing updates are represented.

occurrences of variables, (ii) *there exist at most two variables* $y \in X$ *such that* x *occurs in* $\rho(q,a)(y)$.

The main idea of the proof of Prop. 2 is to summarize into new variables, from a countable set Z, the result of variable updates that occur on *linear branches* (i.e. sequences of nodes with out and indegree at most 1) of dependency graphs. These linear branches represent the evolution of variables by updates that do not involve concatenation or copy. The summary variables in Z can be updated in a copyless manner, while the copying structure of T can be stored in the states of an SST as dependency graphs. There are, however, infinitely many possible dependency graphs in general, therefore the SST T' stores only an abstraction of them, called the *reduced dependency graphs*. Moreover, we show that total number of reduced dependency graphs is finite for a K-bounded SST. Reduced dependency graphs keep only special nodes: copy nodes, concatenation nodes (nodes with indegree 2), and terminal nodes, while the linear branches are summarized by a single edge. Moreover, nodes that are not connected to the terminal level $I_0(X)$ are removed, since their values can not contribute to the output. The values of string variables X can be reconstructed from the values of summary variables Z and the reduced dependency graphs.

The states of the copyless SST T' are pairs (q, G) where q is a state of T and G is a reduced dependency graph. After reading an input symbol $a \in \Sigma$, the next state is $(\delta(q,a), G')$ where G' is the reduction of the extension $G \boxtimes \rho(q,a)$. Moreover, the nodes of the new reduced dependency graph are renamed in such a way that they define an initial segment of $\mathbb{N}$. This ensures, together with the K-bounded assumption, that the number of reduced dependency graphs is finite.

The reduction of dependency graphs is shown in Fig. 5, in which the graphs in the rectangular nodes correspond to the reduction of the corresponding dependency graphs above (nodes are identified by natural numbers in superscripts). The reduction of a graph is defined by successive applications of simple rewriting rules, as shown in Fig. 6 (transitions l_1 to l_4).

We now informally describe the variable mechanism of the copyless SST, and how values of string variables X are reconstructed from summary variables Z and reduced dependency graphs. For all nodes n of a reduced dependency graph, we introduce three fresh summary variables z_n^l, z_n^m and z_n^r (left, middle and right). Since there are finitely many reduced dependency graphs, one needs only a finite number of summary variables for the construction. Suppose that $\text{pred}(n) = n_1 n_2$ for some nodes n_1, n_2. As the graph is reduced, the edges (n_1, n) and (n_2, n) represent two linear branches in the original dependency graph, i.e. two sequences of updates of $\lambda(n_1)$ and $\lambda(n_2)$ that do not involve copy or concatenation. The result of these two sequences of updates are necessarily strings of the form $u_1 \lambda(n_1) u_1'$ and $u_2 \lambda(n_2) u_2'$ where $u_1, u_1', u_2, u_2' \in \Gamma^*$. The fact that (n_1, n) and (n_2, n) have the same target indicates a concatenation update of the form $\lambda(n) := u^l u_1 \lambda(n_1) u_1' u^m u_2 \lambda(n_2) u_2' u^r$, for some $u^l, u^m, u^r \in \Gamma^*$. The variables z_n^l, z_n^m and z_n^r represent the strings $u^l u_1$, $u_1' u^m u_2$ and $u_2' u^r$ respectively.

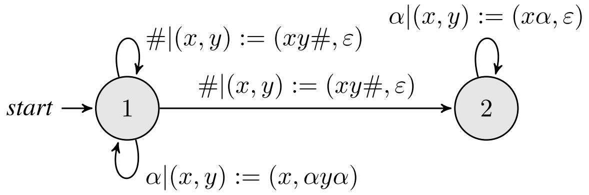
More generally, the value of some variable x at some node n is represented by the value of a string of summary variables, called *shape*, defined by the variables z_0^l, z_0^m and z_0^r and inductively by the shape of its predecessor nodes. For instance, the value of x after the first transition of Fig. 5 is given by the value of $z_0^l z_3^l z_3^m z_3^r z_0^r$. The variables z_i^m are needed for nodes with two incoming edges, as shown before.

Example 6. *In Fig. 5 we show a run of the copyful* SST T *from Example 4, its simulation by a copyless* SST T', *and corresponding variable updates. If initially the values of* x, y *and* z *are* ε, *then after the third transition, the value of* x *is* $baab\#$. *In the copyless* SST T', *the value of* x *after the third transition equals the value of* $z_0^l z_0^m z_0^r$, *which, by* $\sigma_1 \sigma_2 \sigma_3$, *maps to* $z_0^l z_0^m z_0^r ba z_2^l z_2^m z_2^r ab\#$, *which in turn maps to* $baab\#$ *if the summary variables are initialized to* ε. *In Fig. 6 we detail a transition of the copyless SST. It starts with the extension of the reduced dependency graph, followed by the successive rewritings of the reduction, and ends by the renaming of its nodes. Each of these steps induces some summary* Z*-variable update (depicted in the table) such that* $\sigma_3 = l_0 l_1 l_2 l_3 l_4 l_{\mathsf{RN}}$.

B. Functional Non-deterministic SST*s*

Non-deterministic SST (NSST for short) are defined similarly as SST, except that the transitions are defined by a relation $\Delta \subseteq Q\times\Sigma\times Q$ and the update function is a total mapping $\rho : \Delta \to \mathcal{S}$ that maps any transition to a linear substitution. Any NSST T defines a relation $[\![T]\!] \subseteq \Sigma^\omega \times \Gamma^\omega$. We say that an NSST T is *functional* if $[\![T]\!]$ is a partial function, i.e. for all $w \in \Sigma^\omega$, $\{w' \mid (w,w') \in [\![T]\!]\}$ has cardinality at most 1. Note that if an ω-transformation is definable by an NSST, this NSST is functional as ω-transformations are (partial) functions. In this section, we show that any ω-transformation definable by an NSST is definable by a bounded copyful SST, and therefore by an SST, thanks to Prop. 2.

Example 7. *The transformation f_1 of Example 1 can be defined by the following* NSST*, in which non-determinism is used to guess the last occurrence of $\#$:*


$\#|(x,y) := (xy\#,\varepsilon)$
$\alpha|(x,y) := (x\alpha,\varepsilon)$
$\#|(x,y) := (xy\#,\varepsilon)$
start
1
2
$\alpha|(x,y) := (x,\alpha y\alpha)$

Here $\alpha \in \{a,b\}$ and the output function is $F(\{2\}) = x$.

Lemma 1. *Any functional* NSST *with n states and m variables can be transformed into an equivalent mn-bounded copyful* SST.

Proof (Sketch): Let $T = (Q, q_0, \Sigma, \Gamma, \Delta, X, \rho, F)$ be a functional NSST. Wlog we assume that T is trimmed, i.e. any state q occurs in a least one run r such that $F(r)$ is defined. Any NSST can be trimmed in polynomial time.

We construct a bounded copyful SST T' and show its equivalence to T. As T is non-deterministic, T' has to simulate all the runs of T on an input string in a deterministic way. The construction of T' uses determinization of Muller automata as a black box. The key observation is that, since T is trimmed and functional, we have that if two runs of T on an input both reach some state q, then any accepting continuation of those two runs on the same input string will produce the same output. Therefore, T' can stop simulating one of the two runs when they both reach q. This implies that only $|Q|$ runs are simulated in parallel. Then for all $x \in X$, a copy x_q of x is introduced for all $q \in Q$, that denotes the content of variable x for all the runs of T that are in state q. E.g., if there is a transition of T from p to q that updates x to xa, and another transition from p to q' that updates x to xb, then T' updates the variables x_q and $x_{q'}$ as $x_q := x_p a$ and $x_{q'} := x_p b$. Variable updates may introduce copy, but in bounded manner. ■

From Prop. 2 and Lemma 1, we get the following result.

Proposition 3. *An ω-transformation is definable by an* NSST *iff it is definable by an* SST.

C. SST*s with Regular Look-Around*

We introduce NSST with look-around (NSST$_{\mathrm{la}}$) and prove that they can define all SST-definable transformations. In this section we summarize the main features of NSST$_{\mathrm{la}}$ that are required to understand the key ideas of the proof.

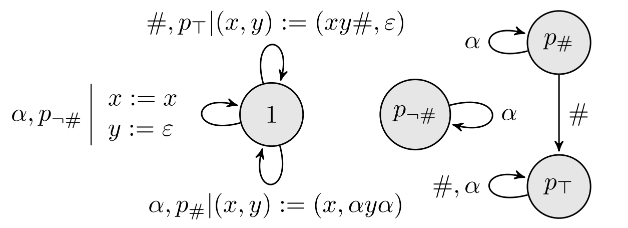


Fig. 7. SST with look-ahead defining the transformation f_1 of Example 1, for all $\alpha \in \{a,b\}$. The output function F is defined by $F(\{1\}) = x$. The accepting sets of the look-ahead are $\{p_\top\}$ and $\{p_{\neg\#}\}$.

Similar to 2-way transducers with regular look-around, NSST$_{\mathrm{la}}$ are equipped with a (look-behind) finite automaton B that can inspect the current (finite) prefix and a (look-ahead) Muller automaton A that can inspect the current infinite suffix. Transitions depend on the result of those inspections. In particular, the transition relation is a set of tuples (q, r, a, p, q') where q, q' are two states of the NSST, a is an input symbol, r (resp. p) is a state of the look-behind (resp. look-ahead) automaton. The transition (q, r, a, p, q') can be triggered if the current symbol is a, prefix up to current position is accepted by B with initial state r, and the infinite suffix starting at the current position is accepted by A with initial state p.

Example 8. *The transformation f_1 of Example 1 is defined by the* SST *with look-around in Fig. 7 (only the look-ahead is represented, the look-behind is assumed to be blind). The look-ahead can accept any suffix (state $p_\top$), or decides if the suffix contains $\#$ (state $p_\#$), or not (state $p_{\neg\#}$).*

Lemma 2. *An ω-transformation is definable by a (functional)* NSST *with look-around if and only if it is definable by a (functional)* NSST.

Sketch of proof: Let T be an NSST with look-ahead A and look-behind B. The simulation of B is easy by a subset construction. To simulate look-ahead A, we view the calls to A as universal transitions: both the look-ahead and the transducer must accept the remaining suffix. We define an alternating Muller automaton U that recognizes the accepting runs of (T, A), i.e. the accepting sequences of tuples (q, a, p, q'), where q, q' are states of T, $a \in \Sigma$ and p is a look-ahead state. Alternation is then removed by standard techniques, so that U is equivalent to some Muller automaton V. The alphabet of V is finally projected on Σ, and the output mechanism of T added. This projection may introduce non-determinism, even if V is deterministic. ■

Combining Lemma 2 with Prop. 3 we get the following.

Proposition 4. *Any ω-transformation definable by a functional* NSST *with look-around is definable by an* SST.

V. MSO Expressiveness of SST

Now we have all the ingredients to prove our main result Theorem 1, i.e. to show that SST = MSOT. From Prop. 1 we know that two-way transducers with look-around can implement any MSO-definable transformation. The following proposition implies that functional SST with look-around can also implement all MSO-definable transformations.

Proposition 5 ($2\mathrm{WST}_{\mathrm{la}} \subseteq \mathrm{NSST}_{\mathrm{la}}$). *Any* $2\mathrm{WST}_{\mathrm{la}}$*-definable transformation is* $\mathrm{NSST}_{\mathrm{la}}$*-definable.*

To prove this proposition we show in two steps that for every $2\mathrm{WST}_{\mathrm{la}}$ there exists an $\mathrm{NSST}_{\mathrm{la}}$ implementing the same transformation. First, we construct an SST with copy implementing the same transformation adapting the classical reduction from two-way finite automata to one-way finite automata. Next, we remove the copy by introducing the nondeterminism.

From Prop. 4 it follows that every functional NSST with regular look-around is effectively equivalent to an SST. Therefore from Prop. 1 and Prop. 5, we get that any MSO-definable transformation is SST-definable. A natural extension of the proof for the finite string case yields the following result.

Proposition 6 ($\mathrm{SST} \subseteq \mathrm{MSOT}$). *Any* SST*-definable* ω*-transformation is* MSOT*-definable.*

VI. Decision Problems

Equivalence Problem. Given two SSTs the equivalence problem is to decide if they implement the same transformation

Theorem 3. *Equivalence problem for* SST *is in* PSPACE*.*

To prove this theorem we first check the equivalence of the domains of T_1 and T_2, and then check functionality of $T_1 \cup T_2$. Since (see Section III-C) a deterministic Muller automaton that recognizes the domain of a given SST can be constructed in linear time, and we can decide equivalence of Muller automata in PTIME [17], it follows that the equivalence of the domains can be decided in polynomial time. We next show that the functionality of an NSST can be decided in polynomial space.

Lemma 3. *Given an* NSST T*, it is decidable in* PSPACE *whether* T *is functional.*

Proof (Sketch): We construct a non-deterministic one-reversal two-counter machine M (on finite strings) such that $L(M) \neq \varnothing$ iff T is not functional. The size of M is exponential in the size of T, and since emptiness of bounded reversal counter machines is in NL [18], the result follows. Clearly, T is not functional iff there exist $u \in dom(T)$, $v_1, v_2 \in T(u)$ and $i \geq 0$ such that $v_1[i] \neq v_2[i]$. Intuitively, machine M guesses a position i, and two finite runs r_1 and r_2 of T on the same finite input string (of length i at least) and checks that r_1 and r_2 can be extended to form infinite runs, whose respective outputs differ at i-th position. The construction generalizes the one for NSST on finite strings [19]. However, as M runs on finite strings and T on infinite strings, we need to make sure that there exist infinite continuations of strings evaluated by M that are consistent with T. ∎

Type-Checking Problem. Given a streaming string transducer T, two ω-regular languages I and O as Muller automata A_I and A_O, respectively, the *type-checking problem* is to decide whether for all strings $u \in I$ we have that $T(u) \in O$.

Theorem 4. *Type-checking problem for* SST *is in* PSPACE*.*

VII. Conclusion

We initiate the study of the regular transformations of infinite strings. We have shown a way to generalize streaming string transducers to associate output with infinite strings that capture the MSO definable global transformations. We also show that model of two-way finite-state transducers extended with ω-regular look-around can express regular transformations of infinite strings. We show that some natural and useful extensions like bounded copy, functional nondeterminism, and regular look-ahead do not increase their expressive power. We prove the decidability of equivalence and type-checking problems for MSO transformations of infinite strings by showing that these problems can be solved for streaming string transducers in PSPACE. At this stage we are not aware of any concrete practical applications of regular transformations of infinite strings, but we note that the theory of ω-regular languages and the theory of regular transformations of finite strings, both have found interesting practical applications.

Acknowledgment

This research was partially supported by NSF awards CNS 0931239, CNS 1035715, CCF 0915777.

References

[1] 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.
[2] 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.
[3] B. A. Trakhtenbrot, “Finite automata and monadic second order logic,” *Siberian Mathematical Journal*, vol. 3, pp. 101–131, 1962.
[4] 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.
[5] R. McNaughton, “Testing and generating infinite sequences by a finite automaton,” *Inform. Contr.*, vol. 9, pp. 521–530, 1966.
[6] B. Courcelle, “Monadic second-order definable graph transductions: a survey,” *Theoretical Computer Science*, vol. 126(1), pp. 53–75, 1994.
[7] 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.
[8] J. Berstel, *Transductions and context-free languages*. Teubner, 1979.
[9] J. Sakarovitch, *Elements of Automata Theory*. Cambridge University Press, 2009.
[10] R. Alur and P. Černý, “Expressiveness of streaming string transducers,” in *FSTTCS*, vol. 8, 2010, pp. 1–12.
[11] A. Alur and P. Černý, “Streaming transducers for algorithmic verification of single-pass list-processing programs,” in *POPL*, 2011, pp. 599–610.
[12] F. Gire, “Two decidability problems for infinite words,” *IPL*, vol. 22, pp. 135–140, 3 Mar. 1986.
[13] S. Varricchio, “A polynomial time algorithm for the equivalence of two morphisms on omega-regular languages,” in *STACS 93*, ser. LNCS. Springer, 1993, vol. 665, pp. 595–606.
[14] Beal and Carton, “Determinization of transducers over infinite words: The general case,” *MST: Mathematical Systems Theory*, vol. 37, 2004.
[15] J. Engelfriet and S. Maneth, “Macro tree transducers, attribute grammars, and mso definable tree translations,” *Inform. and Comput*, vol. 154, pp. 34–91, 1998.
[16] R. Alur, E. Filiot, and A. Trivedi, “Regular transformations of infinite strings,” University of Pennsylvania, Tech. Rep. MS-CIS-12-05, 2012.
[17] E. Clarke, I. Draghicescu, and R. Kurshan, “A unified approach for showing language inclusion and equivalence between various types of omega-automata,” *IPL*, vol. 46, no. 6, pp. 301 – 308, 1993.
[18] Gurari and Ibarra, “A note on finite-valued and finitely ambiguous transducers,” *MST: Mathematical Systems Theory*, vol. 16, 1983.
[19] R. Alur and J. V. Deshmukh, “Nondeterministic streaming string transducers,” in *ICALP*, ser. LNCS, vol. 6756. Springer, 2011, pp. 1–20.