Nondeterministic Streaming String Transducers[⋆]

Rajeev Alur and Jyotirmoy V. Deshmukh

Dept. of Computer and Information Science,
University of Pennsylvania
{alur,djy}@cis.upenn.edu

Abstract. We introduce *nondeterministic streaming string transducers* (NSSTs) – a new computational model that can implement MSO-definable relations between strings. An NSST makes a single left-to-right pass on the input string and uses a finite set of *string variables* to compute the output. In each step, it reads one input symbol, and updates its string variables in parallel with a *copyless assignment*. We show that NSST are closed under sequential composition and that their expressive power coincides with that of nondeterministic MSO-definable transductions. Further, we identify the class of *functional* NSSTs; such an NSST allows nondeterministic transitions, but for every successful run on a given input generates the same output string. We show that deciding functionality of an arbitrary NSST is decidable with PSPACE complexity, while the equivalence problem for functional NSSTs is PSPACE-COMPLETE. We also show that checking if the set of outputs of an NSST is contained within the set of outputs of a finite number of DSSTs is decidable in PSPACE.

1 Introduction

In this paper, we introduce *nondeterministic streaming string transducers* (NSSTs). A run of such a transducer processes an input string in a single left-to-right pass in linear time, and computes an output string. The nondeterminism of an NSST allows it to produce different outputs for different runs on an input string. Thus, an NSST is a natural model for implementing *relations* between strings.

Classical literature on string-to-string transductions largely focusses on finite-state transducers that realize *rational relations* [12]. In each step, such a transducer reads an input symbol and writes zero or more symbols to the output. These transducers and their many variations have been extensively studied in the literature. If restricted to a single left-to-right pass over the input, *i.e.*, to their streaming versions, the expressive power of finite transducers is limited. For example, they can implement transductions such as inserting a symbol in an input string and deleting a substring of the input string, but they cannot implement transductions such as reversing an input string or swapping two substrings within an input string.

[⋆] This research is partially supported by NSF award CCF 0905464 and by the CCC-CRA Computing Innovation Fellows project.

Deterministic streaming string transducers (DSSTs), first introduced in [1,2], in addition to all transductions implemented by deterministic finite transducers, can implement transductions such as reversing a string and swapping substrings. A DSST reads the input in a single left-to-right pass. In addition to a finite set of states, it has a finite set of string variables that it uses to produce the output. In each step, a DSST reads an input symbol, changes its state, and concurrently updates all its string variables using a *copyless assignment*. The right-hand-sides (RHS) in a copyless assignment consist of a concatenation of string variables and output symbols, with the restriction that in a parallel assignment, a variable can appear at most once across all right-hand-sides.

A DSST that reverses a string uses a single state and one string variable x. In each step, if it reads the symbol a, it executes the assignment $[x := a.x]$ (here '.' is the string concatenation operator). At the end of its computation, x contains the string that is the reverse of the input string. As another example, consider the transformation mapping a string of the form l,m_f to the string f_m_l (here l, m and f respectively denote strings corresponding to some last, middle, and first name, and '_' denotes a single space). A DSST with two variables x and y can implement this transduction. It stores the substring till the comma (*i.e.*, l) in x, and stores the subsequent string (m) till it sees the space in y. It then sets y to $y.\_.x$, resets x to the empty string, and stores the remaining substring (f) in x. It finally outputs the string $x.\_.y$.

Compared to their deterministic counterparts that implement partial functions between strings, nondeterministic finite transducers (NFTs) can implement relations between strings; for example, mapping a string to all its substrings. However, NFTs lack the expressiveness to implement transductions such as (1) mapping a string w to all strings w' obtained by swapping some prefix and suffix of w, (2) mapping w to all strings $w' = p.rev(m).s$, where w has the form $p.m.s$ (where p, m and s are substrings of w), and $rev(m)$ denotes the reverse of m[1].

In addition to all transductions realized by NFTs, NSSTs can implement both of these transductions. The first transduction is implemented by an NSST with string variables x and y. It nondeterministically chooses some prefix p to store in x and some following substring m to store in y. It then sets y to $y.x$, resets x, copies the remaining input s to x, and finally outputs $x.y$. The NSST implementing the second transduction copies some prefix p (chosen nondeterministically) to x, and uses y to compute $rev(m)$ for a nondeterministically chosen m. It then sets x to $x.y$, appends the remaining input s to x, and finally outputs x.

The organization of the paper is as follows: After defining the NSST model in Sec. 2, we characterize the expressive power of NSSTs in Sec. 3. We show that NSSTs are closed under sequential composition, and prove that the expressive power of NSSTs is equivalent to that of nondeterministic MSO-definable transductions. We then compare both NSSTs and ε-NSSTs– an extended model that allows ε-transitions – with classical models such as two-way nondeterminis-

[1] For a given prefix p and a given suffix s, the transformation from $p.m.s$ to $p.rev(m).s$ is well-known as the *inversion operator* for a string representation of a chromosome [10]. Thus, this transduction generates all inversions of a string.

tic generalized sequential machines. In Sec. 4, we explore decision problems for NSSTs and their subclasses. An interesting and robust subclass of NSSTs is that of *functional* NSSTs: these can have multiple runs on a given input string, but the output produced in each run is identical. We show that checking whether an NSST is functional is decidable in PSPACE, and show that checking equivalence of functional NSSTs is PSPACE-COMPLETE.

In [2], the authors show how DSSTs can be viewed as natural models for a class of *sequential* heap-manipulating programs, and reduce the verification problem for such programs to decidable problems for DSSTs. A key application for NSSTs would be in understanding the theoretical limits of verification problems for *concurrent* heap-manipulating programs. Here, we wish to verify if the set of behaviors of the concurrent program (say $m_1 \parallel m_2$) is a subset of the set of behaviors engendered by some sequential execution of the programs m_1 and m_2. For instance, checking *linearizability* of a set of methods often entails such a check. With this goal in mind, we show that checking if the set of outputs of an NSST is contained within the set of outputs generated by a fixed number of DSSTs is decidable with PSPACE complexity.

2 Transducer Models

In this section, we introduce definitions related to transductions, review existing models for specifying transductions, and formally define NSSTs. We observe the following convention: the letters R and W denote transductions, and the letters T, D denote the transducers implementing transductions.

Transduction. A transduction R from a finite input alphabet Σ to a finite output alphabet Γ is a subset of $\Sigma^* \times \Gamma^*$. A transduction is *deterministic* if for every $x \in \Sigma^*$, there is at most one y such that $(x, y) \in R$, *i.e.*, R is a partial function. Otherwise, we say that the transduction is *nondeterministic.* Given an input $x \in \Sigma^*$, we use $R(x)$ to denote the set $\{y \mid (x, y) \in R\}$.

Valuedness. A transduction is called *finitary* if for every $x \in \Sigma^*$, the set $R(x)$ is finite. A transduction R is said to be *k-valued* for a given constant k, if the following holds: $\forall x \in \Sigma^* : |R(x)| \leq k$. A transduction is said to be bounded-valued if there *exists* some constant k such that R is k-valued. A 1-valued transduction is also called a *functional* transduction. A transduction is called *bounded-length* if there exists a constant k such that for all $x \in \Sigma^*$ and for each y in $R(x)$, $|y| \leq k.|x|$. Note that a bounded-valued transduction is obviously finitary, while every bounded-length transduction is also finitary: for a constant k, and a given input x, there are finitely many strings of length $k.|x|$.

Example 2.1. Consider the transduction $R_{cp} \subseteq \{a\}^* \times (\Gamma \cup \{\#\})^*$ defined as the set $\{(a^n, w\#w) | n \geq 0, w \in \Gamma^*, |w| = n\}$. R_{cp} maps an input a^n to a string containing two identical copies of some output string of length n, separated by #. As there are only finitely many strings of length n, R_{cp} is finitary. Further, note that R_{cp} is bounded-length: for an input of length n, the output length is

$(2.n + 1)$. However, R_{cp} is not bounded-valued as the number of output strings for an input of length n is $|\Gamma|^{2n}$.

Example 2.2. Consider the transduction $R_{mult} \subseteq \Sigma^* \times \Sigma^*$ defined as the set $\{(w, w^m) \mid m \geq 1, w \in \Sigma^*\}$. R_{mult} is nonfinitary, as it maps the input string w to the set of *all* multiples of the string w. Clearly, R_{mult} is neither bounded-valued nor bounded-length.

2.1 Background

We assume that each machine in the following discussion has finite input and output alphabets Σ and Γ respectively. Each machine reads words from Σ^*, and outputs words in Γ^*. We use the notation $[\![M]\!]$ to define the semantics of M: for an input string $w \in \Sigma^*$, $[\![M]\!](w)$ defines the set of outputs of M on input w.

Finite Transducers. A *finite sequential transducer* (denoted as NFT) is a finite-state device that has a one-way read-only input tape and a one-way output tape. It scans the input tape from the left to the right, and in each state it reads an input symbol, writes a finite string to the output tape, changes its state, and moves its reading head to the next position on the input tape. In each such step, it can nondeterministically choose its next state and the output string that it writes. The output of an NFT is the string on the output tape if the NFT finishes scanning the input tape in some designated final state. Formally, we define an NFT T as the tuple $(Q, \Sigma, \Gamma, E, q_0, F)$ where Q is a finite nonempty set of states, Σ and Γ are input and output alphabets, E is a set of transitions defined as a finite subset of $Q \times (\Sigma \cup \{\varepsilon\}) \times Q \times \Gamma^*$, the state $q_0 \in Q$ is an initial state, and $F \subseteq Q$ is a set of final states. An NFT may contain transitions in which the NFT writes some output string or changes state without moving its reading head. We call such transitions as ε-transitions, and a NFT in which such moves are disallowed is called a ε-free NFT or a *nondeterministic generalized sequential machine* (NGSM).

Two-way Machines. A *two-way nondeterministic generalized sequential machine* (denoted 2NGSM) is a finite-state device with a two-way read-only input tape and a one-way output tape. In each step, the machine reads an input symbol, changes its state, writes a finite string to its output tape, and moves its input head as per its finite-state control. The input head either does not move (denoted by 0), or is moved to the left $(-)$ or to the right $(+)$. In each such move, it has a finite number of nondeterministic choices for the next state, output string written and the direction for moving its head. The string on the input tape is assumed to be $\vdash w \dashv$, where $\vdash$ and $\dashv$ $(\notin \Sigma)$ are special symbols known as the left and right end-markers. The NGSM halts if it moves right from $\dashv$, or left from $\vdash$. It is possible that the computation of the machine may not terminate; however, only halting runs contribute to the output. The output of the machine is the string on the output tape if the machine terminates in a designated final state. Formally, a 2NGSM is specified as the tuple $(Q, \Sigma, \Gamma, E, q_0, F)$, where Q, Σ, Γ, q_0 and F are defined as for finite transducers, while the set of transitions E is defined as a finite subset of $(Q \times \Sigma \cup \{\vdash, \dashv\} \times Q \times \{-, 0, +\} \times \Gamma^*)$.

Determinism and Valuedness. The definition of determinism is the standard one: no two distinct transitions starting in a state q have the same input symbol symbol a. The deterministic versions of the machines defined above are denoted as DFT, DGSM, and 2DGSM. The transductions implemented by a DFT, a DGSM and a 2DGSM are obviously 1-valued (and thus finitary). Further, these transductions can also be shown to be bounded-length, as on an input of length n, these machines take at most n steps, and each step contributes a constant number of symbols to the output. In an NGSM, there are only finitely many paths corresponding to each input string, and the length of output along each such path grows proportional to that of the input. Thus, the transductions implemented by an NGSM are bounded-length (and thus finitary) as well. However, note that the transductions implemented by an NGSM are not bounded-valued in general. The transductions implemented by NFTs and 2NGSMs are neither finitary nor bounded-length.

MSO-definable String Transductions. String transductions can be described using **M**onadic **S**econd **O**rder logic [4,6]. The input to such a transduction is a string $w = w_1 w_2 \dots w_k$ viewed as an input string graph G_i with $k+1$ vertices $v_0, \dots, v_k$, where the label of each edge (v_i, v_{i+1}) is w_i. The labels for the vertices and the edges appearing in the output string graph are expressed in terms of MSO formulas over a fixed number of copies of G_i. Formally, a deterministic MSO-definable string transduction (DMSOS) W is specified by[2] : (1) a finite *copy set* C, (2) for each $c \in C$, vertex formulas $\varphi^c(x)$, which are MSO formulas with one free first-order variable variable x, and (3) for each $a \in (\Gamma \cup \{\varepsilon\})$, $c, d \in C$, edge formulas $\varphi_a^{c,d}(x, y)$, which are MSO formulas with two free first order variables x and y. The output graph (G_o) is defined as follows: for each vertex x in G_i and $c \in C$, the vertex x^c is in G_o if the vertex formula $\varphi^c(x)$ is *true*, and for all vertices x^c, y^d, there is an edge (x^c, y^d) labeled by a if the edge formula $\varphi_a^{c,d}(x)$ is *true*. Finally, the output is defined as the string formed by the sequence of symbols (other than ε) that are the obtained by reading the labels of edges of the output graph G_o in order. If G_o is not a legal string graph the output is undefined. In case of MSO-definable transductions, we use the notation $[\![W]\!](x)$ synonymously with $W(x)$, where x is an input string.

The nondeterministic variant of MSO-definable string transductions is obtained by adding free second-order variables $\mathcal{X}_1, \dots, \mathcal{X}_n$ that range over sets of vertices. The vertex and edge formulas in this case have these variables as additional arguments. To obtain the output string graph, a (global) valuation of the second-order variables $\mathcal{X}_1, \dots, \mathcal{X}_k$ is chosen, and then the output string graph is obtained as in the deterministic case. As the valuation for $\mathcal{X}_1, \dots, \mathcal{X}_k$ is chosen nondeterministically, there can be multiple possible outputs for a given input string.

[2] We present a definition slightly modified from the one in [6]. For ease of exposition, we omit specification of a domain formula φ_{dom}, as it can be included to be a part of the vertex and edge formulas. We also allow edges in the output graph to be labeled with the empty string ε. It can be shown that such MSO-transductions are equivalent to MSO-transductions that do not make use of ε-labels.

Example 2.3. Consider the transduction $R_{ss} = \{(w, u\#v) | u, v$ are subsequences of $w\}$, where $u, v, w \in \Sigma^*$, and $\# \notin \Sigma$. We define R_{ss} as a NMSOS transduction W by choosing the copy set $C = \{1, 2\}$, and two parameters $\mathcal{X}_1$ and $\mathcal{X}_2$. We use the formula $\mathsf{edge}_a(x, y)$ to denote that there is an edge labeled a between vertices x and y, and the formula $\mathsf{last}(x)$ (resp. $\mathsf{first}(x)$) to denote that x is the last (resp. first) node in the input string graph. We now define the edge formulas: $\varphi_{\#}^{1,2}(x, y, \mathcal{X}_1, \mathcal{X}_2) \equiv \mathsf{last}(x) \wedge \mathsf{first}(y)$, and, $\forall j \in C, \forall a \in \Sigma$: $\varphi_a^{j,j}(x, y, \mathcal{X}_1, \mathcal{X}_2) \equiv (x \in \mathcal{X}_j) \wedge \mathsf{edge}_a(x, y)$, and $\forall j \in C$: $\varphi_{\varepsilon}^{j,j}(x, y, \mathcal{X}_1, \mathcal{X}_2) \equiv (x \notin \mathcal{X}_j) \wedge \bigvee_{a \in \Sigma} \mathsf{edge}_a(x, y)$. Intuitively, the second order variable $\mathcal{X}_1$ is used to guess which positions of the original string contribute to u, while $\mathcal{X}_2$ independently guesses which positions contribute to v. □

Hennie Machines. A two-way machine with a writable input tape and a write-only output tape is defined in similar fashion to a 2NGSM. The instructions of such a machine M are of the form $(q, \sigma/\alpha, q', \gamma)$, which means that in state q, M reads the symbol σ from the tape, overwrites it with the symbol α, transitions to the state q' and writes the finite string γ to the output tape. Here, $\sigma, \alpha \in \Sigma$, where Σ is the tape alphabet, and $\gamma \in \Gamma^*$, which is the output alphabet. A computation of M is called k-visiting, if each position of the tape is visited *at most* k times. A *Hennie machine* is such a two-way machine with the property that there exists a constant k such that every computation of the machine is k-visiting. The classes of string transductions realized by nondeterministic and deterministic Hennie machines are denoted by NHM and DHM, respectively.

Example 2.4. Consider the transduction R_{cp} of Ex. 2.1. In [6], it is shown that R_{cp} can be realized by an NHM moving in two left-to-right passes over the tape. In the first pass, it nondeterministically overwrites the input string a^n with some output string w, and also writes w to the output. It then scans backward to the beginning of the (overwritten) input, writes # to the output, and copies the contents of the input tape to the output a second time. The NHM is 3-visit. □

2.2 Streaming String Transducers

Deterministic streaming string transducers (DSST) have been proposed as an effective machine model for MSO-definable string transductions [2,1]. A DSST makes a *single* left-to-right pass over the input string mapping it to the output string, while using a fixed set of string variables that can store strings over the output alphabet. In each step, a DSST reads an input symbol a, changes its state and updates all its string variables simultaneously using a *copyless* assignment. The right-hand-side (RHS) of any assignment to a string variable is a concatenation of output symbols and some string variables, with the restriction that in a given parallel assignment, any string variable can appear at most once across all the right-hand-sides. For instance, let $X = \{x, y\}$ be the set of string variables, and let $\alpha, \beta, \gamma \in \Gamma^*$ be strings of output symbols. Then, the update $(x, y) = (\alpha.x.\beta.y.\gamma,\ \varepsilon)$ is a copyless assignment, as it contains only one occurrence of x and y each. On the other hand, the assignment $(x, y) = (x.y,\ y.\alpha)$ is not copyless as the variable y appears in the RHS twice. The two main extensions to the DGSM model are that (1) DSSTs are not constrained to add output symbols only at the end of the tape, and (2) they can compute the output in multiple chunks that can be extended and concatenated (but not duplicated) as needed.

We now present a nondeterministic extension of DSSTs: the class of *nondeterministic streaming string transducers* (NSSTs). An NSST is defined as the tuple $(Q, \Sigma, \Gamma, X, E, q_0, F)$, where Q is a finite nonempty set of states, Σ and Γ are respectively the input and output alphabets, X is a finite set of *string variables* with A as a set of *copyless assignments* to variables in X (mappings α from X to $(X \cup \Gamma)^*$ such that for any $x \in X$, x appears at most once in the set $\{\alpha(y) \mid y \in X\}$), E is a set of transitions that is a finite subset of $(Q \times \Sigma \times A \times Q)$, q_0 is an initial state, and $F : Q \to (X \cup \Gamma)^*$ is a partial output function such that for every $q \in Q$ and $x \in X$, there is at most one occurrence of x in $F(q)$.

Semantics. We first make an observation: copyless assignments are closed under sequential composition, *i.e.*, if α_1 and α_2 are copyless assignments, the assignment $\alpha_1 \circ \alpha_2$ is also copyless. We define the semantics of an NSST in terms of the *summary* of a computation of the NSST. For an NSST starting in state q and processing the input string w, the summary is a set of pairs of the form (α, q'), where α represents the effect of a sequence of copyless assignments to the string variables, and q' is a possible next state. Let id denote the identity assignment, *i.e.*, it maps each x to x. We inductively define the summary Δ as a mapping from $Q \times \Sigma^*$ to $2^{A \times Q}$, where: $\Delta(q, \varepsilon) = \{(\mathsf{id}, q)\}$, and $\Delta(q, w.a) = \{(\alpha_w \circ \alpha, q') \mid \exists q_1,\ \text{s.t. } (\alpha_w, q_1) \in \Delta(q, w) \ \wedge\ (q_1, a, \alpha, q') \in E\}$.

A valuation of a string variable is defined as the function $\nu : X \to \Gamma^*$, which maps each variable to some string in Γ^*. Such a valuation is extended to map strings in $(X \cup \Gamma)^*$ to Γ^* in a natural fashion. Let ν_ε be the initial valuation, where for all $x \in X$, $\nu_\varepsilon(x) = \varepsilon$. We define the semantics of an NSST T as the set of outputs $[\![T]\!](w)$ generated by T on input string w:

$$[\![T]\!](w) = \{\nu_\varepsilon(\alpha_w \circ F(q)) \mid (\alpha_w, q) \in \Delta(q_0, w) \text{ and } F(q) \text{ is defined}\}.$$

We now illustrate the model using a couple of examples.

Example 2.5. The NSST T_{ss} implementing the transduction R_{ss} from Ex. 2.3 has a single state q and uses two string variables x and y. For each input symbol a, it nondeterministically decides to append a to x or y or both or none. Formally, the transitions of T_{ss} are of the form $(q, a, (x, y) := (x.\gamma_1, y.\gamma_2), q)$, where $\gamma_1, \gamma_2 \in \{\varepsilon, a\}$. The output function is defined as $F(q) = x.\#.y$. □

Example 2.6. The NSST T_{cp} implementing the transduction R_{cp} from Ex. 2.1 has a single state q and uses two string variables x and y. For each input symbol it nondeterministically chooses an output symbol $\gamma \in \{a, b\}$ and appends it to both x and y, *i.e.*, the set of transitions $E = \{(q, a, (x, y) := (x.\gamma, y.\gamma), q) \mid \gamma \in \{a, b\}\}$. The output function is defined as $F(q) = x.\#.y$. □

Extensions and Subclasses. An NSST with ε-transitions (ε-NSST) can update its string variables and change state without consuming an input symbol, *i.e.* E is a finite subset of $(Q \times (\Sigma \cup \{\varepsilon\}) \times A \times Q)$. As we will see in Sec. 3, an ε-NSST is more powerful than an NSST as it can implement nonfinitary transductions.

Lastly, we note some important subclasses of NSSTs, based on their valuedness. An NSST T is said to be *k-valued*, if the underlying transduction that it implements is k-valued, *i.e.* for a given constant k, $\forall w \in \Sigma^*$, $|[\![T]\!](w)| \leq k$. If there exists some constant k such that T is k-valued, we say that T is *bounded-valued*, and if $k = 1$, we say that the NSST is *functional.* In other words, all runs of a functional NSST on the same input string, which end in a state q where $F(q)$ is defined, produce the same output string. Thus, the transduction realized by such an NSST is a partial function. Functional NSSTs are an interesting subclass of NSSTs, as their decision problems have good algorithmic properties.

3 Expressiveness Results

In this section, we first show that NSSTs are closed under sequential composition, and they implement bounded-length transductions. We then compare NSSTs with the class of NMSOS-transductions, and show that they have the same expressive power. Lastly, we compare the expressive power of ε-NSSTs with that of 2NGSMs.

3.1 Properties of NSSTs

Sequential Composition of DSST*s.* Let (Σ, Γ)-DSST be the shorthand for a DSST that has Σ and Γ as its input and output alphabets respectively. Let T_1 be a (Σ_1, Σ_2)-DSST, with the set of states Q_1, and the set of string variables X_1, and let T_2 be a (Σ_2, Σ_3)-DSST with the set of states Q_2 and the set of string variables X_2. We review the construction from [1] to construct the (Σ_1, Σ_3)-DSST $T_s = T_1 \circ T_2$, *i.e.*, T_s is the sequential composition of T_1 and T_2. Let the set of states and the set of string variables of T_s be Q_s and X_s respectively. When computing the sequential composition, the input to T_2 is the output of T_1. Hence, the contents of any string variable $x \in X_1$ are a possible input to T_2. T_s encodes the sequential composition as follows: (1) it simulates the action of T_1 by recording the state of T_1 in its state, and, (2) at each step during the simulation, it records maps f and g in its state, and uses these in conjunction with its string variables to record the summary of T_2 starting in each possible state $r \in Q_2$, on the contents of each string variable $x \in X_1$.

We now formally define these maps. Recall that $\nu(x)$ denotes the valuation of the variable x. We define the map f from $Q_2 \times X_1$ to Q_2, such that $f(r, x) = r'$ if $\Delta_2(r, \nu(x)) = (\alpha', r')$. For each state $r \in Q_2$ and each string variable $x \in X_1$, T_s records the map $f(r, x)$ in its state. A copyless assignment α in T_2 maps each $y \in X_2$ to some concatenation of symbols from Σ_3 (the output language of T_2) and X_2. Thus the *shape* of α is some string of the form $s_1.y_1.s_2.y_2.s_3$, where the s_k's are some finite strings in Σ_3^* and y_j's are variables in X_2. T_s uses its string variables $z \in X_s$ to store these s_k chunks in such a manner that (1) no two variables in X_s appear consecutively, and (2) none of the variables in X_s or X_2 are repeated (in keeping with the copyless restriction on assignments). The shape of each $\alpha(y)$ expression is then a string in $(X_2 \cup X_s)^*$. We define the map g from $(Q_2 \times X_1 \times X_2)$ to $(X_2 \cup X_s)^*$, such that for each state $r \in Q_2$, and for each $x \in X_1$, $g(r, x, y)$ is the string that describes the shape of $\alpha(y)$. Thus, a state of T_s is the tuple $(q, f, g) \in (Q_1 \times [(Q_2 {\times} X_1) \to Q_2] \times [(Q_2 {\times} X_1 {\times} X_2) \to (X_2 \cup X_s)^*]$.

It can be shown that the maximum number of z-variables required for each state $r \in Q_2$ and each string variable $x \in X_1$ is $2.|X_2|$. Thus, the total number of string variables required for T_s is $2.|X_2|.|Q_2|.|X_1|$. Further, as the z and y variables strictly interleave and do not repeat, the number of possible shapes is finite. This is key to ensure that the size of T_s (number of states and string variables) is finite.

Example 3.1. Suppose that T_2 contains the transitions:

$t_1 : (r_0, a,\ (y_1, y_2) {:=} (c.y_1.d,\ e.y_2),\ r_1)$ $\qquad t_2 : (r_0, b,\ (y_1, y_2) {:=} (y_1.y_2.g,\ \varepsilon),\ r_1)$
$t_3 : (r_1, b,\ (y_1, y_2) {:=} (g.y_1.h,\ e.y_2.d),\ r_0)$

The state of T_s when T_1 is in state q and $\nu(x_1) = abb$ encodes the effect of T_2 executing transitions t_1, t_3 and t_2 (in that order), and is of the form (q, f, g). Here $f(r_0, x_1) = r_1$, $g(r_0, x_1, y_1) = z_1.y_1.z_2.y_2.z_3$, and $g(r_0, x_1, y_2) = z_4$. The valuation for the variables of T_s is: $\nu(z_1) = gc$, $\nu(z_2) = dhee$, $\nu(z_3) = dg$, and $\nu(z_4) = \varepsilon$. Similarly, if $\nu(x_2) = bbb$, we encode the effect of T_2 executing t_2, t_3, and t_2, and define $f(r_0, x_2) = r_0$, $g(r_0, x_2, y_1) = v_1.y_1.v_2.y_2.v_3$, and $g(r_0, x_2, y_2) = v_4$. The valuation of the variables is: $\nu(v_1) = g$, $\nu(v_2) = \varepsilon$, $\nu(v_3) = ghedg$, and $\nu(v_4) = \varepsilon$. □

Invariant. Suppose $\Delta_1(q_0^1, w)$ is (α, q) (*i.e.*, the summary of T_1 after reading w, beginning in its initial state q_0^1). Suppose the state of T_s after reading the input word w is (q_s, f, g) and the valuation of its string variables is ν_w. Then the following invariant holds: if $\Delta_1(q_0^1, w) = (\alpha, q)$, then $q_s = q$, and, if $\Delta_2(r, \nu_1(x)) = (\beta, r')$, then $f(r, x) = r'$, and for each y, $\nu_w(g(r, x, y)) = \beta(y)$.

Thus, to model the effect of a transition of T_1 of the form (q, a, α, q'), we add an edge of the form $\big((q, f, g),\ a, \alpha',\ (q', f', g')\big)$, where the maps f', g', and the assignment α' to the variables of T_s are defined in a manner that satisfies the above invariant.

Example 3.2. Consider the state in Ex. 3.1. Now suppose that T_1 executes the transition $(q, a,\ (x_1, x_2) := (x_2.x_1, \varepsilon),\ q')$. Suppose the resulting state of T_s is (q', f', g'). We only explain how the updates to the variable x_1 are encoded. To model the assignment $x_1 := x_2.x_1$, we define $f'(r_0, x_1) = f(f(r_0, x_2), x_1)$ which evaluates to r_1 (as seen from Ex. 3.1). Similarly, the map $g'(r_0, x_1, y_1)$ is obtained by sequentially composing the maps $g(r_0, x_2, y_1)$ and $g(r_0, x_1, y_1)$ from Ex. 3.1. The result of the composition yields the expression $\underline{z_1.v_1}.y_1.v_2.y_2.\underline{v_3.z_2.v_4.z_3}$, which is then compressed by combining adjacent z_i and v_i variables. Thus, $g'(r_0, x_1, y_1)$ is defined to be $z_1.y_1.v_2.y_2.z_3$ with the copyless assignments $z_1 := z_1.v_1$ and $z_3 := v_3.z_2.v_4.z_3$ (and no change to the other variables). Similarly, we can compute $g'(r_0, x_1, y_2) = z_4$ by looking at the maps $g(r_0, x_2, y_2)$ and $g(r_0, x_1, y_2)$. □

Sequential Composition of NSSTs. We now show that NSSTs are closed under sequential composition.

Theorem 3.1. *Let T_1 be a (Σ_1, Γ_1)-*NSST*, T_2 be a (Σ_2, Σ_3)-*NSST*, then one can effectively construct T_s, a (Σ_1, Σ_3)-*NSST *such that for all strings $w \in \Sigma_1^*$, we have* $[\![T_s]\!](w) = \bigcup\limits_{u \in [\![T_1]\!](w)} [\![T_2]\!](u)$.

Proof. The construction for the sequential composition of DSSTs extends to the case of NSSTs in a straightforward fashion. We show how to compute the NSST $T_s = T_1 \circ T_2$. A run of T_s simulates a run of T_1 on the input string w, and simultaneously summarizes the action of some transition of T_2 on the contents of the string variables of T_1. This summary is stored with the help of the maps f and g that are part of its state, and its string variables. Thus, a state of T_s is a finite subset of $(Q_1 \times [f : Q_2 \times X_1 \to Q_2] \times [g : Q_2 \times X_1 \times X_2 \to (X_2 \cup X_s)^*])$.

Unlike the deterministic case where the computation summary is a pair, the summary $\Delta_2(r, \nu(x_1))$ of T_2 is the set of pairs $\{(\beta_1, r_1), (\beta_2, r_2), \ldots\}$. To accommodate this, we define an invariant that all states of T_s satisfy: If T_s is in the state (q, f, g) upon reading input string w, with the valuation ν_w for its string variables, then for some α, $(q, \alpha) \in \Delta_1(q_0^1, w)$, and, if $f(r, x) = r'$, and $\nu_w(g(r, x, y)) = \beta$, then $(\beta, r') \in \Delta_2(r, \nu_1(x))$. For a transition of the form (q, a, α, q') in T_1, we add a transition $((q, f, g), a, \alpha', (q', f', g'))$ to T_s, such that the copyless assignment α', and the maps f' and g' satisfy the invariant (for the input $w.a$). We finally remark that the number of variables of T_s is $2.|X_2|.|Q_1|.|X_1|$, (the same as for DSSTs) and as the maps f, g are finite, the number of states required for T_s is also finite. □

Theorem 3.2. *The transduction implemented by an* NSST *T is bounded-length,* i.e.*, there exists a constant c such that $\forall w \in \Sigma^*$, if $u \in [\![T]\!](w)$, then $|u| \leq c.|w|$.*

Proof. We use the notation $|\alpha(x)|$ to denote the number of output symbols in the expression $\alpha(x)$. We define $|\alpha| = \sum_x (|\alpha(x)|)$. Now let $\ell_{max} = \max_E(|\alpha|)$, *i.e.*, the maximum number of output symbols added by *any* transition in T. Let u_w be the shorthand for some output of T upon reading the string w. We prove the result by induction on the length of the input. The base case is valid; since if $|w| = 0$, the length of u_w is zero. The inductive hypothesis is stated as follows: for input w, if $u_w \in [\![T]\!](w)$, then $|u_w| \leq \ell_{max}.|w|$. Now consider an input of the form $w.a$, and let q be the state of T after reading w. Let E' be the subset of E containing transitions of the form (q, a, β, q'). The number of output symbols added by any transition in E' is bounded by $\max_{E'} |\beta|$. However, by definition, $\ell_{max} \geq \max_{E'} |\beta|$. Thus, for any $u_{w.a} \in [\![T]\!](w.a)$, $|u_{w.a}| \leq \ell_{max}.|w| + \ell_{max}$. Thus, $|u_{w.a}| \leq \ell_{max}.|w.a|$, since $|w.a| = |w| + 1$. □

3.2 Comparison with Existing Models

For classes of transductions C_1 and C_2, we denote by $C_1 \subseteq C_2$ the fact that for every transducer T of type C_1, there is a transducer T' of type C_2 such that

$[\![T]\!] = [\![T']\!]$. We say that C_1 is *equi-expressive* to C_2 if $C_1 \subseteq C_2$ and $C_2 \subseteq C_1$. The following relationships are known: DMSOS = 2DGSM [6], DMSOS = DSST [1], NMSOS $\not\subseteq$ 2NGSM and 2NGSM $\not\subseteq$ NMSOS [6]. We now compare the expressive power of NSSTs with respect to that of existing models/machines for nondeterministic transductions such as NMSOS-transductions, NGSM, and 2NGSM.

Expressive power of NGSM *vs.* NSST. A NGSM is a NSST with a single string variable (say x), and every transition is of the form $(q, a, x := x.\gamma, q')$, where $a \in \Sigma$, and $\gamma \in \Gamma^*$. Thus, NGSM $\subseteq$ NSST.

Expressive power of NMSOS *vs.* NSST. We now show that the the class of NSSTs is equi-expressive to NMSOS-transductions. We first show that for every NSST T we can define a NMSOS transduction W such that $[\![T]\!] = [\![W]\!]$. In [1], the authors shows how a sequence of states and copyless assignments of a DSST can be encoded in the copy set of a DMSOS-transduction. As a run of an NSST T is such a sequence of states and assignments, it can also be encoded in the copy set of a DMSOS-transduction. The key ideas in this construction are: (1) Let $n_x = \max_E |\alpha(x)|$ (*i.e.* the maximum number of output symbols added by any assignment to the string variable x). Then, assignments to x can be encoded within a copy set that has at most $n_x + 2$ copies. In any step, for each string variable x, we maintain a designated start vertex x^b and a designated end vertex x^e for x, and the valuation of x ($\nu(x)$) is obtained from the string graph corresponding to the path beginning at x^b and ending in x^e. This graph is linear and connected by construction. (2) Concatenation of string variables, say $x := x.y$, is encoded by adding an edge between x^e and y^b. (3) A state of the streaming transducer can be encoded by an MSO-formula on the input graph (these formulas essentially capture the next-state function). (4) The output is defined if the output graph – the sequence edges beginning in the start vertex for the output and concluding in the end vertex – is a string graph.

Lemma 3.1. NSST $\subseteq$ NMSOS.

Proof. Recall that a NMSOS transducer W has a finite copy set C, and a finite set of *parameters* $\mathcal{X} = \{\mathcal{X}_1, \ldots, \mathcal{X}_m\}$, *i.e.*, second order variables ranging over sets of vertices. This proof combines two main ideas: (a) a given run of a NSST T can be encoded in the finite copy set of a NMSOS transducer W, and, (b) each run of T can be *guessed* by W by guessing a valuation for its parameters. The construction for part (a) is as discussed, for details see [1]. We focus on part (b).

As T is nondeterministic, there may be multiple transitions on a given input symbol that are enabled in a state q. However, as the assignment in each transition is a copyless assignment over the same set of string variables, the copy set required is the same as that for encoding a single run. However, different runs may label an edge in the copy set with different labels.

To make the NMSOS transduction W mimic a single run of T, we add parameters that consistently choose among the different labels available for a given set of edges. Let the maximum number of transitions of T from any state q on any input symbol a be d. Then, W uses the set of parameters $\mathcal{X} = \{\mathcal{X}_1, \ldots, \mathcal{X}_d\}$.

Intuitively, the j^{th} nondeterministic transition of T is chosen by W if the corresponding vertex of the input graph is present in the valuation chosen for $\mathcal{X}_j$ by W. Choosing the j^{th} transition fixes a particular assignment α_j to each string variable. Now for each edge formula $\varphi_a^{i,j}(x,y)$ that encodes the action of α_j, we define $\varphi_a^{i,j}(x,y,\mathcal{X}_1,\ldots,\mathcal{X}_d) = \varphi_a^{i,j}(x,y) \wedge (x \in \mathcal{X}_j)$. We can similarly encode the next state function in terms of the parameters.

We also assume that each of the vertex and edge formulas is conjoined with a domain formula that specifies that for any vertex $x \in G_i$, x is contained in at most one $\mathcal{X}_j$ in a given valuation, and the union of all $\mathcal{X}_j$ covers the set of all vertices of the input graph[3]. In other words, using the parameters in $\mathcal{X}$, W first *guesses* a possible sequence of choices at each state in T, and then encodes that sequence using the copy set and next-state relations (which are MSO-formulas as in the deterministic case). It is obvious that the number of parameters required is at most d, and thus finite. $\square$

In Lemma 3.2, we show that for every NMSOS W, we can obtain a NSST T such that $[\![W]\!] = [\![T]\!]$. The key idea is that a NMSOS transduction is essentially the sequential composition of a string relabeling with a DMSOS transduction, both of which are expressible as NSSTs.

Lemma 3.2. NMSOS $\subseteq$ NSST.

Proof. We recall from [6], that any NMSOS transduction W can be written as the sequential composition of a string relabeling and a DMSOS transduction. A string relabeling nondeterministically substitutes each symbol in w with a finite string in Γ^*. A string relabeling can be easily encoded by an NSST T_1 that has one state q, one string variable x, and for all $a \in \Sigma$, its transitions are of the form $(q, a, x := x.\gamma, q)$, where γ is a finite string in Γ^*. From [1], we know that for every DMSOS W, there exists a DSST T_2 that implements the same transduction. Every DSST is an NSST, and from Theorem 3.1 we know that NSSTs are closed under sequential composition; thus, $T = T_1 \circ T_2$ is a NSST s.t. $[\![W]\!] = [\![T]\!]$. $\square$

From Lemmas 3.1 and 3.2, we can conclude that NSSTs are equi-expressive to NMSOS-transductions, formalized in Theorem 3.3 below.

Theorem 3.3. NSST = NMSOS.

Comparing ε-NSSTs with existing models. We now investigate the expressive power of ε-NSSTs, *i.e.*, NSSTs that are allowed transitions without consuming any input symbol. The expressive power of ε-NSSTs is greater than that of NSSTs: consider an ε-NSST in which there is a state q reachable from the initial state q_0 on the input string w, and q has an ε-loop (a cycle with only ε-transitions) that adds some symbols to the output. The set of outputs for any input with w as its prefix is infinite, as the ε-NSST can cycle through the ε-loop an infinite number of times. Thus, the transduction realized by a general ε-NSST is nonfinitary.

[3] It further ensures the technicality that if the number of outgoing transitions of T corresponding to a particular state and input symbol is d', where $d' < d$, then in this case, all $\mathcal{X}_j$ where $j > d'$ always evaluate to the empty set.

If we restrict ε-NSSTs so that they are ε-loop-free, then we can show that the class of such transducers coincides with NSSTs. The proof uses a construction similar to the one used for eliminating ε-transitions in NFAs: a finite sequence of transitions of the form $(q_1, \varepsilon, \alpha_1, q_2)$, ..., $(q_n, \varepsilon, \alpha_n, q_{n+1})$, (q_{n+1}, a, β, q') can be replaced by the single transition: $(q_1, a, \alpha_1 \circ \ldots \circ \alpha_n \circ \beta, q')$. Since the transducer is ε-loop-free, an infinite sequence with ε-transitions does not exist.

We now compare ε-NSSTs with 2NGSMs. Though both classes can implement nonfinitary transductions, we show that their expressive power is incomparable. We first note that NMSOS $\not\subseteq$ 2NGSMs [6], and as NSST = NMSOS (Theorem 3.3), and NSST $\subseteq$ ε-NSST, we have that ε-NSST $\not\subseteq$ 2NGSM, *i.e.*, there are transductions implemented by NSSTs and ε-NSSTs that cannot be implemented by 2NGSMs. We further show that 2NGSM $\not\subseteq$ ε-NSST, by an example transduction that can be realized by a 2NGSM but cannot be realized by an ε-NSST. A similar example is used in [6] to show the incomparability of NMSOS with 2NGSM; however, the argument used therein is not applicable in our setting, as the transductions realized by both 2NGSMs and ε-NSSTs are nonfinitary.

Theorem 3.4. *The transduction $R_{mult} = \{(w, w^m) \mid m \geq 1, w \in \Sigma^*\}$ can be realized by a* 2NGSM*, but cannot be realized by an* ε-NSST.

Proof. The 2NGSM implementing the transduction R_{mult} has three states q, q_b and *halt*. It starts in state q scanning the input $\vdash w \dashv$ from left to right, simultaneously copying it to the output. Upon reaching $\dashv$, it either transitions to the state *halt* and moves right from $\dashv$, or transitions to q_b and scans the input from right to left without producing any output. Once it reaches $\vdash$, it transitions back to the state q.

We note that R_{mult} is a nonfinitary transduction. If there is a ε-NSST T_ε that implements it, it must have a ε-loop on some state q that is reachable from the initial state q_0. We denote the summary of T_ε for the ε-loop by $\Delta(q, \varepsilon\text{-loop})$. Suppose $\Delta(q, \varepsilon\text{-loop}) = (\beta, q)$. Let $u, v \in \Sigma^*$ be substrings of w such that $u.v = w$, u is the input that drives T_ε from q_0 to q, and v is the input that drives T_ε from q to some state q' (possibly the same), where $F(q')$ is defined. Now let $\Delta(q_0, u) = (\alpha_1, q)$, $\Delta(q, v) = (\alpha_2, q')$. Then $\Delta(q, w)$ is the set $\{(\alpha_1 \circ \alpha_2, q'), (\alpha_1 \circ \beta \circ \alpha_2, q'), (\alpha_1 \circ \beta \circ \beta \circ \alpha_2, q'), \ldots\}$.

Suppose the number of symbols added to the output by α_1 is t_1, by β is t, and by α_2 is t_2. Then the lengths of the outputs of T_ε in state q' are $t_1 + t_2$, $t_1 + t + t_2$, $t_1 + 2.t + t_2$, and so on. However, R_{mult} requires that for *any* input w, each of the outputs have length that is a multiple of $|w|$. This is possible only if $t_1 = t_2 = 0$ and $t = |w|$. However, t is a fixed number for a given ε-loop, and independent of the input w. Thus, once we fix T_ε, it will produce incorrect outputs for any w where $|w| \neq t$. $\square$

Functional NSST*s.* It is clear that the expressive power of functional NSSTs is identical to that of the class DSST, as a transduction realized by a functional NSST is single-valued and MSO-definable, *i.e.*, a DMSOS transduction, which is the same as that for a DSST.

Theorem 3.5. *Functional* NSST = DSST.

It is also easy to show that functional NSSTs are closed under sequential composition. The results follow from the fact that NSSTs are closed under sequential composition, and single-valued transductions are closed under sequential composition. Finally, the expressivity relations between the different transducer models are as shown in Fig. 1.

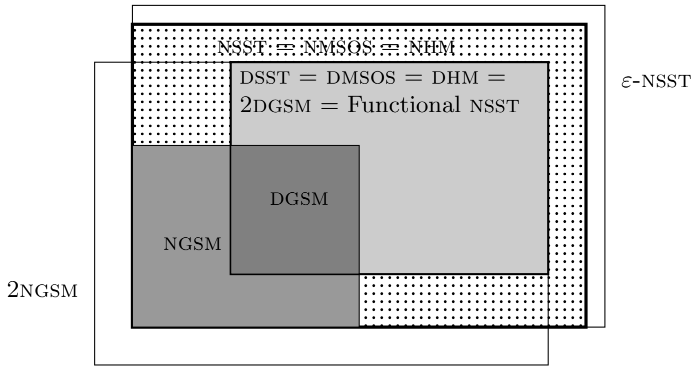

Fig. 1. Expressivity Results

4 Decision Problems

In this section, we discuss decision problems for NSSTs, with special heed to the class of *functional* NSSTs. In what follows, we make use of 1*-reversal-bounded m-counter machines*. Such a machine is an NFA augmented with m counters, where each counter can make at most one reversal (*i.e.*, in any computation each counter can go from increasing to decreasing mode at most once). It has been shown that the nonemptiness problem for such machines is in NLOGSPACE in the size of the machine's description. For further details on reversal-bounded counter machines, please see [8].

Checking functionality. We now address the problem of checking if an arbitrary NSST is functional. We show that this problem is decidable, and present an algorithm with PSPACE complexity. To a large extent, our algorithm draws upon techniques developed for checking equivalence of DSSTs discussed in [2]. We use the following intuition: An NSST is not functional if there exists some input string w such that there are outputs $y_1, y_2 \in [\![T]\!](w)$ and $y_1 \neq y_2$. We reduce the problem of finding such distinct outputs to the nonemptiness problem for a 1-reversal-bounded 2-counter machine.

Theorem 4.1. *For an* NSST T*, checking if* T *is functional is in* PSPACE.

Proof. We construct a 1-reversal-bounded 2-counter machine M that for a given input string w simultaneously simulates two paths in T corresponding to w, and checks if the resulting outputs u_1 and u_2 differ because of either of these conditions: (1) $|u_1| \neq |u_2|$, *i.e.*, the lengths of u_1 and u_2 are different, (2) there exists a position p such that the p^{th} symbol in the string u_1 (denoted $u_1[p]$) differs from $u_2[p]$. M nondeterministically chooses one of the above conditions to check. Checking the first condition is easier and uses similar techniques as for checking the second condition; thus, we focus on the latter. We now explain how the second condition is checked.

M precisely keeps track of the state of T in each of the two paths being simulated in its state. At the beginning of the computation, M pre-loads its counters c_1 and c_2 with the same number; this number is the guess for the position p in which the outputs might differ. For path i ($i = 1, 2$), at some step in the simulation, M nondeterministically guesses that the symbol γ_i added by a copyless assignment in this step appears at the p^{th} position in the corresponding output u_i, and remembers γ_i in its state. It also guesses where each string variable appears with respect to p in this output. Concretely, we define the mapping ϱ (called the string variable class) from X to an element of $\zeta = \{L, R, C, N\}$. Here, $\varrho(x)$ is L, R, C or N, depending on whether x appears to the left of p, to the right of p, contains p, or does not contribute to the output respectively. For each output symbol added to the left of p in the first path, M decrements the counter c_1, and for each output symbol added to the left of p in the second path, M decrements c_2.

Let the symbol $\perp$ denote that M has not made its guess for the symbol in position p. Let $\Gamma' = \Gamma \cup \{\perp\}$. Formally, the set of states S of M is a subset of $(Q \times Q \times (\Gamma' \times \zeta^X) \times (\Gamma' \times \zeta^X))$. To elaborate, the meaning of some state $(q, q', \gamma_1, \varrho_1, \perp, \varrho_2)$ in M is: T reaches states q and q' from its initial state on the same input string; M has guessed that the symbol at $u_1[p]$ is γ_1, M has not yet guessed the symbol at $u_2[p]$, and the mappings from the string variables to classes along the two paths are ϱ_1 and ϱ_2. Except for the initial transitions where M pre-loads c_1 and c_2, all transitions of M decrement the counters by some non-negative amount. Thus, the counters of M are single-reversal.

The transitions of M are defined in a way that ensures that M consistently guesses the classes for each string variable. We illustrate this with two examples. Suppose $(q, a, (x, y) := (x.d.y,\ b), r)$ and (q', a, α, r') are two transitions in T.

Suppose in state r, M guesses that the symbol d added to the string variable x appears at $u_1[p]$. In order to maintain consistency, in state q, M must have assigned x to the class L, and y to the class R. In other words, we add a transition from the state $(q, q', \perp, [\varrho_1(x) = L, \varrho_1(y) = R], \gamma_2, \varrho_2)$ to the state $(r, r', d, [\varrho_1'(x) = C, \varrho_1'(y)], \gamma_2, \varrho_2')$, with no change to the counters. The mapping $\varrho_1'(y)$ maps y to any string class in ζ other than C, and the mappings ϱ_2 and ϱ_2' similarly ensure consistency with the assignment α along the second path.

Now consider the case where in state r, M guesses that the contents of x (which are now $x.d.y$) appear to the left of the position p in u_1. In order to maintain consistency, in state q, M must have assigned x and y to the class L. Thus, we add a transition between the states $(q, q', \gamma_1, [\varrho_1(x) = L, \varrho_1(y) = L], \gamma_2, \varrho_2)$ and $(r, r', \gamma_1, [\varrho_1'(x) = L, \varrho_1'(y)], \gamma_2, \varrho_2')$ in M. As the assignment adds one letter to the left of p, M decrements c_1 by one. Here, $\varrho'_1(y)$ maps y to any string class in ζ, and γ_1 is propagated unchanged.

Initially, no string variable is guessed to be in the class C, and at each step at most one string variable can be guessed to be in the class C. At the end of a computation, if $c_1 = c_2 = 0$, then it means that the symbols γ_1 and γ_2 in the state of M are symbols in the same position p in u_1 and u_2, *i.e.*, $\gamma_1 = u_1[p]$ and $\gamma_2 = u_2[p]$. We define a state $(q, q', \gamma_1, \varrho_1, \gamma_2, \varrho_2)$ of M as *accepting* if (1) $c_1 = c_2 = 0$, (2) $\gamma_1 \neq \gamma_2$, (3) if for some $x \in X$, if $\varrho_1(x) = C$ then $F(q) = x$ [4], and (4) if for some $y \in X$, if $\varrho_2(y) = C$ then $F(q') = y$. M thus accepts only those paths that are witnesses to an input string that leads to two distinct output strings. In other words, T is functional iff M is empty. The problem of checking nonemptiness of 1-reversal bounded m-counter machines is in NLOGSPACE in the size of M. As the total number of states of M is $(|Q|^2.(|\Gamma| + 1)^2.|\zeta|^{2.|X|})$, *i.e.*, exponential in the size of X, the above technique gives us a procedure that is in PSPACE. □

Equivalence Checking. We now discuss the equivalence problem for NSSTs. Given two (Σ, Γ)-NSSTs T_1 and T_2, the equivalence problem is to determine if for all inputs $w \in \Sigma^*$, $[\![T_1]\!](w) = [\![T_2]\!](w)$. We start with a negative result that shows that for the general case of NSSTs, checking equivalence is undecidable.

Theorem 4.2. *The equivalence problem for* NSST*s is undecidable.*

Proof. As mentioned in Section 3.2, an NGSM is a special case of a NSST. The result follows from the fact that the equivalence problem for NGSMs is known to be undecidable by a reduction from the Post Correspondence problem [7]. □

If we restrict our attention to functional NSSTs, the equivalence problem is decidable and a minor modification of the algorithm for checking equivalence of DSSTs in [2] can be directly applied to give a PSPACE complexity procedure. Moreover, we show that the problem is in fact PSPACE-COMPLETE.

Theorem 4.3. *The equivalence problem for the class of functional* NSST*s is* PSPACE-COMPLETE.

Proof. Membership in PSPACE can be shown by using an algorithm similar to the one for checking equivalence of DSSTs [2].

We further show that the problem is PSPACE-HARD by a reduction from the equivalence problem for NFAs. We can encode an NFA as an NSST T which uses only one string variable x and never changes the value of x from the initial value of ε. The states of the NFA are the same as the states of T. We define the output function F of T such that for each final state q_f of the NFA, in the corresponding

[4] Here, we assume that, for all q, $F(q)$ is defined to be some string variable x. An NSST in which $F(q)$ is an arbitrary expression $\alpha \in (X \cup \Gamma)^*$ can be converted to an equivalent NSST of this form as follows: add a new state $q_f \notin Q$; for all q where $F(q) = \alpha$, add a transition from q to q_f with the copyless assignment $x := \alpha$ on that transition; finally, define $F(q_f) = x$, and $F(q)$ as undefined.

state of T, $F(q_f) = x$, and for all other states q of the NFA, $F(q)$ is undefined. Clearly, $[\![T]\!](w) = \{\varepsilon\}$ implies that w is accepted by the NFA. Observe that the NSST defined thus is functional: for all inputs w, $[\![T]\!](w)$ is either empty or contains exactly one symbol. If two such NSSTs T_1 and T_2 are equivalent, it means that either (1) there exist runs of T_1 and T_2 in which both read the same input string w, and produce the same output ε, or (2) all runs of T_1 and T_2 reading the same input string w reach some states q and q' respectively, such that $F_1(q)$ and $F_2(q')$ are undefined. In other words, two such NSSTs are equivalent iff the encoded NFAs are equivalent. □

Containment. The problem of checking if the transduction realized by an NSST is contained within the outputs of a finite set of DSSTs is of particular interest in applications such as the verification of concurrent heap-manipulating methods. Given an NSST T, and DSSTs D_1,...,D_n, we show that the problem of checking if the set of outputs of T is contained within the set of outputs of $D_1, \ldots, D_n$ is in PSPACE by reducing the problem to the nonemptiness problem for 1-reversal m-counter machines.

Theorem 4.4. *Suppose $D_1, \ldots, D_n$ are (Σ, Γ)-DSSTs, and T is a (Σ, Γ)-NSST. Checking if $[\![T]\!] \subseteq \bigcup_{j=1}^{n} [\![D_j]\!](w)$ is in PSPACE.*

Proof. We observe that $[\![T]\!] \not\subseteq \bigcup_{j=1}^{n} [\![D_j]\!](w)$, if there is a run of T on the input string w that produces an output u_t such that for all the DSSTs D_j, either the output of D_j on w is undefined, or the output of D_j is different from u_t. Formally,

$$\exists w \in \Sigma^*, \exists u_t \in [\![T]\!](w) : \bigwedge_{j=1}^{n} \begin{pmatrix} [\![D_j]\!](w) \text{ is not defined} \vee \\ |u_t| \neq |[\![D_j]\!](w)| \qquad \vee \\ \exists p_j : u_t[p_j] \neq [\![D_j]\!](w)[p_j] \end{pmatrix} \tag{4.1}$$

We can expand the above condition to obtain a disjunction over different combinations of conjunctions of the inner disjuncts. Each term in this disjunct illustrates a way in which u_t is not generated by all of the n DSSTs. We construct a 1-reversal $(2.n)$-counter machine M that nondeterministically decides to check one of the conjunctive terms in (4.1). In particular, we show how M checks the conjunction: $\bigwedge_{j=1}^{n} (u_t[p_j] \neq [\![D_j]\!](w)[p_j]))$, the other conjunctions are simpler and can be checked by similar techniques.

M simulates the action of $D_1, \ldots, D_n$, and T in parallel. It precisely keeps track of the states of each of these transducers in its state. For each pair (D_j, T), M maintains a pair of counters (c_{jt}, c_{tj}). For each of these n pairs, M pre-loads the counters with some number p_j (the same for both), which is the guess of M for the position p_j such that $u_t[p_j] \neq [\![D_j]\!](w)[p_j]$. At some point in its simulation of T (resp. D_j), M guesses that it has output the symbol $\gamma_{tj} = u_t[p_j]$ (resp. $\gamma_{jt} = [\![D_j]\!](w)[p_j]$) and remembers it in its state. It also guesses where each string variable of T (resp. D_j) appears with respect to the position p_j, thus mapping each string variable to a class in $\zeta = \{L, C, R, N\}$. For each output symbol added by T (resp. D_j) to the left of p_j, M decrements the counter c_{tj} (resp. c_{jt}) by one.

Let $\Gamma' = \Gamma \cup \{\bot\}$, where $\bot$ is the symbol indicating that M has not guessed the symbol at position p_j yet. The set of states of M is a subset of $((Q_T \times Q_1 \times \ldots \times Q_n) \times (\Gamma'^2 \times \zeta^{X_T} \times \zeta^{X_1}) \times \ldots \times (\Gamma'^2 \times \zeta^{X_T} \times \zeta^{X_n}))$. The transitions of M preserve consistency in its guesses for the string variable classes, using the same technique as in the proof of Theorem 4.1. Note that apart from the initial transitions of M that pre-load the counters, all transitions decrement the counters by some non-negative number; thus M has only one reversal.

Once M reaches the end of the input string w, it checks if the symbol guessed for D_j and T is at the same position p_j, by checking if $c_{tj} = c_{jt} = 0$. We define a state as *accepting* if *for all* j, $c_{jt} = c_{tj} = 0$, and for each pair of symbols recorded in the state, $\gamma_{jt} \neq \gamma_{tj}$. In other words, M accepts a computation iff it encodes the parallel action of T, D_1,. . .,D_m on an input string w, producing an output u_t in $[\![T]\!](w)$ such that u_t is different from each of the n outputs of the transducers $D_1, \ldots, D_n$. Thus $[\![T]\!] \subseteq \bigcup_{i=1}^{n} [\![D_i]\!]$ iff M is empty. From [8], we know that checking the nonemptiness of a 1-reversal m-counter machine is in NLOGSPACE (in the size of M). For a fixed n, the number of states of M is dominated by an exponential in the size of the largest set among $X_T, X_1, \ldots, X_n$. Thus, the above construction yields us a procedure that is in PSPACE. □

5 Discussion

Checking k-valuedness. A k-valued NSST naturally extends a functional NSST. This class is not closed under sequential composition because given two k-valued transducers T_1 and T_2, the NSST $T_1 \circ T_2$ could be k^2-valued. Checking if an arbitrary NSST T is k-valued is decidable. We skip the proof in interest of brevity. Similar to the proof of Theorem 4.1, the basic idea is to reduce k-valuedness of an NSST to the nonemptiness problem for a 1-reversal-bounded $(k.(k+1))$-counter machine M that detects if there is some input w on which T produces $(k+1)$ distinct outputs. M simulates $(k+1)$ copies of T in parallel, and for each pair of the possible $(k+1)$ outputs, M uses a pair of counters to check if the outputs are different. M is empty iff T is k-valued. Our approach can be viewed as an extension of the algorithm to check the k-valuedness of NFTs [9].

Equivalence of k-valued NSSTs. The equivalence problem for k-valued NFTs has been shown to be decidable in [5], and more recently in [13]. We are interested in investigating the same problem for k-valued NSSTs. The approach in [5] reduces the equivalence of k-valued NFTs to the solution of an infinite system of word equations, which by the validity of Ehrenfeucht's conjecture, is equivalent to a finite subsystem. The authors then show how the finite subsystem can be effectively characterized for NFTs, which leads to a decision procedure. The proof of [5] relies on the fact that the valuedness of an NFT is at the most its maximum edge-ambiguity[5]. However, as the maximum edge-ambiguity of an NSST does not

[5] The edge-ambiguity of a transducer is the maximum number of transitions between any two states q and q' that have the same input symbol.

place such a bound on its valuedness, this proof strategy fails for proving the decidability of equivalence checking for k-valued NSSTs. In [13], the authors rely on a procedure to decompose a k-valued NFT T into k functional NFTs whose union is equivalent to T. Whether such a decomposition can be generalized to NSSTs remains open.

Bounded-valued NSST*s.* The class of bounded-valued NFTs has been extensively studied in the literature [14], [11]. The class of bounded-valued NSSTs promises to be a robust class. Unlike the class of k-valued NSSTs, it is closed under sequential composition. In both [14] and [11], it is shown that the problem of checking if an NFT is bounded-valued can be checked in polynomial time by verifying certain conditions on the structure of its transition diagram. We believe that it may be possible to generalize these conditions to characterize bounded-valuedness of NSSTs. However, the extension is not straightforward due to the presence of multiple string variables, and allowing output symbols to be appended to the left as well as to the right.

In summary, some of the interesting open problems are:

- Is the equivalence problem for k-valued NSSTs decidable?
- Given a k-valued NSST T can it be effectively decomposed into k functional NSSTs $T_1, \ldots, T_k$, such that $[\![T]\!] = \bigcup_{i=1}^{k} [\![T_i]\!]$?
- Is the bounded-valuedness of NSSTs decidable?

Applications. One of the motivations for studying NSSTs is their potential as models for verifying concurrent heap-manipulating programs. In [3], the authors study the problem of verifying linearizability of methods that modify linked-lists. Checking linearizability of a pair of methods m_1 and m_2 involves verifying if the interleaved execution of m_1 and m_2 (denoted by $m_1 \parallel m_2$) finishes with the same linked-list as executing m_1 followed by m_2 (denoted by $m_1; m_2$) or m_2 followed by m_1 ($m_2; m_1$). The authors show that with a specific regimen for pointer updates this problem can be algorithmically solved.

In [2], the authors have demonstrated how *sequential* heap-manipulating methods can be modeled as DSSTs. Proving linearizability of methods m_1 and m_2 is thus the same as verifying if the interleaved product of the corresponding DSSTs D_1 and D_2 is contained within their union, *i.e.*, checking if $[\![D_1 \parallel D_2]\!] \subseteq ([\![D_1 \circ D_2]\!] \vee [\![D_2 \circ D_1]\!])$. In general, it may not be possible to model the interleaved product of DSSTs as an NSST. However, if we identify a subclass of DSSTs such their interleaved product is an NSST, then checking linearizability reduces to checking containment of this NSST in the finite union of DSSTs (each of which represents a possible sequential execution of the given DSSTs). We have shown that checking containment is decidable in PSPACE. Thus, we believe that NSSTs could serve as a foundational model of *concurrent* heap-manipulating methods, by helping us better understand the theoretical limits of decidability of the verification problems for such methods.

References

1. Alur, R., Černý, P.: Expressiveness of streaming string transducers. In: Proc. of Foundations of Software Technology and Theoretical Computer Science, pp. 1–12 (2010)
2. Alur, R., Černý, P.: Streaming transducers for algorithmic verification of single-pass list-processing programs. In: Proc. of Principles of Programming Languages, pp. 599–610 (2011)
3. Černý, P., Radhakrishna, A., Zufferey, D., Chaudhuri, S., Alur, R.: Model checking of linearizability of concurrent list implementations. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 465–479. Springer, Heidelberg (2010)
4. Courcelle, B.: Graph Operations, Graph Transformations and Monadic Second-Order Logic: A survey. Electronic Notes in Theoretical Computer Science 51, 122–126 (2002)
5. Culik, K., Karhumäki, J.: The equivalence of finite valued transducers (on HDT0L languages) is decidable. Theor. Comp. Sci. 47, 71–84 (1986)
6. Engelfriet, J., Hoogeboom, H.J.: MSO definable String Transductions and Two-way Finite-State Transducers. ACM Transactions on Computational Logic 2(2), 216–254 (2001)
7. Griffiths, T.V.: The unsolvability of the equivalence problem for ε-Free nondeterministic generalized machines. Journal of the ACM 15, 409–413 (1968)
8. Gurari, E.M., Ibarra, O.H.: The Complexity of Decision Problems for Finite-Turn Multicounter Machines. J. Comput. Syst. Sci. 22(2), 220–229 (1981)
9. Gurari, E.M., Ibarra, O.H.: A Note on Finite-valued and Finitely Ambiguous Transducers. Mathematical Systems Theory 16(1), 61–66 (1983)
10. Gusfield, D.: Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, Cambridge (1997)
11. Sakarovitch, J., de Souza, R.: On the decidability of bounded valuedness for transducers. In: Proc. of Mathematical Foundations of Computer Science, pp. 588–600 (2008)
12. Schützenberger, M.P.: Sur les relations rationelles entre monoïdes libres. Theor. Comput. Sci., 243–259 (1976)
13. de Souza, R.: On the Decidability of the Equivalence for k-Valued Transducers. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 252–263. Springer, Heidelberg (2008)
14. Weber, A.: On the Valuedness of Finite Transducers. Acta Informatica 27(8), 749–780 (1990)