Regular Relations over Infinite Strings

Rajeev Alur
University of Pennsylvania, Philadelphia, USA
alur@cis.upenn.edu

Vrunda Dave
IIT Bombay, Mumbai, India
vrunda@cse.iitb.ac.in

Taylor Dohmen
University of Colorado, Boulder, USA
taylor.dohmen@colorado.edu

Shankara Narayanan Krishna
IIT Bombay, Mumbai, India
krishnas@cse.iitb.ac.in

Ashutosh Trivedi
University of Colorado, Boulder, USA
ashutosh.trivedi@colorado.edu

Abstract

Regular model checking is a symbolic state-exploration technique for infinite state systems where the set of states are expressed as regular languages and the transition relations are expressed using rational relations over infinite (or finite) strings. We generalize regular model checking to express transition relations with the more expressive class of regular relations. We adapt (copyless) streaming string transducers (SSTs) as a computational model to represent regular relations over infinite strings, and show that nondeterministic SSTs over infinite strings (ω-NSSTs) precisely capture regular relations.

We further explore theoretical properties of ω-NSSTs required to effectively carry out regular model checking. Since the post-image of a regular language under a regular relation may not be regular (or even context-free), approaches that iteratively compute the image can not be effectively carried out in this setting. Instead, we demonstrate that regular relations are closed under sequential composition and provide a direct composition construction over ω-NSSTs. A key contribution of this paper is a direct construction for the sequential composition of ω-NSST that requires fixing a known bug in the previous composition constructions for SSTs. To establish this result, we show that transducer models with copies in variable updates in a certain bounded fashion can be reduced to copyless versions. Additionally, we show that the regular type checking problem for ω-NSSTs is decidable in PSPACE. This result, along with a direct construction for sequential composition, provides a theoretical foundation for regular model checking with regular relations.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Regular languages; Theory of computation → Transducers

Keywords and phrases Monadic Second-Order Logic, Streaming String Transducers

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Regular model checking [3, 2, 13, 30, 24] is a symbolic state-space exploration technique where the sets of configurations are expressed as regular languages and the transition relations are encoded as rational relations [28, 27, 26] realized by nondeterministic generalized sequential machines (GSMs), see Figure 1. While regular model checking is undecidable in general, a number of approximation schemes and heuristics [13, 23, 29, 18, 1, 8, 12, 22] have made regular model checking a practical verification approach. Regular model checking has been applied to verify programs with unbounded data structures such as lists and stacks [13, 3]. Moreover, since the infinite strings over $\{0,1\}^\omega$ can be naturally interpreted as real numbers in the interval $[0,1]$, regular model checking over infinite strings provides a theoretical framework [9, 10, 25, 7, 14] to analyze properties of hybrid dynamical systems.

In this paper, we generalize regular model checking approach so that transition relations can be realized using *regular relations* over infinite strings. We propose the computational model of nondeterministic streaming string transducers on infinite strings (ω-NSST), and explore theoretical properties of ω-NSSTs required to effectively carry out regular model checking.

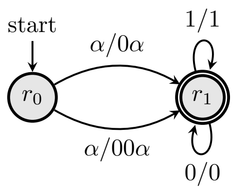


Figure 1 A GSM realizing the relation that shifts a string to the right by 1 or 2, or equivalently realizing division of the binary encoding of real numbers in $[0,1]$ by 2 or 4.

Regular Relations. While rational relations are expressive enough to model a rich set of properties relevant to regular model checking (see, Figure 1), their expressive limitation can be observed by noting their inability to model common transformations such as copy $\stackrel{\text{def}}{=} w \mapsto ww$ and reverse $\stackrel{\text{def}}{=} w \mapsto \overleftarrow{w}$, where the string $\overleftarrow{w}$ is the reverse of the string w. Courcelle [16] initiated the use of monadic second-order logic (MSO) in defining deterministic and nondeterministic graph-to-graph transformations that can easily implement such transformations. Engelfriet and Hoogeboom [19] showed that deterministic MSO-definable deterministic transformations (DMSOT), when restricted to finite strings, are as expressive as two-way generalizations of GSMs (2GSM). They further show that this correspondence does not extend to the set of nondeterministic MSO-definable transformations and nondeterministic 2GSMs.

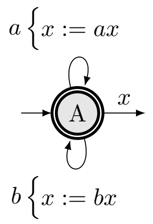
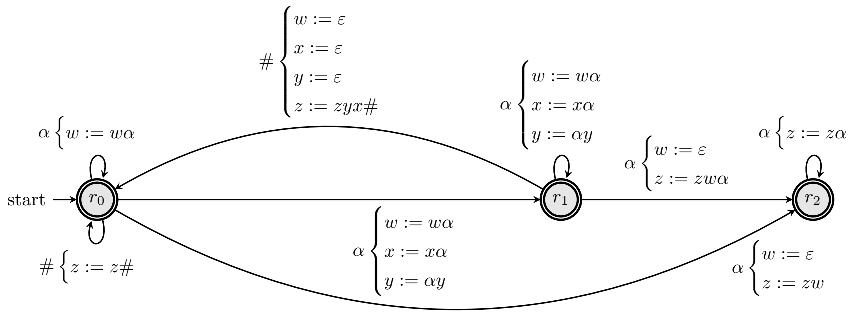


Figure 2 SSTs implementing reverse transformation.

Alur and Černý [4] proposed a one-way machine capable of expressing deterministic MSO-definable transformations over finite strings. These machines, known as *streaming string transducers* (SST), work by storing partial outputs in string variables, and enjoy a number of appealing properties including decidability of functional equivalence and type-checking. Moreover, natural non-deterministic counterparts of SSTs were shown [5] to be equi-expressive to MSO-definable nondeterministic transformations for finite strings. Since the automata and logic connection is often used as a yardstick for regularity, MSO-definable functions and relations over finite strings are often called regular functions and regular relations.

Regular Relations over Infinite Strings. The expressiveness of SSTs and MSO-definable transformations also coincides [6] for functions of infinite strings However, for regular *relations* of infinite strings, no existing computational model is known. We combine and generalize nondeterministic SSTs [5] and SSTs over infinite strings [6] to propose the computational model of nondeterministic streaming ω-string transducers (ω-NSST) to capture regular relations of infinite strings. An example of a regular relation over infinite strings is described below.

▶ **Example 1.1.** Let A be a finite alphabet and $\#$ be a special separator not in A. For



start
r_0
r_1
r_2
$\alpha \{ w := w\alpha$
$\# \{ z := z\#$
$\# \{ w := \varepsilon,\ x := \varepsilon,\ y := \varepsilon,\ z := zyx\#$
$\alpha \{ w := w\alpha,\ x := x\alpha,\ y := \alpha y$
$\alpha \{ w := \varepsilon,\ z := zw\alpha$
$\alpha \{ z := z\alpha$
$\alpha \{ w := \varepsilon,\ z := zw$

Figure 3 An ω-NSST implementing the relation $R_{\overleftarrow{u}u}$ from Example 1.1. Let α denote all symbols in A, excluding $\#$. Variable w remembers the string since the last $\#$, while x and y store the chosen suffix and its reverse. The output variable z is append-only and collects the partial output during each run.

strings $u, v \in A^*$ we say that $v \preceq u$ if v is a suffix of u. Consider a rational relation $R_{\overleftarrow{u}u}$ that transforms strings over $(A \cup \{\#\})^\omega$ such that every $\#$-free finite substring u of the input string is transformed into $\overleftarrow{v}\,v$ for some suffix v of u, formally $R_{\overleftarrow{u}u}$ is defined as:

$$\{(u_1\#\cdots\#u_n\#w, \overleftarrow{v_1}v_1\#\cdots\#\overleftarrow{v_n}v_n\#w) \;:\; u_i, v_i \in A^*, w \in A^\omega \text{ and } v_i \preceq u_i\}$$
$$\cup\, \{(u_1\#u_2\#\ldots, \overleftarrow{v_1}v_1\#\cdots\#\overleftarrow{v_n}v_n\#\ldots) \;:\; u_i, v_i \in A^*, w \in A^\omega \text{ and } v_i \preceq u_i\}$$

$R_{\overleftarrow{u}u}$ can be implemented as an ω-NSST with Büchi acceptance condition as shown in Figure 3.

Contributions. One of our key contributions is to show that the ω-NSST definable relations coincide exactly with MSO-definable relations of infinite strings. To enable regular model checking with regular relations specified using ω-NSSTs, we study the key verification problem of regular model checking. The *type checking problem* for ω-NSSTs asks to decide, given two ω-regular languages L_1, L_2 and an ω-NSST, whether $[\![T]\!](L_1) \subseteq L_2$ where $[\![T]\!]$ is the regular relation implemented by T. We show that type checking for ω-NSSTs is decidable in PSPACE.

Since regular model checking is already undecidable for less expressive rational relations, we studied the problem of bounded model checking in our context. Since, the images of ω-regular languages via regular relations may not be regular (for instance, consider the regular relation that copies strings), we cannot effectively compute the images to approximate the set of reachable configurations. Instead, we turn towards computing an approximation of transitive closure of transition relation by computing sequential composition of ω-NSSTs. While MSO-definable transformations are known to be closed under sequential composition [16], such an approach is not practical as it may require repeated translation of ω-NSSTs to NMSOTs and back. On the other hand, existing sequential compositions of SSTs that work directly with the SSTs are known to be buggy, and in particular, result in copyful SSTs. In this paper, we extend these composition constructions to work on ω-NSSTs, and using a careful renaming of variables observe that while the resulting ω-NSST*s* may contain copyful updates, such copies are bounded in a technical sense. We further provide a construction to remove such bounded copies for ω-NSSTs and provide a direct construction to compute sequential composition for ω-NSSTs. Our construction also fixes the known bugs [4, 5] in sequential compositions for deterministic and nondeterministic SSTs over finite strings. In addition, it provides first direct sequential composition for (both deterministic and nondeterministic) SSTs on infinite strings. The sequential composition together with the decidability of type-checking problem provides theoretical foundations for regular model checking with regular relations.

2 Regular Relations for Infinite Strings

An alphabet A is a finite set of letters. A *string* w over an alphabet A is a finite sequence of symbols in A. We use $w[i]$ to refer to the i^{th} letter in w. We denote the empty string by ε.

We write A^* for the set of all finite strings over A, and for $w \in A^*$ we write $|w|$ for its length. A language L over A is a subset of A^*. An *ω-string* x over A is a function $x : \mathbb{N} \to A$, and written as $x = x(0)x(1)\cdots$. We write A^ω for the set of all ω-strings over A, and A^∞ for $A^* \cup A^\omega$. An ω-language L over A is a subset of A^ω.

2.1 MSO Definable Relations

We may view strings as ordered structures encoded over the signature $\mathcal{S}_A = \{(a)_{a\in A}, <\}$ and interpreted with respect to A^* or A^ω. The domain of a string in this context refers to the set of valid positions in the string, and the relation $<$ in $\mathcal{S}_A$ ranges over this domain. The expression $a(x)$ holds true if the symbol at position x in the current domain is a, and $x < y$ holds if x is a lesser index than y.

Formulae in MSO over $\mathcal{S}_A$ are defined relative to a countable set of first-order variables $x, y, z, \ldots$ that range over individual elements of the domain and a countable set of second-order variables $X, Y, Z, \ldots$ that range over subsets of the domain. The formal syntax for well-formed formulae is given by the following grammar.

$$\phi \quad ::= \quad \exists X.\, \phi(X) \quad | \quad \exists x.\, \phi(x) \quad | \quad \phi \wedge \phi \quad | \quad \phi \vee \phi \quad | \quad \neg\phi \quad | \quad a(x) \quad | \quad x < y \quad | \quad x \in X$$

A deterministic MSO transducer is a tuple $\big(A, B, \mathsf{dom}, N, (\phi_b^n(x))_{b\in B}^{n\in N}, (\psi^{n,m}(x,y))^{n,m\in N}\big)$, where A and B are input and output alphabets, $N = \{1, 2, ..., n\}$ is a finite set of indices, and every ϕ_b^n, every $\psi^{(n,m)}$, and dom are MSO formulae such that the sentence dom defines the domain of the MSOT, the *node formulae* $\big(\phi_b^n(x)\big)_{b\in B}^{n\in N}$ specify the labels on nodes in the output graph, and the *edge formulae* $(\psi^{n,m}(x,y))^{n,m\in N}$ specify edge connections.

The transducer operates over the disjoint union of the string graph of w (its input) with itself, for every index in N. The node formulae determine labels of nodes in the output, and the edge formulae define connections between nodes. Each formula ϕ_b^n has a single free variable and should be interpreted in such a way that if a position satisfies ϕ_b^n, then that position will be labeled by the symbol b in the n^{th} disjoint string graph comprising the output. Each formula $\psi^{(n,m)}$ has two free variables and a satisfying pair of indices indicate that there is a link between the former index in copy n and the latter index in copy m.

Nondeterminism is introduced by allowing additional set variables $X_1, \ldots, X_k$ called *parameters*. For each valuation of these parameters given by sets of positions of the input graph satisfying the domain formula, the rest of the formulae define the output graph as before. Thus, each valuation may result in different output graphs for the same input graph resulting in nondeterminism.

A *nondeterministic* MSO *transducer* (NMSOT) with k free variables $\mathbf{X}_k = (X_0, \ldots, X_{k-1})$ is given as a tuple $\big(A, B, \mathsf{dom}(\mathbf{X}_k), N, (\phi_b^n(x, \mathbf{X}_k))_{b\in B}^{n\in N}, (\psi^{n,m}(x, y, \mathbf{X}_k))^{n,m\in N}\big)$ where all formulae are parameterized by free set variables in addition to the required first-order parameters. For a given input word, two distinct valuations of $\mathbf{X}_k$ may both satisfy the domain formula and result in distinct outputs, and this allows for nondeterministic choice. Two possible outputs from the transducer from Figure 3 are illustrated in Figure 4 and presented to show how an ω-NMSOT would specify these outputs based on the valuation of its second-order parameters.

2.2 Nondeterministic Streaming String Transducers

A nondeterministic streaming string transducer over ω-strings (ω-NSST) is presented as a tuple $(A, B, S, I, \mathsf{Acc}, \Delta, f, X, U)$ where A and B are finite input and output alphabets respectively, S is a finite set of states, $I \subseteq Q$ is the set of initial states, Acc is an *acceptance*

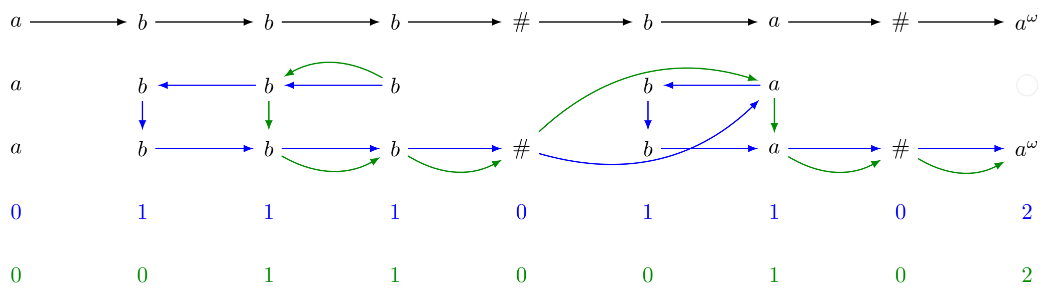


Figure 4 Two possible outputs of the machine from Figure 3, depicted, respectively in blue and green. The numbers below indicate which second-order parameter the corresponding position belongs to for two distinct valuations (blue and green) of these parameters.

condition, X is a finite set of string variables, U is a finite set of variable update functions of type $X \to (X \cup B)^*$, $\Delta : (S \times A) \to 2^{U \times S}$ is a transition function, and $f \in X$ is an append-only output variable.

Such a machine is deterministic if $|\Delta(s, a)| = 1$, for all $s \in S$, $a \in A$, and $|I| = 1$; it is nondeterministic otherwise. A *run* of an ω-NSST on an infinite string $w \in A^\omega$ is an infinite sequence of states and transitions $s_0 \xrightarrow[u_0]{a_0} s_1 \xrightarrow[u_1]{a_1} \dots$ where $s_0 \in I$ and $(s_{k+1}, u_k) \in \Delta(s_k, a_k)$ for all $k \in \mathbb{N}$. Let $\mathsf{Runs}_T(w)$ be the set of all runs in T given input w. Similarly, a finite run is any finite sequence matching the same requirements.

On each transition $s_k \xrightarrow[u_k]{a_k} s_{k+1}$, the transducer changes state and applies the update u_k to each variable of X in parallel. A *valuation* is a function $X \to B^*$ mapping a variable to a string value. The initial valuation val_ε of all variables is the empty string ε. A valuation is well defined after any finite prefix p of a run r and is computed as composition of updates occurring on this prefix: $\mathsf{val}_p = \mathsf{val}_\epsilon \circ u_0 \circ u_1 \circ \dots \circ u_{n-1}$.

We consider both Büchi and Muller acceptance conditions for ω-NSSTs and reference these classes of machines by the initialisms NBT and NMT (DBT and DMT for their deterministic versions), respectively. For a run $r \in \mathsf{Runs}_T(w)$, let $\mathsf{Inf}(r) \subseteq S$ denote the set of states visited infinitely often. A *Büchi acceptance condition* is given by a set of states $F \subseteq S$ and is interpreted such that a NBT is is defined on an input $x \in A^\omega$ if there exists a run $r \in \mathsf{Runs}_T(x)$ for which $\mathsf{Inf}(r) \cap F \neq \emptyset$. A *Muller acceptance condition* is given as a set of sets of states $\mathbb{F} = \{F_0, \dots, F_n\} \subseteq 2^S$, interpreted such that a NMT is defined on input $x \in A^\omega$ if there exists a run $r \in \mathsf{Runs}_T(x)$ for which $\mathsf{Inf}(r) \in \mathbb{F}$.

Given a run r, let $\langle u_k \rangle_{k \in \mathbb{N}}$ be the sequence of partial updates corresponding to finite prefixes of r. The *output* $T(r)$ of a run r of T, $T(r) \overset{\text{def}}{=} \lim_{n \to \infty} \langle u_n(f) \rangle$ is well-defined only if r is an accepting run. Since the output variable f is only ever appended to and never prepended, the limit exists and is an ω-string whenever the acceptance condition is met and $T(r) = \perp$ otherwise. The relation $[\![T]\!]$ realized by an ω-NSST T is given by $[\![T]\!] = \{(w, T(r)) : r \in \mathsf{Runs}_T(w)\}$. We say that an ω-NSST T is functional if $[\![T]\!]$ is a partial function, i.e. for all $w \in A^\omega$, the set $\{w' : (w, w') \in [\![T]\!]\}$ has cardinality at most 1.

An SST is *copyless* if every variable in X occurs at most once in the image $\mathsf{im}(u)$ of every update $u \in U$. Alternately stated, an update $u \in U$ is copyless if the string $u(x_0)u(x_1)\dots u(x_{n-1})$ has at most one occurrence of each $x \in X$, and an SST is copyless if all of its updates are copyless. An update function $u : X \to (X \cup B)^*$ can easily be extended to $\widehat{u} : (X \cup B)^* \to (X \cup B)^*$ such that $\widehat{u}(w) \overset{\text{def}}{=} \varepsilon$ if $w = \varepsilon$, $\widehat{u}(w) \overset{\text{def}}{=} b\widehat{u}(w')$ if $w = bw'$, and $u(x)\widehat{u}(w')$ if $w = xw'$. The effect of two updates $u_1, u_2 \in U$ in sequence can be *summarized* by the function composition $\widehat{u}_1 \circ \widehat{u}_2$; likewise a sequence of updates of arbitrary length would be summarized by $\widehat{u}_0 \circ \widehat{u}_1 \circ \dots \circ \widehat{u}_{n-1}$. For notational convenience, we often omit the hats when the extension is clear from context. Notice that if all updates in a sequence of compositions are copyless, then so is the entire summary.

▶ **Theorem 2.1.** *A relation is* NBT *definable if, and only if, it is* NMT *definable.*

The equivalence of NBT definable relations and NMT definable relations follows from a straightforward application of the equivalence of nondeterministic Büchi automata and nondeterministic Muller automata, since no changes to the variable assignments are required. Equality of these acceptance conditions in transducers allows us to switch between them whenever convenient.

▶ Remark 2.2. Observe that DMTs and functional NMTs, both of which were introduced in [6], have a slightly different output mechanism, which is defined as a function $\Omega : 2^S \rightharpoonup X^*$ such that for all $S' \subseteq S$ for which $\Omega(S') \neq \bot$, the output string $\Omega(S')$ is copyless and of the form $x_1 \dots x_n$. Furthermore, there is the condition that if $s, s' \in S'$ and $a \in A$ s.t. $(u, s') \in \Delta(s, a)$, then (1) $u(x_k) = x_k$ for all $k < n$ and (2) $u(x_n) = x_n w$ for some $w \in (X \cup B)^*$.

In contrast, our definition has a unique append-only output variable $f \in X$. However, our model with the Muller acceptance is as expressive as that studied in [6]. One can use nondeterminism to guess a position in the input after which states in a Muller accepting set S' will be visited infinitely often. The output function can be defined by guessing a muller set, and keeping an extra variable for the output. Upon making the guess, it will move the contents of $x_1 \dots x_n$ to the variable f and make a transition to a copy $T_{S'}$ of the transducer where $\mathsf{Acc} = \{S'\}$. If any state outside the set S' is visited, or the variables $x_1 \dots, x_{n-1}$ are updated, or the variable f is assigned in non-appending fashion, then $T_{S'}$ makes a transition to a rejecting sink state. Alur, Filiot, & Trivedi [6] showed the equivalence of functional NMT with DMT. This implies that the transductions definable using *functional* NMTs or *functional* NBTs (in our definition) are precisely those definable by deterministic ω-MSOT.

2.3 Equivalence of MSO and NSST Definable Relations

Alur and Deshmukh [5] showed that the nondeterministic versions of these models coincide on the set of relations they define over finite strings. We generalize this results by proving the following theorem. We provide symmetric arguments to connect ω-NSST, ω-SST and ω-NMSOT, ω-MSOT using relabellings, resulting in a simple proof.

▶ **Theorem 2.3.** *A transformation is* ω-NMSOT *definable if, and only if, it is* ω-NSST *definable.*

The proof of Theorem 2.3 proceeds in two stages. In the first part (Lemma 2.4), we show that every ω-NSST is equivalent to the composition of a nondeterministic relabeling and a deterministic ω-SST. In the second part (Lemma 2.5), we show that every ω-NMSOT is equivalent to the composition of a (possibly non-functional) relabeling and a deterministic ω-MSOT. These two lemmas, in conjunction with the equivalence of DMTs and functional NMTs [6], allow us to equate these two models of transformation via a simple assignment.

Our arguments use the concept of a relabelling relation, introduced in [19]. A relation $\rho \subseteq A^\omega \times B^\omega$ is a *relabeling*, if there exists another relation $\rho' \subseteq A \times B$ such that $(aw, bv) \in \rho$ iff $(a, b) \in \rho'$ and $(w, v) \in \rho$. In other words, ρ is obtained by lifting the letter-to-letter relation ρ', in a straight-forward manner, to ω-strings. Let $\mathsf{Let}(\rho)$ denote the letter to letter relation $\rho' \subseteq A \times B$ corresponding to ρ and let RL be the set of all such relabelings ρ.

▶ **Lemma 2.4.** ω-NSST $= \omega$-SST $\circ$ RL

Proof. We first show that ω-SST $\circ$ RL $\subseteq \omega$-NSST: for every DMT $T = (B, C, S, I, \mathbb{F}, \Delta, f, X, U)$ and nondeterministic relabeling $\rho \subseteq A^\omega \times B^\omega$, there is a NMT $T' = (A, C, S, I, \mathbb{F}, \Delta_\rho, f, X, U)$

such that $[\![T']\!] = [\![T]\!] \circ \rho$. As indicated by the tuple given to specify T', the only distinct components between the two machines are their input alphabets and their transition functions Δ and Δ_ρ. The latter is given as $\Delta_\rho \overset{\text{def}}{=} (s, a) \mapsto \bigcup_{(a,b)\in \mathsf{Let}(\rho)} \Delta(s, b)$. The nondeterminism of ρ is therefore captured in Δ_ρ. This results in there being one run through T', for every possible relabeling of inputs for T. Since, the remaining machinery of T is untouched in the process of constructing T', it is clear that T' faithfully simulates the relation $[\![T]\!] \circ \rho$.

What remains to be shown is the inclusion ω-NSST $\subseteq \omega$-SST $\circ$ RL: for any NMT T, there exists a DMT T' and a nondeterministic relabeling ρ such that $[\![T]\!] = [\![T']\!] \circ \rho$. Let $T = (A, B, S, I, \mathbb{F}, \Delta, f, X, U)$ be a NMT. From T, we can construct a nondeterministic, letter-to-letter relation $\rho' \subseteq A\times(U\times S)$ as follows: $\rho' \overset{\text{def}}{=} \{(a, (u, s')) \ : \ (u, s') \in \Delta(s, a) \text{ for some } u \in U\}$. Now let $\rho \subseteq A^\omega \times (U \times S)^\omega$ be the extension of ρ' as described previously. The relation ρ contains the set of all possible runs through T for any possible input in A^ω.

Next, we construct a DMT $T' = (U \times S, B, S, I, \mathbb{F}, \Delta_\rho, f, X, U)$ for which Δ_ρ is defined as $(s, (u, s')) \mapsto \{(u, s') \ : \ (u, s') \in \Delta(s, a) \text{ for some } a \in A\}$. In this way, the DMT T' retains only those relations from ρ which correspond to a run of words $w \in A^\omega$ in T and encodes it as an ω-string over the alphabet $S \times U$. The DMT T' then simply follows the instructions encoded in its input and thereby simulates only legitimate runs through T. Thus, we may conclude that $[\![T]\!] = [\![T']\!] \circ \rho$. ◀

▶ **Lemma 2.5.** ω-NMSOT $= \omega$-MSOT $\circ$ RL.

Proof. We begin by showing the inclusion ω-NMSOT $\subseteq \omega$-MSOT $\circ$ RL: for any ω-NMSOT T, there exists an ω-MSOT T' and a relabeling ρ such that $[\![T]\!] = [\![T']\!] \circ \rho$. Nondeterministic choice in T is determined by the choice of assignment to free variables in $\mathbf{X}_k$. The job of facilitating nondeterminism can be placed upon a relabeling relation, thereby allowing us to remove the parameter variables. Define a letter-to-letter relation $\rho' \subseteq A \times (A \times \{0, 1\}^k)$ as follows: $\rho' \overset{\text{def}}{=} \left\{(a, (a, b)) \ : \ b \in \{0, 1\}^k\right\}$, and let the relabeling $\rho \subseteq A^\omega \times (A \times \{0, 1\}^k)^\omega$ be its extension. This relabeling essentially gives us a new alphabet such that each symbol from A is tagged with encodings of its membership status for each set parameter from $\mathbf{X}_k$. Now, we can construct an ω-MSOT T' that is identical to T, apart from two distinctions. The transducer T' is deterministic (i.e. it has no free set variables), and every occurrence of a sub-formula $x \in X_i$ in T is replaced by a subformula $\bigvee_{b\in\{0,1\}^k \wedge b[i]=1}(a, b)(x)$ in T'. As a result of this encoding, the equality $[\![T]\!] = [\![T']\!] \circ \rho$ holds.

The converse inclusion, ω-MSOT $\circ$ RL $\subseteq \omega$-NMSOT, is much simpler. Every relabeling ρ in RL is ω-NMSOT definable: consider $\rho' = \mathsf{Let}(\rho) \subseteq A \times B$. ω-NMSOT defining ρ is similar to identity/copy, except here have the output label as b iff the input label is a and $(a, b) \in \rho'$. This can be implemented using second order variables X_b for all $b \in B$. Let $\mathbf{X}_B$ represents this set. It needs a single copy to produce the output. Node formula $\phi^1_b(x, \mathbf{X}_B) = \bigvee\limits_{a\in A} \bigvee\limits_{(a,b)\in\rho'} (a(x) \wedge x \in X_b)$, while the edge formula $\psi^{1,1}(x, y, \mathbf{X}_B) = x < y$. It is known that ω-NMSOT are closed under composition [17]. Thus, we conclude that any composition of a nondeterministic relabeling and a deterministic ω-MSOT is definable by a ω-NMSOT and that ω-MSOT $\circ$ RL $\subseteq \omega$-NMSOT. ◀

In conjunction Lemmas 2.4 and 2.5 allow us to write the equation ω-NMSOT $= \omega$-MSOT $\circ$ RL $=$ DMT $\circ$ RL $=$ NMT $= \omega$-NSST, thereby proving Theorem 2.3.

3 Regular Model Checking with Regular Relations

We explain how algorithms for deciding properties of regular relations can be used to perform regular model checking. Given two relations T_1 and T_2, their *sequential composition* is defined as $T_2 \circ T_1 \overset{\text{def}}{=} \{(x, z) \,:\, (x, y) \in T_1 \text{ and } (y, z) \in T_2\}$. Let T^k denote the k-fold composition of a relation T with itself. Let T^* denote the transitive closure of T.

Suppose that Init and Bad are regular languages representing sets of states in some system that are initial, and unsafe, respectively. Given a generic transition relation T which captures the dynamics of the system, the *regular model checking problem* asks to decide whether any element of Bad is reachable from any element of Init via repeated applications of T.

In precise terms, the regular model checking problem asks to decide whether the equation $T^*(\mathsf{Init}) \cap \mathsf{Bad} = \emptyset$ holds. Bounded model checking, in this setting, asks to decide, given $n \in \mathbb{N}$, whether $T^k(\mathsf{Init}) \cap \mathsf{Bad} = \emptyset$ holds, for all $k \leq n$. Unbounded model checking is undecidable, even when T is rational, so we restrict our focus to bounded model checking.

When T is rational, its image is always a regular language, and this permits the approach of iteratively applying T from Init and checking whether this set intersects with Bad by standard automata-theoretic methods. If T is regular, images are no longer regular in general, and we must iteratively compute compositions of T with itself and test whether these compositions enter the Bad language. To allow this, we establish decidability of the *type checking problem* for ω-NSSTs: Given two ω-regular languages L_1, L_2 and an ω-NSST T, decid if the inclusions $L_1 \subseteq \mathsf{dom}(T)$ and $T(L_1) \subseteq L_2$ hold.

▶ **Theorem 3.1.** *The type checking problem for ω-NSSTs is decidable in* PSPACE.

Proof. Suppose that $T = (A, B, S, I, F, \Delta, f, X, U)$ is an NBT and L_1 and L_2 are ω-regular languages over A and B respectively, encoded as DMAs M_1 and M_2. We first check whether T is defined for all ω-strings $w \in L_1$, i.e. $L_1 \subseteq \mathsf{dom}(T)$. A NBA C that recognizes the domain of T can be constructed in linear time, and $L_1 \subseteq \mathsf{dom}(T)$ can be checked in PSPACE : It boils down to checking emptiness of $M_1' \cap C$ where M_1' is the NBA equivalent to M_1 and C is the NBA representing complement language of $\mathsf{dom}(T)$. It is known that NBA (M_1') can be constructed from DMA (M_1) with polynomial states blowup [11]. Also, complement automaton can be constructed for NBA in exponential states [11] and hence C has exponentially many states as compared to T. Intersection of M_1' and C is a standard product construction with the flag so that both M_1' and C visit good states infinitely often. Thanks to the fact that emptiness of NBA can be checked in NLOGSPACE [11], the emptiness of this product automaton having exponential states in the input T and M_1, can be checked in PSPACE.

We now assume that T is well-defined on L_1. We construct a nondeterministic Büchi automaton $\mathcal{A}$ such that $L(\mathcal{A}) = \{w \in L_1 \,:\, \exists w' \in [\![T]\!](w) \text{ s.t. } w' \notin M_2\}$. First, we construct a DMA $\overline{M_2}$ for $\overline{L_2}$ by complementing the Acc set. The automaton $\mathcal{A}$ simulates M_1, T and $\overline{M_2}$ in parallel. First we construct NMT T' corresponding to NBT T to keep muller acceptance condition for all three automata. This can be done just by updating the acceptance condition. Let us fix the definition for all three muller automata: (i) $M_1 = (A, S_1, p_0, \mathbb{F}_1, \Delta_1)$, (ii) $T' = (A, B, S, I, \mathbb{F}', \Delta, f, X, U)$, (iii) $\overline{M_2} = (B, S_2, r_0, \mathbb{F}_2, \Delta_2)$.

$\mathcal{A}$ is defined as the product of M_1, T' (without the output mechanism) and it stores the state summary map i.e. the effect of running current valuation of each variable starting from all states of $\overline{M_2}$. Formally the states of $\mathcal{A}$ is a finite subset of $S_1 \times S \times [S_2 \times X \to S_2 \cup \{\perp\}]$. A state (q, p, g) and $g(r, x) = r'$ represents that if we read current value of register x starting from state r, we reach r'. If $g(r, x) = \perp$, it indicates there is no run on valuation of x starting from r. This information can be updated along the run of $\mathcal{A}$ similar to the composition proof presented in Section 4. For instance, if a transition of T updates x as $aybx$, summary map g

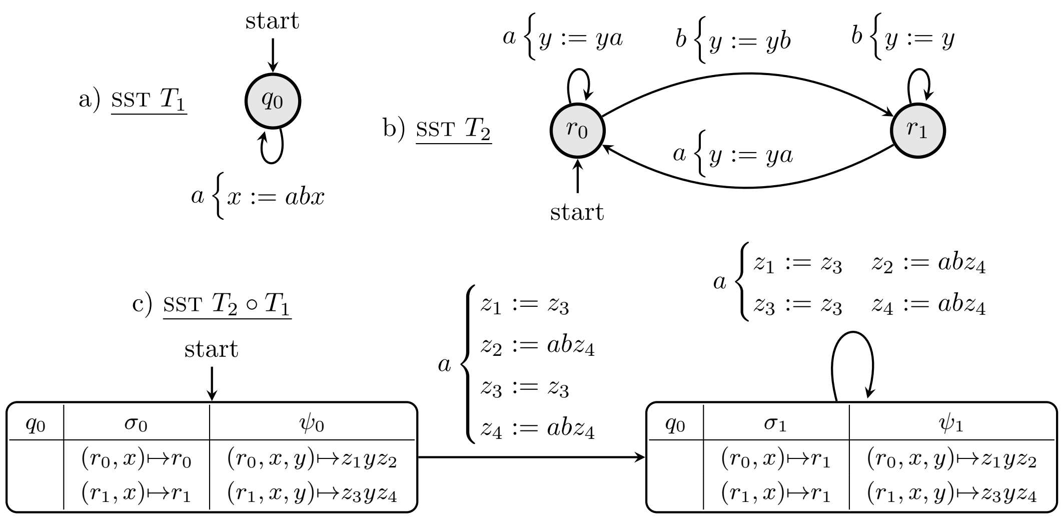

a) SST T_1
start
q_0
$a \{ x := abx$
b) SST T_2
$a \{ y := ya$
$b \{ y := yb$
$b \{ y := y$
r_0
r_1
$a \{ y := ya$
start
c) SST $T_2 \circ T_1$
start
q_0 | σ_0 | ψ_0
$(r_0, x) \mapsto r_0$ | $(r_0, x, y) \mapsto z_1 y z_2$
$(r_1, x) \mapsto r_1$ | $(r_1, x, y) \mapsto z_3 y z_4$
$a \{ z_1 := z_3,\ z_2 := abz_4,\ z_3 := z_3,\ z_4 := abz_4$
$a \{ z_1 := z_3 \quad z_2 := abz_4,\ z_3 := z_3 \quad z_4 := abz_4$
q_0 | σ_1 | ψ_1
$(r_0, x) \mapsto r_1$ | $(r_0, x, y) \mapsto z_1 y z_2$
$(r_1, x) \mapsto r_1$ | $(r_1, x, y) \mapsto z_3 y z_4$

Figure 5 The SSTs T_1, T_2, and their composition $T_3 = T_2 \circ T_1$.

is updated to g' such that $g'(r, x) = g(\Delta_2(g(\Delta_2(r, a), y), b), x)$. It summarizes the effect of reading $x = aybx$ in a sequence in $\overline{M_2}$ starting from state r.

The set of states of $\mathcal{A}$ is $S_{\mathcal{A}} = S_1 \times S \times [S_2 \times X \to S_2 \cup \{\perp\}]$. Here S_1, S and S_2 represent the set of states of M_1, T' and $\overline{M_2}$ respectively. Transition relation $\Delta_{\mathcal{A}}$ is defined as $(u, (q', p', g')) \in \Delta_{\mathcal{A}}(q, p, g)$ iff (1) $\Delta_1(q, a) = q'$, (2) $(u, p') \in \Delta_1(p, a)$, and (3) for all $x \in X$, for all $r \in S_2$, $g'(r, x) = r'$ and $\Delta_2(r, \mathsf{val}_{u(x)}) = r'$. Initial states are product of initial states i.e. a set $I_{\mathcal{A}} = \{(q_0, p_0, r_0) : q_0 \in I\}$. Output variable f_{out} and variable set X remains unchanged. The muller accepting set of $\mathcal{A}$ is defined as all sets $P \subseteq S_{\mathcal{A}}$ such that (i) $\pi_1(P) \in \mathbb{F}_1$, (ii) $\pi_2(P) \in \mathbb{F}$, and (iii) $(\pi_3(P))(r_0, f) \in \mathbb{F}_2$. The size of $\mathcal{A}$ is exponential in M_1, M_2 and T, so its emptiness can be tested in PSPACE. ◀

Since regular relations are definable in MSO, they are closed under sequential composition. In combination with Theorem 3.1, this establishes everything necessary for bounded regular model checking with regular relations to be possible. Thus, we have the following corollary.

▶ **Corollary 3.2.** *Bounded regular model checking with regular relations is decidable.*

4 Sequential Composition

Alur et al. proposed a direct composition for deterministic SSTs [4] and for nondeterministic SST [5] on finite strings. However, Engelfriet identified a counterexample to this construction by showing that the resulting SSTs may produce copyful updates.

To understand the key point of Engelfriet's counterexample, consider the three SSTs shown in Figure 5, where transducer T_3 is the composition of T_1 and T_2 using the construction from [4]. The composite T_3 determines which variables to use in an update as a function of the variables of T_1 and the states of T_2. The transitions with copyful updates (outlined in red) are caused by the machine trying to simulate, simultaneously, two runs in T_2 starting from distinct states that meet at a common state and must process the same variable from T_1 as input. At this point, summarizing these two runs requires the use of summary variables from Z to compute the proper updates. Notice, also, that all transducers in Figure 5 are deterministic, illustrating the fact that this issue is not isolated to the nondeterministic case.

4.1 Bounded Copy

An SST is called *copyful* when it does not adhere to the copyless restriction, i.e. it has at least one copyful update. Unlike copyless SSTs, copyful SSTs do not have output linear in size to their input in general [5, 20]. Consider, for example, an SST with two states (q_{in}, q_f) and two transitions: from q_{in} to q_f while reading a and update $x = a$, and self loop on q_f with the update $x = xx$; the length of the output is exponential relative to the input. In this way, copyful SSTs are strictly more expressive than copyless SSTs. SSTs with *bounded copy* are defined in terms of live and dead copies formalized using dependency graphs.

A *dependency graph* $G_T(r)$, corresponding to a run $r = s_0 \xrightarrow[u_1]{a_1} s_1 \xrightarrow[u_2]{a_2} s_2 \dots s_{n-1} \xrightarrow[u_n]{a_n} s_n$ of an SST T, is a directed acyclic graph presented as a tuple (V, E) in which $V \stackrel{\text{def}}{=} X \times \{0, \dots, n\}$ is a set of vertices, and, for $0 \le i \le n$, $E((x, i), (y, i+1)) = k$ if x occurs k times in $u_i(y)$. Thus, $E : V \times V \mapsto \mathbb{N}$ is a multiset, with each edge labeled by a natural number $k \in \mathbb{N}$. A path through a dependency graph is a sequence $(v_0, v_1), (v_1, v_2), \dots, (v_{n-1}, v_n)$ such that, for all $0 \le i < n$, $E(v_i, v_{i+1}) = k > 0$. If there is a vertex $v = (x, i)$ such that there are k outgoing edges from v, then k *live copies* of (x, i) are present at $i + 1$. More generally, for a given vertex $v = (x, i)$ there are k live copies of (y, j) in v if there are k distinct paths from (y, j) to v. We often say that a copy is *born* in an update u when a variable is copied by u and we say that a copy has *died* if it has no outgoing edge and is at level $i < n$. A vertex $v \in V$ is called a *copy vertex* if $|vE| \ge 2$. The maximum number of copies of any variable that are simultaneously live during any possible accepting computation is denoted $\Lambda(T)$, and takes a value in $\mathbb{N} \cup \{\infty\}$. A bounded-copy SST is a copyful SST for which there exists a $k \in \mathbb{N}$ such that $\Lambda(T) \le k$. If $\Lambda(T) = k$, then we call T a k-copying SST.

Bounded copy transducers are no more powerful than copyless ones. The proof proceeds by constructing an equivalent transducer in which each state is a product of a state from the original machine and a reduced dependency graph. We show that number of reduced dependency graphs is finite for a bounded copy ω-NSST and can be stored in the states. The construction follows the approach of [6] where a similar result was presented for ω-SSTs.

▶ **Theorem 4.1.** *For any bounded copy ω-NSST, one can effectively construct an equivalent, copyless ω-NSST with a polynomial increase in the number of states.*

4.2 Intuition Behind the Construction

Let $T_{12} = (A_1, A_2, Q, I_Q, \mathsf{Acc}_Q, \Delta_{12}, f_{12}, X, U_X)$, $T_{23} = (A_2, A_3, R, I_R, \mathsf{Acc}_R, \Delta_{23}, f_{23}, Y, U_Y)$ be copyless NSSTs to be composed. The construction relies on storing two *summaries* as finite maps in the states of the composite machine. One of these maps, $\sigma : R \times X \to R$*–the state summary map*—stores a state in T_{23} that can be reached by reading the contents of the string stored in variable $x \in X$, for every possible combination of starting states in T_{23} and variables $x \in X$ of T_{12}. This is necessary since the outputs of T_{12} are the inputs of T_{23}. The other map, $\psi : R \times X \to [Y \to (Y \cup Z)^*]$—*the shape summary map*—stores the shape of each variable update $y \in Y$, after one possible run induced by reading a variable $x \in X$ from a starting point $r \in R$. The variables Z are used to store the chunks of strings from A_3 appearing interspersed in the updates of T_{23}, there by resulting in shapes over $(Y \cup Z)^*$. This helps in computing the outputs of T_{23} using the state summary map, and updating variables of Y appropriately. The composed SST simulates T_{12} by keeping the state of T_{12} in its finite control, and in each simulation step, records the state and shape summary maps, and uses these to to record the summary of T_{23} starting in each state of T_{23} on the contents of each $x \in X$. We write St for the set of all state summary maps $[R \times X \to R]$ and Sh for the set of all shape summary maps $[R \times X \to [Y \to (Y \cup Z)^*]]$.

4.3 Composition Construction

Below are defined a number of functions that will facilitate the construction of T_{13}. For a set X and an alphabet A, let $\langle X, A^* \rangle$ be the set of all strings over $(X \cup A)^*$ that are copyless in X, i.e. at most one of each element of X occurs in any string in $\langle X, A^* \rangle$. To minimize notation, we use $\mathbf{x}$ to represent strings from $\langle X, A_2^* \rangle$, $\mathbf{y}$ to denote strings from $\langle Y \cup Z, A_3^* \rangle$ or $\langle Y, A_3^* \rangle$,and $\mathbf{z}$ to denote strings from $\langle Z, A_3^* \rangle$.
The summary helper $\eta : (\langle X, A_2^* \rangle \times \mathsf{St} \times \mathsf{Sh} \times 2^{R \times [Y \to \langle Y \cup Z, A_3^* \rangle]}) \to 2^{R \times [Y \to \langle Y \cup Z, A_3^* \rangle]}$ is defined:

$$(\mathbf{x}, \sigma, \psi, \Sigma) \mapsto \begin{cases} \Sigma & \text{if } \mathbf{x} = \varepsilon \\ \eta(\mathbf{x}', \sigma, \psi, \{(r', \varphi \circ u) \ : \ (r, \varphi) \in \Sigma \text{ and } (u, r') \in \Delta_{23}(r, a)\}) & \text{if } \mathbf{x} = a\mathbf{x}' \\ \eta(\mathbf{x}', \sigma, \psi, \{(\sigma(r), \varphi \circ \psi(r, x)) \ : \ (r, \varphi) \in \Sigma)\} & \text{if } \mathbf{x} = x\mathbf{x}'. \end{cases}$$

The next sets of shape summaries and their corresponding variable updates in T_{13}, following an update $u \in U_X$, are computed, respectively, by the functions $\mathbb{S}_x : R \times [Y \to \langle Y \cup Z, A_3^* \rangle] \to [Y \to \langle Y \cup Z \rangle]$ and $\mathbb{U}_x : R \times [Y \to \langle Y \cup Z, A_3^* \rangle] \to [Z \to \langle Y \cup Z, A_3^* \rangle]$, defined as follows:

$$\mathbb{S}_x(r, \varphi) \stackrel{\text{def}}{=} \bigcup_{y_i \in Y} \left\{ y_i \mapsto z_{i,0}^{r,x} y_1 z_{i,1}^{r,x} \ldots y_n z_{i,n}^{r,x} \ : \ \varphi(y_i) = \mathbf{z_0} y_1 \mathbf{z_1} \ldots y_n \mathbf{z_n} \right\},$$
$$\mathbb{U}_x(r, \varphi) \stackrel{\text{def}}{=} \bigcup_{y_i \in Y} \left\{ z_{i,0}^{r,x} \mapsto \mathbf{z_0}, z_{i,1}^{r,x} \mapsto \mathbf{z_1}, \ldots, z_{i,n}^{r,x} \mapsto \mathbf{z_n} \ : \ \varphi(y_i) = \mathbf{z_0} y_1 \mathbf{z_1} \ldots y_n \mathbf{z_n} \right\},$$

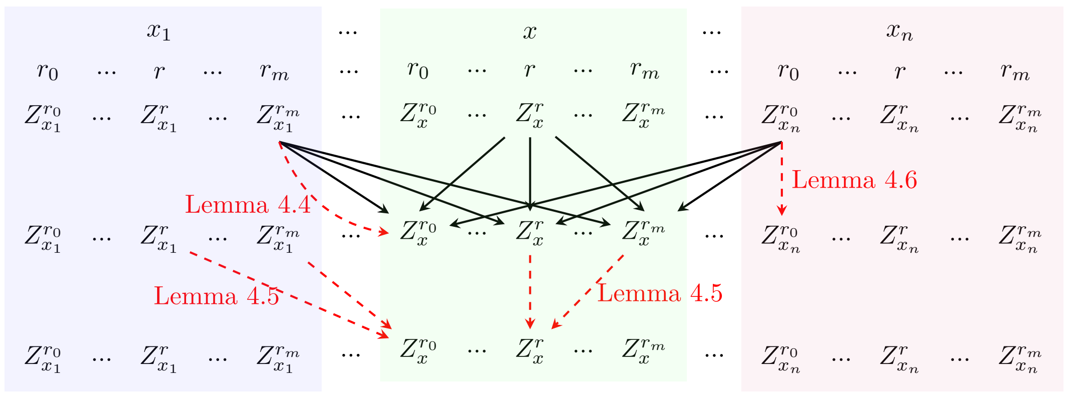
where $(r, \varphi) \in \eta(x, \sigma, \psi, \{(r, y \mapsto y)\})$, for some $\sigma \in \mathsf{St}$ and $\psi \in \mathsf{Sh}$.

4.4 Sequential Composition

For $T_{12} = (A_1, A_2, Q, I_Q, \mathsf{Acc}_Q, \Delta_{12}, f_{12}, X, U_X)$ and $T_{23} = (A_2, A_3, R, I_R, \mathsf{Acc}_R, \Delta_{23}, f_{23}, Y, U_Y)$, copyless ω-NSSTs, $T_{23} \circ T_{12}$ is the ω-NSST $T_{13} = (A_1, A_3, S, I_S, \mathsf{Acc}_S, \Delta_{13}, f_{13}, Z, U_Z)$ where:

$$S \stackrel{\text{def}}{=} Q \times \mathsf{St} \times \mathsf{Sh}; \ I_S \stackrel{\text{def}}{=} \left\{ \big(q, (r, x) \mapsto r, (r, x, y_k) \mapsto z_{k,0}^{r,x} y_k z_{k,1}^{r,x}\big) \ : \ q \in I_Q, z_{k,0}^{r,x}, z_{k,1}^{r,x} \in Z_x^r \right\};$$
$$\mathsf{Acc}_S \stackrel{\text{def}}{=} \{P \subseteq S \colon q \in \mathsf{Acc}_Q, \sigma(r_0, f_{12}) \in \mathsf{Acc}_R, \text{ for some } r_0 \in I_R, \text{ and } (q, \sigma, \psi) \in P\};$$
$$\Delta_{13} \stackrel{\text{def}}{=} \big((q, \sigma, \psi), a\big) \mapsto \big\{ \big(u, (q', \sigma', \psi')\big) \ : \ (v, q') \in \Delta_{12}(q, a),$$
$$\textit{and } (\sigma', \psi') = \bigcup_{r \in R} \bigcup_{x \in X} \left\{ \big((r, x) \mapsto r', \mathbb{S}_x(r', \varphi)\big) : (r', \varphi) \in \eta\big(v(x), \sigma, \psi, \{(r, y \mapsto y)\}\big) \right\},$$
$$\textit{and } u = \bigcup_{r \in R} \bigcup_{x \in X} \left\{ \mathbb{U}_x(r', \varphi) \ : \ (r', \varphi) \in \eta\big(v(x), \sigma, \psi, \{(r, y \mapsto y)\}\big) \right\} \big\};$$
$$Z \stackrel{\text{def}}{=} \bigcup_{(r,x) \in R \times X} Z_x^r, \text{ where } |Z_x^r| = 2|Y|; U_Z \stackrel{\text{def}}{=} \bigcup_{(s,a) \in S \times A} \{u : (u, s') \in \Delta_{13}(s, a)\}.$$

The acceptance condition of T_{13} consists of sets of states P, whose first component is part of the acceptance condition of T_{12}. If Acc_Q was Buchi, then $P \in \mathsf{Acc}_Q$ represents $P \cap \mathsf{Acc}_Q \neq \emptyset$, and if Acc_Q was Muller, then, $P \in \mathsf{Acc}_Q$ represents that P is one of the sets in the Muller set of T_{12}. The notation is same for T_{23} as well. The outputs computed by T_{12} using its acceptance condition, are in turn, accepted by T_{23}. The set of states in the state summary of (r_0, f_{12}), are those reached from an initial state r_0 of T_{23} on a valuation of the output variable f_{12} of T_{12}. This set of states must be a part of acceptance condition of T_{23}. Acc_S has been defined as a set consisting of sets of states, so a Muller condition. By


x_1
x
x_n
r_0 … r … r_m
$Z_{x_1}^{r_0}$ … $Z_{x_1}^{r}$ … $Z_{x_1}^{r_m}$
$Z_{x}^{r_0}$ … Z_{x}^{r} … $Z_{x}^{r_m}$
$Z_{x_n}^{r_0}$ … $Z_{x_n}^{r}$ … $Z_{x_n}^{r_m}$
Lemma 4.4
Lemma 4.5
Lemma 4.5
Lemma 4.6

■ **Figure 6** Dependency graph: Potential copies in a single update (black) and examples of information flows that cannot occur (dashed red) labelled by lemmas prohibiting them.

Remark 2.2, we know that this is without loss of generality. The output function can be defined by guessing a muller set, and keeping an extra variable for the output.

▶ Remark 4.2. The number of states in T_{13} is upper bounded by $|Q||R|^{|R\times X|}|\langle Y\cup Z\rangle|^{|Y|^{|R\times X|}}$.

4.5 Establishing the Bounded Copy Property

Here we provide an analysis showing that the composite machine produced by the construction presented above always has the bounded copy property. To establish the bounded copy property in the composite machine, we define two binary relations $\sim$ and $\equiv$ over the Z variables of T_{13}:

$$\sim \overset{\text{def}}{=} \left\{(z_1,z_2)\ :\ z_1\in Z_x^r \text{ and } z_2\in Z_x^{r'}\right\} \text{ and } \equiv \overset{\text{def}}{=} \{(z_1,z_2)\ :\ z_1\in Z_x^r \text{ and } z_2\in Z_x^r\},$$

from some $x\in X$ and $r,r'\in R$. It is easy to see that $\sim$ and $\equiv$ are equivalence relations. The relation $\sim$ partitions Z into $|X|$ classes, each of size $2|Y\times R|$. Similarly, the relation $\equiv$ partitions Z into $|X\times R|$ classes each of size $2|Y|$. Abusing notation, we sometimes write $[z]_\sim = x$ to denote that z belongs to the class corresponding to the subset $\cup_{r\in R}Z_x^r$. Similarly, we write $[z]_\equiv = (x,r)$ to denote that the equivalence class of z corresponds to the subset Z_x^r.

We next utilize these relations and the structure they impose on Z to show that the composite transducer is bounded copy. In the following we assume that variables in a given $\sim$-class all function to summarize how updates to a single variable in X effect Y variables in T_{23}, and that the variables in a given $\equiv$-class all function to summarize the effects of updates in T_{12} due to a single variable of X, when executed from a single starting state in R. This is consistent with the definition of Δ_{13} given above.

We proceed by considering a single, arbitrary run through the transducer and prove that it is bounded copy. In turn, this implies that any run through the machine is bounded copy and thus that the machine itself is bounded copy overall.

▶ **Lemma 4.3** (One Step Copies are Bounded.). *Every update $u\in U_{13}$ in T_{13} gives birth to at most $2|R\times X\times Y|$ copies during a single transition.*

Next, we develop three observations (Lemmas 4.4 to 4.6) about these equivalence relations that imply that T_{13} is bounded copy.

▶ **Lemma 4.4** (Copies exists in same x-class and different (r,x)-classes.). *If there is a copyful update u in T_{13} that copies a variable z, then for every $z_1,z_2\in Z$ such that $z\in u(z_1)$ and $z\in u(z_2)$, we have that $z_1\sim z_2$ and $z_1\not\equiv z_2$.*

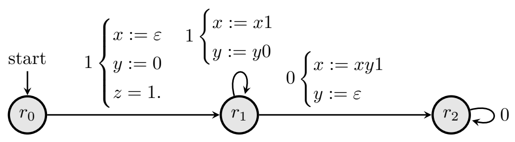

start
$1 \begin{cases} x := \varepsilon \\ y := 0 \\ z = 1. \end{cases}$
$1 \begin{cases} x := x1 \\ y := y0 \end{cases}$
$0 \begin{cases} x := xy1 \\ y := \varepsilon \end{cases}$
r_0
r_1
r_2
0

Figure 7 An ω-SST squaring a number of form 0.1^n0^ω. The output at r_1 is zy and at r_2 is x.

▶ **Lemma 4.5** (No two $\sim$-equivalent variables are ever concatenated in an update.). *If two variables $z_1, z_2 \in Z$ flow into a common target variable z, i.e. $z_1, z_2 \in u(z)$, during an update u, then $z_1 \nsim z_2$.*

▶ **Lemma 4.6** (Updates preserve $\sim$-equivalence.). $z_1 \sim z_2 \wedge z_1 \in u(z) \wedge z_2 \in u(z') \Rightarrow z \sim z'$.

▶ **Theorem 4.7.** *T_{13} is a bounded copy ω-NSST.*

Proof. We show that T_{13} is k-copying where $k \leq 2|R{\times}X{\times}Y|$. Notice that if the number of live copies is to grow unboundedly, then there must be some transition that causes two copies to flow into a common target variable. Otherwise, the maximum number of live copies at any given time would be bounded by $|Z| = 2|R{\times}X{\times}Y|$. In terms of dependency graphs, this idea can be conceptualized in terms of a *diamond* pattern. A diamond occurs when there are two distinct paths from one node to another. This kind of merging cannot occur in T_{13} as a consequence of the conjunction of Lemma 4.5: only two variables that are not $\sim$-equivalent can merge, and Lemma 4.6: all copies of any variable must go into $\sim$-equivalent partitions. To summarize: a variable may be copied to many sets Z_x^r for a unique x, and variables may only be merged from distinct $\sim$-classes, and so the only possibility of a diamond occurring in the dependency graph is if both paths are contained within a single $\sim$-class. Due to Lemma 4.4, however, we know that no two copies of a variable can go to the same $\equiv$-class, and this eliminates this final potential source of diamonds. Figure 6 gives an example dependency graph in which $2|R{\times}X{\times}Y|$ copies are born and shows the diamond-producing situations that the above lemmas prohibit. The absence of the diamond patterns in the dependency graph for any run of the transducer implies that the number of live copies in T_{13} is bounded. ◀

Together, Theorems 4.7 and 4.1 give an alternate proof of closure under composition for regular relations that does not require translating between logic and machine specifications.

5 Conclusion

We introduced ω-NSSTs as a computational model for regular relations, and showed that the relations definable by ω-NSST coincide exactly with those definable in MSO. Motivated by potential applications of SSTs in formal verification, we studied algorithmic properties of these objects and established the minimal theoretical results required for bounded regular model checking to be possible with regular transition relations. Regular functions and relations provide an appealing class of models for real valued functions, as can be seen in [15, 21], where the continuity of regular functions encoded by ω-automata has been studied. Going beyond ω-automata, and developing efficient algorithms for checking continuity and differentiability of regular functions definable by transducers have a potential application in analyzing dynamical systems. The underlying model to consider here can be ω-SST; for instance, a non-trivial example of an ω-SST that squares certain binary numbers is given in Figure 7. With this application in mind, it would be worthwhile to study the approximation techniques developed for traditional regular model checking to see if they generalize to handle regular relations.

References

1 Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d'Orso. Regular model checking made simple and effcient. In *CONCUR — Concurrency Theory*. Springer Berlin Heidelberg, 2002.

2 Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, Julien d'Orso, and Mayank Saksena. Regular model checking for ltl(mso). *International Journal on Software Tools for Technology Transfer*, 14, 2012.

3 Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Mayank Saksena. A survey of regular model checking. In *CONCUR - Concurrency Theory*. Springer Berlin Heidelberg, 2004.

4 Rajeev Alur and Pavol Černý. Expressiveness of streaming string transducers. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010.

5 Rajeev Alur and Jyotirmoy V Deshmukh. Nondeterministic Streaming String Transducers. In *Proceedings of ICALP 2011*. Springer Berlin Heidelberg, 2011.

6 Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular transformations of infinite strings. In *Proceedings of LICS 2012*. IEEE, 2012.

7 Bernard Boigelot, Sébastien Jodogne, and Pierre Wolper. An effective decision procedure for linear arithmetic over the integers and reals. *ACM Trans. Comput. Logic*, 6, 2005.

8 Bernard Boigelot, Axel Legay, and Pierre Wolper. Iterating transducers in the large. In *Computer Aided Verification*. Springer Berlin Heidelberg, 2003.

9 Bernard Boigelot, Axel Legay, and Pierre Wolper. Omega-regular model checking. In *Tools and Algorithms for the Construction and Analysis of Systems*, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

10 Bernard Boigelot and Pierre Wolper. Representing arithmetic constraints with finite automata: An overview. In *Logic Programming*. Springer Berlin Heidelberg, 2002.

11 Udi Boker. Why these automata types? In *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018*, pages 143–163, 2018.

12 Ahmed Bouajjani, Peter Habermehl, and Tomas Vojnar. Abstract regular model checking. In *Computer Aided Verification*. Springer Berlin Heidelberg, 2004.

13 Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular Model Checking. In *Proceedings of CAV*. Springer, Berlin, Heidelberg, 2000.

14 Ahmed Bouajjani, Axel Legay, and Pierre Wolper. Handling liveness properties in (ω-)regular model checking. *Electronic Notes in Theoretical Computer Science*, 138, 2005.

15 S. Chaudhuri, S. Sankaranarayanan, and M. Y. Vardi. Regular real analysis. In *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 509–518, June 2013.

16 B. Courcelle. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126, 1994.

17 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Cambridge University Press, 2012.

18 Dennis Dams, Yassine Lakhnech, and Martin Steffen. Iterating transducers. *J. Log. Algebr. Program.*, 52-53, 2002.

19 Joost Engelfriet and Hendrik Jan Hoogeboom. Mso definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Logic*, 2, 2001.

20 Emmanuel Filiot and Pierre-Alain Reynier. Copyful streaming string transducers. In *Reachability Problems*. Springer International Publishing, 2017.

21 Alexi Block Gorman, Philipp Hieronymi, Elliot Kaplan, Ruoyu Meng, Erik Walsberg, Zihe Wang, Ziqin Xiong, and Hongru Yang. Continuous regular functions. *arXiv preprint arXiv:1901.03366*, 2019.

22 Peter Habermehl and Tomáš Vojnar. Regular model checking using inference of regular languages. *Electronic Notes in Theoretical Computer Science*, 138, 2005. Proceedings of the 6th International Workshop on Verification of Infinite-State Systems (INFINITY 2004).

23 Bengt Jonsson and Marcus Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer Berlin Heidelberg, 2000.

24 Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *Theoretical Computer Science*, 256, 2001.

25 Axel Legay. Extrapolating (omega-)regular model checking. *International Journal on Software Tools for Technology Transfer*, 14, 2012.

26 Christof Löding and Christopher Spinrath. Decision problems for subclasses of rational relations over finite and infinite words. *Discrete Mathematics and Theoretical Computer Science*, 21:3, 2019.

27 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

28 M.P Schützenberger. Sur les relations rationelles entre monoïdes libres. *Theoretical Computer Science*, pages 243–259, 1976.

29 Tayssir Touili. Regular model checking using widening techniques. *Electr. Notes Theor. Comput. Sci.*, 50, 2001.

30 Pierre Wolper and Bernard Boigelot. Verifying systems with infinite but regular state spaces. In *Computer Aided Verification*. Springer Berlin Heidelberg, 1998.

A Proofs Omitted from Section 4

A.1 Equivalence of Bounded Copy and Copyless NSSTs

The following lemma establishes that there are finitely many reduced dependency graphs for any bounded copy transducer.

▶ Remark A.1. Without loss of generality, we assume in this section that all update function u of SST satisfies the following conditions: for each $x \in X$ (1) $u(x) \in \Gamma^*(X+\epsilon)\Gamma^*(X+\epsilon)\Gamma^*$ i.e. at most two variables can be used on right side of the update. (2) There are at most two variables y such that x occurs in $u(y)$. This can be done in such a way that boundedness of copies is preserved, as described in [6].

▶ **Lemma A.2.** *For any reduced dependency graph $R = (V, E)$ with at most k copy nodes, $|V| \leq 2k + |X|$.*

Proof. We prove this lemma by induction on number of copy nodes. As a base case, suppose there is only one copy node, then number of vertices are $|X|$ (all leaves) and 1 node which is a copy node. Thus $|X| + k$ nodes suffice in this case.

Consider a graph $R = (V, E) \in \mathcal{R}_k$ with $k > 1$ copy nodes. There must exist a copy node n which has outdegree two. Assume n has two successors n' and n''. Consider new graph $G = (V', E')$ where $V' = V \setminus \{n\}$ and $E' = E \cup \{(v, v') \mid v$ is predecessor of n in R and $v' \in \{n', n''\}\} \setminus \{(n, n'), (n, n'')\}$. G is a reduced dependency graph with $k-1$ copy nodes and by induction hypothesis, G contains at most $2(k-1) + |X|$ nodes. Hence R has at most $2(k-1) + |X| + 1 = 2k + |X| - 1 \leq 2k + |X|$ nodes. ◀

We define now an *extension operator* $\otimes$ which, given a dependency graph G for a run r and an update function $u \in U$, returns the dependency graph $G \otimes u$ such that $(V, E) \otimes u \stackrel{\text{def}}{=} (V', E')$ where $V' \stackrel{\text{def}}{=} V \cup (X \times \{n+1\})$ and $E' \stackrel{\text{def}}{=} E \cup \{((x, n), (y, n+1)) \mapsto k \ : \ x$ occurs k times in $u(y)\}$.

▶ **Theorem 4.1.** *For any bounded copy ω-NSST, one can effectively construct an equivalent, copyless ω-NSST with a polynomial increase in the number of states.*

Proof. Suppose that $T = (A, B, S, I, \mathsf{Acc}, \Delta, f, X, U)$ is an arbitrary, bounded copy NBT and $\Lambda(T) = k$. Suppose, further, that $\max\{|V| \ : \ (V, E) \in \mathcal{R}_k\} = n$.

The construction is centered around storing a reduced dependency graph in each state, introducing additional variables, and using these graphs to guide the copyless updates of the new variables. We introduce three additional variables, for every vertex of the largest reduced dependency graph in $\mathcal{R}_k$. By assumption, a vertex cannot have more than 2 incoming edges, and so we require three variables in order to store the non-variable sub-strings that interleave variables in updates of the form $x \mapsto w_0 x_1 w_1 x_2 w_2$.

To deal with variable updates, we introduce a naming scheme for the vertices of the reduced dependency graphs in question. In the initial dependency graph $(X \times \{0\}, \emptyset)$, we give each vertex a unique name from the set $\{0, \ldots, |X| - 1\}$. When a graph is extended by an update u, a new level is added to the graph and the vertices on the new level are named from the set $\{m, \ldots, m + |X| - 1\}$, where m is the number of vertices in the graph prior to the extension. Next, going in numerical order by name, we use the vertices on this new level to faithfully encode the update u. If $u(x_i) = w_0 x_1 w_1 x_2 w_2$, then the three variables associated to vertex x_i are assigned the strings w_0, w_1, w_2 respectively. If there are more variables than strings, then the left over variables are initialized to contain ε. We will refer to variables associated to a vertex named i with a superscript i and we will identify the order on these variables with a subscript, so $\{x_0^i, x_1^i, x_2^i\}$ would be the set of variables used in the given example.

After an extension is computed, the next step is to reduce the resultant graph and compute the accompanying updates. The induced updates are constructed as the compositions of all of the updates along paths such that each step uses the appropriate subset of variables based on the name of the current vertex. Finally, the nodes in the reduced dependency graph are renamed so that the set of names form a finite subset of $\mathbb{N}$, and the variables are renamed to match. Let $\Upsilon(G, u)$ be the update computed as described above, for a given reduced dependency graph G and update $u \in U$.

We now construct $T' = (A, B, S', I', \mathsf{Acc}', \Delta', f', X', U')$, a copyless NBT such that $[\![T]\!] = [\![T']\!]$, as follows:

$$\begin{aligned}
S' &\stackrel{\text{def}}{=} S \times \mathcal{R}_T, \\
I' &\stackrel{\text{def}}{=} \{(s, ((X \times \{0\}), \emptyset) \in S' \,:\, s \in I\}, \\
\mathsf{Acc}' &\stackrel{\text{def}}{=} \{(s, G) \in S' \,:\, s \in F\}, \\
\Delta' &\stackrel{\text{def}}{=} \big((s, G), a\big) \mapsto \big\{\big(u', (s', G')\big) \,:\, (u, s') \in \Delta(s, a) \text{ and } G' = R_{G \otimes u} \text{ and } u' = \Upsilon(G, u)\big\}, \\
X' &\stackrel{\text{def}}{=} \bigcup_{i=0}^{n} X_i, \text{ where } |X_i| = 3, \text{ and} \\
U' &\stackrel{\text{def}}{=} \bigcup_{(s,a) \in S \times A} \{u \,:\, u \in \Delta'(s, a)\}.
\end{aligned}$$

The correctness of this construction follows from the fact that it is correct for any single run (since it holds for deterministic SST) and we never require more variables than are necessary for a single run. Lemma A.2 established that the number of reduced dependency graphs in $\mathcal{R}_T$ is linear with respect to $\Lambda(T)$ and thus the state space increases by a linear factor, while the set of variables increases by a polynomial factor in size to $n(|X| + 1) \leq (|X|k + |X|)(|X| + 1) = (k + 1)|X|^2 + (k + 1)|X|$. ◀

A.2 Proofs from Section 4.5

▶ **Lemma 4.3** (One Step Copies are Bounded.). *Every update $u \in U_{13}$ in T_{13} gives birth to at most $2|R \times X \times Y|$ copies during a single transition.*

Proof. Assume that $(v, q') \in \Delta_{12}^*(q, a)$ and $(u, (q', \psi', \sigma')) \in \Delta_{13}^*((q, \sigma, \psi), a)$. We proceed by a case analysis.

Case 1: $v(x) = \varepsilon$.

In this situation, $v(x)$ induces the identity function on both the variables and states of T_{23}, because reading ε has no effect. In other terms, $\eta(\varepsilon, \sigma, \psi, \{(r, y \mapsto y)\}) = \{(r, y \mapsto y)\}$. Thus, we get the following derivation:

$$\begin{aligned}
u &= \bigcup_{r \in R} \{\mathbb{U}_x(r', \varphi) \,:\, (r', \varphi) \in \eta\big(v(x), \sigma, \psi, \{(r, y \mapsto y)\}\big)\} \\
&= \bigcup_{r \in R} \mathbb{U}_x(r, y \mapsto y) \\
&= \bigcup_{r \in R} \{z^{r,x} \mapsto z^{r,x} \,:\, z^{r,x} \in Z_x^r\},
\end{aligned}$$

showing that u results in no new copies born; u is copyless.

Case 2: $v(x) = \mathbf{x}x'$.

If $(r_*, \varsigma) \in \eta(v(x), \sigma, \psi, \{(r, \varphi)\})$ and $(r_*, \varsigma') \in \eta(v(x), \sigma, \psi, \{(r', \varphi')\})$, then $\mathbf{x}$ synchronizes r and r' to the same target state r_*. Following the construction, the update corresponding to the run starting at r is computed by $\mathbb{U}_x(r, \varphi)$ and the update for the run starting at r' is computed by $\mathbb{U}_x(r', \varphi')$. Respectively, these subsets of u contain only assignments to variables from Z_x^r and $Z_x^{r'}$ and summarize T_{23} from states r and r' on the input $\mathsf{val}_w(v(x))$. Applying some judicious rewriting yields the following two equations:

$$\begin{aligned}\mathbb{U}_x(r, \varphi) &= \mathbb{U}_x\big(r, \eta(v(x), \sigma, \psi, \{(r, y \mapsto y)\})\big)\\ &= \mathbb{U}_x\big(r, \eta(\mathbf{x}x', \sigma, \psi, \{(r, y \mapsto y)\})\big)\\ &\subseteq \mathbb{U}_x\big(r, \varsigma \circ \psi(r_*, x')\big),\end{aligned}$$

$$\begin{aligned}\mathbb{U}_x(r', \varphi') &= \mathbb{U}_x\big(r', \eta(v(x), \sigma, \psi, \{(r', y \mapsto y)\})\big)\\ &= \mathbb{U}_x\big(r', \eta(\mathbf{x}x', \sigma, \psi, \{(r', y \mapsto y)\})\big)\\ &\subseteq \mathbb{U}_x\big(r', \varsigma' \circ \psi(r_*, x')\big)\end{aligned}$$

Because $\mathbf{x}$ synchronizes r and r', both sets of updates are dependent upon $\psi(r_*, x')$, and consequently the same set of variables $Z_{x'}^{r_*}$. If every variable in $Z_{x'}^{r_*}$ is utilized, then $2|Y|$ copies are birthed. If this occurs for every state in R for a fixed variable x, then $2|Y \times R|$ copies are birthed. The maximum number of copies birthed in this case occurs when $v(x) = w_0 x_1 w_1 ... w_{n-1} x_n$, where $|X| = n$ and each $w_i \in v(x)$ synchronizes every state in R. If this happens, then it is possible that every variable in $Z^{r_*} x_i$ gets copied for each $x_i \in X$, resulting in $2|R \times X \times Y|$ copies born. Since T_{12} is copyless by assumption, x' will never appear in $v(x'')$, for any $x'' \in X$ where $x'' \neq x$, and therefore variables in $Z_{x'}^{r_*}$ will not be used in any components of the update that are not simulating a run of $T_{2,3}$ on $v(x)$.

If $\mathbf{x}$ is non-synchronizing, then we will get the same derivation as before except that there will be different parameters in place of r_*, and thus will not result in a copyful update.

Case 3: $v(x) = \mathbf{x}a$.

If $(r_*, \varsigma) \in \eta(v(x), \sigma, \psi, \{(r, \varphi)\})$ and $(r_*, \varsigma') \in \eta(v(x), \sigma, \psi, \{(r', \varphi')\})$, then $\mathbf{x}$ synchronizes r and r' to the same target state r_*. The definition of the summary helper function yields the following re-writings, for some $u \in U_Y$.

$$\begin{aligned}\mathbb{U}_x(r, \varphi) &= \mathbb{U}_x\big(r, \eta(v(x), \sigma, \psi, \{(r, y \mapsto y)\})\big)\\ &= \mathbb{U}_x\big(r, \eta(\mathbf{x}a, \sigma, \psi, \{(r, y \mapsto y)\})\big)\\ &\subseteq \mathbb{U}_x\big(r, \varsigma \circ u\big),\end{aligned}$$

$$\begin{aligned}\mathbb{U}_x(r', \varphi') &= \mathbb{U}_x\big(r', \eta(v(x), \sigma, \psi, \{(r', y \mapsto y)\})\big)\\ &= \mathbb{U}_x\big(r', \eta(\mathbf{x}a, \sigma, \psi, \{(r', y \mapsto y)\})\big)\\ &\subseteq \mathbb{U}_x\big(r', \varsigma' \circ u\big)\end{aligned}$$

Because both are dependent upon u, which cannot have Z variables occurring in it by definition, no new copies are born. ◀

▶ **Lemma 4.4** (Copies exists in same x-class and different (r, x)-classes.). *If there is a copyful update u in T_{13} that copies a variable z, then for every $z_1, z_2 \in Z$ such that $z \in u(z_1)$ and $z \in u(z_2)$, we have that $z_1 \sim z_2$ and $z_1 \not\equiv z_2$.*

Proof. Consider an update of the form $v(x) = \mathbf{x}x'$ in T_{12} where such that $\mathbf{x}$ synchronizes T_{23} to state r_*. Since T_{12} is copyless, x' can only appear once in $v(x)$ and must not appear in the update of any other variable in X. Therefore $[z]_\sim = x'$, and the partitioning of Z implies that z_1 and z_2 both contribute to a summary of x. Thus, we get $[z_1]_\sim = [z_2]_\sim = x$ and consequently $z_1 \sim z_2$.

There are two states r, r' such that the summaries $\sigma'(r, x)$ and $\sigma'(r', x)$ converge on a third state r_*. We saw in Lemma 4.3, that for a copy to occur, z_1 and z_2 must be summarizing effects from two distinct states, and so we infer that $z_1 \in Z_x^r$ and $z_2 \in Z_x^{r'}$ such that $r \neq r'$. Therefore, as a result of the partitioning of Z, we get that $[z_1]_\equiv = (r, x) \neq (r', x) = [z_2]_\equiv$, and consequently $z_1 \not\equiv z_2$. ◀

▶ **Lemma 4.5** (No two $\sim$-equivalent variables are ever concatenated in an update.). *If two variables $z_1, z_2 \in Z$ flow into a common target variable z, i.e. $z_1, z_2 \in u(z)$, during an update u, then $z_1 \nsim z_2$.*

Proof. Suppose that $z_1, z_2 \in u(z)$. If $z_1 \sim z_2$, then both variables are contributing to summaries indexed by the same variable, x, from two potentially different states in R. Both z_1 and z_2 occurring in $u(z)$ implies that T_{13} is simulating some run of T_{23} in which it process x from two distinct states. In other words, T_{13} relies on both of the summaries $\psi(x, r)$ and $\psi(x, r')$ to compute $u(z)$. This contradicts the assumption that T_{12} is copyless, because it would require some update $v(x)$ to be present in T_{12} in which two copies of the same variable occur. Therefore, we conclude that $z_1, z_2 \in u(z)$ implies $z_1 \nsim z_2$. ◀

▶ **Lemma 4.6** (Updates preserve $\sim$-equivalence.). $z_1 \sim z_2 \wedge z_1 \in u(z) \wedge z_2 \in u(z') \Rightarrow z \sim z'$.

Proof. Because $z_1 \sim z_2$, both registers must contribute to the summary of the effects of the same $x \in X$. On the other hand, if $z \nsim z'$, then z and z' are summarizing effects resulting from distinct registers of X. If $z_1 \in u(z)$ and $z_2 \in u(z')$, then there must be some information from x flowing into both $[z]_\sim$ and $[z']_\sim$. In terms of T_{12}, this would imply that there are $x, x_1, x_2 \in X$ such that $[z]_\sim = x_1$ and $[z']_\sim = x_2$ and $[z_1]_\sim = [z_2]_\sim = x$ and $x \in v(x_1)$ and $x \in v(x_2)$. So, if $z \nsim z'$, then there must be copyful update in T_{12}. This is a contradiction, so we conclude that the conjunction $z_1 \sim z_2$ and $z_1 \in u(z)$ and $z_2 \in u(z')$ implies $z \sim z'$. ◀