MSO-Definable Regular Model Checking

Vrunda Dave*[1], Taylor Dohmen†[2], Shankara Narayana Krishna‡[1], and Ashutosh Trivedi§[2]

[1]Indian Institute of Technology, Bombay
[2]University of Colorado, Boulder

October 22, 2019

Abstract

Regular Model Checking (RMC) is a symbolic model checking technique where the set of system states are expressed as regular languages over strings and the transition relation is expressed using rational string-to-string relations. RMC permits verification of non-trivial properties in systems with infinite state spaces. We introduce monadic second-order logic (MSO) definable regular model checking (MSO-RMC), a framework that generalizes RMC by enabling the modeling of systems with more complex transition relations which are definable using nondeterministic MSO-definable string-to-string transformations. While MSO-RMC is in general undecidable, we recover decidability of the bounded model checking problem within this framework. For this decidability result, we introduce nondeterministic streaming ω-string transducers and establish their expressive equivalence to nondeterministic MSO-definable ω-string transformations. We also proof of the decidability of the regular type checking problem for nondeterministic streaming string transducers, both in the setting of finite strings and ω-strings. Since MSO-definable relations are closed under composition, this result implies decidability of the bounded model checking in MSO-RMC.

1 Introduction

Regular Model Checking [3, 2, 12, 30, 23] (RMC) is a formal verification technique where a configuration of a system is expressed as a string over a finite alphabet and the system's transition relation is encoded as a *rational relation* in the form of a nondeterministic generalized sequential machine (NGSM). Briefly, a GSM is a deterministic that writes a word to an output tape on every transition and defines a function. An NGSM is the nondeterministic version of this and defines a relation (non-functional in general). Given a set of initial and undesirable configurations as regular languages I and B, respectively, and the transition relation as a NGSM T, the *regular model checking* problem asks whether a configuration in B can be reached in any number of steps from a configuration in I by following the transition relation. This task reduces to computing the transitive closure T^* of the transition relation and deciding whether $T^*(I) \cap B = \emptyset$. Computing the transitive closure T^* is generally undecidable, and a number of approximation techniques and heuristics have been developed [12, 22, 28, 18, 1, 8, 11, 21] to compute reasonably precise representations of this relation. As a result, RMC has proven to be a practical model checking approach, despite the theoretical limitations. RMC has been useful in the verification of programs acting on unbounded lists and stacks as well as in the verification of mutual exclusion protocols.

Example 1. As an example of RMC, consider a finite-state program operating on a priority queue. Let Q be the set of locations of the program, M be a finite set of messages that can be enqueued, and $P = \{1, \ldots, n\}$ be a finite set of priority rankings. Configurations of this program can be encoded as strings over the

*vrunda@cse.iitb.ac.in
†taylor.dohmen@colorado.edu
‡krishnas@cse.iitb.ac.in
§ashutosh.trivedi@colorado.edu

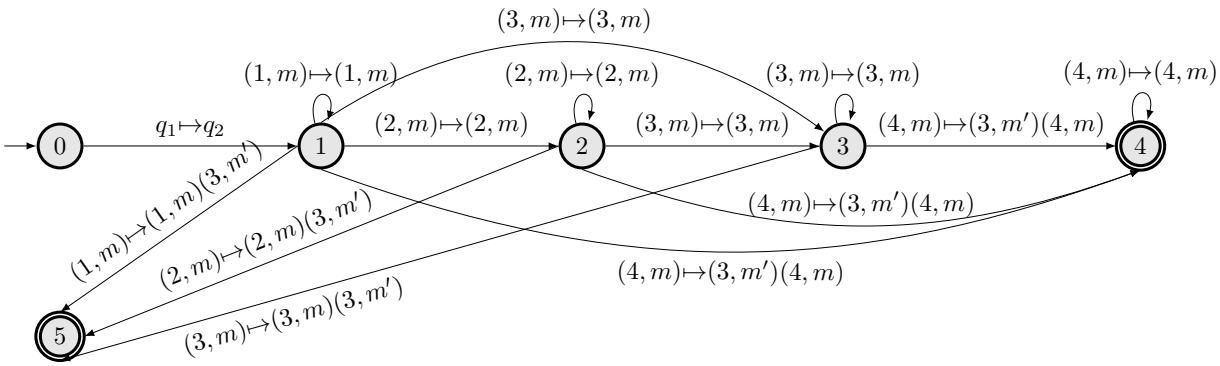

Figure 1: A NGSM encoding on possible action on the priority queue as described in Example 1.

alphabet $Q \cup (P \times M)$ where the first letter is in Q and the subsequent letters are in $(P \times M)$, ordered by a fixed priority. An example of a configuration for a queue with three priority rankings is the string $q(1,m_1)(1,m_2)(2,m_3)(3,m_4)(3,m_5)(3,m_6)$ where $m_i \in M$ and $q \in Q$. The set of valid configurations of this system can be expressed as a regular expression $Q(1,M)^*(2,M)^* \cdots (n,M)^*$. An example of an action where the program sends a message m' of priority 3 to the queue and changes its state from q_1 to q_2 is shown in Figure 1 as a GSM.

The RMC approach has also been generalized to the setting of ω-regular languages over infinite strings [9, 13], and thus can be applied to systems with uncountably large state-spaces. This framework has been demonstrated to be applicable [10, 7] in representing arithmetic problems and algorithms over real numbers. Legay and Wolper [25, 24], among others, have developed efficient, specialized techniques for computing transitive closures of transitions which can be encoded with machines weaker than general ω-automata. Furthermore, there is substantial work on extending the utility of RMC for the verification of properties in dynamical systems such as hybrid dynamical systems [7] and one-dimensional cellular automata [15]. Our goal is to generalize RMC, in the case of both finite strings and ω-strings, in terms of expressiveness by utilizing the full power of MSO-definable transition relations.

1.1 Limitations of Regular Model Checking

The limitations of using rational relations to define transitions in RMC can be observed when considering two simple functions: $\mathsf{reverse} : \Sigma^* \to \Sigma^*$ and $\mathsf{copy} : \Sigma^* \to \Sigma^*$ defined as $w \mapsto \overleftarrow{w}$ and $w \mapsto ww$, respectively, where string $\overleftarrow{w}$ is the reverse of the string w. It is clear that neither of these functions is rational. The function reverse cannot be rational, because GSMs have a one-way input tape, and to reverse a string the machine must read the last symbol of the input and then proceed to read the input in reverse. The irrationality of copy follows from the fact that regular languages are closed under rational transformations and the language $\{ww \ : \ w \in \Sigma^*\}$ is not regular (it is context-sensitive). The fact that both of these operations are relatively simple and potentially useful for modeling more complex systems motivates us to explore more expressive models of transitions. The following two examples provide instances of systems which have transition relations depending upon copy and reverse.

Example 2 (Variable Priority Queue)**.** Suppose that there is a system where a single finite-state process acts on a priority queue with variable priority rankings. Let $\Sigma = Q \cup (M \times P) \cup P$ where Q is the set of possible program states, M a finite set of messages that may be enqueued, and P a finite set of priority markers. A configuration in this sytem would be a word of the form $P^{|P|}Q(M,P)^*$ such that the prefix $P^{|P|}$ is a subword with every symbol from P occurring exactly once. The order of the symbols in this prefix is used to denote the current priority ranking. If $P = \{1,2,3\}$ and q and m denote arbitrary elements of Q and M, a good configuration might look like $213q(m,2)(m,1)(m,3)$, while a bad configuration would be of the form $213q(m,1)(m,2)(m,3)$. Since priority rankings are not fixed, one possible transition of this system involves the reordering of elements in the queue when the ranking is altered. For example, if the ranking changes from

213 to 312, then the resulting transition would look like $213q(m,2)(m,1)(m,3)\mapsto 312q(m,3)(m,1)(m,2)$. As we have seen earlier, such transitions can not be captured using NGSMs.

Example 3 (Unix-like Processes)**.** In unix systems, a process may spawn another process with the fork system call. The new process is called a child of its creator, and upon creation gets an exact copy of the memory space of its parent. Most such systems have what is called copy-on-write semantics, meaning that a child shares its parent's address space until either process wants to write some data to memory. At this point, the parent's address space is copied before the write operation so that both processes have their own distinct spaces in memory. Assuming that the processes in question are finite state, this method of forking can be modeled in MSO-RMC. Let $\Sigma = Q \cup \{0,1\}$, where Q is the finite set of possible program states. Thus, a configuration can be captured by the regular expression

$$\bigcup_{k=0}^{k\leq n} \underbrace{QQ^*(0+1)(0+1)^*QQ^*(0+1)(0+1)^*\dots QQ^*(0+1)(0+1)^*}_{k \text{ times}},$$

where n is a fixed limit on the number of concurrent processes and any two adjacent symbols from Q denote two processes sharing the next address space (i.e. the next subword of zeros and ones). Transitions in this context could be of the form $q01001100\mapsto qp01001100$ when a new process is created, or $qp01001100\mapsto q01001100p11001100$ when a process sharing an address space writes to memory. The first type of transition can easily be described as a rational relation, but since the second type involves subword replication, it is beyond the capabilities of NGSMs. The MSOT defining the transition where the second process in state p (child of the process in state q) overwrites the leftmost bit of its address space is given in Appendix A.

As we show in Example 5, both reverse and copy are MSO-definable relations and hence we are able to model the transition relations from both examples discussed above.

1.2 From Rational Relations to ω-Regular Relations

The classical Büchi-Elgot-Trakhtenbrot theorem [14, 19, 29] establishes the equivalence of MSO and regular languages. Engelfriet and Hoogeboom [20] later showed that this equivalence works for transformations as well. They used the logical transformation framework of Courcelle, called MSO transductions (MSOT), and showed the equivalence of MSOT with deterministic two way transducers (2GSM). The class of transformations defined by deterministic two way transducers are called regular transformations. More recently, Alur et al. [4] introduced streaming string transducers (SST), and proved the equivalence of this model and the MSOT model. Unlike automata, where two-wayness does not increase expressiveness, regular transformations are strictly more expressive than the transformations computed by GSMs (the rational transformations).

Regular transformations, characterized by SSTs and 2GSM, can only accept regular functions. To express regular relations, Engelfriet and Hoogeboom [20] introduced a nondeterministic variant of monadic second-order transducers (NMSOT) and showed that the classes of relations NMSOT and N2GSM are incomparable in terms of expressiveness. Later, Alur and Deshmukh [5] introduced a nondeterministic variant of streaming string transducers (NSST) and showed their equivalence to NMSOT string-to-string transformations.

Alur, Filiot, and Trivedi [6] studied SSTs over infinite strings, captured by SSTs with Muller acceptance condition, and showed their equivalence with MSO-definable infinite string-to-string transformations. A key contribution of our paper is the generalization of these models to capture the notion of ω-regular relations by studying nondeterministic MSO-definable infinite string-to-string transformations (ωNMSOT). We introduce nondeterministic SST over infinite strings (ωNSST) and show their equivalence with ωNMSOTs. This equivalence enables the MSO-definable regular model checking framework, and allows us to computationally tackle the following problem.

Definition 1 (MSO-Definable Regular Model Checking)**.** Given an ωNMSOT T, and two ω-regular languages I and B, the MSO-Definable Regular Model Checking (MSO-RMC) problem is to decide, for all $w \in I$ and $i \in \mathbb{N}$, whether $T^i(w) \notin B$.

Using a reduction from two-counter machines, we show that MSO-RMC is undecidable. This is not surprising, as the problem is already undecidable for transition functions over finite strings expressed as

GSMs. However, since ωNMSOTs are closed under composition, the following contribution yields a recipe for solving the bounded variant of MSO-RMC.

Theorem 1 (Decidability of Type-Checking and Bounded Regular Model Checking)**.** *Given an* ωNMSOT T *and two* ω*-regular languages* I *and* B*, the* k*-bounded model checking problem reduces to the* regular type checking problem*: "Is* $T^i(I)$ *contained in the complement of* B*, for all* $i \leq k$*?" The regular type checking problem for* ωNMSOT*s is decidable.*

2 Preliminaries

For sets A and B, we write $[A \to B]$ for the set of functions $A \to B$. The set/function/relation $\overline{X}$ is the complement of X. Given a function or relation ρ and a subset D of its domain $\mathsf{dom}(\rho)$, the notation $\rho(D)$ is interpreted as the image $\mathsf{im}(\rho)$ of ρ restricted to the elements of D. If D is a singleton, then this is the same as standard function/relation application. If the set $\rho(x)$ is finite for every $x \in \mathsf{dom}(\rho)$, then ρ is *finitely valued*. More precisely, ρ is k-valued if the maximum size of $\rho(x)$ is k over all $x \in \mathsf{dom}(\rho)$.

An alphabet Σ is a finite set of letters. An ω*-string* w on an alphabet Σ is a function $w\colon \mathbb{N} \to \Sigma$. We abbreviate $w(i)$ by w_i. A finite string is defined in a similar way. We denote by ε the empty string. We write Σ^* and Σ^ω for the set of finite and ω-strings over Σ. We write Σ^∞ for $\Sigma^* \cup \Sigma^\omega$. For a string $w \in \Sigma^\infty$ we write $|w|$ for its length; note that for ω-strings we have that $|w| = \infty$.

A language of finite strings is called *regular* if there is a deterministic finite state automaton (DFA) that accepts it. Similarly, ω-automata, defined below, are finite state models accepting ω-regular languages.

Definition 2 (Nondeterministic ω-Automata)**.** A *nondeterministic* ω*-automaton* is a tuple $\mathcal{A} = (Q, \Sigma, E, q_0, \mathsf{Acc})$, where

- Q is a finite set of *states*,
- Σ is a finite *alphabet*,
- $E \subseteq Q \times \Sigma \times Q$ is the set of *transitions*,
- $q_0 \in Q$ is the *initial state*, and
- Acc is the *acceptance condition*.

A *run* r of $\mathcal{A}$ on $w \in \Sigma^\omega$ is an ω-string $r_0, w_0, r_1, w_1, \ldots$ in $(Q \cup \Sigma)^\omega$ such that $r_0 = q_0$ and, for $i > 0$, $(r_{i-1}, w_{i-1}, r_i) \in E$. A *Büchi* (*Muller*) automaton is an ω-automaton equipped with a Büchi (Muller) acceptance condition. We write $\Omega(r)$ for the set of states that appear infinitely often in the run r. The *Büchi* acceptance condition defined by $F \subseteq Q$ accepts the set of runs $\{r \in (Q \cup \Sigma)^\omega \colon \Omega(r) \cap F \neq \emptyset\}$, in which at least one state in F is visited infinitely often. A *Muller* acceptance condition is defined by k sets of subsets of Q, i.e. the set $\mathcal{F} = \{F_0, \ldots, F_{k-1}\}$, is the set of runs $\{r \in (Q \cup \Sigma)^\omega \colon \exists i < k \text{ s.t. } \Omega(r) = F_i\}$. A run r of $\mathcal{A}$ is *accepting* if $r \in \mathsf{Acc}$. The *language* of $\mathcal{A}$ is the subset of strings in Σ^ω that have accepting runs in $\mathcal{A}$. A language is ω*-regular* if it is accepted by an ω-automaton.

An automaton $\mathcal{A} = (Q, \Sigma, E, q_0, \mathsf{Acc})$ is *deterministic* if $(q, a, q') \in E$ and $(q, a, q'') \in E$ implies $q' = q''$ for all $q, q', q'' \in Q$ and all $a \in \Sigma$. $\mathcal{A}$ is *complete* if for all $q \in Q$ and $a \in \Sigma$ there is $q' \in Q$ such that $(q, a, q') \in E$. A word in Σ^ω has exactly one run in a deterministic, complete automaton. We use (N)BA and (N)MA, respectively, to denote the set of all (nondeterministic) Büchi automata and the set of all (nondeterministic) Muller automata. Every MSO-definable language is accepted by some MA and by some NBA. In contrast, there are MSO-definable languages that are not accepted by any BA.

3 MSO-Definable Relations

In this section, we will introduce MSO-definable relations over ω-words (ωNMSOT). For an NMSOT to operate over finite words, the only definitional change needed is to the interpretation of formulae. This change is minimal, since it requires only that the domain of the formulae is comprised of finitely many

positions. We will keep our presentation mostly (some examples will be given for the finite string setting) limited to ω-strings, as all of our results can be easily transferred to a setting with finite strings.

Expressing relations in MSO was originally introduced in the more general context of graph-to-graph transformations [16, 17]. This approach was later specialized to words by considering only string graphs. String graphs are directed acyclic graphs in which every node has at most one entering and one exiting edge. By labeling the vertices with symbols from an alphabet, string graphs can represent arbitrary words. When defining word relations in MSO it is useful to consider string graphs rather than strings themselves.

A *signature* is a collection of relational symbols, each with a designated arity (constants being viewed as relations of arity zero). Signatures are to logics as alphabets are to languages; they are basic syntactic constructs. An *interpretation* of a signature is a means of associating well-formed combinations of relational symbols with elements of some external set or universe.

MSO-Definable Languages. Before discussing MSO-definable relations, we first must introduce the notion of MSO-definable languages. We may view words as *logical structures* by defining logics which are encoded over the signature $\mathcal{S}_\Sigma = \{(\sigma)_{\sigma\in\Sigma}, <\}$ and interpreted with respect to strings in Σ^* or Σ^ω. The domain of a word in this context refers to the set of valid positions/indices in the word, and relations in $\mathcal{S}_\Sigma$ range over this domain. The expression $\sigma(x)$ holds true if the symbol at position x in the current domain is σ, and $x < y$ holds if x is a lesser index than y.

Formulae in MSO over the signature $\mathcal{S}_\Sigma$ 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.

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

As is standard, define universal quantification by $\forall x.\ \phi(x) \stackrel{\text{def}}{=} \neg\exists x.\ \neg\phi(x)$, implication as $\phi \Rightarrow \psi \stackrel{\text{def}}{=} \neg\phi \vee \psi$, and logical equivalence as $\phi \Leftrightarrow \psi \stackrel{\text{def}}{=} (\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$. We define the xor operation as $\phi \oplus \psi \stackrel{\text{def}}{=} (\phi \wedge \neg\psi) \vee (\neg\phi \wedge \psi)$ and we use $\bigoplus_{\phi\in\Phi} \phi$ to express that ϕ is true for exactly one of the members of Φ. We use $\top$ and $\bot$ as symbols for absolute truth and falsehood. For any logical structure s, the semantics $s \models \top$ and $s \not\models \bot$ hold. By the term *sentence*, we refer to a logical formula in which every variable is bound to a quantifier. Unbound variables are *free*, and we write $\widehat{x}_n$ as shorthand for a tuple $x_1, \ldots, x_n$ of first-order variables (resp. $\widehat{X}_n$ for second-order variables).

An MSO formula ϕ is *satisfied/modeled* by a word w if it holds true when interpreted with respect to w as a logical structure. We write $w \models \phi$ to capture this notion. The formula ϕ specifies or recognizes the language L if, and only if $w \models \phi$ for all $w \in L$. The classical Büchi-Elgot-Trakhtenbrot theorem [14, 19, 29] establishes the equivalence of MSO-definable and ω-regular languages.

We will use the following shorthand in the rest of the paper: We write expression $x = y$ for $\neg(x < y) \wedge \neg(y < x)$, expression $\mathsf{edge}(x, y)$ for $x < y \wedge \neg\exists z.\ x < z \wedge z < y$, expression $\mathsf{first}(x)$ and $\mathsf{last}(x)$ for $\neg\exists y.\ y < x$ and $\neg\exists y.\ x < y$, respectively.

Example 4. Consider the priority queue from Example 1. We can express the regular language of the configuration space in MSO using the sentence pq defined as:

$$\forall x.\ \big(\mathsf{first}(x) \Leftrightarrow \bigvee_{q\in Q} q(x)\big) \wedge \Big(\exists y.\ \big(\mathsf{edge}(x,y) \wedge \neg\mathsf{first}(x)\big) \Rightarrow \bigvee_{p\in P} \bigvee_{m,m'\in M} (p,m)(x) \wedge \big((p,m')(y) \vee (p{+}1,m')(y)\big)\Big)$$

where $(p + 1)$ indicates the next most significant priority ranking. Any word satisfying pq is a valid configuration in the system.

MSO-Definable Functions. An deterministic MSO *transducer* (ωMSOT) is a tuple

$$T = \Big(\Sigma, \Gamma, \mathsf{dom}, C, \big(\phi^c_\gamma(x)\big)^{c\in C}_{\gamma\in\Gamma}, \big(\psi^{c,d}(x,y)\big)^{c,d\in C}\Big),$$

where Σ and Γ are input and output alphabets, $C = \{1, 2, ..., n\}$ is a finite set of indices, and every ϕ^c_γ, every $\psi^{(c,d)}$, and dom are MSO formulae such that

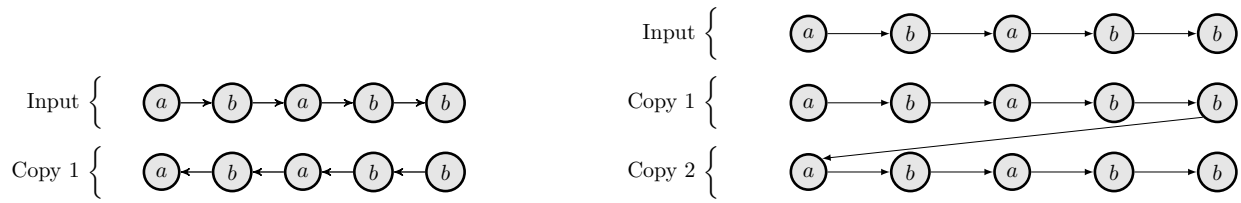

Figure 2: The MSO transformations reverse (left) and copy (right) on input word *ababb*.

- the sentence dom defines the domain of the ωMSOT,
- the *node formulae* $\left(\phi^c_\gamma(x)\right)^{c\in C}_{\gamma\in\Gamma}$ specify the labels on nodes in the output graph, and
- the *edge formulae* $\left(\psi^{c,d}(x,y)\right)^{c,d\in C}$ specify edge connectivity in the output graph.

For a word w such that $w \models \mathsf{dom}$, the transducer operates over the graph sum or disjoint union of the string graph of w with itself, for every index in C. This sum can be viewed as n disjoint copies of w, each indexed by a number in C. The node formulae then determine which nodes will be present in the output and how they are relabeled in the output, and the edge formulae add, remove, and rearrange the connections between nodes. Each formula ϕ^n_γ has a single free variable and should be interpreted in such a way that if a position satisfies ϕ^n_γ, then that position will be labeled by the symbol γ in the nth 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.

Example 5. We give MSO definitions of the regular functions reverse and copy discussed in the introduction. The MSOT in Equation 1 defines reverse, while the MSOT in Equation 2 defines copy. The corresponding transformations are depicted in Figure 2.

$$\begin{aligned} \Sigma = \Gamma &= \{a,b\} \\ C &= \{1\} \\ \mathsf{dom} &\stackrel{\text{def}}{=} \top \end{aligned} \qquad \begin{aligned} \phi^1_a(x) &\stackrel{\text{def}}{=} a(x) \\ \phi^1_b(x) &\stackrel{\text{def}}{=} b(x) \end{aligned} \qquad \psi^{(1,1)}(x,y) \stackrel{\text{def}}{=} \mathsf{edge}(y,x) \tag{1}$$

$$\begin{aligned} \Sigma = \Gamma &= \{a,b\} \\ C &= \{1,2\} \\ \mathsf{dom} &\stackrel{\text{def}}{=} \top \end{aligned} \qquad \begin{aligned} \phi^1_a(x) &\stackrel{\text{def}}{=} \phi^2_a(x) \stackrel{\text{def}}{=} a(x) \\ \phi^1_b(x) &\stackrel{\text{def}}{=} \phi^2_b(x) \stackrel{\text{def}}{=} b(x) \end{aligned} \qquad \begin{aligned} \psi^{(1,1)}(x,y) &\stackrel{\text{def}}{=} \psi^{(2,2)}(x,y) \stackrel{\text{def}}{=} \mathsf{edge}(x,y) \\ \psi^{(1,2)}(x,y) &\stackrel{\text{def}}{=} \mathsf{last}(x) \wedge \mathsf{first}(y) \\ \psi^{(2,1)} &\stackrel{\text{def}}{=} \bot \end{aligned} \tag{2}$$

MSO-Definable Relations. The transducers defined in the previous paragraph are completely deterministic. This is because the definition gives no mechanism to facilitate nondeterministic choice. Nondeterminism is introduced to the model by modifying the definition such that dom is no longer required to be a sentence, and the parameters of dom are also parameters for the node and edge formulae. By specifying these formulae with free variables, the action of nondeterministic choice can be realized by the choice of interpretation or valuation of these variables in terms of sets of positions in the input.

A *nondeterministic monadic second-order transducer* (ωNMSOT) is a tuple

$$T = \left(\Sigma, \Gamma, \mathsf{dom}(\widehat{X}_n), C, \left(\phi^c_\gamma(x, \widehat{X}_n)\right)^{c\in C}_{\gamma\in\Gamma}, \left(\psi^{c,d}(x,y,\widehat{X}_n)\right)^{c,d\in C}\right)$$

where all formulae are parameterized by some number of free set variables in addition to the first-order parameters of their deterministic counterparts. For a given input word, two distinct valuations of $\widehat{X}_n$ may both satisfy the domain formula and result in distinct outputs, and this allows for nondeterministic choice. While deterministic ωMSOTs can define only functional relations, ωNMSOTs can define a broader class of relations that are non-functional in general. It is also easy to see that every ωMSOT is an ωNMSOT in which $n = 0$ for $\widehat{X}_n$. An important result for ωNMSOTs is their closure under composition, which is particularly relevant for our model checking procedure.

Theorem 2 (Courcelle-Engelfriet [17]). *For any two ωNMSOTs, T and T', the composition $T \circ T'$ is an ωNMSOT.*

Since MSO-definability is often used [5, 4, 6] as a yardstick for regularity, we call the class of ωMSOT definable transformations the *regular functions/transformations* and the class of ωNMSOT definable transductions the *regular relations*.

4 Machine models accepting Regular Relations

4.1 Transducers on finite strings

A *finite state transducer* (FST) is a machine model of string relations similar to how DFAs are machine models of languages. The most basic kind of such machine is the *generalized sequential machine* (GSM). GSMs generalize DFAs by including a write-only output tape in addition to the input tape, and by allowing the machine to append a string on this tape upon every transition. The relation given by a GSM, G, is defined as $\{(w, x) \ : \ w \in \mathcal{L}(\mathcal{A})\}$ where $\mathcal{A}$ is the underlying automaton (inferred by ignoring all output directives in G) and x is the contents of the output tape after an accepting run on the input w. It is well known that nondeterministic GSMs (NGSM) are more expressive than their deterministic counterparts. The set of functions (relations) definable by a GSM (NGSM) is called the *rational functions* (*rational relations*).

A generalization of the GSM is the *two-way generalized sequential machine* (2GSM), in which the machine is allowed to make multiple passes in any direction over the input string. In this model, the reading head can arbitrarily change direction and read various substrings of the input repeatedly. To achieve this, the input is wrapped by end markers (so that the machine can know when it must change direction) and an extra dimension is added to the transitions to indicate which position the input head should read next. nondeterminism in this model again adds expressive power. Every GSM is 2GSM and every NGSM is a N2GSM, but not every NGSM is equivalent to a 2GSM.

An alternative generalization of the GSM is the model of *streaming string transducers* (SST), in which the underlying automaton is augmented with a set of write-only registers which can store partial outputs. Much like the GSM, an SST reads its input in a single pass, but instead of writing to the output tape on transitions it updates its registers when changing state. If the machine accepts an input, then there is output function that determines, based on the final state, how to combine the contents of the registers to form the final output string. As is the norm for transducers, nondeterministic SSTs (NSST) are more expressive than deterministic SSTs.

Theorem 3 (Engelfriet and Hoogeboom [20] and Alur et al. [4, 5]). *The following results summarize connections between various models for transformations on finite strings.*

- *A transformation of finite strings is* MSOT*-definable if and only if it* 2GSM*-definable.*
- *A transformation of finite strings is* MSOT*-definable if and only if it* SST*-definable.*
- *A transformation of finite strings is* NMSOT*-definable if and only if it* NSST*-definable.*

4.2 Streaming Streaming Transducers on ω-Strings

Alur et al. [6] introduced deterministic SSTs on infinite strings with Muller acceptance condition, and showed that the set of functions definable by deterministic MSO on ω-words coincides with the set of functions definable by deterministic Muller SSTs.

Theorem 4 (Alur, Filiot, and Trivedi [6]). *A transformation of infinite strings is definable as a deterministic Muller* SST *iff it is definable as a deterministic* ωMSO *transduction.*

We introduce nondeterministic ωSSTs with both Büchi and Muller acceptance conditions, and prove an analogous result in this setting.

Let X be finite set of string variables and Γ be an alphabet. A variable assignment α over X is a mapping $\alpha : X \to (\Gamma \cup X)^*$. A valuation is a function $\alpha : X \to \Gamma^*$. Any assignment α can be extended

to $\hat{\alpha} : (\Gamma \cup X)^* \to (\Gamma \cup X)^*$ in a straightforward manner. The composition $\alpha_1\alpha_2$ of two assignments α_1 and α_2 is defined by the function composition $\hat{\alpha}_1\alpha_2$, i.e. $\hat{\alpha}_1\alpha_2(x) = \hat{\alpha}_1(\alpha_2(x))$ for all $x \in X$. We say that a string $u \in (\Gamma \cup X)^*$ is *copyless* if each $x \in X$ occurs at most once in u. An assignment α is copyless if $\hat{\alpha}(u)$ is copyless, for all copyless $u \in (\Gamma \cup X)^*$. We write A_X for the set of copyless assignments over X. We say that a variable $x \in X$ is *appended* (resp. *prepended*) in an assignment α if $\alpha(x) := x(\Gamma \cup X)^*$ (resp. $\alpha(x) := (\Gamma \cup X)^*x$).

Definition 3 (Nondeterministic Streaming String Transducer)**.** A nondeterministic streaming string transducer over ω-strings (ωNSST) is a tuple $T = (Q, \Sigma, \Gamma, X, E, q_0, \mathsf{Acc}, f)$ where

- Q is a finite set of states,
- Σ and Γ are input and output alphabets, respectively,
- X is a finite set of string variables,
- $E \subseteq Q \times \Sigma \times A_X \times Q$ is the finite set of transitions,
- $q_0 \in Q$ is the initial state,
- Acc is the *acceptance condition*, and
- $f \in X$ is an append-only output variable.

We consider both Büchi and Muller acceptance conditions for ωNSSTs and reference these classes of machines by the initialism DBST and DMST, respectively. The notion of a run and of an accepting run for a ωNSST with each kind of condition is defined in an analogous manner to that of an ω-automaton. The notion of deterministic ωSST is also specified in a fashion similar to deterministic ω-automata.

The sequence $\langle \alpha_{r,i} \rangle_{0 \leq i}$ of assignments induced by a run $r = r_0, w_0, r_1, w_1, \ldots$ is given inductively as the following: $\alpha_{r,i} = \hat{\alpha}\alpha_{r,i-1}$ for $0 < i$ and $\alpha_{r,0}(x) = \varepsilon$ for any $x \in X$. The output $T(r)$ of a run r of T is well-defined only if r is an accepting run and equals:

$$T(r) \stackrel{\text{def}}{=} \lim_{i \to \infty} \langle \alpha_{r,i}(f) \rangle.$$

Since the output variable f is only ever appended to and never prepended, the limit exists and is an ω-string. The transformation $[\![T]\!]$ realized by an ωNSST T is the relation

$$[\![T]\!] = \{(w, T(r)) \,:\, r \text{ is an accepting run of } T \text{ over } w\}.$$

We say that an NSST T is *functional* if $[\![T]\!]$ is a partial function, i.e. for all $w \in \Sigma^\omega$, $\{w' \,:\, (w, w') \in [\![T]\!]\}$ has cardinality at most 1.

Theorem 5. *A transformation is definable as an* ωNSST *with a Büchi acceptance condition iff it is definable as an* ωNSST *with a Muller acceptance condition.*

The equivalence of NBST-definable transformations and NMST-definable transformations follows from a straightforward application of the equivalence of NBA and NMA since no changes to the variable assignments are required. Equality of expressiveness with these acceptance conditions in transducers allows us to switch between them whenever convenient.

Remark 1. Observe that DMSTs and functional NMSTs, both of which were introduced in [6], have a slightly different output mechanism, which is defined as $F : 2^Q \rightharpoonup X^*$ such that for all $P \in \mathsf{dom}(F)$ the string $F(P)$ is copyless and of the form $x_1 \ldots x_n$, and for $q, q' \in P$ and $a \in \Sigma$ s.t. $q' = \delta(q, a)$ we have

- $\rho(q, a)(x_i) = x_i$ for all $i < n$ and
- $\rho(q, a)(x_n) = x_n u$ for some $u \in (\Gamma \cup X)^*$.

In contrast, our definition has a unique append-only output variable $f \in X$. However, for functional NMSTs, our model 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 P will be visited infinitely often. 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_P of the transducer where the only accepting set of states is P. If any state outside the set P 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_P makes a transition to a rejecting sink state.

Alur et al. [6] showed the equivalence of functional NMST with DMST. This result and Theorem 5 imply that the class of transformations definable using *functional* NMST or *functional* NBST (in our definition) are precisely deterministic ωMSOT-definable.

4.3 Equivalence of ωNMSOT and ωNSST

Alur et al. [6] showed that DMSTs are equivalent in expressiveness to deterministic MSO-definable transformations. Similarly, Alur and Deshmukh [5] showed that these models are equivalent in expressiveness for their nondeterministic, finite string definitions as well. We generalize both of these results by proving the following theorem.

Theorem 6. *A transformation is* ωNMSOT*-definable if, and only if, it is* ωNSST*-definable.*

The proof of Theorem 6 is in two parts. In the first part (Lemma 1), we show that every ωNSST is equivalent to the composition of a nondeterministic relabeling and a deterministic ωSST, i.e. $\omega\text{NSST} = \omega\text{SST} \circ R$. Our proof follows the structure of a similar result in [5]. In the second part (Lemma 2), we show that every ωNMSOT is equivalent to the composition of a non-functional relabeling and a deterministic ωMSOT, i.e. $\omega\text{NMSOT} = \omega\text{MSOT} \circ R$. The proof of this portion follows the approach of the proof for similar result in [20]. These two lemmas, in conjunction with Theorem 4, allow us to equate these two models of transformation via a simple assignment.

We call a transformation $\lambda : \Sigma^\omega \times \Gamma^\omega$ a *relabeling*, if there exists another relation $\lambda' : \Sigma \times \Gamma$ such that $(aw, bv) \in \lambda$ when $(a, b) \in \lambda'$ and $(w, v) \in \lambda$. In other words, λ is a letter-to-letter relation, λ', lifted in a straight-forward manner to ω-words. Let R be the set of all such relabelings.

Lemma 1. $\omega\text{NSST} = \omega\text{SST} \circ R$.

Proof. Let $S : \Pi^\omega \to \Gamma^\omega$ be a ωSST, and $\lambda : \Sigma^\omega \times \Pi^\omega$ a relabeling extending the relation $\lambda' : \Sigma \times \Pi$. We can construct a ωNSST, $T : \Sigma^\omega \times \Gamma^\omega$ such that $T = S \circ \lambda$ in the following way. Begin with $T = S$, and, for every transition in S of the form $q \xrightarrow[\alpha]{\pi} p$, substitute in the set of transitions $\{q \xrightarrow[\alpha]{\sigma} p \ : \ (\sigma, \pi) \in \lambda'\}$. In general, the resulting machine T is nondeterministic, since the relabeling λ is nondeterministic (i.e. non-functional) in general. If λ' maps two distinct symbols from Σ to single symbol from Π, then λ might pass along to S many outputs for a single input. All possible relabelings are taken into account by replacing single transitions with multiple transitions, and each distinct run of T on an input w corresponds to a distinct relabeling of the input which is then run through S. Thus, we have $T = S \circ \lambda$, and since S and λ were chosen arbitrarily, we have shown that $\omega\text{SST} \circ R \subseteq \omega\text{NSST}$.

For the opposite inclusion, suppose that $T : \Sigma^\omega \times \Gamma^\omega$ is a ωNSST. From T, we can construct a nondeterministic, letter-to-letter relation $\lambda' : \Sigma \times \Pi$, where $\Pi = Q_T$. If there is a transition in T of the form $q \xrightarrow[\alpha]{\sigma} p$, then $(\sigma, p) \in \lambda'$, and $\lambda : \Sigma^\omega \times \Pi^\omega$ is the associated relabeling. Now, define $S : \Pi^\omega \to \Gamma^\omega$ as an ωSST such that every set of transitions $\{q \xrightarrow[\alpha]{\sigma} p \in T\}$, for fixed q, p, and α, is replaced by a single transition $q \xrightarrow[\alpha]{p} p$. This construction is essentially the first portion of the proof in reverse, and so we have shown that $\omega\text{NSST} \subseteq \omega\text{SST} \circ R$. □

Lemma 2. $\omega\text{NMSOT} = \omega\text{MSOT} \circ R$.

Proof. We first show that $\omega\text{NMSOT} \subseteq \omega\text{MSOT} \circ R$. Suppose that T is a ωNMSOT with n free set variables $\widehat{X}_n$. nondeterminism in T is determined by the valuation of the free variables $\widehat{X}_n$, and this valuation can be summarized via a preprocessing step. Define the letter-to-letter relation $\lambda' \subseteq \Sigma \times \{0, 1\}^n$ and its extended relabeling $\lambda \subseteq \Sigma^\omega \times (\{0, 1\}^n)^\omega$, where $(\sigma, \beta) \in \lambda'$, for all possible $\beta \in \{0, 1\}^n$. For any input word, w, the set

$\lambda(w)$ represents the input word interpreted over all possible valuations for the free parameters. Now, we can construct $S : \Sigma^\omega \times (\{0,1\}^n)^\omega \to \Gamma^\omega$, an ωMSOT, that operates over the language induced by λ. For every occurrence of a sub-formula of the form $x \in X_i$ in T, we insert the formula $\bigvee_{\beta \in \{0,1\}^n \wedge \beta[i]=1} (\sigma, \beta)(x)$ in S, and all other sub-formulae are copied exactly from T. Since λ relabels the input string in a way that encodes the valuations of $\widehat{X}_n$ and S takes this into account via the above disjunction, we have the equality $T = S \circ \lambda$. Since the components of this equality where selected arbitrarily, we conclude that $\omega\text{NMSOT} \subseteq \omega\text{MSOT} \circ R$.

The converse inclusion is much simpler. Every relabeling in R is ωNMSOT-definable, since a relabeling is a nondeterministic, rational transformation. Because the relabeling relation has many outputs for a single input and each such output is fed to a deterministic ωSST in the sequential composition, the resulting transformation is both ωNMSOT-definable and nondeterministic. Thus, we can conclude that $\omega\text{MSOT} \circ R \subseteq \omega\text{NMSOT}$, and finally that $\omega\text{NSST} = \omega\text{MSOT} \circ R$. □

In conjunction, Theorem 4 and Lemmas 1 and 2 allow for the derivation

$$\omega\text{NMSOT} = \omega\text{MSOT} \circ R = \omega\text{SST} \circ R = \omega\text{NSST}$$

thereby proving Theorem 6.

5 MSO-Definable Regular Model Checking

Theorem 6 enables MSO-definable regular model checking framework, and allows us to computationally tackle the regular model checking problem when transitions are encoded as nondeterministic regular relations. Using a reduction from two-counter machines, we show that MSO-RMC is undecidable. This is not surprising, as the problem is already undecidable for transition functions over finite strings expressed as GSMs. We also show decidability of the regular type checking problem for ωNSSTs in Section 5.2 and exploit this result to provide an algorithm for a bounded variant of MSO-definable model checking.

5.1 Undecidability of MSO-Definable Regular Model Checking

We show that the transitive closure of an arbitrary regular relation is not computable. The proof is a reduction from the halting problem for two-counter machines to computing the transitive closure of a NMSOT.

A *two-counter machine* $\mathcal{M}$ is a tuple (L, C) where: $L = \{\ell_1, \ell_2, \ldots, \ell_{n-1}, \ell_{halt}\}$ is the set of instructions and $C = \{c_1, c_2\}$ is the set of two *counters*. There is a distinguished terminal instruction ℓ_{halt} called HALT and the instructions in L are one of the following types (here $c \in C$, $\ell_i, \ell_k, \ell_m \in L$):

1. `increment`. $\ell_i : c := c + 1$; goto ℓ_k,
2. `decrement`. $\ell_i : c := c - 1$; goto ℓ_k,
3. `zero-test`. ℓ_i : if (c>0) then goto ℓ_k else goto ℓ_m,
4. `halt`. ℓ_{halt} : HALT.

Let I, D, O represent the set of increment, decrement and zero-check instructions s.t. $L = I \cup D \cup O$. A configuration of a two-counter machine is a tuple (ℓ, c, d) where $\ell \in L$ is an instruction, and c, d are natural numbers that specify the value of counters c_1 and c_2, respectively. The initial configuration is $(\ell_1, 0, 0)$. A run of a two-counter machine is a (finite or infinite) sequence of configurations $\langle k_1, k_2, \ldots \rangle$ where k_1 is the initial configuration, and the relation $\vdash$ between subsequent configurations is governed by transitions between respective instructions. Note that a two-counter machine has exactly one run starting from the initial configuration. We assume without loss of generality that ℓ_0 is an increment instruction. Clearly, it cannot be a decrement instruction. If ℓ_1 were a zero check instruction, we add two dummy instructions ℓ_1', ℓ_1'' such that ℓ_1' increments a counter, and ℓ_1'' decrements it, and passes control to ℓ_1. The dummy instructions ℓ_1', ℓ_1'' will never again be encountered in the two counter machine after control passes to ℓ_1. The *halting problem* for a two-counter machine asks whether its unique run ends at the terminal instruction ℓ_{halt}. It is well known that the halting problem for two-counter machines is undecidable [26].

Theorem 7. *The transitive closure of a regular relation is uncomputable in general.*

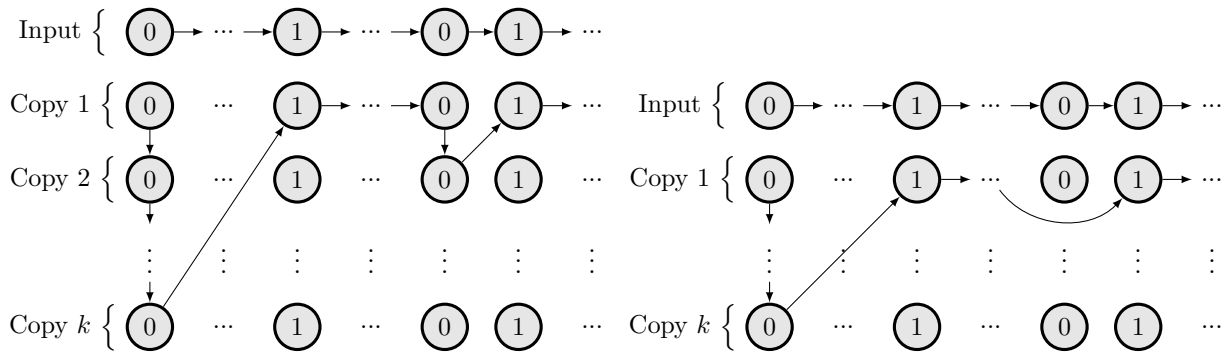

Figure 3: Partial views of the transformations for `increment` (left) and `decrement` (right).

Proof. Let our alphabet be $\Sigma = \{0, 1\}$ and let $\mathcal{M} = (L, C)$ be a two-counter machine. A configuration $k_i = (\ell_i, c, d)$ of $\mathcal{M}$ can be represented as the string $0^i 1 0^c 1 0^d$, where the substrings 0^c and 0^d encode the values of the counters in unary and 0^i encodes the index of the current instruction. The initial configuration is then 011. The outline of the proof is the following. We will show that every instruction of a two-counter machine is definable as an MSOT. As a consequence, it follows that every computation of a two-counter machine is the composition of some sequence of these MSOTs, starting from the initial word 011. To test if a given Minsky machine halts, we could then construct the transitive closure of the union of all instructions and test if it is defined on the input 011.

We begin by defining some auxiliary formulae that will make the subsequent pieces of the proof easier to understand.

$$\begin{aligned} \mathsf{instr}(x) &\equiv \neg\exists y.\ 1(y) \wedge y < x \\ \mathcal{C}(x) &\equiv \exists y, z.\ 1(y) \wedge 1(z) \wedge y < x \wedge x < z \\ \mathcal{D}(x) &\equiv \neg\exists y.\ 1(y) \wedge x < y \end{aligned} \tag{3}$$

The following components are common to the MSOTs for each type of Minsky machine instruction. Because every MSOT in question has these components, we only specify their edge formulae individually. The definitions of the edge formulae for each type of instruction are given in Figure 5, and the corresponding transformations are shown in Figures 3 and 4.

$$\begin{aligned} &\Sigma = \Gamma = \{0, 1\} && \forall c \in C.\ \phi_0^c(x) \equiv \phi_0^c \equiv 0(x) && \mathsf{dom} \equiv \exists x, y.\ \forall z.\ 1(x) \wedge 1(y) \wedge 0(z) \\ &C = \{1, ..., k\} && \forall c \in C.\ \phi_1^c(x) \equiv \phi_1^c \equiv 1(x) && \qquad \wedge \big(\mathsf{instr}(z) \oplus \mathcal{C}(z) \oplus \mathcal{D}(z)\big) \end{aligned}$$

For the zero-test instruction, one must specify a distinct MSOT for each possible branching result (for every pair of instructions ℓ_k and ℓ_m), so that each one has a fixed copy set. Instead of giving these explicitly,

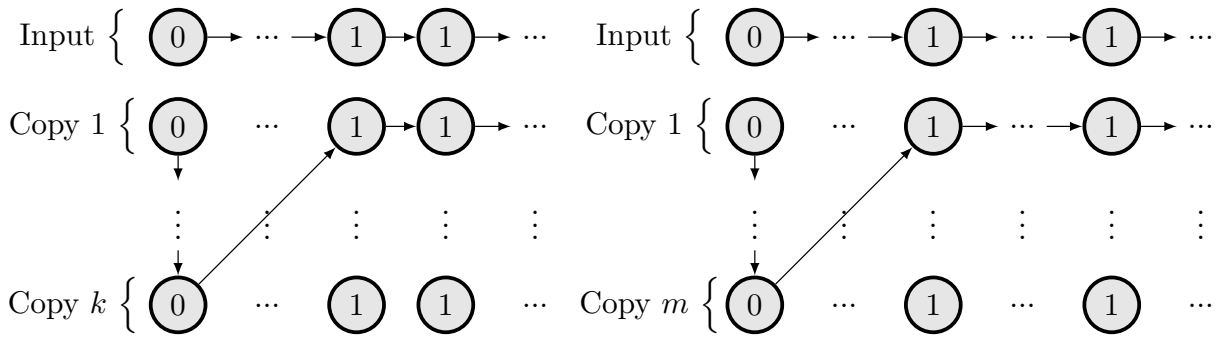

Figure 4: Partial views of the branching instruction, for both possible cases.

$$\underline{\texttt{goto } \ell_k} \begin{cases} \forall i < k.\ \psi^{(i,i+1)}(x,y) \equiv \mathsf{first}(x) \wedge \mathsf{first}(y) \\ \psi^{(k,1)}(x,y) \equiv \mathsf{first}(x) \wedge 1(y) \wedge \big(\forall z.\ z < y \iff \mathsf{instr}(z)\big) \end{cases}$$

$$\underline{\ell_i : c := c+1} \begin{cases} \psi^{(1,1)}(x,y) \equiv \mathsf{edge}(x,y) \wedge \big(\mathcal{C}(x) \oplus \mathcal{D}(x)\big) \\ \psi^{(1,2)}(x,y) \equiv \mathcal{C}(x) \wedge x = y \wedge \exists z.\ \mathsf{edge}(x,z) \wedge 1(z) \\ \psi^{(2,1)}(x,y) \equiv \mathcal{C}(x) \wedge \exists z.\ \mathsf{edge}(x,z) \wedge 1(z) \wedge y = z \end{cases}$$

$$\underline{\ell_i : c := c-1} \begin{cases} \psi^{(1,1)}(x,y) \equiv \Big(\mathsf{edge}(x,y) \wedge \big(\mathcal{C}(x) \vee \mathcal{D}(y)\big) \wedge \neg\exists z.\ \mathsf{edge}(y,z) \wedge 1(z)\Big) \\ \qquad\qquad \oplus\Big(\mathcal{C}(x) \wedge 1(y) \wedge \exists z.\ \mathsf{edge}(x,z) \wedge \mathsf{edge}(z,y)\Big) \end{cases}$$

$$\underline{\ell_i :\ \texttt{if } (c{>}0) \texttt{ then goto } \ell_k \texttt{ else goto } \ell_m} \Big\{ \quad \psi^{(1,1)}(x,y) \equiv \mathsf{edge}(x,y) \wedge \neg\mathsf{instr}(x)$$

Figure 5: MSO specifications for each type of counter machine instruction.

we provide the general form of the goto instruction, and simply require that the MSOT that handles branching to ℓ_k or ℓ_m has the larger number between k and m as its number of copies.

Any computation of $\mathcal{M}$ is of the form $k_1 \vdash k_2 \vdash ... \vdash k_{halt}$ if the run is halting and $k_1 \vdash k_2 \vdash ...$ otherwise. We can view the relation $\vdash$ as a the union of all the instructions in L, and thus as the union of the MSOTs encoding this instruction. Suppose that T is the union of all of these transducers. In this way, we rephrase arbitrary computations of $\mathcal{M}$ as $011Tw_2T...Tw_{halt}$ or $011Tw_1Tw_2T...$ for halting and non-halting runs, respectively. In more standard notation this is expressed $T(...(T(T(011)))...)$. Thus, if we were able to compute the transitive closure T^*, then we would be able to solve the halting problem for two-counter machines by checking whether $T^*(011)$ is defined or not. Since the halting problem for two-counter machines is undecidable, we have proven T^* to be uncomputable. □

5.2 Decidability of Type Checking

Given an NBST $T = (Q, \Sigma, \Gamma, X, E, q_0, F, f)$ with $F \subseteq Q$ as the set of accepting states, two ω-regular languages I and O given as NBAs $\mathcal{A}_I$ and $\mathcal{A}_O$, respectively, the *type-checking problem* is to decide whether for all strings $u \in I$ we have that $T(u) \subseteq O$.

Theorem 8. *Regular type checking is decidable for* ωNSST*s.*

Proof. We first check whether T is defined for all strings $u \in I$, i.e. $I \subseteq dom(T)$. An NBW that recognizes the domain of T can be constructed in linear time, and therefore $I \subseteq \mathsf{dom}(T)$ can be checked in PSPACE.

We now assume that T is defined on I. We construct a nondeterministic Büchi automaton $\mathcal{A}$ such that $L(\mathcal{A}) = \{w \in I \ : \ \exists w' \in T(u) \text{ s.t. } w' \notin O\}$. First, we construct a NBW for $\overline{O}$. Suppose that its set of states is $Q_{\overline{O}}$ and its set of accepting sets is $F_{\overline{O}}$. Similarly, let Q_I and F_I be the set of states and accepting sets of NBW $\mathcal{A}_I$. Finally assume that Q and F denote the set of states and output function of T respectively.

The automaton $\mathcal{A}$ simulates I, T and $\overline{O}$ (on output strings) in parallel. It is defined as the product of I, T (without the output mechanism) and an automaton that computes set of all the possible evaluations of variables of T by $\overline{O}$. The automaton $\mathcal{A}$ maintains a set of functions $\mathbb{T} \subseteq [Q_{\overline{O}} \times X \to Q_{\overline{O}} \cup \{\bot\}]$ such that for all $\tau \in \mathbb{T}$ there is a run of T such that for all $(q, x) \in Q_{\overline{O}} \times X$ the entry $\tau(q, x)$ gives the state of $\overline{O}$ after reading the content of x starting from q, and $\bot$ if no run exists. This information can be easily updated along the run of $\mathcal{A}$. For instance, if a transition of T updates x as ayx, all of the functions $\tau \in \mathbb{T}$ are updated for all states q by $\tau'(q, x) = \tau(\tau(q', y), x)$ where (q, a, q') is a transition of $\overline{O}$.

Moreover, since it is not necessary that the product visits accepting states of $\mathcal{A}_I$, $\mathcal{A}_{\overline{O}}$ and T simultaneously, we use an idea similar to the intersection of NBAs to have a circular layered construction

Algorithm 1: Bounded Model Checking

Data: ωNMSOT T, sets of initial and bad states I and B, and a bound k
Result: whether a system is safe within k transitions

1 compute $\overline{B}$;
2 let $i = 0$;
3 **while** $i \leq k$ **do**
4 | **if** $T^i(I) \not\subseteq \overline{B}$ **then**
5 | | **return** `unsafe`;
6 | **end**
7 | **if** T `is functional and` $T^{i-1} = T^i$ **then**
8 | | **return** `universally safe`;
9 | **end**
10 | let $i = i + 1$;
11 **end**
12 **return** `safe within` n `transitions`;

where after seeing an accepting state of one component, the automaton makes a transition to the next layer with going back to the first layer after seeing the last one. Therefore, the set of states of $\mathcal{A}$ is $Q_{\mathcal{A}} = Q_I \times Q \times 2^{[Q_{\overline{O}} \times X \to Q_{\overline{O}} \cup \{\bot\}]} \times \{0, 1, 2\}$. The final component of the state is keeping track of the accepting states visited in Q_I, Q, and $\overline{O}$ using the idea just described. The accepting set of $\mathcal{A}$ is defined by the tuples $(p, q, \mathbb{T}, 2) \in Q_{\mathcal{A}}$ such that if q_0 is the initial state of $\overline{O}$ then $\tau(q_0, f) \in F_{\overline{O}}$ for some $\tau \in \mathbb{T}$. The size of $\mathcal{A}$ is doubly exponential in I, O and T, and its emptiness can be tested in EXPSPACE. □

5.3 Bounded MSO-Regular Model Checking

Definition 4 (Bounded MSO-RMC)**.** Given an ωNMSOT T, two ω-regular languages I and B, and a bound $k \geq 0$, the bounded MSO-RMC problem is to decide whether for all $w \in I$ and $i \leq k$ if $T^i(w) \notin B$.

The bounded model checking procedure is shown in Algorithm 1. At line 4, it uses regular type-checking subroutines discussed in the previous subsection to decide whether $T^i(I) \not\subseteq \overline{B}$. At every iteration, the algorithm computes compositions of the i-step transition relation T^i with the transition relation T itself. This procedure can be performed at the ωNMSOTlevel. Finally, if the transducer T is functional, the check for $T^{i-1} = T^i$ is decidable [6] for NSSTs on ω-strings. Moreover, a recent result by Muscholl and Puppis [27] showed that for bounded valued SSTs on finite strings, the equivalence remains decidable. For these classes, at every iteration we can decide equivalence between T^n and T^{n+1}, and if we find that the machines define the same relation, then a fixed point has been reached and $T^n = T^{n+1} = T^*$ is the actual transitive closure. If this is the case, then the system is proven universally safe assuming the initial configurations given by I are fixed.

6 Conclusion

We presented a characterization of ω-regular relations by introducing two equi-expressive formalisms capable of expressing infinite string-to-string relations, namely nondeterministic MSO-definable transducers on ω-strings (ωNMSOT) and nondeterministic streaming string transducers on ω-strings (ωNSST). The regular type checking problem was shown to be decidable for arbitrary ω-regular relations.

Based on these results, we introduced a generalization of regular model checking to encompass the class of regular transition relations. Since the language expressing the sets of states and the relations expressing the transition relations are characterized by MSO definability, this extended framework can be thought of as a "more regular" version of classical *regular model checking*. By allowing for the expression of all parts of a model checking problem — the state space, initial states, bad states, and transition relation — to be encoded in MSO, we have a single language in which to define verification problems. Moreover, we have

shown that there are interesting systems with regular but non-rational transition relations, and MSO-RMC yields a new approach to verification of these types of systems.

One possibility for further work is the extension of the algorithmic and heuristic methods that have been developed over the last two decades to MSO-RMC. Techniques involving widening [12, 28], extrapolation [3, 24], and abstraction [11] present themselves as methods that could potentially translate to our more general framework. Additionally, there is the avenue of applying MSO-RMC to the line of research championed by Boigelot and Wolper around representing arithmetical systems in automata-based frameworks. If it is possible to represent linear arithmetic over the field of reals with MSO-definable functions on ω-words, then a new avenue of verification would be available for a large class of dynamical systems which are currently out of reach for RMC.

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

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

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

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

[15] Joakim Byg and Kenneth Yrke Jørgensen. Regular model checking and verification of cellular automata. 2008.

[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] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *In Transactions of the American Mathematical Society*, 98(1):21–51, 1961.

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

[21] Peter Habermehl and Tom 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).

[22] 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.

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

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

[25] Axel Legay and Pierre Wolper. On (omega-)regular model checking. *ACM Trans. Comput. Logic*, 12, 2010.

[26] Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.

[27] Anca Muscholl and Gabriele Puppis. Equivalence of finite-valued streaming string transducers is decidable. *CoRR*, abs/1902.06973, 2019.

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

[29] B. A. Trakhtenbrot. Finite automata and monadic second order logic. *Siberian Mathematical Journal*, 3:101–131, 1962.

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

A Examples from the Paper

A.1 MSOT definitions for Example 3

$$\Sigma = \Gamma = Q \cup \{0, 1\}$$
$$C = \{1, 2\}$$
$$\mathsf{dom} \equiv \exists X, Y.\ \forall x.\ \Big(x \in X \iff \bigvee_{q \in Q} q(x)\Big) \wedge \Big(x \in Y \iff 0(x) \vee 1(x)\Big)$$
$$\wedge \Big(\mathsf{first}(x) \implies x \in X\Big) \wedge \Big(\mathsf{last}(x) \implies x \in Y\Big)$$

$$\phi_0^1(x) \equiv 0(x)$$
$$\phi_1^1(x) \equiv 1(x)$$
$$\phi_0^2(x) \equiv 0(x) \wedge \neg\mathsf{first}(x)$$
$$\phi_1^2(x) \equiv 1(x) \vee \mathsf{first}(x)$$
$$\phi_q^1(x) \equiv \mathsf{first}(x)$$
$$\phi_q^2(x) \equiv \bot$$
$$\phi_p^1(x) \equiv \bot$$
$$\phi_p^2(x) \equiv \exists y.\ \mathsf{first}(y) \wedge \mathsf{edge}(y, x)$$

$$\psi^{(1,1)}(x, y) \equiv \exists z.\ \Big(\mathsf{first}(x) \wedge \mathsf{edge}(x, z) \wedge \mathsf{edge}(z, y)\Big) \oplus \mathsf{edge}(x, y)$$
$$\psi^{(1,2)}(x, y) \equiv \exists z.\ \mathsf{last}(x) \wedge \mathsf{first}(z) \wedge \mathsf{edge}(z, y)$$
$$\psi^{(2,1)}(x, y) \equiv \bot$$
$$\psi^{(2,2)}(x, y) \equiv \neg\mathsf{first}(x) \wedge \mathsf{edge}(x, y)$$