Decidable models of integer-manipulating programs with recursive parallelism

Matthew Hague and Anthony Widjaja Lin

[1] Royal Holloway, University of London, UK
[2] Yale-NUS College, Singapore

Abstract. We study safety verification for multithreaded programs with recursive parallelism (i.e. unbounded thread creation and recursion) as well as unbounded integer variables. Since the threads in each program configuration are structured in a hierarchical fashion, our model is state-extended ground-tree rewrite systems equipped with shared unbounded integer counters that can be incremented, decremented, and compared against an integer constant. Since the model is Turing-complete, we propose a decidable underapproximation. First, using a restriction similar to context-bounding, we underapproximate the global control by a weak global control (i.e. DAGs possibly with self-loops), thereby limiting the number of synchronisations between different threads. Second, we bound the number of reversals between non-decrementing and non-incrementing modes of the counters. Under this restriction, we show that reachability becomes NP-complete. In fact, it is poly-time reducible to satisfaction over existential Presburger formulas, which allows one to tap into highly optimised SMT solvers. Our decidable approximation strictly generalises known decidable models including (i) weakly-synchronised ground-tree rewrite systems, and (ii) synchronisation/reversal-bounded concurrent pushdown systems systems with counters. Finally, we show that, when equipped with reversal-bounded counters, relaxing the weak control restriction by the notion of senescence results in undecidability.

1 Introduction

Verification of multithreaded programs is well-known to be a challenging problem. One approach that has proven effective in addressing the problem is to bound the number of context switches [36, 38]. [Recall that a *context switch* occurs when the CPU switches from executing one thread to executing a different thread.] When the number of context switches is fixed, one may adopt pushdown systems as a model of a single thread and show that reachability for the concurrent extension of the abstraction (i.e. multi-pushdown systems) is NP-complete [38]. This result has paved the way for an efficient use of highly optimised SMT solvers in verifying concurrent programs (e.g. see [1, 18, 24]). Note that without bounding the number of context switches the model is undecidable [37].

In the past decade the work of Qadeer and Rehof [38] has spawned a lot of research in underapproximation techniques for verifying multithreaded programs, e.g., see [1, 2, 4, 5, 7, 14, 18, 20, 22, 24, 27, 28, 31, 33, 35, 40, 42] among many others.

Other than unbounded recursions, some of these results simultaneously address other sources of infinity, e.g., unbounded thread creation [5, 22, 31], unbounded integer variables [24], and unbounded FIFO queues [1, 2].

Contributions. In this paper we generalise existing underapproximation techniques [23, 31] so as to handle both shared unbounded integer variables and recursive parallelism (unbounded thread creation and unbounded recursions). The paper also provides a cleaner proof of the result in [24]: an NP upper bound for synchronisation/reversal-bounded reachability analysis of concurrent pushdown systems with counters. We describe the details below.

We adopt state-extended ground-tree rewrite systems (sGTRS) [31] as a model for multithreaded programs with recursive parallelism (e.g. programming constructs including `fork/join`, `parbegin/parend`, and `Parallel.For`). Ground-tree rewrite systems (GTRS) are known (see [21]) to strictly subsume other well-known sequential and concurrent models like pushdown systems [11], PA-processes [19], and PAD-processes [34], which are known to be suitable for analysing concurrent programs. [One may think of GTRS as an extension of PA and PAD processes with return values to parent threads [21].] We then equip sGTRS with unbounded integer counters that can be incremented, decremented, and compared against an integer constant.

Since our model is Turing-powerful, we provide an underapproximation of the model for which safety verification becomes decidable. First, we underapproximate the global control by a weak global control [26, 31] (i.e. DAGs possibly with self-loops), thereby limiting the number of synchronisations between different threads. To this end, we may simply unfold the *underlying control-state graph* of the sGTRS (see Section 3) in the standard way, while preserving self-loops. This type of underapproximation is similar to *loop acceleration* in the symbolic acceleration framework of [8]. Second, we bound the number of reversals between non-decrementing and non-incrementing modes of the counters [25]. Under these two restrictions, reachability is shown to be NP-complete; in fact, it is poly-time reducible to satisfaction over existential Presburger formulas, which allows one to tap into highly optimised SMT solvers. Our result strictly generalises the decidability (in fact, NP-completeness) of reachability for (i) weakly-synchronised ground-tree rewrite systems [31, 41], and (ii) synchronisation/reversal-bounded concurrent pushdown systems with counters [24].

Finally, we show one negative result that delineates the boundary of decidability. If we relax the weak control underapproximation by the notion of senescence (with age restrictions associated with nodes in the trees) [22], then the resulting model becomes undecidable.

Related Work. Recursively-parallel program analysis was analysed in detail by Bouajjani and Emmi [10]. However, in contrast to our systems, their model does not allow processes to communicate during execution. Instead, processes hold handles to other processes which allow them to wait on the completion of others, and obtain the return value. They show that when handles can be passed to child processes (during creation) then the state reachability problem is undecidable. When handles may only be returned from a child to its parent,

state reachability is decidable, with the complexity depending on which of a number of restrictions are imposed.

The work of Bouajjani and Emmi is closely related to branching vector addition systems [43] which can model a stack of counter values which can be incremented and decremented (if they remain non-negative), but not tested. While it is currently unknown whether reachability of a configuration is decidable, control-state reachability and boundedness are both 2ExpTime-complete [17].

Another variant of vector addition systems with recursion are pushdown vector addition systems, where a single (sequential) stack and several global counters are permitted. As before, these counters can be incremented and decremented, but not compared with a value. Reachability of a configuration, and control-state reachability in these models remain open problems, but termination (all paths are finite) and boundedness are known to be decidable [30]. For reachability of a configuration, an under-approximation algorithm is proposed by Atig and Ganty where the stack behaviour is approximated by a *finite index* context-free language [6].

Lang and Löding study boundedness problems over sequential pushdown systems [29]. In this model, the pushdown system is equipped with a counter that can be incremented, reset, or recorded. Their model differs from ours first in the restriction to sequential systems, and second because the counter cannot effect execution or be decremented: it is a recording of resource usage. These kind of cost functions have also been considered over static trees [9, 13], however, to our knowledge, they have not been studied over tree rewrite systems.

2 Preliminaries

We write $\mathbb{N}$ to denote the set of natural numbers and $\mathbb{Z}$ the set of integers.

Trees A *ranked alphabet* is a finite set of characters Σ together with a rank function $\rho : \Sigma \mapsto \mathbb{N}$. A *tree domain* $D \subset \mathbb{N}^*$ is a non-empty finite subset of $\mathbb{N}^*$ that is both *prefix-closed* and *younger-sibling-closed*. That is, if $\eta i \in D$, then we also have $\eta \in D$ and, for all $1 \leq j \leq i$, $\eta j \in D$ (respectively). A *tree* over a ranked alphabet Σ is a pair $t = (D, \lambda)$ where D is a tree domain and $\lambda : D \mapsto \Sigma$ such that for all $\eta \in D$, if $\lambda(\eta) = a$ and $\rho(a) = n$ then η has exactly n children (i.e. $\eta n \in D$ and $\eta(n+1) \notin D$). Let $\mathcal{T}_\Sigma$ denote the set of trees over Σ.

Context Trees A *context tree* over the alphabet Σ with a set of context variables $x_1, \ldots, x_n$ is a tree $C = (D, \lambda)$ over $\Sigma \uplus \{x_1, \ldots, x_n\}$ such that for each $1 \leq i \leq n$ we have $\rho(x_i) = 0$ and there exists a unique *context node* η_i such that $\lambda(\eta_i) = x_i$. By unique, we mean $\eta_i \neq \eta_j$ for all $i \neq j$. We will denote such a tree $C[x_1, \ldots, x_n]$. Given trees $t_i = (D_i, \lambda_i)$ for each $1 \leq i \leq n$, we denote by $C[t_1, \ldots, t_n]$ the tree t' obtained by filling each variable x_i with t_i. That is, $t' = (D', \lambda')$ where

$$D' = D \cup \eta_1 \cdot D_1 \cup \cdots \cup \eta_n \cdot D_n \quad \text{and} \quad \lambda'(\eta) = \begin{cases} \lambda(\eta) & \text{if } \eta \in D \wedge \forall i.\eta \neq \eta_i \\ \lambda_i(\eta') & \text{if } \eta = \eta_i\eta' \ . \end{cases}$$

Tree Automata A *bottom-up non-deterministic tree automaton* (NTA) over a ranked alphabet Σ is a tuple $\mathcal{T} = (\mathcal{Q}, \Delta, \mathcal{F})$ where $\mathcal{Q}$ is a finite set of states, $\mathcal{F} \subseteq \mathcal{Q}$ is a set of final (accepting) states, and Δ is a finite set of rules of the form $(q_1, \ldots, q_n) \xrightarrow{a} q$ where $q_1, \ldots, q_n, q \in \mathcal{Q}$, $a \in \Sigma$ and $\rho(a) = n$. A *run* of $\mathcal{T}$ on a tree $t = (D, \lambda)$ is a mapping $\pi : D \mapsto \mathcal{Q}$ such that for all $\eta \in D$ labelled $\lambda(\eta) = a$ with $\rho(a) = n$ we have $(\pi(\eta 1), \ldots, \pi(\eta n)) \xrightarrow{a} \pi(\eta)$. It is accepting if $\pi(\varepsilon) \in \mathcal{F}$. The *language* defined by a tree automaton $\mathcal{T}$ over alphabet Σ is a set $\mathcal{L}(\mathcal{T}) \subseteq \mathcal{T}_\Sigma$ of trees over which there exists an accepting run of $\mathcal{T}$.

Parikh images Given an alphabet $\Sigma = \{\gamma_1, \ldots, \gamma_n\}$ and a word $w \in \Sigma^*$, we write $\mathcal{P}(w)$ to denote a mapping $\rho : \Sigma \to \mathbb{N}$, where $\rho(a)$ is defined to be the number of occurrences of a in w. Given a language $L \subseteq \Sigma^*$, we write $\mathcal{P}(L)$ to denote the set $\{\mathcal{P}(w) \mid w \in L\}$. We say that $\mathcal{P}(L)$ is the *Parikh image* of L.

Presburger Arithmetic Presburger formulas are first-order formulas over integers with addition. Here, we use existential Presburger formulas $\varphi(\mathbf{x}, \mathbf{y}) := \exists \mathbf{x} \varphi$, where (i) $\mathbf{x}$ and $\mathbf{y}$ are sets of variables, and (ii) φ is a boolean combination of expressions $\sum_{i=1}^{m} a_i z_i \sim b$ for variables $z_1, \ldots, z_m \in \mathbf{x} \cup \mathbf{y}$, constants $a_1, \ldots, a_m, b \in \mathbb{Z}$, and $\sim \in \{\leq, \geq, <, >, =\}$ with constants represented in binary. A *solution* to φ is a valuation $\mathbf{b} : \mathbf{y} \mapsto \mathbb{Z}$ to $\mathbf{y}$ such that $\varphi(\mathbf{x}, \mathbf{b})$ is true. The formula φ is *satisfiable* if it has a solution. Satisfiability of existential Presburger formulas is known to be NP-complete [39].

3 Formal Models

In this section, we will define our formal models, which are based on ground-tree rewrite systems. Ground-tree rewrite systems (GTRSs) [15] permit subtree rewriting where rules are given as a pair of ground-trees. In the sequel, we use the extension proposed by Löding [32] where NTA (instead of ground trees) appear in the rewrite rules. Hence, a single rule may correspond to an infinite number of *concrete rules* (i.e. containing concrete trees).

Ground Tree Rewrite Systems with State and Reversal-Bounded Counters. To capture synchronisations between different subthreads, we follow [26, 31, 41] and extend GTRS with state (a.k.a. global control). The resulting model is denoted by sGTRS (state-extended GTRS). To capture integer variables, we further extend the model with unbounded integer counters, which can be incremented, decremented, and compared against an integer constant. Since Minsky's machines can easily be encoded in such a model, we apply a standard underapproximation technique: *reversal-bounded analysis of the counters* [23, 25]. This means that one only analyses executions of the machines whose number of reversals between nondecrementing and nonincrementing modes of the counters is bounded by a given constant $r \in \mathbb{N}$ (represented in unary). The resulting model will be denoted by rbGTRS. We will now define this model in more detail.

An *atomic counter constraint* on counter variables $C = \{c_1, \ldots, c_k\}$ is an expression of the form $c_i \sim v$, where $v \in \mathbb{Z}$ and $\sim \in \{<, \leq, =, \geq, >\}$. A *counter*

constraint θ on C is a boolean combination of atomic counter constraints on C. Given a valuation $\nu : C \mapsto \mathbb{Z}$ to the counter variables, we can determine whether $\theta[\nu]$ is true or false by replacing a variable c by $\nu(c)$ and evaluating the resulting boolean expressions in the obvious way. Let Cons_C denote the set of all counter constraints on C. Intuitively, these formulas will act as guards to determine whether certain transitions can be fired. Given two counter valuations ν and μ we define $\nu + \mu$ as the pointwise addition of the valuations. That is, $(\nu + \mu)(c) = \nu(c) + \mu(c)$.

Given a sequence of counter values, a reversal occurs when a counter switches from being incremented to being decremented or vice-versa. For example, if the values of a counter c along a run are $1, 1, 1, 2, 3, 4, 4, \overline{4, 3}, 2, \overline{2, 3}$, then the number of reversals of c is 2 (reversals occur in between the overlined positions). A sequence of valuations is reversal-bounded whenever the number of reversals is the sequence is bounded.

Definition 1 (r-Reversal-Bounded). *For a counter c from a set of counters C, a sequence $\nu_1, \ldots, \nu_n$ of counter valuations over C is* r-reversal-bounded *for c whenever we can partition $\nu_1, \ldots, \nu_n$ into $(r+1)$ sequences $A_1, \ldots, A_{r+1}$ (with $\nu_0, \ldots, \nu_n = A_1, \ldots, A_{r+1}$) such that for all $1 \leq i \leq r$ there is some $\sim \in \{\leq, \geq\}$ such that for all ν_j, ν_{j+1} appearing together in A_i, we have $\nu_j(c) \sim_c \nu_{j+1}(c)$.*

We define sGTRS with reversal-bounded counters (rbGTRS).

Definition 2 (sGTRSs with r-Reversal-Bounded Counters). *We define* state-extended ground tree rewrite system with r-reversal-bounded counters *(rbGTRS) as a tuple $G = (\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, C, r)$ where $\mathcal{P}$ is a finite set of control-states, Σ is a finite ranked alphabet, Γ is a finite alphabet of output symbols (i.e. transition labels), C is a finite set of counters, $\mathcal{R}$ is a finite set of rules of the form $(p_1, \mathcal{T}_1, \theta) \xrightarrow{\gamma} (p_2, \mathcal{T}_2, \mu)$ where $p_1, p_2 \in \mathcal{P}$, $\gamma \in \Gamma$, $\theta \in \text{Cons}_C$, $\mu \in C \mapsto \mathbb{Z}$, and $\mathcal{T}_1, \mathcal{T}_2$ are NTAs over Σ.*

In the sequel, we will omit mention of the number r in the tuple G if it is clear from the context.

A *configuration* of an sGTRS with counters is a tuple $\alpha = (p, t, \nu)$ where p is a control-state, t a tree, and ν a valuation of the counters. We have a *transition* $(p_1, t_1, \nu_1) \xrightarrow{\gamma} (p_2, t_2, \nu_2)$ whenever there is a rule $(p_1, \mathcal{T}_1, \theta) \xrightarrow{\gamma} (p_2, \mathcal{T}_2, \mu) \in \mathcal{R}$ such that: (i) *(dynamics of counters)* $\theta[\nu_1]$ is true and $\nu_2 = \nu_1 + \mu$, and (ii) *(dynamics of trees)* $t_1 = C[t_1']$ for some context C and tree $t_1' \in \mathcal{L}(\mathcal{T}_1)$ and $t_2 = C[t_2']$ for some tree $t_2' \in \mathcal{L}(\mathcal{T}_2)$. A *run* π over $\gamma_1 \ldots \gamma_{n-1}$ is a sequence

$$(p_1, t_1, \nu_1) \xrightarrow{\gamma_1} \cdots \xrightarrow{\gamma_{n-1}} (p_n, t_n, \nu_n)$$

such that for all $1 \leq i < n$ we have $(p_i, t_i, \nu_i) \xrightarrow{\gamma_i} (p_{i+1}, t_{i+1}, \nu_{i+1})$ is a transition of G and for each $c \in C$ the sequence $\nu_1, \ldots, \nu_n$ is r-reversal-bounded for c. We say that $\gamma_1 \ldots \gamma_{n-1}$ is the output string of π. We write $(p, t, \nu) \xrightarrow{\gamma_1 \cdots \gamma_n} (p', t', \nu')$ (or simply $(p, t, \nu) \to^* (p', t', \nu')$) whenever there is a run from (p, t, ν) to (p', t', ν') over $\gamma_1 \ldots \gamma_n$. Let ε denote the empty output symbol.

Whenever we wish to discuss sGTRSs without counters, we simply omit the counter components. That is, we have configurations of the form (p, t) and transitions of the form $(p_1, \mathcal{T}_1) \xrightarrow{\gamma} (p_2, \mathcal{T}_2)$. The standard notion of GTRS (i.e. not state-extended) [32] is simply sGTRS without counters with only one state.

We next define the problems of *(global) reachability*. To this end, we use a tree automaton $\mathcal{T}$ (resp. an existential Presburger formula φ) to represent the tree (resp. counter) component of a configuration. More precisely, a *symbolic config-set* of an rbGTRS $G = (\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, C, r)$ is a tuple $(p, \mathcal{T}, \varphi)$, where $p \in \mathcal{P}$, $\mathcal{T}$ is an NTA over Σ, and $\varphi(\bar{x})$ is an existential Presburger formula with free variables $\bar{x} = \{x_c\}_{c \in C}$ (i.e. one free variable for each counter). Each symbolic config-set $(c, \mathcal{T}, \varphi)$ represents a set of configurations of G defined as follows: $[\![(p, \mathcal{T}, \varphi)]\!] := \{(p, t, \nu) : t \in \mathcal{L}(\mathcal{T}),\ \varphi(\nu) \text{ is true}\}$.

Global Reachability

Instance: an rbGTRS G and two symbolic config-sets $(p_1, \mathcal{T}_1, \varphi_1)$ $(p_2, \mathcal{T}_2, \varphi_2)$
Question: Decide whether $(p_1, t_1, \nu_1) \rightarrow^* (p_2, t_2, \nu_2)$, for some $(p_1, t_1, \nu_1) \in [\![(p_1, \mathcal{T}_1, \varphi_1)]\!]$ and $(p_2, t_2, \nu_2) \in [\![(p_2, \mathcal{T}_2, \varphi_2)]\!]$

The problem of *control-state reachability* can be defined by restricting (i) the tree automata $\mathcal{T}_1$ and $\mathcal{T}_2$ to accept, respectively, a singleton tree and the set of all trees, and (ii) the solutions to the formulas φ_1 and φ_2 are, respectively, $\{\nu_0\}$ (where ν_0 is the valuation assigning 0 to all counters) and the set of all counter valuations.

Remark 3. When we measure the complexity of reachability for rbGTRS, the number r of reversals is represented in unary, while the numbers in counter constraints and valuations are represented in binary. This is consistent with the standard representation of numbers in previous work on reversal-bounded counter machines (e.g. see [23, 24]). The unary representation for r can be justified by the fact that bugs can often be discovered within a small number of reversals.

Weakly-Synchronised Ground Tree Rewrite Systems The control-state and global reachability problems for sGTRS are known to be undecidable [12, 21]. The problems become NP-complete for *weakly-synchronised* sGTRS [31, 41], where the underlying control-state graph (where there is an edge between p_1 and p_2 whenever there is a transition $(p_1, \mathcal{T}_1) \xrightarrow{\gamma} (p_2, \mathcal{T}_2)$) may only have cycles of length 1 (i.e. self-loops), i.e., a DAG (directed acyclic graph) possibly with self-loops. Underapproximation by a weak control is akin to loop acceleration in the symbolic acceleration framework of [8]. We extend the definition to rbGTRSs. The original definition can be easily obtained by omitting the counter components.

We define the *underlying control graph* of an rbGTRS $G = (\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, C)$ as a tuple $(\mathcal{P}, \Delta)$ where $\Delta = \left\{ (p_1, p_2) \;\middle|\; (p_1, \mathcal{T}_1, \theta) \xrightarrow{\gamma} (p_1, \mathcal{T}_2, \mu) \in \mathcal{R} \right\}$.

Definition 4 (Weakly-Synchronised rbGTRS). *An rbGTRS is said to be* weakly-synchronised *if its underlying control graph $(\mathcal{P}, \Delta)$ is a DAG possibly with self-loops.*

4 Decidability

In this section we will prove the main result of the paper:

Theorem 5. *Global reachability for weakly synchronised rbGTRS is NP-complete. In fact, it is poly-time reducible to satisfiability over existential Presburger formulas.*

To prove this theorem, we fix notation for the input to the problem: an rbGTRS $G = (\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, C, r)$ and two symbolic config-sets $(p_1, \mathcal{T}_1, \varphi_1)$, $(p_2, \mathcal{T}_2, \varphi_2)$ of G. Let $C = \{c_i\}_{i=1}^k$. The gist of the proof is as follows. From G, we construct a new sGTRS G' (without counters) by encoding the dynamics of the counters in the output symbols of G'. Of course, G' has no way of comparing the values of counters with constants. [In this sense, G' only overapproximates the behavior of G.] To deal with this problem, we use the result of [31] to compute an existential Presburger formula ψ capturing the Parikh images of the set of all output strings of G' from $(p_1, \mathcal{T}_1, \varphi_1)$ to $(p_2, \mathcal{T}_2, \varphi_2)$. The final formula is $\psi \wedge \psi'$, where ψ is a constraint asserting that the desired counter comparisons are performed throughout runs of G'. We sketch the details of the construction below.

Modes of the counters. The first notion that is crucial in our proof is that of *mode* of a counter [23, 25], which is an abstraction of the values of a counter in a run of an rbGTRS containing three pieces of information: (i) the *region* of the counter value (i.e. how it compares to constants occurring in counter constraints), (ii) the number of reversals that has been performed by each counter (between 0 and r), and (iii) whether a counter is currently non-decrementing ($\uparrow$) or non-incrementing ($\downarrow$). A *mode vector* is simply a k-tuple of modes, one mode for each of the k counters. We now formalise these notions.

Let $d_1 < \ldots < d_m$ be the integer constants appearing in the counter constraints in G. This sequence of constants gives rise to the set $\texttt{REG}$ of *regions* defined as $\texttt{REG} := \{A_0, \ldots, A_m, B_1, \ldots, B_m\}$, where $B_i := \{d_i\}$ (where $1 \leq i \leq m$), $A_i := \{n \in \mathbb{Z} : d_i < n < d_{i+1}\}$ (where $1 \leq i < m$), $A_0 := \{n \in \mathbb{Z} : n < d_1\}$, and $A_m := \{n \in \mathbb{Z} : n > d_m\}$. A *mode* is simply a tuple in $\texttt{REG} \times [0, r] \times \{\uparrow, \downarrow\}$. A *mode vector* is simply a tuple in $\mathbf{Modes} := \texttt{REG}^k \times [0, r]^k \times \{\uparrow, \downarrow\}^k$.

Building the sGTRS G'. We might be tempted to build G' by first removing the counters from G and then embedding $\mathbf{Modes}$ into the control-states G'. This, however, causes two problems. First, the number of control-states becomes exponential in k. Second, the resulting system is no longer weakly synchronised even though G originally was weakly synchronised. To circumvent this problem, we adapt a technique from [23]. Every run π of G from $(p_1, \mathcal{T}_1, \varphi_1)$ to $(p_2, \mathcal{T}_2, \varphi_2)$ can be associated with a sequence σ of mode vectors recording the information (i)–(iii) for each counter. The crucial observation is that there are at most $N_{\max} := 2mk(r + 1)$ different mode vectors in σ. This is because a counter can

only go through at most $2m$ regions without incurring a reversal. For this reason, we may use the control-states of G' to store the number of mode vectors that G has gone through, while the actual mode vector guessed by G' will be made "visible" in the output strings of G'. That way, we can use an additional existential Presburger formula ψ' (see below) to enforce that the run of G' faithfully simulates runs of G. In addition, the shape of the control-states (DAG with self-loops) of G' is preserved. [The product graph of two DAGs with self-loops is also a DAG with self-loops.] We detail the construction below.

Define the weakly-synchronised sGTRS $G' = (\mathcal{P}', \Sigma, \Gamma', \mathcal{R}')$ as follows. Let $\mathcal{P}' := \mathcal{P} \times [0, N_{\max}]$. The output alphabet Γ' is defined as $\Gamma \times \mathcal{R} \times [0, N_{\max}] \times \{0, 1\}$, where the boolean flag is used to denote whether the transition taken changes the mode. We define $\mathcal{R}'$ as follows. For each rule $\tau = (p, \mathcal{T}, \theta) \xrightarrow{\gamma} (p', \mathcal{T}', \mu)$ in $\mathcal{R}$, we add the rule $((p, i), \mathcal{T}) \xrightarrow{(\gamma,\tau,i,0)} ((p', i), \mathcal{T}')$ for each $i \in [0, N_{\max}]$, and $((p, i), \mathcal{T}) \xrightarrow{(\gamma,\tau,i,1)} ((p', i+1), \mathcal{T}')$ for each $i \in [0, N_{\max})$. Since G is weakly-synchronised and the mode counter never decreases, it follows that G' is weakly-synchronised too. Note also that this construction can be performed in polynomial-time.

Constructing the formula $\psi \wedge \psi'$. As we mentioned, ψ is an existential Presburger formula encoding the Parikh image $\mathcal{P}(L)$ of the set L of all output strings of G' from $((p_1, 0), \mathcal{T}_1)$ to $(S, \mathcal{T}_2)$, where $S = \{p_2\} \times [0, N_{\max}]$. More precisely, the set $\mathbf{z}$ of free variables of ψ include z_a for each $a \in \Gamma'$. Furthermore, for each valuation $\mu \in \mathbf{z} \mapsto \mathbb{Z}$, it is the case that $\psi(\mu)$ is true iff $\mu \in \mathcal{P}(L)$. Such a formula is known to be polynomial-time computable since G' is a weakly-synchronised sGTRS [31].

Recall that ψ' should assert that the desired counter comparisons are performed throughout runs of G'. To this end, the formula ψ' will have extra variables for guessing the existence of a sequence of $N_{\max}$ distinct mode vectors through runs of G'. More precisely, the formula ψ' is the conjunction

$$\varphi_1(\mathbf{x}) \wedge \varphi_2(\mathbf{y}) \wedge \texttt{Dom}(\mathbf{m}_0, \ldots, \mathbf{m}_{N_{\max}}) \wedge \texttt{Init}(\mathbf{m}_0) \wedge \\ \texttt{GoodSeq}(\mathbf{m}_0, \ldots, \mathbf{m}_{N_{\max}}) \wedge \texttt{Respect}(\mathbf{z}, \mathbf{m}_0, \ldots, \mathbf{m}_{N_{\max}}) \wedge \texttt{EndVal}(\mathbf{x}, \mathbf{y}, \mathbf{z}).$$

The set $\mathbf{x}$ consists of variables x_i $(1 \leq i \leq k)$ which contain the initial value of the ith counter. Similarly, the set $\mathbf{y}$ consists of variables y_i $(1 \leq i \leq k)$ which contain the final value of the ith counter. Each $\mathbf{m}_i$ denotes a set of variables for the ith mode vector defined as follows:

- reg^i_j (for each $j \in [1, k]$) — to encode which of the $2m + 1$ possible regions the jth counter is in.
- rev^i_j (for each $j \in [1, k]$) — to encode how many reversals have been used up by the jth counter.
- arr^i_j (for each $j \in [1, k]$) — to encode whether the jth counter is non-incrementing or non-decrementing.

We detail each subformula below.

The subformula $\texttt{Dom}$ asserts that each variable in $\mathbf{m}_i$ (for each i) has the right domain (i.e. range of integer values). More precisely, for each $j \in [1, k]$, we add

the conjuncts: (i) $0 \leq reg_j^i \leq 2m$, (ii) $0 \leq rev_j^i \leq r$, and (iii) $0 \leq arr_j^i \leq 1$. For the first constraint, we use an even number of the form $2i$ to represent the region A_i, and an odd number $2i-1$ to represent the region B_i. The last constraint simply encodes non-decrementing ($\uparrow$) as 1, and non-incrementing ($\downarrow$) as 0.

The subformula `Init` asserts that $\mathbf{m}_0$ is an initial mode vector. More precisely, for each $j \in [1,k]$, we add the conjuncts $rev_j^0 = 0$.

The subformula `GoodSeq` asserts that $\mathbf{m}_0, \ldots, \mathbf{m}_{N_{\max}}$ forms a valid sequence of mode vectors. More precisely, for each $i \in [0, N_{\max})$ and each $j \in [1,k]$, we add the conjuncts: (i) $arr_j^i \neq arr_j^{i+1} \Rightarrow rev_j^{i+1} = rev_j^i + 1$, (ii) $arr_j^i = arr_j^{i+1} \Rightarrow rev_j^{i+1} = rev_j^i$, (iii) $reg_j^i < reg_j^{i+1} \Rightarrow arr_j^{i+1} = 1$, and (iv) $reg_j^i > reg_j^{i+1} \Rightarrow arr_j^{i+1} = 0$. For example, the first constraint asserts that a change in the direction (non-incrementing or non-decrementing) of the counter incurs one reversal. The other constraints are similar.

The subformula `Respect` asserts that the Parikh image $\mathbf{z}$ of the run of G' respects the sequence $\mathbf{m}_0, \ldots, \mathbf{m}_{N_{\max}}$ of mode vectors. In effect, this subformula ensures that G' faithfully simulates G. Firstly, we need to assert that the jth counter values at the *start* and at the *end* of the ith mode of G' (which are encoded in $\mathbf{z}$) are in the right regions reg_j^i. To state this more precisely, for each rule $\tau = (p, \mathcal{T}, \theta) \xrightarrow{\gamma} (p', \mathcal{T}', \mu)$ in $\mathcal{R}$, we let $\mu_j(\tau)$ denote the value $\mu(c_j)$. For each $i \in [0, N_{\max}]$ and $j \in [1,k]$, we denote by the notation $\texttt{StartCounter}_j^i$ the term $x_j + \sum_{s=0}^{i-1} \sum_{(\gamma,\tau,s,l)} \mu_j(\tau) \times z_{(\gamma,\tau,s,l)}$, where γ, τ, and l, range over, respectively, Γ, $\mathcal{R}$, and $\{0,1\}$. Similarly, we denote by $\texttt{EndCounter}_j^i$ the term $\texttt{StartCounter}_j^i + \sum_{(\gamma,\tau,i,0)} \mu_j(\tau) \times z_{(\gamma,\tau,i,0)}$. We add the conjuncts: (i) $reg_j^i = 2h \Rightarrow \texttt{EndCounter}_j^i \in A_h$, for each $h \in [0,m]$, and (ii) $reg_j^i = 2h+1 \Rightarrow \texttt{EndCounter}_j^i \in B_h$, for each $h \in [0,m)$. [Note that formulas of the form $g \in A$, for a Presburger term g and a set $S \in \{A_0, \ldots, A_m, B_1, \ldots, B_m\}$, can be easily replaced by quantifier-free Presburger formulas, e.g., $g \in A_0$ stands for $g < d_1$.] To ensure that the initial condition is correct, for each $j \in [1,k]$, we add the following conjuncts: (1) $\texttt{StartCounter}_j^0 \in A_h \Rightarrow reg_j^0 = 2h$, and (2) $\texttt{StartCounter}_j^0 \in B_h \Rightarrow reg_j^0 = 2h+1$. Secondly, we need to state that the transitions executed in each mode are valid (i.e. satisfy the counter constraints). More precisely, for each $\gamma \in \Gamma$, $\tau \in \mathcal{R}$, $i \in [0, N_{\max}]$, and $l \in \{0,1\}$, if θ is the counter constraint in τ, we add the conjunct $z_{(\gamma,\tau,i,l)} > 0 \Rightarrow \theta(\texttt{StartCounter}_1^i, \ldots, \texttt{StartCounter}_k^i)$. Next we assert that, when the jth counter is non-incrementing (resp. non-decrementing), only non-negative (resp. non-positive) counter increments are permitted. More precisely, for each $i \in [0, N_{\max}]$, $j \in [1,k]$, $l \in \{0,1\}$, and $\tau \in \mathcal{R}$, if $\mu_j(\tau) > 0$, then add the conjunct $arr_j^i = 0 \Rightarrow z_{(\gamma,\tau,i,l)} = 0$; if $\mu_j(\tau) < 0$, then add the conjunct $arr_j^i = 1 \Rightarrow z_{(\gamma,\tau,i,l)} = 0$.

Finally, the subformula `EndVal` simply asserts that, starting from the initial counter value $\mathbf{x}$ and following the transitions $\mathbf{z}$, the end counter values are $\mathbf{y}$. To this end, we can simply add the conjunct $y_j = \texttt{EndCounter}_j^{N_{\max}}$ for each $j \in [1,k]$.

This concludes the formula construction. It is immediate that G' faithfully simulates G iff $\psi \wedge \psi'$ is true. In addition, the formula construction runs in

polynomial-time. Since satisfiability over existential Presburger formulas is NP-complete [39], the NP upper bound for Theorem 5 follows. NP-hardness already holds for the restricted model where the tree component is a stack [23].

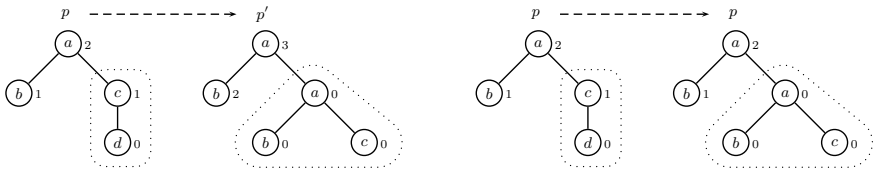
5 Senescent Ground-Tree Rewrite Systems

A natural question arising from the result on weakly synchronised rbGTRS is whether the "weakly synchronised" restriction can be relaxed while maintaining decidability. It is known that allowing arbitrary underlying control-state graphs leads to undecidability of reachability even without reversal bounded counters. In this section we explore the notion of *senescence* [22], which is more general than the weakly synchronised restriction, but still permits a decidable reachability problem (without counters). After giving the formal definition of senescent GTRS, we show the following result.

Theorem 6 (Control-State Reachability of Senescent rbGTRS). *The control-state reachability problem for senescent rbGTRS is undecidable.*

5.1 Model Definition

Senescence allows the underlying control-state graph to have arbitrary cycles (instead of only self-loops). For sGTRS, control-state reachability is decidable under an "age restriction" that is imposed on the nodes that can be rewritten. That is, when the control-state changes, the nodes in the tree age by one timestep. Once a node reaches an *a priori* fixed age r, it becomes fixed (i.e. cannot be rewritten by further transitions in the run).



(a) A transition changing the control-state.

(b) A transition that does not change the control-state.

Fig. 1: Transitions of a senescent GTRS.

Before the formal definition, two example transitions of a senescent rbGTRS are shown in Figure 1. A configuration is written as its control-state and counter values ((p, ν) or (p', ν')) with the tree appearing below. In the tree, the label of each node appears in the centre of the node. The ages of each node is depicted as a subscript on the right. Dotted lines are used to indicate the part of the tree rewritten by a rule. In Figure 1a the transition changes the control-state, causing the age of the nodes that are not rewritten to increase by 1. The rewritten nodes are given the age 0 as they are new, *fresh*, nodes. The situation when the control-state does not change is shown in Figure 1b. In this case, the nodes that are not rewritten maintain the same age. The senescence restriction disallows runs where nodes older than a fixed age are rewritten.

More formally, given a run

$$(p_1, t_1, \nu_1) \xrightarrow{\gamma_1} \cdots \xrightarrow{\gamma_{n-1}} (p_n, t_n, \nu_n)$$

of an rbGTRS, let $C_1, \ldots, C_{n-1}$ be the sequence of tree contexts used in the transitions from which the run was constructed. That is, for all $1 \leq i < n$, we have $t_i = C_i[t_i^{\text{out}}]$ and $t_{i+1} = C_i\left[t_{i+1}^{\text{in}}\right]$ where $(p_i, \mathcal{T}_i, \theta_i) \xrightarrow{\gamma_i} (p_{i+1}, \mathcal{T}_i', \mu_i)$ was the rewrite rule used in the transition and $t_i^{\text{out}} \in \mathcal{L}(\mathcal{T}_i)$, $t_{i+1}^{\text{in}} \in \mathcal{L}(\mathcal{T}_i')$ were the trees that were used in the tree update.

For a given position (p_i, t_i, ν_i) in the run and a given node η in the domain of t_i, the *birthdate* of the node is the largest $1 \leq j \leq i$ such that η is in the domain of $C_j\left[t_j^{\text{in}}\right]$ and η is in the domain of $C_j[x]$ only if its label is x. The *age* of a node is the cardinality of the set $\{i' \mid j \leq i' < i \wedge p_{i'} \neq p_{i'+1}\}$. That is, the age is the number of times the control-state changed between the jth and the ith configurations in the run.

A lifespan-restricted run with a lifespan of r is a run such that each transition $(p_i, C_i[t_i^{\text{out}}], \nu_i) \xrightarrow{\gamma_i} \left(p_{i+1}, C_i\left[t_{i+1}^{\text{in}}\right], \nu_{i+1}\right)$ has the property that all nodes η in t_i^{out} have an age of at most r. That is, more precisely, that all nodes η in the domain of $C_i[t_i^{\text{out}}]$ but only in the domain of $C_i[x]$ if the label is x have an age of at most r.

Definition 7 (Senescent rbGTRS). *A* senescent rbGTRS *with* lifespan r *is an rbGTRS* $G = (\mathcal{P}, \Sigma, \mathcal{R}, C)$ *where runs are* lifespan-restricted *with a lifespan of* r.

Note that the senescence restriction is weaker than the weakly-synchronised restriction in that the number of times the finite control could change state is unbounded. In fact, a node could be affected by an unbounded number of control-state changes so long as it is always rewritten without becoming fixed (i.e. reaches age r).

5.2 Undecidability

We show control-state reachability for senescent rbGTRSs is undecidable in the full version, and give the intuition here. In the following, we refer to nodes whose age is within the age bound as *live*. We refer to nodes that are not live as *fixed*. Note, each time a node is rewritten, its age is reset to zero. Thus, we can keep leaves of the tree live by allowing them to rewrite to themselves. That is, for all symbols a we wish to keep live and all control-states p, we have a transition $(p, a, \theta) \xrightarrow{\gamma} (p, a, \mu)$ where θ is a formula that is always satisfied, and μ assigns 0 to all counters (i.e. the rule does not depend on, nor change the counter values). In addition, by omitting the above rules for certain control-states, we can prevent a node from keeping itself fresh in certain situations.

We follow the proof that reachability for reset Petri nets is undecidable [3]. We simulate a two-counter machine. Testing whether such a machine can reach a given control-state while having counters with value zero is undecidable.

Let the two counters be c_1 and c_2. In the tree, we track the value of a counter $c \in \{c_1, c_2\}$ by the number of live leaves labelled with the counter

name c. E.g. the tree $\bullet(c_1, \bullet(c_2, *))$ represents the situation where both counters have value 1, assuming these leaves are live. We will always use internal nodes labelled $\bullet$. The node $*$ is for adding new leaves when required. To increment a counter we add a new leaf labelled c. To decrement a counter, we rewrite a leaf labelled c to a null label. Thus, we can easily increment and decrement counters. Zero tests, however, are more subtle. To help with this, we track, using reversal-bounded counters, the number of increments made to each counter, and in separate reversal-bounded counters, the number of decrements. That is, we have reversal bounded counters $\{c_1^+, c_1^-, c_2^+, c_2^-\}$. When we simulate an increment of c_1 we add a leaf and increment c_1^+. When we simulate a decrement of c_2 we rewrite a leaf to a null character and increment c_1^-. Similarly for c_2. We simulate zero tests as follows.

To simulate a zero test on a counter c we perform the following checks. First, we "reset" the counter to zero by forcing enough control-state changes to fix the nodes corresponding to the counter. That is, we move to a control-state p where all leaf labels may rewrite to themselves, except those labelled c. After the move to p all leaves will have age 1. Leaves not labelled c can refresh their age to 0 by rewriting themselves. Leaves labelled c will stay aged 1. Then, we move to the target control-state of the transition we are simulating. Thus, after these moves, all leaves labelled c will reach age 2, while all other nodes will only reach age 1. Thus, if our lifespan is 2, nodes labelled c will no longer be live. That is, the simulated value of c in the tree has been forced to 0.

After this reset operation, the counter value is definitely zero. However, we did not enforce that the counter value was zero before the transition. Recall, we track the number of increments and decrements to c in the reversal bounded counters. If the counter was not zero before the test, there will be a discrepancy with the reversal bounded counters: more increments will be recorded than decrements. E.g. for counter c_1 we will have $c_1^+ > c_1^-$. This cannot be corrected by the simulation. Thus, at the end of the run, we check whether the number of increments is equal to the number of decrements. If not, we know the run made a spurious transition. That is, it performed a zero test transition when the counter was not zero. If no spurious transitions were made, we know the two-counter machine has a corresponding run. This completes the gist of the simulation of a two-counter machine.

6 Extensions and Future Work

We proposed sGTRS with counters as a model of recursively parallel programs with unbounded recursion, thread creation, and integer variables. To obtain decidability, we gave an underapproximation in the form of weak sGTRS with reversal-bounded counters. We showed that the reachability problem for this model is NP-complete; in fact, polynomial-time reducible to satisfiability of linear integer arithmetic, for which highly optimised SMT solvers are available (e.g. Z3 [16]). Additionally, we explored the possibility of relaxing the weakly-synchronised constraint to that of senescence, and showed that the resulting model has an undecidable control-state reachability problem.

One possible avenue of future work is to investigate what happens when *local* integer values are permitted. That is, reversal-bounded counters can be stored on the nodes of the tree. We may also study techniques that allow nodes to contain multiple labels, permitting the modelling of multiple local variables without an immediate exponential blow up.

Acknowledgments We thank anonymous reviewers for their helpful feedback. This work was supported by the Engineering and Physical Sciences Research Council [EP/K009907/1] and Yale-NUS College Startup Grant.

References

1. P. A. Abdulla, M. F. Atig, and J. Cederberg. Analysis of message passing programs using SMT-solvers. In *ATVA*, pages 272–286, 2013.
2. C. Aiswarya, P. Gastin, and K. N. Kumar. Verifying communicating multi-pushdown systems via split-width. In *ATVA*, pages 1–17, 2014.
3. T. Araki and T. Kasami. Some Decision Problems Related to the Reachability Problem for Petri Nets. *Theoretical Computer Science*, 3(1):85–104, 1977.
4. M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2etime-complete. In *DLT*, pages 121–133, 2008.
5. M. F. Atig, A. Bouajjani, and S. Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *Logical Methods in Computer Science*, 7(4), 2011.
6. M. F. Atig and P. Ganty. Approximating petri net reachability along context-free traces. In *FSTTCS*, pages 152–163, 2011.
7. M. F. Atig, K. N. Kumar, and P. Saivasan. Adjacent ordered multi-pushdown systems. *Int. J. Found. Comput. Sci.*, 25(8):1083–1096, 2014.
8. S. Bardin, A. Finkel, J. Leroux, and P. Schnoebelen. Flat acceleration in symbolic model checking. In *ATVA*, pages 474–488, 2005.
9. A. Blumensath, T. Colcombet, D. Kuperberg, P. Parys, and M. Vanden Boom. Two-way cost automata and cost logics over infinite trees. In *CSL-LICS*, pages 16:1–16:9, 2014.
10. A. Bouajjani and M. Emmi. Analysis of recursively parallel programs. *ACM Trans. Program. Lang. Syst.*, 35(3):10, 2013.
11. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, pages 135–150, 1997.
12. L. Bozzelli, M. Kretínský, V. Rehák, and J. Strejcek. On decidability of LTL model checking for process rewrite systems. *Acta Inf.*, 46(1):1–28, 2009.
13. T. Colcombet and C. Löding. Regular cost functions over finite trees. In *LICS*, pages 70–79, 2010.
14. W. Czerwinski, P. Hofman, and S. Lasota. Reachability problem for weak multi-pushdown automata. *Logical Methods in Computer Science*, 9(3), 2013.
15. M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *LICS*, pages 242–248, 1990.
16. L. M. de Moura and N. Bjørner. Z3: An efficient smt solver. In *TACAS*, pages 337–340, 2008.
17. S. Demri, M. Jurdzinski, O. Lachish, and R. Lazic. The covering and boundedness problems for branching vector addition systems. *J. Comput. Syst. Sci.*, 79(1):23–38, 2013.
18. J. Esparza, P. Ganty, and T. Poch. Pattern-based verification for multithreaded programs. *ACM Trans. Program. Lang. Syst.*, 36(3):9:1–9:29, 2014.

19. J. Esparza and A. Podelski. Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In *POPL*, pages 1–11, 2000.
20. P. Ganty, R. Majumdar, and M. Monmege. Bounded underapproximations. *FMSD*, 40(2), 2012.
21. S. Göller and A. W. Lin. Refining the process rewrite systems hierarchy via ground tree rewrite systems. In *CONCUR*, pages 543–558, 2011.
22. M. Hague. Senescent ground tree rewrite systems. In *CSL-LICS*, pages 48:1–48:10, 2014.
23. M. Hague and A. W. Lin. Model checking recursive programs with numeric data types. In *CAV*, pages 743–759, 2011.
24. M. Hague and A. W. Lin. Synchronisation- and reversal-bounded analysis of multithreaded programs with counters. In *CAV*, pages 260–276, 2012.
25. O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978.
26. M. Kretínský, V. Rehák, and J. Strejcek. Extended process rewrite systems: Expressiveness and reachability. In *CONCUR*, pages 355–370, 2004.
27. S. La Torre, M. Napoli, and G. Parlato. Scope-bounded pushdown languages. In *DLT*, pages 116–128, 2014.
28. A. Lal, T. Touili, N. Kidd, and T. Reps. Interprocedural analysis of concurrent programs under a context bound. In *TACAS*, pages 282–298, 2008. Springer-Verlag.
29. M. Lang and C. Löding. Modeling and verification of infinite systems with resources. *Logical Methods in Computer Science*, 9(4), 2013.
30. J. Leroux, M. Praveen, and G. Sutre. Hyper-ackermannian bounds for pushdown vector addition systems. In *CSL-LICS*, pages 63:1–63:10, 2014.
31. A. W. Lin. Weakly-synchronized ground tree rewriting (with applications to verifying multithreaded programs). In *MFCS*, pages 630–642, 2012.
32. C. Löding. Reachability problems on regular ground tree rewriting graphs. *Theory Coput. Syst.*, 39(2):347–383, 2006.
33. P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In *POPL*, pages 283–294, 2011.
34. R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, TU-Munich, 1998.
35. M. Musuvathi and S. Qadeer. Iterative context bounding for systematic testing of multithreaded programs. In *PLDI*, pages 446–455, 2007.
36. S. Qadeer. The case for context-bounded verification of concurrent programs. In *SPIN*, pages 3–6, 2008.
37. G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *Transactions on Programming Languages and Systems (TOPLAS)*, 2000.
38. S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, pages 93–107, 2005.
39. B. Scarpellini. Complexity of subcases of Presburger arithmetic. *Trans. of AMS*, 284(1):203–218, 1984.
40. D. Suwimonteerabuth, J. Esparza, and S. Schwoon. Symbolic context-bounded analysis of multithreaded java programs. In *SPIN*, pages 270–287, 2008.
41. A. W. To and L. Libkin. Algorithmic metatheorems for decidable LTL model checking over infinite systems. In *FOSSACS*, pages 221–236, 2010.
42. S. L. Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *In LICS*, pages 161–170. IEEE Computer Society, 2007.
43. K. N. Verma and J. Goubault-Larrecq. Karp-Miller trees for a branching extension of VASS. *Discrete Mathematics & Theoretical Computer Science*, 7(1):217–230, 2005.