Decidable models of integer-manipulating programs with recursive parallelism

Matthew Hague

Royal Holloway, University of London

Anthony W. Lin

Department of Computer Science, University of Oxford

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.

Keywords: Reversal-bounded counters, Ground Tree Rewrite Systems, Context-bounded, Presburger Arithmetic, Parallelism, Senescence, Automata

[☆]This is the journal version of an article in RP 2016 [1]. The article has been extended with the inclusion of all proofs and an example to illustrate the model considered.

Email addresses: matthew.hague@rhul.ac.uk (Matthew Hague), anthony.lin@cs.ox.ac.uk (Anthony W. Lin)

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 [2, 3]. [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 [2]. This result has paved the way for an efficient use of highly optimised SMT solvers in verifying concurrent programs (e.g. see [4, 5, 6]). Note that without bounding the number of context switches the model is undecidable [7].

In the past decade the work of Qadeer and Rehof [2] has spawned a lot of research in underapproximation techniques for verifying multithreaded programs, e.g., see [4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21] among many others. Other than unbounded recursions, some of these results simultaneously address other sources of infinity, e.g., unbounded thread creation [12, 13, 9], unbounded integer variables [4], and unbounded FIFO queues [5, 16].

Contributions. In this paper we generalise existing underapproximation techniques [22, 12] 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 [4]: 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) [12] 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 [23]) to strictly subsume other well-known sequential and concurrent models like pushdown systems [24], PA-processes [25], and PAD-processes [26], 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 [23].] 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 [27, 12] (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 3.2) in the standard way, while preserving self-loops. This type of underapproximation is similar to *loop acceleration* in the symbolic acceleration framework of [28]. Second, we bound the number of reversals between non-decrementing and non-incrementing modes of

the counters [29]. 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 [12, 30], and (ii) synchronisation/reversal-bounded concurrent pushdown systems with counters [4].

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) [13], then the resulting model becomes undecidable.

Related Work. Recursively-parallel program analysis was analysed in detail by Bouajjani and Emmi [31]. 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 [32] 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 [33].

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 [34]. 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 [35].

Lang and Löding study boundedness problems over sequential pushdown systems [36]. 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 [37, 38], 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 $\{0, 1, 2, \ldots\}$ and $\mathbb{Z}$ the set of integers.

2.1. 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 $0 \leq j \leq i$, $\eta j \in D$ (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.

2.2. Context Trees

A *context tree* over the alphabet Σ with a set of context variables $x_1, \ldots, x_n$ is a tree $\mathcal{C} = (D, \lambda)$ 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 $\mathcal{C}[x_1, \ldots, x_n]$. Given trees $t_i = (D_i, \lambda_i)$ for each $1 \leq i \leq n$, we denote by $\mathcal{C}[t_1, \ldots, t_n]$ the tree t' obtained by filling each variable x_i with t_i. That is, $t' = (D', \lambda')$ where

$$D' = D \cup \eta_1 \cdot D_1 \cup \cdots \cup \eta_n \cdot D_n$$

and

$$\lambda'(\eta) = \begin{cases} \lambda(\eta) & \text{if } \eta \in D \wedge \forall i. \eta \neq \eta_i \\ \lambda_i(\eta') & \text{if } \eta = \eta_i \eta' \ . \end{cases}$$

2.3. 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 $\mathcal{T}$.

2.4. 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.

2.5. Presburger Arithmetic

Presburger formulas are first-order formulas over integers with addition. Here, we use existential Presburger formulas $\varphi(\mathbf{y}) = \exists \mathbf{x} \varphi$, 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) [40] permit subtree rewriting where rules are given as a pair of ground-trees. In the sequel, we use the extension proposed by Löding [41] 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).

3.1. Ground Tree Rewrite Systems with State and Reversal Bounded Counters.

To capture synchronisations between different subthreads, we follow [12, 27, 30] 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* [22, 29]. 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 (where $\top$ denotes "true"). Given a valuation $\nu : C \mapsto \mathbb{Z}$ to the counter variables, we can determine whether $\theta[\nu]$ is true or false by replacing each variable c by $\nu(c)$ and evaluating the resulting boolean expression 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 3.1 (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 $1 \leq i \leq (r+1)$ 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.

Definition 3.2 (rbGTRS). *A* state-extended ground tree rewrite system with r-reversal-bounded counters *(rbGTRS) is 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 $p_1, p_2 \in \mathcal{P}$, $\gamma \in \Gamma$, $\theta \in \mathrm{Cons}_C$, $\mu \in C \mapsto \mathbb{Z}$, 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)* $t_1 = \mathcal{C}[t_1']$ for some context $\mathcal{C}$ and tree $t_1' \in \mathcal{L}(\mathcal{T}_1)$ and $t_2 = \mathcal{C}[t_2']$ 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) [41] is simply sGTRS without counters with only one state.

We next define the problem 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 $(p, \mathcal{T}, \varphi)$ 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 3.1. 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 [22, 4]). The unary representation for r can be justified by the fact that bugs can often be discovered within a small number of reversals.

3.2. Weakly Synchronised Ground Tree Rewrite Systems

The control-state and global reachability problems for sGTRS are known to be undecidable [42, 23]. The problems become NP-complete for *weakly-synchronised* sGTRS [12, 30], 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 [28]. 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 3.3 (Weakly-Synchronised rbGTRS). *An rbGTRS is* weakly synchronised *if its underlying control graph $(\mathcal{P}, \Delta)$ is a DAG possibly with self-loops.*

3.3. A simple example

We now provide a simple example of how weakly-sycnhronised rbGTRS can be used in concurrent program verification. Consider the concurrent program in Algorithm 1 (taken from [43]) that computes the nth Fibonacci number using recursive paralleism. Note, the variable m takes the initial value 0. One question we might be interested in is to check whether the value m is *functionally determined* by n, i.e., whether the value of m can be different on the input n

Algorithm 1 A concurrent program $Fib(n)$ with a global int variable m

Input: A number n
Task: Put the nth Fibonacci number $Fib(n)$ in the variable m
  if $n < 2$ **then**
    $m := m + n$
  else
    Spawn $Fib(n-1)$
    Spawn $Fib(n-2)$
    Sync (i.e. wait till both finished)
  end if

due to concurrency. We can use the algorithm in the paper to prove a weaker property: whether m is functionally determined for a *given* value of n (e.g. $n = 100$). This is clearly not something that can be easily proven by testing. It is possible to prove this using a finite-state model checker, but the search space is exponential. To prove this using our algorithm (an NP algorithm that may take advantage of highly optimised SMT-solvers), we first model the above program as a weakly synchronised rbGTRS with two reversal-bounded counters.

Intuitively, we will use the tree of the rbGTRS to model the parent/child relationships of the processes created by the spawn actions. The m variable will be tracked over two runs of the program using two counters m_1 and m_2. The system will run in three stages. During the ith ($i = 1, 2$) stage, it guesses a possible output value of the above program and stores it in m_i. During the third stage, the system checks if the two output values are the same.

We provide the details of our weakly-synchronised rbGTRS model

$$G = (\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, C, r)\,.$$

The set $\mathcal{P}$ of control states consists of three states q_1, q_2, and q_3. As we shall see, q_3 is the "bad" state. The finite ranked alphabet consists of labels f_i and f'_i for each $i = 0, \ldots, n$. [Recall that n is a number that is provided in the input to our analysis.] The ranks of f_i and f'_i are, respectively, 0 and 2. Intuitively, the label f_i indicates that the function $Fib(i)$ is about to be called, while f'_i indicates that the function $Fib(i)$ has been called and is currently executing. We do not need special output symbols, so $\Gamma = \{a\}$ and we will omit mention of a below. We have two 1-reversal-bounded counters m_1 and m_2. We use a term representation of trees in our description of rules for G, e.g., $f'_7(f_6, f_5)$ means a tree with root labeled f'_7 and two children labeled f_6 and f_5 respectively. Moreover, we denote by $\mathbf{0}$ the function that maps both m_1 and m_2 to 0. The rules are given as follows:

- $(q_i, f_j, \top) \to \big(q_i, f'_j(f_{j-1}, f_{j-2}), \mathbf{0}\big)$, for each $i \in \{1, 2\}$, and $j \in \{2, \ldots, n\}$. This covers the case when $Fib(j)$ is called with $2 \le j \le n$.
- $(q_i, f_j, \top) \to \big(q_i, f'_j, \mu\big)$, for each $i \in \{1, 2\}$, $j \in \{0, 1\}$, and μ with $\mu(m_i) = j$ (and $\mu(x) = 0$ for $x \in C \setminus \{m_i\}$). This covers the case when $Fib(j)$ is called with $j = 0, 1$, in which case m_i will be incremented by j.

- $\big(q_i, f'_j(f'_{j-1}, f'_{j-2}), \top\big) \to \big(q_i, f'_j, \mathbf{0}\big)$, for each $i \in \{1, 2\}$, and $j \in \{2, \ldots, n\}$. This covers the case of the synchronisation step (i.e. two subtasks spawned were completed).
- $(q_1, f'_n, \top) \to (q_2, f_n, \mathbf{0})$. This indicates that the system goes to the second stage.
- $(q_2, f'_n, m_1 > 0 \wedge m_2 > 0) \to (q_2, f'_n, \mu)$, for μ with $\mu(x) = -1$ for all $x \in C$. This indicates that the second stage is complete and the two counters are being decremented whenever none of them have emptied.
- $(q_2, f'_n, (m_1 = 0 \wedge m_2 > 0) \vee (m_1 > 0 \wedge m_2 = 0)) \to (q_3, f'_n, \mathbf{0})$. This indicates that the values of the two counters are different and the system goes to q_3 (a bad state).

Let $\alpha_0 = (q_1, f_n, \nu)$ with $\nu(x) = 0$ for each $x \in C$. Observe that α_0 may reach the control state q_3 iff the value of m on input n is not functionally determined.

4. Decidability

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

Theorem 4.1 (Global Reachability for rbGTRS). *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 [12] 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.

4.1. Modes of the counters

The first notion that is crucial in our proof is that of *mode* of a counter [22, 29], 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

$$\mathbf{Modes} = \texttt{REG}^k \times [0, r]^k \times \{\uparrow, \downarrow\}^k .$$

4.2. Building the sGTRS G'

We might be tempted to build G' by first removing the counters from G and then embedding **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 [22]. 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.

4.3. 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 [12].

Recall that ψ' should assert that the desired counter comparisons are performed throughout a run 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 a run of G'. More precisely,

$$\psi' = \left(\begin{array}{c} \varphi_1(\mathbf{x}) \wedge \varphi_2(\mathbf{y}) \wedge \texttt{Dom}(\mathbf{m}_0, \ldots, \mathbf{m}_{N_{\max}}) \wedge \texttt{Init}(\mathbf{m}_0) \wedge \\ \texttt{GoodSeq}(\mathbf{m}_0, \ldots, \mathbf{m}_{N_{\max}}) \wedge \texttt{Respect}(\mathbf{z}, \mathbf{m}_0, \ldots, \mathbf{m}_{N_{\max}}) \wedge \\ \texttt{EndVal}(\mathbf{x}, \mathbf{y}, \mathbf{z}) \end{array} \right) .$$

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:

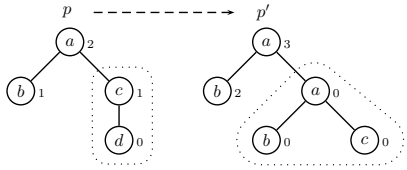
- reg^i_j (for each $j \in [1, k]$) — to encode which of the $2m+1$ possible regions the jth counter is in.
- rev^i_j (for each $j \in [1, k]$) — to encode how many reversals have been used up by the jth counter.
- arr^i_j (for each $j \in [1, k]$) — to encode whether the jth counter is non-incrementing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

We detail each subformula below.

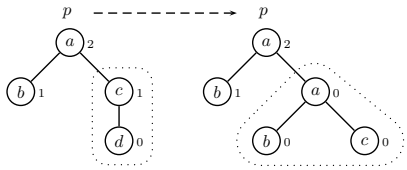
- The subformula `Dom` asserts that each variable in $\mathbf{m}_i$ (for each i) has the right domain (i.e. range of integer values). More precisely, for each $j \in [1, k]$, we add the conjuncts: (i) $0 \leq reg^i_j \leq 2m$, (ii) $0 \leq rev^i_j \leq r$, and (iii) $0 \leq arr^i_j \leq 1$. For the first 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 $rev^0_j = 0$.
- 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) $arr^i_j \neq arr^{i+1}_j \Rightarrow rev^{i+1}_j = rev^i_j + 1$, (ii) $arr^i_j = arr^{i+1}_j \Rightarrow rev^{i+1}_j = rev^i_j$, (iii) $reg^i_j < reg^{i+1}_j \Rightarrow arr^i_j = 1$, and (iv) $reg^i_j > reg^{i+1}_j \Rightarrow arr^i_j = 0$. 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^i_j. 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 $\texttt{StartCounter}^i_j$ 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 $\texttt{EndCounter}^i_j$ the term $\texttt{StartCounter}^i_j + \sum_{(\gamma,\tau,i,0)} \mu_j(\tau) \times z_{(\gamma,\tau,i,0)}$. We add the conjuncts: (i) $reg^i_j = 2h \Rightarrow (\texttt{EndCounter}^i_j \in A_h \wedge \texttt{StartCounter}^i_j \in A_h)$, for each $h \in [0, m]$, and (ii) $reg^i_j = 2h + 1 \Rightarrow (\texttt{EndCounter}^i_j \in B_h \wedge \texttt{StartCounter}^i_j \in B_h)$, 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) $\texttt{StartCounter}^0_j \in A_h \Rightarrow reg^0_j = 2h$, and (2) $\texttt{StartCounter}^0_j \in B_h \Rightarrow reg^0_j = 2h + 1$. 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 $z_{(\gamma,\tau,i,l)} > 0 \Rightarrow \theta(\texttt{StartCounter}^i_1, \ldots, \texttt{StartCounter}^i_k)$. Note that it is not necessary to add $\theta(\texttt{EndCounter}^i_1, \ldots, \texttt{EndCounter}^i_k)$ as a consequence of this implication because we have asserted that $\texttt{StartCounter}^i_j$ is in the same region as $\texttt{EndCounter}^i_j$, and so we have $\theta(\texttt{StartCounter}^i_1, \ldots, \texttt{StartCounter}^i_k) \iff \theta(\texttt{EndCounter}^i_1, \ldots, \texttt{EndCounter}^i_k)$. 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 $arr^i_j = 0 \Rightarrow z_{(\gamma,\tau,i,l)} = 0$; if $\mu_j(\tau) < 0$, then add the conjunct $arr^i_j = 1 \Rightarrow z_{(\gamma,\tau,i,l)} = 0$.
- 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 $y_j = \texttt{EndCounter}^{N_{\max}}_j$ 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 4.1 follows. NP-hardness already holds for the restricted model where the tree component is a stack [22].



(a) A transition changing the control-state.



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

Figure 1: Transitions of a senescent GTRS.

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* [13], 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 5.1 (Undecidability for 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).

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 age 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 $\mathcal{C}_1, \ldots, \mathcal{C}_{n-1}$ 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 $t_i = \mathcal{C}_i[t_i^{\text{out}}]$ and $t_{i+1} = \mathcal{C}_i\big[t_{i+1}^{\text{in}}\big]$ 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 $\mathcal{C}_j\big[t_j^{\text{in}}\big]$ and η is in the domain of $\mathcal{C}_j[x]$ 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 $(p_i, \mathcal{C}_i[t_i^{\text{out}}], \nu_i) \xrightarrow{\gamma_i} \big(p_{i+1}, \mathcal{C}_i\big[t_{i+1}^{\text{in}}\big], \nu_{i+1}\big)$ 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 $\mathcal{C}_i[t_i^{\text{out}}]$ but only in the domain of $\mathcal{C}_i[x]$ if the label is x have an age of at most r.

Definition 5.1 (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. First, we give the intuition behind the construction and then give the full details.

5.2.1. Intuition

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 changes 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 [44]. 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_0 and c_1. In the tree, we track the value of a counter $c \in \{c_0, c_1\}$ by the number of live leaves labelled with the counter name c. E.g. the tree $\bullet(c_0, \bullet(c_1, *))$ 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 $\{c_0^+, c_0^-, c_1^+, c_1^-\}$. When we simulate an increment of c_0 we add a leaf and increment c_0^+. When we simulate a decrement of c_0 we rewrite a leaf to a null character and increment c_0^-. Similarly for c_1. 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 at least 1. Leaves not labelled c can refresh their age to 0 by rewriting themselves. Leaves labelled c will stay aged at least 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 at least 2, while all node that have refreshed 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. Note, other counters may see their value reduced if not all nodes refresh. This is handled below.

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_0 we will have $c_0^+ > c_0^-$. 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. This test will also catch the case noted above where

some counters fail to fully refresh. If no spurious transitions were made, we know the two-counter machine has a corresponding run.

5.2.2. Full Construction

A two-counter machine is a tuple $\mathcal{M} = (\mathcal{S}, \Delta)$ where $\mathcal{S}$ is a finite set of control-states, Δ is a finite set of rules of the form $s_1 \underset{op}{\longrightarrow} s_2$ where $s_1, s_2 \in \mathcal{S}$, and $op \in \{\mathrm{inc}_0, \mathrm{inc}_1, \mathrm{dec}_0, \mathrm{dec}_1, \mathrm{zero}_0, \mathrm{zero}_1\}$. A configuration of $\mathcal{M}$ is a tuple $(s, v_0, v_1) \in \mathcal{S} \times \mathbb{N} \times \mathbb{N}$. We have a transition $\left(s_1, v_0^1, v_1^1\right) \longrightarrow \left(s_2, v_0^2, v_1^2\right)$ if we have a rule $s_1 \underset{op}{\longrightarrow} s_2$ and

- if $op = \mathrm{inc}_i$, $v_i^2 = v_i^1 + 1$ and $v_{1-i}^2 = v_{1-i}^1$,
- if $op = \mathrm{dec}_i$, $v_i^2 = v_i^1 - 1 \geq 0$ and $v_{1-i}^2 = v_{1-i}^1$,
- if $op = \mathrm{zero}_i$, $v_i^2 = v_i^1 = 0$, and $v_{1-i}^2 = v_{1-i}^1$.

Let ν_0 be the valuation assigning 0 to all counters. For given two-counter machine $\mathcal{M}$ and control-states s_0 and s_f we define a senescent rbGTRS $G_{\mathcal{M}}$ such that there is a run

$$(s_0, t_0, \nu_0) \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} (f, t, \nu)$$

for some t and ν iff there is a run

$$(s_0, 0, 0) \longrightarrow \cdots \longrightarrow (s_f, 0, 0)$$

of $\mathcal{M}$. Since this latter problem is well-known to be undecidable, we obtain undecidability of control-state reachability for senescent rbGTRS.

In the following definition we use the following 1-reversal-bounded counters: c_0^+, c_1^+, c_0^- and c_1^-. We use $\mathcal{R}_{\mathrm{fresh}}$ to keep leaf nodes within the lifespan, $\mathcal{R}_{\mathrm{inc}}, \mathcal{R}_{\mathrm{dec}}$, and $\mathcal{R}_{\mathrm{zero}}$ to simulate the counter operations, and $\mathcal{R}_{\mathrm{fin}}$ to check $c_i^+ = c_i^-$ for both i at the end of the run. Furthermore, let

$$\begin{aligned}
\mu_i^+(c) &= \begin{cases} 1 & c = c_i^+ \\ 0 & \text{otherwise,} \end{cases} \\
\mu_i^-(c) &= \begin{cases} 1 & c = c_i^- \\ 0 & \text{otherwise, and} \end{cases} \\
\mu_i^=(c) &= \begin{cases} -1 & c \in \left\{c_i^+, c_i^-\right\} \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}$$

Recall ν_0 maps all counters to zero.

Given a node η and trees $t_1, \ldots, t_n$, we will often write $\eta(t_1, \ldots, t_n)$ to denote the tree with root node η and left-to-right child sub-trees $t_1, \ldots, t_n$. When η is labelled a, we may also write $a(t_1, \ldots, t_n)$ to denote the same tree. We will often simply write a to denote the tree with a single node labelled a.

For a tree t, let $\mathcal{T}_t$ be an NTA accepting only t. For example, $\mathcal{T}_{a(b)}$ is the automaton accepting only the tree $a(b)$, and $\mathcal{T}_a$ accepts only the tree containing

a single node labelled a. Note, we do not use natural numbers as tree labels, hence $\mathcal{T}_1, \mathcal{T}_2, \ldots$ may range over all NTAs.

Definition 5.2 ($G_{\mathcal{M}}$). *Given a two-counter machine $\mathcal{M} = (\mathcal{S}, \Delta)$ and two control-states $s_0, s_f \in \mathcal{S}$, we define a senescent rbGTRS with lifespan 1 and 1 counter reversal*

$$G_{\mathcal{M}} = (\mathcal{P}, \Sigma, \Gamma, \mathcal{R}, C, 1)$$

where

$$\begin{array}{rcl} \mathcal{P} & = & \mathcal{S} \uplus \{(s,i) \mid s \in \mathcal{S} \wedge i \in \{0,1\}\} \uplus \{f, p_{=}\} \\ \Sigma & = & \left\{\bullet, *, \circ, \overline{0}, \overline{1}\right\} \\ \Gamma & = & \{\varepsilon\} \\ C & = & \left\{c_0^+, c_1^+, c_0^-, c_1^-\right\} \\ \mathcal{R} & = & \mathcal{R}_{\text{fresh}} \cup \mathcal{R}_{\text{inc}} \cup \mathcal{R}_{\text{dec}} \cup \mathcal{R}_{\text{zero}} \cup \mathcal{R}_{\text{fin}} \end{array}$$

where the character $\bullet$ has rank 2 and all other characters have rank 0 and

$$\begin{array}{rcl} \mathcal{R}_{\text{fresh}} & = & \left\{(s, \mathcal{T}_\eta, \top) \xrightarrow{\varepsilon} (s, \mathcal{T}_\eta, \nu_0) \;\middle|\; s \in \mathcal{S} \wedge \eta \in \left\{*, \overline{0}, \overline{1}\right\}\right\} \cup \\ & & \left\{((s,i), \mathcal{T}_\eta, \top) \xrightarrow{\varepsilon} ((s,i), \mathcal{T}_\eta, \nu_0) \;\middle|\; s \in \mathcal{S} \wedge \eta \in \left\{*, \overline{1}, \overline{2}\right\} \setminus \left\{\overline{i}\right\}\right\} \\ \mathcal{R}_{\text{inc}} & = & \left\{(s_1, \mathcal{T}_*, \top) \xrightarrow{\varepsilon} \left(s_2, \mathcal{T}_{\bullet(\overline{i},*)}, \mu_i^+\right) \;\middle|\; s_1 \xrightarrow[\text{inc}_i]{} s_2 \in \Delta\right\} \\ \mathcal{R}_{\text{dec}} & = & \left\{(s_1, \mathcal{T}_{\overline{i}}, \top) \xrightarrow{\varepsilon} \left(s_2, \mathcal{T}_\circ, \mu_i^-\right) \;\middle|\; s_1 \xrightarrow[\text{dec}_i]{} s_2 \in \Delta\right\} \\ \mathcal{R}_{\text{zero}} & = & \left\{\begin{array}{l} (s_1, \mathcal{T}_*, \top) \xrightarrow{\varepsilon} ((s_2,i), \mathcal{T}_*, \nu_0)\,, \\ ((s_2,i), \mathcal{T}_*, \top) \xrightarrow{\varepsilon} (s_2, \mathcal{T}_*, \nu_0)\,, \end{array} \;\middle|\; s_1 \xrightarrow[\text{zero}_i]{} s_2 \in \Delta\right\} \\ \mathcal{R}_{\text{fin}} & = & \left\{\begin{array}{c} (s_f, \mathcal{T}_*, \top) \xrightarrow{\varepsilon} (p_{=}, \mathcal{T}_*, \nu_0)\,, \\ (p_{=}, \mathcal{T}_*, \top) \xrightarrow{\varepsilon} (p_{=}, \mathcal{T}_*, \mu_{\overline{0}}^{=})\,, \\ (p_{=}, \mathcal{T}_*, \top) \xrightarrow{\varepsilon} (p_{=}, \mathcal{T}_*, \mu_{\overline{1}}^{=})\,, \\ \left(p_{=}, \mathcal{T}_*, c_0^+ = 0 \wedge c_0^- = 0 \wedge c_1^+ = 0 \wedge c_1^- = 0\right) \xrightarrow{\varepsilon} (f, \mathcal{T}_*, \nu_0) \end{array}\right\} \end{array}$$

Property 5.1 ($G_{\mathcal{M}}$ simulates $\mathcal{M}$). *For a given two-counter machine $\mathcal{M}$ and control-states s_0 and s_f there is a run*

$$(s_0, 0, 0) \longrightarrow \cdots \longrightarrow (s_f, 0, 0)$$

of $\mathcal{M}$ iff there is a run

$$(s_0, t_0, \nu_0) \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} (f, t, \nu)$$

for some t and ν of $G_{\mathcal{M}}$ where $t_0 = *$ is the tree containing only a single node labelled $*$.

Proof. Let $s_1 = s_0$ and $s_n = s_f$ and suppose we have a run

$$(s_1, 0, 0) \longrightarrow \cdots \longrightarrow (s_n, 0, 0)\ .$$

We build the required run of $G_{\mathcal{M}}$ by induction such that for configuration $\left(s_j, v_0^j, v_1^j\right)$ along the run of $\mathcal{M}$, we have a run to a configuration (s_j, t_j, ν_j) of $G_{\mathcal{M}}$ such that

- there is one leaf node labelled $*$, this node has age 0,
- the number of nodes $\bar{i}$ in t_j is v_i^j for each $j \in \{0,1\}$, each having age 0, and
- $\nu_j(c_i^+) - \nu_j(c_i^-) = v_i^j$ for each $i \in \{0,1\}$.

In the base case the result holds trivially for the configuration $(s_1, *, \nu_0)$. Now take a transition (s_j, op, s_{j+1}) from the run of $\mathcal{M}$. By induction we have a run to (s_j, t_j, ν_j) as above. We show how to extend this run to $(s_{j+1}, t_{j+1}, \nu_{j+1})$. There are several cases depending on op. In each case we show how to reach a tree satisfying the induction hypothesis, except the age of the leaf nodes. After the case analysis we show how to satisfy the age requirement also.

- When $op = \mathrm{inc}_i$, we use $(s_j, \mathcal{T}_*, \top) \xrightarrow{\varepsilon} \left(s_{j+1}, \mathcal{T}_{\bullet(\bar{i},*)}, \mu_i^+\right)$. It is easy to verify we reach $(s_{j+1}, t_{j+1}, \nu_{j+1})$ as required.
- When $op = \mathrm{dec}_i$, we know the ith counter must have a value greater than zero, hence we can apply $(s_j, \mathcal{T}_{\bar{i}}, \top) \xrightarrow{\varepsilon} \left(s_{j+1}, \mathcal{T}_\circ, \mu_i^-\right)$. It is easy to verify we reach $(s_{j+1}, t_{j+1}, \nu_{j+1})$ as required.
- When $op = \mathrm{zero}_i$, we know the ith counter must have value zero, hence there are no leaves labelled $\bar{i}$ in t_j. We can apply the following sequence of rules.
  1. $(s_j, \mathcal{T}_*, \top) \xrightarrow{\varepsilon} ((s_{j+1}, i), \mathcal{T}_*, \nu_0)$,
  2. $((s_{j+1}, i), \mathcal{T}_\eta, \top) \xrightarrow{\varepsilon} ((s_{j+1}, i), \mathcal{T}_\eta, \nu_0)$ to each leaf labelled by some $\eta \in \{*, \bar{0}, \bar{1}\} \setminus \{\bar{i}\}$,
  3. $((s_{j+1}, i), \mathcal{T}_*, \top) \xrightarrow{\varepsilon} (s_{j+1}, \mathcal{T}_*, \nu_0)$.

  It is easy to verify we reach $(s_{j+1}, t_{j+1}, \nu_{j+1})$ as required.

Finally, to obtain the age restriction on all leaf nodes, we apply $(s_{j+1}, \mathcal{T}_\eta, \top) \xrightarrow{\varepsilon} (s_{j+1}, \mathcal{T}_\eta, \nu_0)$ to each leaf labelled by some $\eta \in \{*, \bar{0}, \bar{1}\}$.

Thus, by induction, we can reach a configuration (s_f, t, ν) such that, for each i we have $\nu(c_i^+) = \nu(c_i^-)$. Thus, we can apply a sequence of rules from $\mathcal{R}_{\mathrm{fin}}$ to reach (f, t, ν_0). In particular, we apply $(s_f, \mathcal{T}_*, \top) \xrightarrow{\varepsilon} (p_=, \mathcal{T}_*, \nu_0)$ and then simultaneously reduce each reversal-bounded counter to zero using $(p_=, \mathcal{T}_*, \top) \xrightarrow{\varepsilon} (p_=, \mathcal{T}_*, \mu_i^=)$ repeatedly for each i, and then finally apply

$$\left(p_=, \mathcal{T}_*, c_0^+ = 0 \wedge c_0^- = 0 \wedge c_1^+ = 0 \wedge c_1^- = 0\right) \xrightarrow{\varepsilon} (f, \mathcal{T}_*, \nu_0)$$

to complete this direction of the proof.

We prove the opposite direction via two inductions. First, take some run of $G_{\mathcal{M}}$, which necessarily has the form

$$(p_1, t_1, \nu_1) \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} (p_n, t_n, \nu_n) \xrightarrow{\varepsilon} (p_=, t_n, \nu_n) \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} (p_=, t_n, \nu_0) \xrightarrow{\varepsilon} (f, t_n, \nu_0)$$

where the last sequence of transitions (from p_n) are all from $\mathcal{R}_{\text{fin}}$, $p_1 = s_0$, $t_1 = *$, $\nu_1 = \nu_0$, and $p_n = s_f$. Let $\#_i(t)$ be the number of leaves labelled $\bar{i}$ in t. We first prove by induction over the run that for all $1 \leq j \leq n$ and $i \in \{0, 1\}$ we have $\#_i(t_j) = \nu_j\left(c_i^+\right) - \nu_j\left(c_i^-\right)$. This is a straightforward induction that can be seen by observing

- the base case is immediate,
- all rules in $\mathcal{R}_{\text{fresh}} \cup \mathcal{R}_{\text{zero}}$ do not change $\#_i(t_j)$, $\nu_j\left(c_i^+\right)$, or $\nu_j\left(c_i^-\right)$,
- all rules in $\mathcal{R}_{\text{inc}}$ increase both $\#_i(t_j)$, and $\nu_j\left(c_i^+\right)$, by one, and leave $\nu_j\left(c_i^-\right)$ unchanged,
- all rules in $\mathcal{R}_{\text{dec}}$ decrease $\#_i(t_j)$ by one, increase $\nu_j\left(c_i^-\right)$ by one, and leave $\nu_j\left(c_i^+\right)$, unchanged, and
- there are no rules from $\mathcal{R}_{\text{fin}}$.

Given $\#_i(t_j) = \nu_j\left(c_i^+\right) - \nu_j\left(c_i^-\right)$ for all j and i, we construct, also by induction, a sequence

$$\left(s_1, v_0^1, v_1^1\right), \ldots, \left(s_n, v_0^n, v_1^n\right)$$

of $\mathcal{M}$ such that for all j and i we have $\#_i(t_j) = v_i^j$ and $p_j \in \{s_j, (s_j, 0), (s_j, 1)\}$ and, either

- $\left(s_j, v_0^j, v_1^j\right) \longrightarrow \left(s_{j+1}, v_0^{j+1}, v_1^{j+1}\right)$ is a transition of $\mathcal{M}$, or
- $\left(s_j, v_0^j, v_1^j\right) = \left(s_{j+1}, v_0^{j+1}, v_1^{j+1}\right)$.

In the base case we set $\left(s_1, v_0^1, v_0^1\right) = (s_0, 0, 0)$. Next, take a transition

$$(p_j, t_j, \nu_j) \xrightarrow{\varepsilon} (p_{j+1}, t_{j+1}, \nu_{j+1})$$

of $G_{\mathcal{M}}$. There are several cases depending on which rule τ was applied.

- If $\tau \in \mathcal{R}_{\text{fresh}}$ then we set $\left(s_j, v_0^j, v_1^j\right) = \left(s_{j+1}, v_0^{j+1}, v_1^{j+1}\right)$ and the properties follow from $\left(s_j, v_0^j, v_1^j\right)$ by induction.
- If $\tau \in \mathcal{R}_{\text{inc}}$ then for some i we have $\tau = (s_j, \mathcal{T}_*, \top) \xrightarrow{\varepsilon} \left(s_{j+1}, \mathcal{T}_{\bullet(\bar{i},*)}, \mu_i^+\right)$ and $s_j \xrightarrow[\text{inc}_i]{} s_{j+1}$ is a rule of $\mathcal{M}$. We apply this rule to obtain $\left(s_j, v_0^j, v_1^j\right) \longrightarrow \left(s_{j+1}, v_0^{j+1}, v_1^{j+1}\right)$ and we can directly verify $\#_i(t_{j+1}) = v_i^{j+1}$ for each i as required.
- If $\tau \in \mathcal{R}_{\text{dec}}$ then for some i we have $\tau = (s_j, \mathcal{T}_{\bar{i}}, \top) \xrightarrow{\varepsilon} \left(s_{j+1}, \mathcal{T}_\circ, \mu_i^-\right)$ and $s_j \xrightarrow[\text{dec}_i]{} s_{j+1}$ is a rule of $\mathcal{M}$. We apply this rule to obtain $\left(s_j, v_0^j, v_1^j\right) \longrightarrow \left(s_{j+1}, v_0^{j+1}, v_1^{j+1}\right)$ and we can directly verify $\#_i(t_{j+1}) = v_i^{j+1}$ for each i as required.

- If $\tau \in \mathcal{R}_{\text{zero}}$ there are two sub-cases.
  - In the first case, for some i we have $\tau = (s_j, \mathcal{T}_*, \top) \xrightarrow{\varepsilon} ((s_{j+1}, i), \mathcal{T}_*, \nu_0)$ and $s_j \xrightarrow[\text{zero}_i]{} s_{j+1}$ is a rule of $\mathcal{M}$. If we apply this rule we obtain $\left(s_j, v_0^j, v_1^j\right) \longrightarrow \left(s_{j+1}, v_0^{j+1}, v_1^{j+1}\right)$ and we can directly verify $\#_i(t_{j+1}) = v_i^{j+1}$ for each i as required. However, we need to prove $s_j \xrightarrow[\text{zero}_i]{} s_{j+1}$ can be applied. That is, we need to prove v_i^j is zero. Here we use $\#_i(t_{j'}) = \nu_{j'}\left(c_i^+\right) - \nu_{j'}\left(c_i^-\right)$ for all j'. From the definition of $G_{\mathcal{M}}$ we know that the run from $((s_{j+1}, i), t_{j+1}, \nu_{j+1})$ must eventually reach s_{j+1} where (s_{j+1}, i) is the only control-state seen before s_{j+1} is reached. During this time, we cannot refresh any node labelled $\bar{i}$. Thus, assume for contradiction that v_i^j is not zero. Since $\#_i(t_j) = v_i^j$ we know there is at least one leaf labelled $\bar{i}$. Since this node cannot refresh while the control-state is (s_{j+1}, i) this node will have age 2 once s_{j+1} is reached. Thus, since the lifespan is 1, this node cannot be rewritten by the end of the run. This means t_n has at least one node labelled $\bar{i}$. Since $1 \leq \#_i(t_n) = \nu_n\left(c_i^+\right) - \nu_n\left(c_i^-\right)$ we know $\nu_n\left(c_i^+\right) \neq \nu_n\left(c_i^-\right)$. However, the final transitions of the run of $G_{\mathcal{M}}$ use rules from $\mathcal{R}_{\text{fin}}$ and have the effect of ensuring $\nu_n\left(c_i^+\right) = \nu_n\left(c_i^-\right)$. Hence, we have a contradiction, and $v_i^j = 0$. Thus we can apply $s_j \xrightarrow[\text{zero}_i]{} s_{j+1}$ as needed.
  - If we have $\tau = ((s_j, i), \mathcal{T}_*, \top) \xrightarrow{\varepsilon} (s_{j+1}, \mathcal{T}_*, \nu_0)$ then we set $\left(s_j, v_0^j, v_1^j\right) = \left(s_{j+1}, v_0^{j+1}, v_1^{j+1}\right)$ which we know satisfies the required properties since $\left(s_j, v_0^j, v_1^j\right)$ did by induction.

Thus, we have a sequence $\left(s_1, v_0^1, v_1^1\right), \ldots, \left(s_n, v_0^n, v_1^n\right)$ from which we can immediately extract a run of $\mathcal{M}$ from $\left(s_1, v_0^1, v_1^1\right) = (s_0, 0, 0)$ to $\left(s_n, v_0^n, v_1^n\right) = (s_f, v_0^n, v_1^n)$. That $v_0^n = v_1^n = 0$ follows since the final transitions from p_n have the effect of asserting $\nu_n\left(c_i^+\right) - \nu_n\left(c_i^-\right) = 0$ from which we conclude $\#_i(t_n) = 0$ and since $v_i^n = \#_i(t_n)$ we complete the proof as required. □

Thus, via Property 5.1 we can reduce the reachability problem for two-counter machines to the control-state reachability problem for senescent rbGTRS. Thus, we show the control-state reachability problem is undecidable and complete the proof of Theorem 5.1.

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 [45]). 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 of the conference and journal versions for their helpful feedback. This work was supported by the Engineering and Physical Sciences Research Council [EP/K009907/1] and Oxford University Startup Fund.

References

[1] M. Hague, A. W. Lin, Decidable models of integer-manipulating programs with recursive parallelism, in: Reachability Problems - 10th International Workshop, RP 2016, Aalborg, Denmark, September 19-21, 2016, Proceedings, 2016, pp. 148–162. doi:10.1007/978-3-319-45994-3_11.
URL http://dx.doi.org/10.1007/978-3-319-45994-3_11

[2] S. Qadeer, J. Rehof, Context-bounded model checking of concurrent software., in: TACAS, 2005, pp. 93–107.

[3] S. Qadeer, The case for context-bounded verification of concurrent programs, in: SPIN, 2008, pp. 3–6.

[4] M. Hague, A. W. Lin, Synchronisation- and reversal-bounded analysis of multithreaded programs with counters, in: CAV, 2012, pp. 260–276. doi:10.1007/978-3-642-31424-7_22.
URL http://dx.doi.org/10.1007/978-3-642-31424-7_22

[5] P. A. Abdulla, M. F. Atig, J. Cederberg, Analysis of message passing programs using SMT-solvers, in: ATVA, 2013, pp. 272–286. doi:10.1007/978-3-319-02444-8_20.
URL http://dx.doi.org/10.1007/978-3-319-02444-8_20

[6] J. Esparza, P. Ganty, T. Poch, Pattern-based verification for multithreaded programs, ACM Trans. Program. Lang. Syst. 36 (3) (2014) 9:1–9:29. doi:10.1145/2629644.
URL http://doi.acm.org/10.1145/2629644

[7] G. Ramalingam, Context-sensitive synchronization-sensitive analysis is undecidable, Transactions on Programming Languages and Systems (TOPLAS).

[8] D. Suwimonteerabuth, J. Esparza, S. Schwoon, Symbolic context-bounded analysis of multithreaded java programs, in: SPIN, 2008, pp. 270–287.

[9] M. F. Atig, A. Bouajjani, S. Qadeer, Context-bounded analysis for concurrent programs with dynamic creation of threads, Logical Methods in Computer Science 7 (4).

[10] A. Lal, T. Touili, N. Kidd, T. Reps, Interprocedural analysis of concurrent programs under a context bound, in: TACAS, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 282–298.
URL http://portal.acm.org/citation.cfm?id=1792734.1792761

[11] M. Musuvathi, S. Qadeer, Iterative context bounding for systematic testing of multithreaded programs, in: PLDI, 2007, pp. 446–455.

[12] A. W. Lin, Weakly-synchronized ground tree rewriting (with applications to verifying multithreaded programs), in: MFCS, 2012, pp. 630–642.

[13] M. Hague, Senescent ground tree rewrite systems, in: CSL-LICS, 2014, pp. 48:1–48:10.

[14] S. L. Torre, P. Madhusudan, G. Parlato, A robust class of context-sensitive languages, in: In LICS, IEEE Computer Society, 2007, pp. 161–170.

[15] M. F. Atig, B. Bollig, P. Habermehl, Emptiness of multi-pushdown automata is 2etime-complete, in: DLT, 2008, pp. 121–133.

[16] C. Aiswarya, P. Gastin, K. N. Kumar, Verifying communicating multi-pushdown systems via split-width, in: ATVA, 2014, pp. 1–17.

[17] M. F. Atig, K. N. Kumar, P. Saivasan, Adjacent ordered multi-pushdown systems, Int. J. Found. Comput. Sci. 25 (8) (2014) 1083–1096.

[18] P. Ganty, R. Majumdar, M. Monmege, Bounded underapproximations, FMSD 40 (2).

[19] P. Madhusudan, G. Parlato, The tree width of auxiliary storage, in: POPL, 2011, pp. 283–294.

[20] S. La Torre, M. Napoli, G. Parlato, Scope-bounded pushdown languages, in: DLT, 2014, pp. 116–128. doi:10.1007/978-3-319-09698-8_11.
URL http://dx.doi.org/10.1007/978-3-319-09698-8_11

[21] W. Czerwinski, P. Hofman, S. Lasota, Reachability problem for weak multi-pushdown automata, Logical Methods in Computer Science 9 (3). doi:10.2168/LMCS-9(3:13)2013.
URL http://dx.doi.org/10.2168/LMCS-9(3:13)2013

[22] M. Hague, A. W. Lin, Model checking recursive programs with numeric data types, in: CAV, 2011, pp. 743–759.

[23] S. Göller, A. W. Lin, Refining the process rewrite systems hierarchy via ground tree rewrite systems, in: CONCUR, 2011, pp. 543–558.

[24] A. Bouajjani, J. Esparza, O. Maler, Reachability analysis of pushdown automata: Application to model-checking, in: CONCUR, 1997, pp. 135–150.

[25] J. Esparza, A. Podelski, Efficient algorithms for pre* and post* on interprocedural parallel flow graphs, in: POPL, 2000, pp. 1–11.

[26] R. Mayr, Decidability and complexity of model checking problems for infinite-state systems, Ph.D. thesis, TU-Munich (1998).

[27] M. Kretínský, V. Rehák, J. Strejcek, Extended process rewrite systems: Expressiveness and reachability, in: CONCUR, 2004, pp. 355–370.

[28] S. Bardin, A. Finkel, J. Leroux, P. Schnoebelen, Flat acceleration in symbolic model checking, in: ATVA, 2005, pp. 474–488.

[29] O. H. Ibarra, Reversal-bounded multicounter machines and their decision problems, J. ACM 25 (1) (1978) 116–133.

[30] A. W. To, L. Libkin, Algorithmic metatheorems for decidable LTL model checking over infinite systems, in: FOSSACS, 2010, pp. 221–236.

[31] A. Bouajjani, M. Emmi, Analysis of recursively parallel programs, ACM Trans. Program. Lang. Syst. 35 (3) (2013) 10.

[32] K. N. Verma, J. Goubault-Larrecq, Karp-Miller trees for a branching extension of VASS, Discrete Mathematics & Theoretical Computer Science 7 (1) (2005) 217–230.
URL http://www.dmtcs.org/volumes/abstracts/dm070113.abs.html

[33] S. Demri, M. Jurdzinski, O. Lachish, R. Lazic, The covering and boundedness problems for branching vector addition systems, J. Comput. Syst. Sci. 79 (1) (2013) 23–38. doi:10.1016/j.jcss.2012.04.002.
URL http://dx.doi.org/10.1016/j.jcss.2012.04.002

[34] J. Leroux, M. Praveen, G. Sutre, Hyper-ackermannian bounds for pushdown vector addition systems, in: CSL-LICS, 2014, pp. 63:1–63:10. doi:10.1145/2603088.2603146.
URL http://doi.acm.org/10.1145/2603088.2603146

[35] M. F. Atig, P. Ganty, Approximating petri net reachability along context-free traces, in: FSTTCS, 2011, pp. 152–163. doi:10.4230/LIPIcs.FSTTCS.2011.152.
URL http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2011.152

[36] M. Lang, C. Löding, Modeling and verification of infinite systems with resources, Logical Methods in Computer Science 9 (4). doi:10.2168/LMCS-9(4:22)2013.
URL http://dx.doi.org/10.2168/LMCS-9(4:22)2013

[37] T. Colcombet, C. Löding, Regular cost functions over finite trees, in: LICS, 2010, pp. 70–79. doi:10.1109/LICS.2010.36.
URL http://dx.doi.org/10.1109/LICS.2010.36

[38] A. Blumensath, T. Colcombet, D. Kuperberg, P. Parys, M. Vanden Boom, Two-way cost automata and cost logics over infinite trees, in: CSL-LICS, 2014, pp. 16:1–16:9. doi:10.1145/2603088.2603104.
URL http://doi.acm.org/10.1145/2603088.2603104

[39] B. Scarpellini, Complexity of subcases of Presburger arithmetic, Trans. of AMS 284 (1) (1984) 203–218.

[40] M. Dauchet, S. Tison, The theory of ground rewrite systems is decidable, in: LICS, 1990, pp. 242–248.

[41] C. Löding, Reachability problems on regular ground tree rewriting graphs, Theory Coput. Syst. 39 (2) (2006) 347–383.

[42] L. Bozzelli, M. Kretínský, V. Rehák, J. Strejcek, On decidability of LTL model checking for process rewrite systems, Acta Inf. 46 (1) (2009) 1–28.

[43] https://www.cilkplus.org/tutorial-cilk-plus-reducers (cited in April 2017), Cilk with reducers.

[44] T. Araki, T. Kasami, Some Decision Problems Related to the Reachability Problem for Petri Nets, Theoretical Computer Science 3 (1) (1977) 85–104.

[45] L. M. de Moura, N. Bjørner, Z3: An efficient smt solver, in: TACAS, 2008, pp. 337–340.