Theoretical Computer Science 410 (2009) 4169–4179

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Deciding branching time properties for asynchronous programs

Rohit Chadha *, Mahesh Viswanathan

Department of Computer Science, University of Illinois at Urbana–Champaign, United States

ARTICLE INFO

Article history:
Received 7 May 2008
Received in revised form 30 October 2008
Accepted 21 January 2009
Communicated by J. Esparza

Keywords:
Asynchronous programs
Well-structured transition systems
Model checking

ABSTRACT

Asynchronous programming is a paradigm that supports asynchronous function calls in addition to synchronous function calls. Programs in such a setting can be modeled by automata with counters that keep track of the number of pending asynchronous calls for each function, as well as a call stack for synchronous recursive computation. These programs have the restriction that an asynchronous call is processed only when the call stack is empty. The decidability of the control state reachability problem for such systems was recently established. In this paper, we consider the problems of checking other branching time properties for such systems. Specifically we consider the following problems — *termination*, which asks if there is an infinite (non-terminating) computation exhibited by the system; *control state maintainability*, which asks if there is a maximal execution of the system, where all the state visited lie in some "good" set; whether the system can be simulated by a given finite state system; and whether the system can simulate a given finite state system. We present decision algorithms for all these problems.

1. Introduction

Asynchronous programming [11,8,10,15,16,5,13] is a programming paradigm for writing multi-threaded applications, as well as event driven embedded programs. In this paradigm, programs have the ability to make two types of function calls: conventional *synchronous* calls where a caller waits until the callee completes computation, and *asynchronous* calls that return immediately, but are processed later when dispatched by a scheduler. Such systems can be conveniently abstracted (using standard techniques like predicate abstraction [9]) by automata with a stack to model recursive computation, and a collection of pending asynchronous calls, with the restriction that a new asynchronous call is processed only when the stack is empty.

The invariant verification problem for such systems has been extensively studied in the recent past. Sen and Viswanathan [17] showed that the control state reachability problem is decidable, when no assumption is made about the order in which pending asynchronous calls are serviced; in other words, the pending asynchronous calls are stored in a bag or multi-set. The proof of decidability was simplified in [12], and the decidability result was extended to systems with real time clocks in [4]. Another set of results concerns systems where the asynchronous calls are dispatched in order. In [4], we showed that the control state reachability problem is decidable if the pending asynchronous calls are stored in a lossy FIFO buffer. In addition, it has been shown in [18] that the control state reachability problem within a bounded number of context switches is decidable when the asynchronous calls are stored in a FIFO buffer (that is not lossy).

In this paper, we consider the problem of verifying asynchronous programs with respect to branching time properties. The asynchronous calls are not assumed to be processed in any particular order. More specifically we consider systems to be modeled by *counter automata with an auxiliary store*. These machines have a finite collection of counters, one for each asynchronous function — the value of the counter stores the number of pending calls of that function. In addition, we assume

* Corresponding author. Tel.:+1 2172650982; fax: +1 2172654035.
E-mail addresses: rch@cs.uiuc.edu (R. Chadha), vmahesh@cs.uiuc.edu (M. Viswanathan).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.01.021

that the automaton has access to auxiliary data store into which it can store and retrieve information. The additional data store could be any data structure; for example, it could be a stack to model synchronous recursive calls, or more generally, it could be a higher-order stack to model synchronous (safe) higher-order recursive schemes.

It is well-known that the model checking problem for counter automata without zero-test (or petri nets) is undecidable when considering specification logics in which it is possible to express a zero test for counters [6]. This is shown by reducing the halting problem of counter machines to the model checking problem for counter automata/petri nets. Thus, verifying general CTL, CTL*, or modal μ-calculus properties is known to be undecidable. In all these logics a zero test can be expressed by checking the enabled-ness of decrement transitions.

Therefore, we consider verifying 4 types of properties for counter automata with auxiliary stores. The first is termination, which asks whether the system has an infinite (non-terminating) computation. The second property we consider is that of control state maintainability, which asks whether the system has a (maximal) computation such that the control states at all points during the computation belong to a "good" set. The last two properties we investigate relate to checking simulation. Given a finite state system, we ask whether the counter automaton with auxiliary store is simulated by it, and whether the finite state system is simulated by the counter automaton. All these properties are natural and have been long studied within the context of algorithmic verification of well structured transition systems. Thus our results here can be seen as a continuation of the work, initiated in [4], wherein one tries to identify sufficient structural conditions on transition system that allow decision procedures for well-structured transition systems to be combined with other decision procedures for infinite state systems to yield decidability results. Other than this theoretical reason, the 4 properties considered here are also practically relevant in the context asynchronous programs. Termination is a basic property which is often at the foundation of proving many liveness properties [1]. Control state maintainability if true can be seen as asserting that there is an execution, where you can never "escape" to a recovery state. Finally, simulation with finite state systems, allows one to express properties like "is there a computation that reaches an unsafe state", or "is there a computation that visits a sequence of states in order, either consecutively or not". Such specifications often need to be verified of any system, including asynchronous programs.

The main observation proved here is that all these problems are *decidable* provided the corresponding problems are decidable for automata with the same auxiliary store, but without additional counters. More specifically the termination and control state maintainability problem are decidable, if the termination problem is decidable for automata without counters but with the data store. And for checking simulations we require that reachability games be decidable for game graphs generated by automata without counters. An immediate consequence of these results is that these problems are decidable when the auxiliary store is either a stack or a higher-order stack, and hence for asynchronous programs.

Before presenting the technical details of our results, we would like to highlight a few aspects of our results. Previous decidability results on reachability for asynchronous programs [17,4] are based on algorithms that follow the same template as algorithms for *well-structured transition systems (WSTS)* [2,7] — they do exactly what the reachability algorithm for WSTSs would do, except that some backward steps of the WSTS algorithm are replaced by calls to decide reachability on automata without counters but with an auxiliary data store. The same is true for most of the algorithms presented in this paper; they follow the same template as the algorithm for the corresponding problem on WSTS, with some of the basic steps being replaced by solving a similar problem on automata without counters. The only exception to this pattern is our algorithm for deciding if a finite state system is simulated by a counter automata with stores. The algorithm we present is very different than the standard algorithm for the same problem on WSTS [7]. Next, for the control state reachability problem we were able to generalize the decidability result even when the system has real time clocks or when the pending asynchronous calls are stored in a lossy FIFO buffer [4], by leveraging properties of *well-quasi-orders (wqo)* [14]. While we can do the same for the termination and control state maintainability problems, we cannot extend the results for the simulation problems. Our proofs rely on crucial properties exhibited by the special wqo of counters.

Paper outline. The rest of the paper is organized as follows. We start with some preliminaries in Section 2 and formally define counter automaton with an auxiliary store in Section 3. We show the decidability of termination and control state maintainability in Section 4. We consider the problems of simulation of a counter automaton with store by a finite state transition system and of a finite transition system by a counter automaton with store in Sections 5 and 6 respectively.

2. Preliminaries

Transition systems. We shall use labeled transition systems as models of temporal behavior of infinite-state systems. Formally, a transition system $\mathbf{S}$ is a tuple $(S, \Lambda, \rightarrow, s_0)$ such that $\mathbf{S}$ is a set of *configurations*; Λ is a *finite* set of *labels*; $\rightarrow \subseteq S \times \Lambda \times S$ is a set of *transitions* and $s_0 \in S$ is the *initial configuration*. A transition system is said to be *finite* if S is finite. A transition system is said to be *finitely branching* if for each $s \in S$, the set $\mathsf{Succ}_s = \{s' \mid \exists \lambda \in \Lambda.\ (s, \lambda, s') \in \rightarrow\}$ is finite. For the rest of the paper, all transition systems considered will be finitely branching.

We shall often write $s \xrightarrow{\lambda} s'$ if $(s, \lambda, s') \in \rightarrow$. As usual, we shall also use Λ^n to denote words of length n over the alphabet Λ and Λ^* to denote the set of all finite words over Λ. For any word $w = \lambda_1\lambda_2 \ldots \lambda_n \in \Lambda^n$, we say that $s \xrightarrow{w} s'$ iff there exists $s = s_0, s_1, \ldots s_n = s'$ such that $s_i \xrightarrow{\lambda_{i+1}} s_{i+1}$ for all $0 \leq i < n$. For $n = 0$, $\Lambda^n = \{\epsilon\}$, where ϵ is the empty word. We say that $s \xrightarrow{\epsilon} s'$ iff $s = s'$. We say that $s \rightarrow^n s'$ if there is some word $w \in \Lambda^n$ such that $s \xrightarrow{w} s'$. Finally, we say that $s \rightarrow^* s'$ if there is some $n \geq 0$ such that $s \rightarrow^n s'$.

Simulation relation. Given two transition systems $\mathbf{S}_1 = (S_1, \Lambda, \rightarrow_1, s_{i_1})$ and $\mathbf{S}_2 = (S_2, \Lambda, \rightarrow_2, s_{i_2})$, a relation $R \subseteq S_1 \times S_2$ is said to be a *simulation relation* if for each $(s_1, s_2) \in R$ and each $s_1 \xrightarrow{\lambda}_1 s_1'$ there is a s_2' such that $s_2 \xrightarrow{\lambda}_2 s_2'$ and $(s_1', s_2') \in R$. We shall say that $s_1 \in S_1$ is *simulated* by $s_2 \in S_2$ (written as $s_1 \sqsubseteq^{\mathbf{S}_1 \times \mathbf{S}_2} s_2$) if there is a simulation relation R such that $(s_1, s_2) \in R$; when the transition systems $\mathbf{S}_1$ and $\mathbf{S}_2$ are clear from the context, we will drop the superscript and simply write $s_1 \sqsubseteq s_2$, instead of $s_1 \sqsubseteq^{\mathbf{S}_1 \times \mathbf{S}_2} s_2$. We say that the transition system $\mathbf{S}_1$ is *simulated* by the transition system $\mathbf{S}_2$ if $s_{i_1} \sqsubseteq s_{i_2}$.

Well-quasi-orders. A pre-order (reflexive and transitive binary relation) $\preceq$ on a set Q is said to be a *well-quasi-order* if every countably infinite sequence of elements $q_0, q_1, \ldots$, from Q contains elements $q_r \preceq q_s$ for some $0 \leq r < s$. The pair $(\mathsf{Q}, \preceq)$ is said to be a w.q.o. if $\preceq$ is a well-quasi-order on Q. We shall often write $q \succeq q'$ if $q' \preceq q$. Given $\mathsf{Q}' \subseteq \mathsf{Q}$, we say that $\mathsf{M}_{\mathsf{Q}'} \subseteq \mathsf{Q}'$ is a *minor set* for Q' if (i) for all $q_1 \in \mathsf{Q}'$ there is a $q \in \mathsf{M}_{\mathsf{Q}'}$ such that $q \preceq q_1$, and (ii) for all $q, q' \in \mathsf{M}_{\mathsf{Q}'}$, $q \neq q'$ implies $q \not\preceq q'$. Every subset of Q has a *finite* minor set. If the pre-order $\preceq$ is also a partial order (i.e., $\preceq$ is also anti-symmetric) then every subset of Q has a *unique* minor set.

A set $\mathsf{U} \subseteq \mathsf{Q}$ is said to be *upward closed* if for every $q_1 \in \mathsf{U}$ and $q_2 \in \mathsf{Q}$, $q_1 \preceq q_2$ implies that $q_2 \in \mathsf{U}$. An upward closed set is completely determined by its minor set: if M_U is a minor set for U then $\mathsf{U} = \{q \in \mathsf{Q} \mid \exists q_m \in \mathsf{M}_\mathsf{U} \text{ s.t. } q_m \preceq q\}$. Also any subset $\mathsf{Q}' \subseteq \mathsf{Q}$ determines an upward closed set, $\mathsf{U}_{\mathsf{Q}'} = \{q \mid \exists q' \in Q' \text{ s.t. } q' \preceq q\}$. The following important observation follows from w.q.o. theory [14].

Proposition 2.1. *For every infinite sequence of upward closed sets* $\mathsf{U}_0, \mathsf{U}_1, \ldots$ *such that* $\mathsf{U}_r \subseteq \mathsf{U}_{r+1}$ *for each* $r \geq 0$ *there is a* $j \geq 0$ *such that* $\mathsf{U}_i = \mathsf{U}_j$ *for all* $i \geq j$.

Counter w.q.o.'s, ranks and projections. The usual $\leq$ relation on the set of natural numbers $\mathbb{N}$ is a well-quasi-order. The set $\mathbb{N}^l$ (Cartesian product on l copies of $\mathbb{N}$) also forms a w.q.o. with the usual pointwise ordering. Given a finite set Q, the pointwise ordering on $\mathbb{N}^l$ can be extended to $\mathsf{Q} \times \mathbb{N}^l$ as follows—$(q, n_1, \ldots n_l) \leq (q', n_1', \ldots n_l')$ iff $q = q'$ and $n_j \leq n_j'$ for all $1 \leq j \leq l$. The resulting order is also a w.q.o.. The resulting order is also easily seen to be a partial order. For the rest of the paper we shall be mainly concerned with such w.q.o.'s and henceforth refer to them as *counter* w.q.o.'s.

Given a counter w.q.o. $\mathsf{C} = (\mathsf{Q} \times \mathbb{N}^l, \leq)$, we define the monotonic function $\mathsf{rank} : \mathsf{Q} \times \mathbb{N}^l \to \mathbb{N}$ as $\mathsf{rank}(q, n_1, \ldots n_l) = \max(\{n_i \mid 1 \leq i \leq l\})$. The function rank can be extended to upward closed sets [4] as follows : given any upward closed subset $\mathsf{U} \subseteq \mathsf{Q} \times \mathbb{N}^l$, the function rank is the maximum rank of minimal elements of U. In other words, $\mathsf{rank}(\mathsf{U}) = \max\{\mathsf{rank}(c) \mid c \in \mathsf{M}_\mathsf{U}\}$. Note that as the order $\leq$ on $\mathsf{Q} \times \mathbb{N}^l$ is a partial order, the function rank is well-defined.[1]

For each $k \geq 0$, we define a monotonic function $\mathsf{pr}_k : \mathsf{Q} \times \mathbb{N}^l \to \mathsf{Q} \times \mathbb{N}^l$ as $\mathsf{pr}_k((q, n_1, n_2, \ldots n_l)) = (q, \min(n_1, k), \min(n_2, k), \ldots \min(n_2, k))$. The important property of the function pr_k is as follows.

Proposition 2.2. *If* $c_1 \leq c$ *and* $\mathsf{rank}(c_1) \leq k$ *then* $c_1 \leq \mathsf{pr}_k(c)$. *If* U *is an upward-closed set,* $c \in \mathsf{U}$ *and* $k_0 = \mathsf{rank}(\mathsf{U})$ *then* $\mathsf{pr}_k(c) \in \mathsf{U}$ *for each* $k \geq k_0$.

Data structures. We shall model the auxiliary store as pointed data structures which are defined in [4]. Formally, a *pointed data structure* is a tuple $\mathsf{D} = (D, \widetilde{op}, \widetilde{pred}, \overline{d})$ such that D is a set, elements of which are called *data values*; $\widetilde{op}$ is a finite collection of functions $f : D \to D$; $\widetilde{pred}$ is a finite collection of unary predicates on D; and $\overline{d}$ is an element of D, called the *initial data value*. It is assumed that $\widetilde{op}$ contains the identity function id and $\widetilde{pred}$ contains a predicate $\overline{p}$ such that $\overline{p}(d)$ is true iff $d = \overline{d}$.

As described in [4], pushdown stores and higher order pushdown stores can be seen as instances of a pointed data structure. For example, a pushdown store on a finite alphabet Γ in a pushdown automata can be formalized as follows. The set Γ^* (set of all finite strings over Γ) can be taken as the set of data values with the empty string ϵ as the initial value. The set of predicates $\widetilde{pred}$ can be chosen as $\{\mathsf{empty}\} \cup \{\mathsf{top}_\gamma \mid \gamma \in \Gamma\} \cup \{\mathsf{any}\}$, where $\mathsf{empty} = \{\epsilon\}$, $\mathsf{top}_\gamma = \{w\gamma \mid w \in \Gamma^*\}$ (the top of stack is γ) and $\mathsf{any} = \Gamma^*$ (any stack). The set of functions $\widetilde{op}$ can be defined as $\{\mathsf{id}\} \cup \{\mathsf{push}_\gamma \mid \gamma \in \Gamma\} \cup \{\mathsf{pop}_\gamma \mid \gamma \in \Gamma\}$ where id is the identity function and the functions push_γ and pop_γ are defined as follows. For all $w \in \Gamma^*$, $\mathsf{push}_\gamma(w) = w\gamma$ and $\mathsf{pop}_\gamma(w) = w_1$ if $w = w_1\gamma$ and w otherwise. In a pushdown system the function pop_γ will be enabled only when the store satisfies top_γ. The function push_γ is enabled when the store satisfies any.

Effective data structures. A pointed data structure $\mathsf{D} = (D, \widetilde{op}, \widetilde{pred}, \overline{d})$ is said to be *effective* if there is a finite binary representation of every d in D; for every $g \in \widetilde{op}$, there is an algorithm APPLY_g which given the binary representation of $d \in D$ returns the binary representation of $g(d)$; and for every $p \in \widetilde{pred}$, there is an algorithm CHECK_p which given the binary representation of $d \in D$ returns true if $p(d)$ holds and returns false otherwise. For the rest of paper, we shall assume that our data structures are effective. We shall now formally define counter automata with auxiliary stores.

3. Counter automata with stores

Counter automata with stores are automata which have counter w.q.o.'s as the set of states and have an auxiliary store. There are two kinds of transitions: transitions that *increment* counters, and transitions that *decrement* counters. We require that decrement transitions only happen when the data stored is the initial value $\overline{d}$.

[1] In fact, given a w.q.o. $(\mathsf{Q}, \preceq)$, any *monotonic* function $\mathsf{rank} : \mathsf{Q} \to \mathbb{N}$, can be extended to the set of upward closed sub-sets of Q in a similar fashion even if the underlying order is not a partial-order [4].

Definition. Given a pointed data structure $\mathsf{D} = (D, \widetilde{op}, \widetilde{pred}, \overline{d})$, a *counter* D*-automaton with store (CAS) and l counters* is a 5-tuple $\mathbf{C} = (Q, \Lambda, \delta_{\mathsf{inc}}, \delta_{\mathsf{dec}}, \overline{q})$ such that

(1) Q is a *finite* set of *control states* and $\overline{q} \in Q$ is an *initial control state*.
(2) Λ is a *finite* set of *labels*.
(3) $\delta_{\mathsf{inc}} \subseteq Q \times \Lambda \times \widetilde{pred} \times \widetilde{op} \times Q \times 2^{\{0,\ldots,l-1\}}$ is the set of *increment transitions*.
(4) $\delta_{\mathsf{dec}} \subseteq Q \times \Lambda \times Q \times (2^{\{0,\ldots,l-1\}} \setminus \emptyset)$ is the set of *decrement transitions*.

The set $Q \times \mathbb{N}^l$ is said to be the set of *states* of $\mathbf{C}$. We shall say that $\mathbf{C}$ is a *finite* D*-automaton* if the number of counters is 0 (i.e., $l = 0$).

The semantics of a CAS is described in terms of a transition system with the set of configurations as $\{(c, d) \mid c \in Q \times \mathbb{N}^l \text{ and } d \in \mathsf{D}\}$. The configuration $((\overline{q}, 0, \ldots, 0), \overline{d})$ is the initial configuration. Formally, the transition relation $(c, d) \xrightarrow{\lambda}_{\mathbf{C}} (c', d')$ is the union of two relations $\xrightarrow{\lambda}_{\mathbf{C},\mathsf{inc}}$ and $\xrightarrow{\lambda}_{\mathbf{C},\mathsf{dec}}$.

- The increment transitions, $\xrightarrow{\lambda}_{\mathbf{C},\mathsf{inc}}$, which increment counters, is formally defined as follows: $((q, n_1, \ldots n_l), d) \xrightarrow{\lambda}_{\mathbf{C},\mathsf{inc}} ((q', n'_1, \ldots n'_l), d')$ if there is a $(q, \lambda, p, g, q', \mathsf{B}) \in \delta_{\mathsf{inc}}$ such that $p(d)$ is true, $g(d) = d'$, $n'_i = n_i + 1$ for all $i \in \mathsf{B}$ and $n'_i = n_i$ for all $i \notin \mathsf{B}$.
- The decrement transitions, $\xrightarrow{\lambda}_{\mathbf{C},\mathsf{dec}}$, decrement counters and are enabled only when the data value is $\overline{d}$. Formally, $((q, n_1, \ldots n_l), d) \xrightarrow{\lambda}_{\mathbf{C},\mathsf{dec}} ((q', n'_1, \ldots n'_l), d')$ if $d = d' = \overline{d}$ and there is a $(q, \lambda, q', \mathsf{B}) \in \delta_{\mathsf{dec}}$ such that $n_i > 0$ for all $i \in \mathsf{B}$, $n'_i = n_i - 1$ for all $i \in \mathsf{B}$ and $n'_i = n_i$ for all $i \notin \mathsf{B}$.

Notation. For a word $w = \lambda_1\lambda_2 \cdots \lambda_n \in \Lambda^*$, we say $(c, d) \xrightarrow{w}_{\mathbf{C},\mathsf{inc}} (c', d')$ iff there are configurations $(c, d) = (c_0, d_0), (c_1, d_1), \ldots, (c_n, d_n) = (c', d')$ such that $(c_i, d_i) \xrightarrow{\lambda_{i+1}}_{\mathbf{C},\mathsf{inc}} (c_{i+1}, d_{i+1})$ for each i. For the empty word ϵ, we say that $(c, d) \xrightarrow{\epsilon}_{\mathbf{C},\mathsf{inc}} (c', d')$ iff $c = c'$ and $d = d'$. We say $(c, d) \rightarrow^*_{\mathbf{C},\mathsf{inc}} (c', d')$ if there is $w \in \Lambda^*$ such that $(c, d) \xrightarrow{w}_{\mathbf{C},\mathsf{inc}} (c', d')$. Similarly, $(c, d) \xrightarrow{w}_{\mathbf{C},\mathsf{dec}} (c', d')$ and $(c, d) \rightarrow^*_{\mathbf{C},\mathsf{dec}} (c', d')$, can be defined as expected.

Next, given an upward-closed set $U \subseteq Q \times \mathbb{N}^l$, we say that $(c, d) \xrightarrow{\lambda}_{\mathbf{C},\mathsf{inc},U} (c', d')$ iff $(c, d) \xrightarrow{\lambda}_{\mathbf{C},\mathsf{inc}} (c', d')$ and $c, c' \in U$. We can similarly define $(c, d) \xrightarrow{w}_{\mathbf{C},\mathsf{inc},U} (c', d')$ and $(c, d) \rightarrow^*_{\mathbf{C},\mathsf{inc},U} (c', d')$ (all the states in the intermediate steps must belong to U). The relations $(c, d) \xrightarrow{w}_{\mathbf{C},\mathsf{dec},U} (c', d')$, $(c, d) \rightarrow^*_{\mathbf{C},\mathsf{dec},U} (c', d')$, $(c, d) \xrightarrow{w}_{\mathbf{C},U} (c', d')$ and $(c, d) \rightarrow^*_{\mathbf{C},U} (c', d')$ are similarly defined.

The next proposition shows that the transition relation of the CAS is *upward compatible*. The proof of this proposition follows immediately from the definition.

Proposition 3.1. *The relation* $\rightarrow_{\mathbf{C}}$ *is upward compatible, that is,* $(c, d) \xrightarrow{\lambda}_{\mathbf{C}} (c', d')$ *implies that for any* $c_1 \geq c$, *there is a* $c'_1 \geq c'$ *such that* $(c_1, d) \xrightarrow{\lambda}_{\mathbf{C}} (c'_1, d')$.

We say that there is a *non-terminating computation* of $\mathbf{C}$ from a configuration (c, d) if there is an infinite sequence of configurations $(c_0, d_0), (c_1, d_1), \ldots$ and an infinite sequence of labels $\lambda_i \in \Lambda$ such that $(c_0, d_0) = (c, d)$ and $(c_i, d_i) \xrightarrow{\lambda_i}_{\mathbf{C}} (c_{i+1}, d_{i+1})$ for all i. We shall often write $(c, d) \uparrow_{\mathbf{C}}$ if $\mathbf{C}$ has a non-terminating computation from (c, d). We can define $(c, d) \uparrow_{\mathbf{C},U}$, $(c, d) \uparrow_{\mathbf{C},\mathsf{inc}}$, $(c, d) \uparrow_{\mathbf{C},\mathsf{inc},U}$, $(c, d) \uparrow_{\mathbf{C},\mathsf{dec}}$ and $(c, d) \uparrow_{\mathbf{C},\mathsf{dec},U}$ similarly.

Please note that we shall often drop the subscript $\mathbf{C}$ whenever it is clear from the context. Also note that there is no zero-test on the counters (zero-test is not an upward compatible). However, the decrement transitions do have an implicit non-zero test. The CAS's are instances of w.q.o. automata discussed in [4]. Before we proceed to the decidability problems, we first consider an under-approximation of a CAS which will be useful in the decision algorithms.

3.1. A finite approximation

We shall now give an under-approximation of CAS which captures the increment transitions in a finite D-automaton. This under-approximation which essentially cuts-off the counter values at a given k is a modification of the rank k under-approximation defined in [4].

Definition. Let $\mathsf{D} = (D, \widetilde{op}, \widetilde{pred}, \overline{d})$ be a pointed data structure and let $\mathbf{C} = (Q, \Lambda, \delta_{\mathsf{inc}}, \delta_{\mathsf{dec}}, \overline{q})$ be a counter D-automaton with l counters. The *k-bounded semi-approximation* is the finite D-automaton $\mathbf{C}_{\leq k} = (C_{\leq k}, \Lambda, \delta_k, \emptyset, \overline{c})$ where $C_{\leq k} = Q \times [0, k]^l$, $\overline{c} = (\overline{q}, 0, \ldots, 0)$ and δ_k is defined as follows:

- Given $c = (q, n_1, n_2, \ldots, n_l) \in C_{\leq k}$ and $c' = (q', n'_1, n'_2, \ldots, n'_l) \in C_{\leq k}$, we say $(c, \lambda, p, g, c', \emptyset) \in \delta_k$ iff there is a $\mathsf{B} \subseteq \{0, 1, \ldots, l-1\}$ such that
  (1) $(q, \lambda, p, g, q', \mathsf{B}) \in \delta_{\mathsf{inc}}$, and
  (2) $n'_i = \min(n_i + 1, k)$ for all $i \in \mathsf{B}$ and $n'_i = n_i$ all $i \notin \mathsf{B}$.

The following proposition shows how increment transitions are captured faithfully in k-bounded semi-approximations (please recall the function pr_k defined in Section 2 cuts off the counter values at k).

Proposition 3.2. *Let $\mathbf{C} = (Q, \Lambda, \delta_{\mathsf{inc}}, \delta_{\mathsf{dec}}, \overline{q})$ be a CAS with l counters on data structure D and let $\mathbf{C}_{\leq k}$ be the k-bounded semi-approximation of $\mathbf{C}$. For every $c, c' \in Q \times \mathbb{N}^l$, if $(c, d) \xrightarrow{\lambda}_{\mathbf{C},\mathsf{inc}} (c', d')$ then $(\mathsf{pr}_k(c), d) \xrightarrow{\lambda}_{\mathbf{C}_{\leq k}} (\mathsf{pr}_k(c'), d')$. Furthermore if $(\mathsf{pr}_k(c), d) \xrightarrow{\lambda}_{\mathbf{C}_{\leq k}} (c_1, d')$ for some $c_1 \in Q \times [0, k]^l$ then there is some $c' \in Q \times \mathbb{N}^l$ such that $c_1 = \mathsf{pr}_k(c')$ and $(c, d) \xrightarrow{\lambda}_{\mathbf{C},\mathsf{inc}} (c', d')$.*

The above result can be bootstrapped to (finite and infinite) computations as follows.

Lemma 3.3. *Let $\mathbf{C} = (Q, \Lambda, \delta_{\mathsf{inc}}, \delta_{\mathsf{dec}}, \overline{q})$ be a CAS with l counters on data structure D and let $U \subseteq Q \times \mathbb{N}^l$ be an upward closed set. Let $k = \mathsf{rank}(U)$ and let $\mathbf{C}_{\leq k}$ be the k-bounded semi-approximation of $\mathbf{C}$. Let $U_{\leq k} = U \cap (Q \times [0, k]^l)$.*

Given $c, c' \in U$, data values d, d' and word $w \in \Lambda^*$, if $(c, d) \xrightarrow{w}_{\mathbf{C},\mathsf{inc},U} (c', d')$ then $(\mathsf{pr}_k(c), d) \xrightarrow{w}_{\mathbf{C}_{\leq k}, U_{\leq k}} (\mathsf{pr}_k(c'), d')$. Furthermore, if $(\mathsf{pr}_k(c), d) \xrightarrow{w}_{\mathbf{C}_{\leq k}, U_{\leq k}} (c_1, d')$ for some $c_1 \in U_k$ then $(c, d) \xrightarrow{w}_{\mathbf{C},\mathsf{inc},U} (c_1', d')$ for some $c_1' \geq c_1$.

Thus, $(c, d)\uparrow_{\mathbf{C},\mathsf{inc},U}$ iff $(\mathsf{pr}_k(c), d) \uparrow_{\mathbf{C}_{\leq k}, U_{\leq k}}$.

Proof. Please note that the observation on infinite computations follow directly from the result on finite computations and Koning's Lemma. The result on finite words follows directly from Proposition 3.2 by induction and the observation that for any $c_2 \in U$, it must the case that $\mathsf{pr}_k(c_2) \in U_{\leq k}$ (see Proposition 2.2). □

We get as a corollary that for checking termination with increment operations we can simply ignore counters and consider just the underlying automata.

Corollary 3.4. *Let $\mathbf{C}$ be a CAS on D with l counters. Let $\mathbf{C}_{\leq 0}$ be the 0-bounded semi-approximation $\mathbf{C}$. Then, given a state c and a data value d, $(c, d) \uparrow_{\mathbf{C},\mathsf{inc}}$ iff $(\mathsf{pr}_0(c), d) \uparrow_{\mathbf{C}_{\leq 0}}$.*

We are now ready to discuss the question of deciding termination and control state maintainability.

4. Termination and control state maintainability

We shall now show that termination and control state maintainability for CAS is decidable if the termination for finite D-automaton is decidable. The decision procedure for these problems follow ideas similar to the ones behind the algorithms for WSTS [7,3]. For the rest of the section we shall assume a fixed pointed-data structure $\mathsf{D} = (D, \widetilde{op}, \widetilde{pred}, \overline{d})$.

The termination problem for general automata asks if all computations of the automata terminate. For WSTS, the problem of termination is solved by constructing the reachability tree in a breadth-first manner. If all computations terminate, then the reachability tree constructed is finite. Otherwise, by properties of well-quasi-orders, we shall find a self-covering path, i.e., two reachable states q_1 and q_2 such that q_1 is a parent of q_2 and q_1 is dominated by q_2 in the well-quasi-order. We now adapt the decision algorithm for termination of WSTS for CAS's.

Theorem 4.1 (*Termination*). *Assume that for an effective data structure D, the termination problem is decidable for finite D-automaton. Then the termination problem is decidable for CAS's on D.*

Proof. Let $\mathbf{C} = (Q, \Lambda, \delta_{\mathsf{inc}}, \delta_{\mathsf{dec}}, \overline{q})$ be a CAS. Let the initial data value be $\overline{d}$ and consider the (possibly infinite) tree, $\mathcal{T}$, of reachable configurations where each node is labeled by a configuration of $\mathbf{C}$. The root is labeled by $((\overline{q}, 0, \ldots, 0), \overline{d})$; and for every node labeled by a configuration (c, d) it must be the case that the set of labels of its children is the set $\{(c', d') \mid \exists \lambda \in \Lambda \text{ s.t. } (c, d) \xrightarrow{\lambda} (c', d')\}$. Using Koning's lemma, it is easy to see that $\mathbf{C}$ has a non-terminating computation (*i.e.*, $\mathcal{T}$ has an infinite path) iff $\mathcal{T}$ is infinite.

If $\mathcal{T}$ has an infinite path $n_0, n_1 \ldots$, then one of the following must happen.

(1) For an infinite number of nodes along the infinite path, the data value in the labeling configuration is the initial data value $\overline{d}$. In other words, there is an infinite sub-sequence of nodes $n_{i_1} n_{i_2} \ldots$ with $i_j < i_{j+1} \forall j \geq 0$ such that if (c_{i_j}, d_{i_j}) labels n_{i_j} then $d_{i_j} = \overline{d}$. Now since $Q \times \mathbb{N}^l$ is a w.q.o., there must be some $i_k < i_{k'}$ such that $c_{i_k} \leq c_{i_{k'}}$. Also, please note that by upward-compatibility of transition relation if there is a path in $\mathcal{T}$ such that there are two nodes n and n' labeled by $(c, \overline{d})$ and $(c', \overline{d})$ respectively such that n is an ancestor of n' and $c \leq c'$ then $\mathcal{T}$ must necessarily have an infinite path.

(2) There is a node n_i on this path labeled by (c_i, d_i) such that $d_i = \overline{d}$ and for all $j > i$, if n_j is labeled by (c_j, d_j) then $d_j \neq \overline{d}$.

Then by definition, $\forall j \geq i$ the transition $(c_j, d_j) \xrightarrow{\lambda_i} (c_{j+1}, d_{j+1})$ must use increment transitions. Also please note that if there a reachable configuration $(c, \overline{d})$ such that there is an infinite path from $(c, \overline{d})$ that just uses increment transitions then $\mathcal{T}$ must necessarily have an infinite path.

The above observations suggest the following algorithm to solve the termination problem. The algorithm starts by constructing the tree $\mathcal{T}$ in a breadth-first manner. Whenever we add a node n labeled by $(c, \overline{d})$ we check-

(1) if there is an ancestor of the node n labeled by $(c', \overline{d})$ such that $c' \leq c$; or
(2) if there is an infinite path from $(c, \overline{d})$ that uses only increment transitions. Please note that by Corollary 3.4, this is reducible to checking termination in a finite D-automaton.

If either of the above two conditions hold, then the algorithm returns that there is a non-terminating computation of $\mathbf{C}$.

If there is no such node, the reachability tree $\mathcal{T}$ must be finite and the algorithm returns that all computations of $\mathbf{C}$ must terminate once we cannot extend $\mathcal{T}$. □

A maximal computation of a CAS is either an infinite computation or a finite computation that cannot be extended. Given, an upward closed set of states $\mathsf{U} \subseteq \mathsf{Q} \times \mathbb{N}^l$ (in terms of its minimal elements), the control state maintainability problem for a CAS $\mathbf{C}$ asks if there is a maximal computation $(c_0, \overline{d}) \xrightarrow{\lambda_0} (c_1, d_1) \xrightarrow{\lambda_1} (c_2, d_2) \ldots$ such that $c_0 = (\overline{q}, 0, \ldots, 0)$ and $c_i \in U$ for all i. The algorithm for termination can be modified to give the following result.

Theorem 4.2 (*Control State Maintainability*). Assume that for an effective data structure D, the termination problem is decidable for finite D-automaton. Then the control state maintainability problem is decidable for CAS's on D.

Proof. Let $\mathbf{C}$ be a CAS and $\mathsf{U} \subseteq \mathsf{Q} \times \mathbb{N}^l$ be an upward closed set of states of $\mathbf{C}$. As in the proof of Theorem 4.1, consider the infinite reachability tree $\mathcal{T}$ with nodes labeled by configurations and the parent-child relation reflecting the transition relation of $\mathbf{C}$. As in the proof of Theorem 4.1, we can show that there is a maximal computation of $\mathbf{C}$ maintainable in U iff one of the following holds.

(1) There is a maximal finite path such that if (c, d) labels a node along this path then $c \in \mathsf{U}$.
(2) There are two reachable nodes n and n' labeled by $(c, \overline{d})$ and $(c', \overline{d})$ respectively such that n is an ancestor of n', $c \leq c'$, and if (c'', d'') labels a node from the root to n' then $c'' \in \mathsf{U}$.
(3) There is a reachable node n labeled by $(c, \overline{d})$ such that if (c', d') labels a node from the root to n, then $c' \in \mathsf{U}$; and $(c, \overline{d}) \uparrow_{\mathbf{C},\mathsf{inc},\mathsf{U}}$, which means that there is an infinite path starting from $(c, \overline{d})$ using only increment transitions and always remaining in U. Now by Lemma 3.3, please note that if $k = \mathsf{rank}(\mathsf{U})$, $\mathbf{C}_{\leq k} = (C_{\leq k}, \Lambda, \delta_k, \emptyset, \overline{c})$ is the k-bounded semi-approximation of $\mathbf{C}$ and $\mathsf{U}_{\leq k} = \mathsf{U} \cap C_{\leq k}$ then $(c, \overline{d}) \uparrow_{\mathbf{C},\mathsf{inc},\mathsf{U}}$ iff $(\mathsf{pr}_k(c), \overline{d}) \uparrow_{\mathbf{C}_{\leq k},\mathsf{U}_{\leq k}}$. By restricting $\mathbf{C}_{\leq k}$ to states $\mathsf{U}_{\leq k}$, the question whether $(\mathsf{pr}_k(c), \overline{d}) \uparrow_{\mathbf{C}_{\leq k},\mathsf{U}_{\leq k}}$ is reducible to a question of termination of the restricted automaton.

Now the desired algorithm can be constructed similar to the one constructed in the proof of Theorem 4.1. We start by constructing the reachability tree. If one of the above three conditions hold at any point then the algorithm returns that $\mathcal{T}$ has a path that is maintainable in U. Otherwise, if U is not maintainable, then every computation would have a node labeled by some configuration (c', d) with $c' \notin \mathsf{U}$ and we do not need to extend the tree from that node. We can call such nodes unsuccessful nodes. The algorithm returns that U is not maintainable if all computations lead to unsuccessful nodes. □

5. Simulation by finite transition system

A CAS $\mathbf{C}$ is said to be simulated by a finite transition system $\mathbf{S}$ if the transition system defined by $\mathbf{C}$ is simulated by $\mathbf{S}$. The algorithm for deciding whether a WSTS is simulated by a $\mathbf{S}$ (which we shall adapt here) depends on the fact that the non-simulation can be computed as a fix-point.

Proposition 5.1. *Let $\mathbf{S}_1 = (S_1, \Lambda, \rightarrow_1, s_{i_1})$ and $\mathbf{S}_2 = (S_2, \Lambda, \rightarrow_2, s_{i_2})$ be two transition systems. For each $s_2 \in S_2$, consider the (increasing) sequence $I_{s_2,0}, I_{s_2,1}, \ldots$ of sub-sets of S_1 constructed as follows:*

- $I_{s_2,0} = \emptyset$.
- $I_{s_2,j+1} = I_{s_2,j} \cup \{s_1 \in S_1 \mid \exists \lambda \in \Lambda, s_1' \in S_1 \text{ s.t. } s_1 \xrightarrow{\lambda}_1 s_1' \text{ and } \forall s_2' \in S_2.((s_2 \xrightarrow{\lambda}_2 s_2') \Rightarrow (s_1' \in I_{s_2',j}))\}$.

For $s_1 \in S_1$ and $s_2 \in S_2$, $s_1 \not\sqsubseteq^{\mathbf{S}_1 \times \mathbf{S}_2} s_2$ iff there exists $j \geq 0$ such that $s_1 \in I_{s_2,j}$.

Intuitively, the fix-point characterization can be thought of as follows. Initially, we do not know if $s_1 \in S_1$ is simulated by $s_2 \in S_2$ or not. At the first step we check if there is some label λ such that s_1 has a transition labeled by λ, but s_2 has no such transition. If this is the case, we know that s_1 is not simulated by s_2. At the end of step j, we know that for each pair (s_1', s_2') in the set $F^j = \{(s_1', s_2') \mid s_2' \in S_2 \text{ and } s_1' \in I_{s_2',j}\}$, it is the case that s_1' is not simulated by s_2'. At step $j+1$, we check if s_1 has a transition $s_1 \xrightarrow{\lambda}_1 s_1'$ such that whenever s_2 tries to match the transition by a transition $s_2 \xrightarrow{\lambda}_2 s_2'$, it is the case that $(s_1', s_2') \in F^j$. If this is the case we can conclude that s_1 is not simulated by s_2. This is because even if s_2 can match $s_1 \xrightarrow{\lambda}_1 s_1'$, it cannot match subsequent transitions. We can think of F^j as an "assisting set" which helps to prove that s_1 is not simulated by s_2 by just considering 1-step transitions out of s_1.

The algorithm for checking whether a WSTS is simulated by a finite transition system exploits the above fixed point characterization as follows: if S_1 is a WSTS then $I_{s_2,j}$ is upward-closed for each $s_2 \in S_2, j \geq 0$. By properties of well-quasi-orders and the fact S_2 is a finite, there is a j_1 such that $I_{s_2,j} = I_{s_2,j_1} \forall j \geq j_1, s_2 \in S$. The algorithm computes this j_1 by a backward search.

We shall adopt a similar approach for deciding whether a CAS is simulated by a finite transition system. However, we first extend the definition of simulations for our purposes.

Definition. Given transition systems $\mathbf{S}_1 = (S_1, \Lambda, \rightarrow_1, s_{i_1}), \mathbf{S}_2 = (S_2, \Lambda, \rightarrow_2, s_{i_2})$ and a set $\mathsf{F} \subseteq S_1 \times S_2$, we say that $R \subseteq S_1 \times S_2$ is *a simulation relation with forbidden* F if R is a simulation relation and $R \cap \mathsf{F} = \emptyset$. We say that $s_1 \sqsubseteq_{\mathsf{F}}^{\mathbf{S}_1 \times \mathbf{S}_2} s_2$ if there is a simulation relation R with forbidden F such that $(s_1, s_2) \in R$.

Although the definition of the simulation with forbidden F may seem contrived, we argue that it is quite natural. First note that the definition of simulation coincides exactly with the definition of simulation with forbidden F where the forbidden set F is the empty set. Intuitively, a (possibly non-empty) forbidden F can be thought of as an "assisting set" which gives a collection of pairs (s'_1, s'_2) such that s'_1 is not simulated by s'_2. Indeed if F is a set such that s'_1 is not simulated by s'_2 for each $(s'_1, s'_2) \in \mathsf{F}$ then $s_1 \sqsubseteq^{\mathbf{S}_1 \times \mathbf{S}_2} s_2$ iff $s_1 \sqsubseteq_{\mathsf{F}}^{\mathbf{S}_1 \times \mathbf{S}_2} s_2$. Indeed for such an "assisting set" we could have started the backward induction in Proposition 5.1 by taking $I_{s'_2,0} = \{s'_1 \mid (s'_1, s'_2) \in \mathsf{F}\}$ for each $s'_2 \in S_2$.

The following are easy consequences of the definition of simulation with forbidden sets.

Proposition 5.2. *Given transition systems* $\mathbf{S}_1 = (S_1, \Lambda, \rightarrow_1, s_{i_1})$, $\mathbf{S}_2 = (S_2, \Lambda, \rightarrow_2, s_{i_2})$ *and sets* $\mathsf{F}, \mathsf{F}_0 \subseteq S_1 \times S_2$.

- *If* $s_1 \not\sqsubseteq_{\mathsf{F}}^{\mathbf{S}_1 \times \mathbf{S}_2} s_2$ *and* $\mathsf{F} \subseteq \mathsf{F}_0$ *then* $s_1 \not\sqsubseteq_{\mathsf{F}_0}^{\mathbf{S}_1 \times \mathbf{S}_2} s_2$ *also.*
- *If* $s_1 \not\sqsubseteq_{\mathsf{F}}^{\mathbf{S}_1 \times \mathbf{S}_2} s_2$ *and* $s \not\sqsubseteq^{\mathbf{S}_1 \times \mathbf{S}_2} s'$ *for all* $(s, s') \in \mathsf{F}$ *then* $s_1 \not\sqsubseteq^{\mathbf{S}_1 \times \mathbf{S}_2} s_2$.

The algorithm for deciding whether a WSTS is simulated by a finite transition system cannot be immediately extended to deciding whether a CAS **C** is simulated by a finite transition system **S**. This is because the set of configurations of **C** may not necessarily form a *wqo* and hence we will not know when to stop the backward induction in Proposition 5.1. However, given a fixed data value (say initial data), the sub-set of configurations with that data value is easily see to be a *wqo*. We shall make use of this important observation in our algorithm to decide if a CAS is simulated by a finite transition system. We shall first describe this algorithm informally. We first fix some notation. For the rest of this section, we shall fix a *finite* transition system $\mathbf{S} = (S, \Lambda, \rightarrow, s_i)$. We shall also fix a CAS $\mathbf{C} = (\mathsf{Q}, \Lambda, \delta_{\mathsf{inc}}, \delta_{\mathsf{dec}}, \overline{q})$ on a data structure $\mathsf{D} = (D, \widetilde{op}, \widetilde{pred}, \overline{d})$ with l counters. Let $\mathsf{Conf} = (Q \times \mathbb{N}^l) \times D$. Let $\overline{c} \in (Q \times \mathbb{N}^l)$ be the element $(\overline{q}, 0, \ldots, 0)$. Let $\mathbf{S}_{\mathbf{C}} = (\mathsf{Conf}, \Lambda, \rightarrow, (\overline{c}, \overline{d}))$ be the transition system generated by **C** and let $\mathbf{S}_{\mathbf{C},\mathsf{inc}} = (\mathsf{Conf}, \Lambda, \rightarrow_{\mathsf{inc}}, (\overline{c}, \overline{d}))$ be the transition system generated by **C** which uses only increment transitions.

Now every computation $(c, \overline{d}) \rightarrow_{\mathbf{C}}^* (c', d)$ of **C** is of the form $(c, \overline{d}) \rightarrow_{\mathbf{C},\mathsf{inc}}^* (c_1, \overline{d}) \rightarrow_{\mathbf{C},\mathsf{dec}} (c_2, \overline{d}) \rightarrow_{\mathbf{C},\mathsf{inc}}^* \cdots \rightarrow_{\mathbf{C},\mathsf{dec}} (c_n, \overline{d}) \rightarrow_{\mathbf{C},\mathsf{inc}}^* (c', d)$. Keeping this observation in mind, our algorithm proceeds informally as follows.

- For each $j \in \mathbb{N}$ and $s \in S$, we compute an increasing sequence $I_{s,j} \subseteq \mathsf{Conf}$ such that if $(c, d) \in I_{s,j}$ then d is the initial data $\overline{d}$ and $(c, d) \not\sqsubseteq^{\mathbf{S}_{\mathbf{C}} \times S} s$. (In the actual algorithm, we will just need to keep track of the set $U_{s,j} = \{c \mid (c, \overline{d}) \in I_{s,j}\}$.)
- Initially $I_{s,0} = \emptyset$ for each $s \in S$.
- For $s \in S$ and even j, let $F^j = \{((c, \overline{d}), s) \mid (c, \overline{d}) \in I_{s,j}\}$. Now, we have that $(c, \overline{d}) \not\sqsubseteq s$ for each $((c, \overline{d}), s) \in F^j$. Indeed, F^j is the set of all pairs $((c, \overline{d}), s)$ such that our algorithm has concluded (within the first j steps) that $(c, \overline{d})$ is not simulated by s. The set $I_{s,j+1}$ will be the set of configurations $(c, \overline{d})$ such that given the "assisting set" F^j, the non-simulation of $(c, \overline{d})$ by s can be derived by considering only the increment transitions of **C**. More precisely, $I_{s,j+1} = I_{s,j} \cup \{(c, \overline{d}) \mid (c, \overline{d}) \not\sqsubseteq_{F^j}^{\mathbf{S}_{\mathbf{C},\mathsf{inc}} \times \mathbf{S}} s\}$.
- For $s \in S$ and odd j, the construction of $I_{s,j+1}$ is very much like in Proposition 5.1 except we only consider decrement transitions. More precisely, $I_{s,j+1} = I_{s,j} \cup \{(c, \overline{d}) \mid \exists \lambda \in \Lambda, (c', \overline{d}) \in \mathsf{Conf} \text{ s.t. } (c, \overline{d}) \xrightarrow{\lambda}_{\mathsf{dec}} (c', \overline{d}) \text{ and } \forall s' \in S.((s \xrightarrow{\lambda} s') \Rightarrow ((c', \overline{d}) \in I_{s',j}))\}$.
- As in the case of WSTS, we can argue that there is a j_1 such that $I_{s,j_1} = I_{s,j_1+1} = I_{s,j_1+2}$ for each $s \in S$ (and thus $I_{s,j_1} = I_{s,j'}$ for each $j' \geq j_1, s \in S$). **C** will be simulated by **S** if $(\overline{c}, \overline{d}) \notin I_{s_i,j_1}$.

We shall now carry out the above algorithm formally and show that the algorithm does indeed solve the problem we set out to solve. For this, as we have seen above, our algorithm needs to check if $\mathbf{S}_{\mathbf{C},\mathsf{inc}}$ is simulated by **S** for some forbidden sets. The forbidden sets used are of certain kind.

Definition. A set $\mathsf{F} \subseteq \mathsf{Conf} \times S$ is said to be a *pointed forbidden set* if $((c, d), s) \in \mathsf{F}$ implies that $d = \overline{d}$. A pointed forbidden set F is said to be *upward-closed* if for each $s \in S$ the set $U_s = \{c \mid ((c, \overline{d}), s) \in \mathsf{F}\}$ is upward-closed. If F is pointed and upward-closed, we let $\mathsf{rank}(\mathsf{F}) = 0$ if $\mathsf{F} = \emptyset$, otherwise $\mathsf{rank}(\mathsf{F}) = \max\{\mathsf{rank}(U_s) \mid s \in S\}$. For $k \geq 0$, we let $\mathsf{F}_k = \{((c, d), s) \mid ((c, d), s) \in \mathsf{F} \text{ and } c \in \mathsf{Q} \times [0, k]^l\}$.

The following Lemma says that in order to check if $\mathbf{S}_{\mathbf{C},\mathsf{inc}}$ is simulated by a finite transition system with pointed and upward-closed forbidden sets, it suffices to consider k-bounded semi-approximations.

Lemma 5.3. *Let* $\mathbf{S}_{\mathbf{C},\mathsf{inc}}$ *be the transition system generated by* **C** *and increment transitions with* Conf *as the set of configurations. Let* **S** *be a finite transition system with S as the set of configurations. Let* $\mathsf{F} \subseteq \mathsf{Conf} \times S$ *be a pointed and upward-closed forbidden set, k be* rank(F), $\mathbf{C}_{\leq k}$ *be the k-bounded semi-approximation of* **C** *and* $\mathbf{S}_{\mathbf{C}_{\leq k}}$ *be the transition system generated by* $\mathbf{C}_{\leq k}$. *For any* $(c, d) \in \mathsf{Conf}$, $(c, d) \sqsubseteq_{\mathsf{F}}^{\mathbf{S}_{\mathbf{C},\mathsf{inc}} \times \mathbf{S}} s$ *iff* $(\mathsf{pr}_k(c), d) \sqsubseteq_{\mathsf{F}_k}^{\mathbf{S}_{\mathbf{C}_{\leq k}} \times \mathbf{S}} s$.

Proof. For each $s \in S$, we define a sequence $I_{s,0}, I_{s,1}, \ldots$ of sub-sets of Conf as follows.

(1) $I_{s,0} = \{(c, \overline{d}) \mid ((c, \overline{d}), s) \in \mathsf{F}\}$.

(2) $I_{s,j+1} = I_{s,j} \cup \{(c, d) \mid \exists \lambda \in \Lambda, (c', d') \in \mathsf{Conf} \text{ s.t. } (c, d) \xrightarrow{\lambda}_{\mathbf{C},\mathsf{inc}} (c', d') \text{ and } \forall s' \in S \text{ we have that } ((s \xrightarrow{\lambda} s') \Rightarrow ((c', d') \in I_{s',j}))\}$.

Please note for each $s \in S$, $d \in D$ and $j \geq 0$ the set $\{c \mid (c,d) \in I_{s,j}\}$ is upward closed. It can be shown that $(c,d) \not\preceq_{\mathsf{F}}^{\mathbf{S}_{\mathbf{C},\mathsf{inc}} \times \mathbf{S}} s$ iff there is a j such that $(c,d) \in I_{s,j}$.

Now, we also define a sequence $I^k_{s,0}, I^k_{s,1}, \ldots$ of sub-sets of $\mathsf{Conf}_{\leq k} = (\mathsf{Q} \times [0,k]^l) \times D$ as follows.

(1) $I^k_{s,0} = \{(c,\overline{d}) \mid c \in \mathsf{Q} \times [0,k]^l, ((c,\overline{d}),s) \in \mathsf{F}_k\}$.

(2) $I^k_{s,j+1} = I^k_{s,j} \cup \{(c,d) \in \mathsf{Conf}_{\leq k} \mid \exists \lambda \in \Lambda, (c',d') \in \mathsf{Conf}_{\leq k}$ s.t. $(c,d) \xrightarrow{\lambda}_{\mathbf{C}_{\leq k}} (c',d')$ and $\forall s' \in S$ we have that $((s \xrightarrow{\lambda} s') \Rightarrow ((c',d') \in I^k_{s',j}))\}$.

It can be shown that for $c_1 \in \mathsf{Conf}_{\leq k}$, $(c_1,d) \not\preceq_{\mathsf{F}_k}^{\mathbf{S}_{\mathbf{C}_{\leq k}} \times \mathbf{S}} s$ iff there is a j such that $(c_1,d) \in I^k_{s,j}$.

The result thus follows if we can show that for each $c \in \mathsf{Q} \times \mathbb{N}^l$, $d \in \mathsf{D}$ and each $j \geq 0$, $(c,d) \in I_{s,j}$ iff $(\mathsf{pr}_k(c),d) \in I^k_{s,j}$. We prove this by induction on j. Please note that for $j = 0$, the claim is true by construction.

Assume that the claim is true for $j \geq 0$. If $(c,d) \in I_{s,j+1} \setminus I_{s,j}$ then there is a transition $(c,d) \xrightarrow{\lambda}_{\mathbf{C},\mathsf{inc}} (c',d')$ such that $\forall s' \in S$ we have that $((s \xrightarrow{\lambda} s') \Rightarrow ((c',d') \in I_{s',j}))\}$. By induction hypothesis, for any $s' \in S$ we have that $(\mathsf{pr}_k(c'),d') \in I^k_{s',j}$ if $(c',d') \in I_{s',j}$. Furthermore, we also have by Proposition 3.2 that $(\mathsf{pr}_k(c),d) \xrightarrow{\lambda}_{\mathbf{C}_{\leq k}} (\mathsf{pr}_k(c'),d')$. Thus, by definition of $I^k_{s,j+1}$, $(\mathsf{pr}_k(c),d) \in I^k_{s,j+1}$.

Similarly, if $(\mathsf{pr}_k(c),d) \in I^k_{s,j+1} \setminus I^k_{s,j}$, we can show that $(c,d) \in I_{s,j+1}$. □

Please observe that the set F_k in Lemma 5.3 is finite. We are ready to show the main result of this section.

Theorem 5.4 (*Simulation by Finite Transition Systems*). *Assume that the simulation of a finite* D*-automaton by a finite transition system with a finite pointed forbidden set is decidable. Then there is an algorithm that given a CAS,* **C***, and a finite state transition* **S** *returns true if* **C** *is simulated by* **S** *and false otherwise.*

Proof. Consider the following increasing sequence (constructed inductively for each $s \in S$) of upward-closed sets $\mathsf{U}_{s,0}, \mathsf{U}_{s,1}, \ldots \subseteq \mathsf{Q} \times \mathbb{N}^l$:

- $\mathsf{U}_{s,0} = \emptyset$.
- When j is odd, $\mathsf{U}_{s,j+1} = \mathsf{U}_{s,j} \cup \{c \in \mathsf{Q} \times \mathbb{N}^l \mid \exists \lambda \in \Lambda, c' \in \mathsf{Q} \times \mathbb{N}^l$ s.t. $(c,\overline{d}) \xrightarrow{\lambda}_{\mathsf{dec}} (c',\overline{d})$ and $\forall s' \in S.((s \xrightarrow{\lambda} s') \Rightarrow (c' \in \mathsf{U}_{s',j}))\}$.
- When j is even, let $\mathsf{F}^j = \{((c_1,\overline{d}),s_1) \in \mathsf{Conf} \times S \mid c_1 \in U_{s_1,j}\}$ and $\mathsf{U}_{s,j+1} = \mathsf{U}_{s,j} \cup \{c \in \mathsf{Q} \times \mathbb{N}^l \mid (c,\overline{d}) \not\preceq_{\mathsf{F}^j}^{\mathbf{S}_{\mathbf{C},\mathsf{inc}} \times \mathbf{S}} s\}$. Note that checking $(c,\overline{d}) \not\preceq_{\mathsf{F}^j}^{\mathbf{S}_{\mathbf{C},\mathsf{inc}} \times \mathbf{S}} s$ involves the transition system $\mathbf{S}_{\mathbf{C},\mathsf{inc}}$ and not $\mathbf{S}_{\mathbf{C}}$.

We have the following claim.

Claim: For each $c \in \mathsf{Q} \times \mathbb{N}^l$ and $s \in S$, $(c,\overline{d}) \not\preceq^{\mathbf{S}_{\mathbf{C}} \times \mathbf{S}} s$ iff there is a $j \geq 0$ such that $c \in \mathsf{U}_{s,j}$.

We postpone the proof of the claim. We show first how the claim allows us to get a decision procedure for the simulation. Please note that since S has only a finite number of elements and that each $\mathsf{U}_{s,j}$ is an upward closed set, there must be a j_1 such that $\mathsf{U}_{s,j_1} = \mathsf{U}_{s,j} \forall s \in S$ and $j \geq j_1$. Thus, if we can compute this j_1 and minor sets for $\mathsf{U}_{s,j}$ for $s \in S$ and $j \in \mathbb{N}$ we can decide the simulation problem.

Now, please note we can compute the minor sets for $\mathsf{U}_{s,j}$ by backward induction. When j is odd, we can compute the minor set for $\mathsf{U}_{s,j+1}$ by inspection of the decrement edges. When j is even, we can compute the minor set for $\mathsf{U}_{s,j}$ by taking recourse to Lemma 5.3. We stop when $\mathsf{U}_{s,j} = \mathsf{U}_{s,j+1} = \mathsf{U}_{s,j+2}$ for each s. This j is the required j_1. Now, we prove the above claim.

Proof the claim:

($\Rightarrow$) We show by induction on j that if $c \in \mathsf{U}_{s,j}$ then $(c,\overline{d}) \not\preceq^{\mathbf{S}_{\mathbf{C}} \times \mathbf{S}} s$. Please note that the claim is true for $j = 0$. Assume that the claim is true for any $j \leq k$. We now consider $j = k+1$. There are two cases:

(1) k is even. Pick $c \in \mathsf{U}_{s,k+1}$. By definition, $(c,\overline{d}) \not\preceq_{\mathsf{F}^k}^{\mathbf{S}_{\mathbf{C},\mathsf{inc}} \times \mathbf{S}} s$. Observe that the transition system $\mathbf{S}_{\mathbf{C},\mathsf{inc}}$ has the same set of configuration as $\mathbf{S}_{\mathbf{C}}$ but has fewer transitions. Thus, $(c,\overline{d}) \not\preceq_{\mathsf{F}^k}^{\mathbf{S}_{\mathbf{C}} \times \mathbf{S}} s$. Also, by induction hypothesis $(c_0,d_0) \not\preceq^{\mathbf{S}_{\mathbf{C}} \times \mathbf{S}} s_0$ for all $((c_0,d_0),s_0) \in \mathsf{F}^k$. Hence, we get by Proposition 5.2 that $(c,\overline{d}) \not\preceq^{\mathbf{S}_{\mathbf{C}} \times \mathbf{S}} s$.

(2) k is odd. This case is straightforward. If $c \in \mathsf{U}_{s,k+1}$ then either it is already in $\mathsf{U}_{s,k}$ and the result follows from induction hypothesis or there is a transition $(c,\overline{d}) \xrightarrow{\lambda} (c',\overline{d})$ which cannot be matched by s.

($\Leftarrow$) We need to show that if $(c_0,\overline{d}) \not\preceq^{\mathbf{S}_{\mathbf{C}} \times \mathbf{S}} s_0$ then there is a $j \geq 0$ such that $c_0 \in \mathsf{U}_{s_0,j}$. Given $s \in S$, consider the following increasing sequence of sets $\mathsf{M}_{s,k} \subseteq \mathsf{Conf}$:

- $\mathsf{M}_{s,0} = \emptyset$.
- When j is odd, $\mathsf{M}_{s,j+1} = \mathsf{M}_{s,j} \cup (\mathsf{U}_{s,j+1} \times \{\overline{d}\})$.
- When j is even, let $\mathsf{M}_{s,j+1} = \mathsf{M}_{s,j} \cup \{(c,d) \in \mathsf{Conf} \mid (c,d) \not\preceq_{\mathsf{F}^j}^{\mathbf{S}_{\mathbf{C},\mathsf{inc}} \times \mathbf{S}} s\}$.

Please note that it is easy to see by construction that for all $j \in \mathbb{N}$, $c \in Q \times \mathbb{N}^l$ and $s \in S, c \in U_{s,j}$ iff $(c, \overline{d}) \in M_{s,j}$. Now, consider the relation $R = \{((c, d), s) \in \mathsf{Conf} \times S \mid (c, d) \notin \cup_{j=0}^{\infty} M_{s,j}\}$. We claim that the result will follow if we can demonstrate that R is a simulation relation. Indeed, if R is a simulation relation then as $(c_0, \overline{d}) \not\preceq^{\mathbf{S_C} \times \mathbf{S}} s_0$, we get that $((c_0, \overline{d}), s_0) \notin R$. By definition of R, this implies that $(c_0, \overline{d}) \in \cup_{j=0}^{\infty} M_{s_0,j}$. Thus, there is a j_0 such that $(c_0, \overline{d}) \in M_{s_0,j_0}$ and hence $c_0 \in U_{s_0,j_0}$. Therefore, the result follows if we can show that R is a simulation relation.

In order to show that R is a simulation relation, we need to show that whenever $((c, d), s) \in R$ and $(c, d) \xrightarrow{\lambda} (c', d')$ then there is a s' such that $((c', d'), s') \in R$ and $s \xrightarrow{\lambda} s'$. There are two cases.

(1) The transition $(c, d) \xrightarrow{\lambda} (c', d')$ is a decrement transition. In this case $d = d' = \overline{d}$. Let $S_1 = \{s' \mid s \xrightarrow{\lambda} s'\}$. Please note that if $S_1 = \emptyset$ then $(c, \overline{d}) \in M_{s,2}$ and hence $((c, \overline{d}), s) \notin R$. Hence, $S_1 \neq \emptyset$. Assume now, by way of contradiction, that for each $s' \in S_1$ it is the case that $((c', \overline{d}), s') \notin R$. Hence for each $s' \in S_1$, there is a unique $j_{s'} > 0$ such that $c' \in U_{s',j_{s'}} \setminus U_{s',j_{s'}-1}$. Now, S_1 is finite and thus there is an odd j_0 such that $j_0 \geq j_{s'}$ for all $s' \in S_1$. Now, we get $c' \in U_{s',j_0}$ for all $s' \in S_1$. Hence, by construction $c \in U_{s,j_0+1}$ which implies that $((c, \overline{d}), s) \notin R$. A contradiction.

(2) The transition $(c, d) \xrightarrow{\lambda} (c', d')$ is an increment transition. Again, let $S_1 = \{s' \mid s \xrightarrow{\lambda} s'\}$. Please note that $S_1 \neq \emptyset$ (otherwise $(c, d) \in M_{s,1}$). Assume now, by way of contradiction, that for each $s' \in S_1$ it is the case that $((c', d'), s') \notin R$. Hence for each $s' \in S_1$, there is a unique $j_{s'} > 0$ such that $(c', d') \in M_{s',j_{s'}} \setminus M_{s',j_{s'}-1}$. Pick j_0 such that j_0 is an even number $\geq j_{s'}$ for all $s' \in S_1$. Now there are two cases depending on d'.

The first case is that d' is not the initial data value $\overline{d}$. We have by definition, that $(c', d') \not\preceq^{\mathbf{S_{C,inc}} \times \mathbf{S}}_{\mathsf{F}^{j_{s'}-1}} s'$ for all $s' \in S_1$. Please note that since F^k is an increasing sequence, we have by Proposition 5.2 that $(c', d') \not\preceq^{\mathbf{S_{C,inc}} \times \mathbf{S}}_{\mathsf{F}^{j_0}} s'$ for all $s' \in S_1$. It is easy to see that this implies that $(c, d) \not\preceq^{\mathbf{S_{C,inc}} \times \mathbf{S}}_{\mathsf{F}^{j_0}} s$ and hence $((c, d), s) \in M_{s,j_0+1}$. Thus, $((c, d), s) \notin R$. A contradiction.

If d' is the initial data value $\overline{d}$ then $((c', d'), s') \in \mathsf{F}^{j_0}$ for all $s' \in S_1$ and we again get by definition $(c, d) \in M_{s,j_0+1}$. Hence $((c, d), s) \notin R$ which contradicts $((c, d), s) \in R$. □

We observe here that for a finite D-automaton $\mathbf{C}$, finite pointed F and finite transition system $\mathbf{S}$ the question of deciding whether $\mathbf{C} \sqsubseteq_{\mathsf{F}} \mathbf{S}$ can be restated as a reachability game and hence is decidable if D is pushdown store or a higher-order store.

6. Simulation of finite transition systems

A CAS $\mathbf{C}$ is said to simulate a finite transition system $\mathbf{S}$ if $\mathbf{S}$ is simulated by the transition system defined by the CAS $\mathbf{C}$. The key ingredient in the proofs of decidability for WSTS's of termination, control state maintainability and simulation by finite state automaton is the existence of a trace with special properties. This allows the algorithms for WSTS's to be extended to CAS's. Each trace of a CAS is of special form in which increment and decrement transitions alternate and the decrement only happens at the initial data value. This property essentially allows one to combine the counter w.q.o. properties and adapt the algorithms for WSTSs. The algorithm for whether a WSTS simulates a finite transition system [3] is, however, based on the construction of an and-or reachability tree; and essentially exploits the properties of the whole tree rather than just traces. For this reason, we were not able to extend the algorithm for WSTS to CAS.

We were, however, able to come up with an algorithm for deciding whether a finite transition system is simulated by a CAS by using new under-approximations. These under-approximations capture both the increment *and* decrement transitions in a finite D-automaton. The states of the under-approximation are obtained by bounding the counter values. Since they capture the decrement transitions also, they are different from the under-approximations discussed in Section 3.1. Formally,

Definition. Let $\mathsf{D} = (D, \widetilde{op}, \widetilde{pred}, \overline{d})$ be a pointed data structure and let $\mathbf{C} = (Q, \Lambda, \delta_{\mathsf{inc}}, \delta_{\mathsf{dec}}, \overline{q})$ be a CAS with l counters. The *k-bounded approximation* is a finite D-automaton $\mathbf{C}^f_{\leq k} = (C_{\leq k}, \Lambda, \delta_k, \emptyset, \overline{c})$ where $C_{\leq k} = Q \times [0, k]^l$, $\overline{c} = (\overline{q}, 0, \ldots, 0)$ and δ_k is defined as follows.

- For each $(q, \lambda, q', \mathsf{B}) \in \delta_{\mathsf{dec}}$, $(c, \lambda, \overline{p}, \mathsf{id}, c', \emptyset) \in \delta_k$ if $c = (q, n_1, \ldots, n_l)$, $c' = (q', n'_1, \ldots n'_l)$ for some $n_1, \ldots, n_l$ and $n'_1, \ldots, n'_l$ such that $n'_i = n_i - 1 \geq 0$ for $i \in \mathsf{B}$ and $n'_i = n_i$ for $i \notin \mathsf{B}$.
- For each $(q, \lambda, p, g, q', \mathsf{B}) \in \delta_{\mathsf{inc}}$, $(c, \lambda, p, g, c', \emptyset) \in \delta_k$ if $c = (q, n_1, \ldots, n_l)$, $c' = (q', n'_1, \ldots n'_l)$ for some $n_1, \ldots, n_l$ and $n'_1, \ldots, n'_l$ such that $n'_i = \min(n_i + 1, k)$ for $i \in \mathsf{B}$ and $n'_i = n_i$ for $i \notin \mathsf{B}$.

Remark. Please note that the under-approximations used for proving decidability results in this paper differ from the approximations used by Jhala and Majumdar [12] to prove control-state decidability for the special case of CAS with a pushdown store. They use an over-approximation and an under-approximation both of which cut-off the counter values at a given k. For the over-approximation, once the cut-off k for a counter is reached, both increment and decrement edges do not change the counter value. For the under-approximation, there is a transition in the under-approximation only if there is a transition in the original CAS—if an increment edge causes a counter value to go beyond the cut-off k then the transition is not reflected in the under-approximation.

For the rest of this Section, we shall fix the data structure $\mathsf{D} = (D, \widetilde{op}, \widetilde{pred}, \overline{d})$ and a CAS $\mathbf{C} = (Q, \Lambda, \delta_{\mathsf{inc}}, \delta_{\mathsf{dec}}, \overline{q})$ on D. The set $Q \times \mathbb{N}^l$ shall be denoted by C and $C \times D$, the set of configurations of $\mathbf{C}$ shall be denoted by Conf. Given k, $\mathbf{C}^f_{\leq k} = (C_{\leq k}, \Lambda, \delta_k, \emptyset, \overline{c})$ will denote the k-bounded approximation of $\mathbf{C}$. The k-bounded approximation is sound for simulation relation.

Lemma 6.1 (*Soundness of Bounded Approximation*). *If the k-bounded approximation $\mathbf{C}^f_{\leq k}$ of a CAS $\mathbf{C}$ simulates a transition system $\mathbf{S}$ then $\mathbf{C}$ simulates the transition system $\mathbf{S}$.*

Proof. Let S be the set of configurations of $\mathbf{S}$. If the relation $\mathsf{R} \subseteq S \times (C_{\leq k} \times \mathsf{D})$ witnesses the simulation of $\mathbf{S}$ by $\mathbf{C}^f_{\leq k}$, then the relation $R^{\uparrow} \subseteq S \times (C \times \mathsf{D})$ defined as

$$\{(s, (c, d)) \mid \exists c' \leq c \text{ s.t. } (s, (c', d)) \in R\}$$

witnesses the simulation of $\mathbf{S}$ by $\mathbf{C}$. □

We next show that if $\mathbf{S}$ is simulated by $\mathbf{C}$, then there must be some bounded approximation that simulates it.

Lemma 6.2 (*Faithfulness of Bounded Approximations*). *Assume that a CAS $\mathbf{C}$ simulates a finite transition system $\mathbf{S}$. Then there is a k_0 such that $\mathbf{S}$ is simulated by the k_0-bounded approximation $\mathbf{C}^f_{\leq k_0}$.*

Proof. Let S be the set of configurations of $\mathbf{S}$. Recall that $\overline{d}$ is the initial data value. Given $s \in S$, let $\mathsf{Sim}_s \subseteq C$ be the set $\{c \mid s \sqsubseteq (c, \overline{d})\}$. The set Sim_s is upward closed. Let $k_0 = \max(\{\mathsf{rank}(\mathsf{Sim}_s) \mid s \in S, \mathsf{Sim}_s \neq \emptyset\})$. Please note that since $\mathbf{S}$ is a *finite* transition system, k_0 exists and is finite.

Recall that $\mathbf{C}^f_{\leq k_0}$ is the k_0-bounded approximation of $\mathbf{C}$ and has $C_{\leq k_0} = Q \times [0, k]^l$ as the set of control states. Consider the relation $\mathsf{Sim}_{\downarrow} \subseteq S \times (C_{\leq k_0} \times D)$ defined as the union of two relations $\mathsf{Sim}_{<k_0}$ and $\mathsf{Sim}_{=k_0}$. The relation $\mathsf{Sim}_{<k_0} = \{(s, (c, d)) \mid \mathsf{rank}(c) < k_0 \text{ and } s \sqsubseteq (c, d)\}$. The definition of $\mathsf{Sim}_{=k_0}$ is more subtle and uses the function pr_{k_0} defined in Section 2 which cuts-off counter values at k_0. The relation $\mathsf{Sim}_{=k_0} = \{(s, (c, d)) \mid \mathsf{rank}(c) = k_0 \text{ and } \exists c' \geq c \text{ s.t. } (s \sqsubseteq (c', d) \text{ and } \mathsf{pr}_{k_0}(c') = c)\}$.

We claim that the relation $\mathsf{Sim}_{\downarrow}$ is a simulation of S by $\mathbf{C}^f_{\leq k_0}$. In order to prove this, we check the definition of what it means to be a simulation relation. Thus given $(s, (c, d)) \in \mathsf{Sim}_{\downarrow}$, we need to check that every transition of s is matched by (c, d). There are two cases depending on $\mathsf{rank}(c)$.

- $(\mathsf{rank}(c) < k_0)$. In this case, every step of s can be matched as it would be matched in the original CAS $\mathbf{C}$.
- $(\mathsf{rank}(c) = k_0)$. Then there must exist c' such that $\mathsf{rank}(c') \geq k_0$, $\mathsf{pr}_{k_0}(c') = c$ and $(s \sqsubseteq (c', d))$. There are two cases depending on the data value d.
  - $(d = \overline{d})$. Then $c' \in \mathsf{Sim}_s$. Since Sim_s is upward closed and $\mathsf{rank}(\mathsf{Sim}_s) \leq k_0$ (by definition of k_0), we get by Proposition 2.2 that $c = \mathsf{pr}_{k_0}(c') \in \mathsf{Sim}_s$. Thus $s \sqsubseteq (c, \overline{d})$ also and $(c, \overline{d})$ will match each step of s in $\mathbf{C}^f_{\leq k_0}$ as it would match it in $\mathbf{C}$.
  - If $d \neq \overline{d}$, then we observe that all transitions out of (c', d) in $\mathbf{C}$ must be increment transitions. Furthermore, if $(c', d) \xrightarrow{\lambda}_{\mathbf{C},\mathsf{inc}} (c'', d_1)$, then it can be shown that $(\mathsf{pr}_{k_0}(c'), d) \xrightarrow{\lambda}_{\mathbf{C}^f_{\leq k_0}} (\mathsf{pr}_{k_0}(c''), d_1)$. Hence any step of s can be matched in the k_0-bounded approximation.

Thus, we get the desired result. □

We get as a consequence of Lemmas 6.1 and 6.2:

Corollary 6.3. *Assume that the simulation between a finite transition system and a finite D-automaton is decidable. There is a semi-decision procedure that given a finite transition system $\mathbf{S}$ and a CAS $\mathbf{C}$ returns true if $\mathbf{S}$ is simulated by $\mathbf{C}$.*

In order to get the algorithm for deciding simulation, we need another semi-decision procedure which checks if $\mathbf{S}$ is not simulated by $\mathbf{C}$:

Proposition 6.4. *There is a semi-decision procedure that given a finite transition system $\mathbf{S}$ and a CAS $\mathbf{C}$ returns true if $\mathbf{S}$ is not simulated by $\mathbf{C}$.*

Proof. Let S and Conf be the set of configurations of $\mathbf{S}$ and $\mathbf{C}$ respectively. For each $n \in \mathbb{N}$ consider the relation $\sqsubseteq_n \subseteq S \times \mathsf{Conf}$ defined inductively as

- $s \sqsubseteq_0 (c, d)$ for all $s \in S$ and $(c, d) \in \mathsf{Conf}$.
- $s \sqsubseteq_{n+1} (c, d)$ iff $s \sqsubseteq_n (c, d)$, and for each $s \xrightarrow{\lambda} s'$ there is a configuration (c', d') such that $(c, d) \xrightarrow{\lambda}_{\mathbf{C}} (c', d')$ and $s' \sqsubseteq_n (c', d')$.

Since the CAS $\mathbf{C}$ is finitely branching, it can be shown that $s \sqsubseteq (c, d)$ iff $s \sqsubseteq_n (c, d)$ for all n. Hence, in order to check if $s \not\sqsubseteq (c, d)$ one needs to find a n such that $s \not\sqsubseteq_n (c, d)$. Also note that given s,(c, d) and n there is a decision procedure to check if $s \not\sqsubseteq_n (c, d)$ which gives us a semi-decision procedure that returns true if $\mathbf{S}$ is not simulated by $\mathbf{C}$. □

Combining Corollary 6.3 and Proposition 6.4, we get:

Theorem 6.5 (*Simulation of Finite Transition Systems*). *Assume that the simulation of a finite transition system by a finite D-automaton is decidable. There is an algorithm that given a finite transition system $\mathbf{S}$ and a CAS $\mathbf{C}$ returns true if $\mathbf{S}$ is simulated by $\mathbf{C}$, otherwise it returns false.*

7. Conclusions and future work

We gave sufficient conditions under which the problems of termination, control state maintainability, simulation of and by finite state systems for counter automata with store (CAS) are decidable; the problem of verifying general CTL, CTL*, and modal μ-calculus problems is known to be undecidable. An immediate consequence of our observation is that these problems are decidable for asynchronous programs. We can also show that the algorithms to decide termination and control state reachability can be generalized to w.q.o. automata (automata whose control states are general w.q.o.s rather than counter w.q.o.), though we do not report these results here. Problems for future research include verifying LTL (or ω-regular) properties of CAS, and checking if the simulation algorithms can be generalized to w.q.o.. automata as well.

Acknowledgements

The authors would like to thank the anonymous referees for their useful comments. Their suggestions have greatly helped the exposition of the proofs in Section 5. Rohit Chadha was supported partially by NSF CCF 04-29639 and Mahesh Viswanathan was supported partially by NSF CCF 04-48178.

References

[1] The terminator project http://research.microsoft.com/terminator/.
[2] P.A. Abdulla, K. Cerans, B. Jonsson, Y.-K. Tsay, Algorithmic analysis of programs with well quasi-ordered domains, Information and Computation 160 (1) (2000) 109–127.
[3] P.A. Abdulla, K. Cerans, B. Jonsson, T.Y -Kuen, General decidability theorems for infinite state systems, in: Proc. of LICS, 1996, pp. 313–321.
[4] R. Chadha, M. Viswanathan, Decidability results for well-structured transition systems with auxiliary storage, in: Proc. of CONCUR, 2007, pp. 136–150.
[5] R. Cunningham, Eel: Tools for debugging, visualization, and verification of event-driven software, University of California, Los Angeles, 2005.
[6] J. Esparza, Reduction and synthesis of live and bounded free choice Petri nets, Information and Computation 114 (1) (1994) 50–87.
[7] A. Finkel, Ph. Schnoebelen, Well-structured transition systems everywhere!, Theoretical Computer Science 256 (1–2) (2001) 63–92.
[8] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, D. Culler, The nesC language: A holistic approach to networked embedded systems, in: Proc. of PLDI, 2003, pp. 1–11.
[9] S. Graf, H. Saidi, Construction of abstract state graphs with pvs, in: Proc. of CAV, 1997, pp. 72–83.
[10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, System architecture directions for networked sensors, in: Proc. of the Conf. on Architectural support for Prog. Lang. and Operating Systems, 2000, pp. 93–104.
[11] Allen Holub, Taming Java Threads, APress, 2000.
[12] R. Jhala, R. Majumdar, Interprocedural analysis of asynchronous programs, in: Proc. of POPL, 2007, pp. 339–350.
[13] C. Killian, J. W. Anderson, R. Braud, R. Jhala, A. Vahdat, Mace: Language support for building distributed systems, in: Proc. of PLDI, 2007.
[14] J.B. Kruskal, The theory of well-quasi-ordering: A frequently discovered concept, Journal of Combinatorial Theory: Series A 13 (3) (1972) 297–305.
[15] Libasync http://pdos.csail.mit.edu/6.824-2004/async/.
[16] Libevent http://www.monkey.org/~provos/libevent/.
[17] K. Sen, M. Viswanathan, Model checking multithreaded programs with asynchronous atomic methods, in: Proc. of CAV, 2006, pp. 300–314.
[18] S. La Torre, P. Madhusudan, G. Parlato, Context-bounded analysis of concurrent queue systems, in: Proc. of TACAS, 2008.