Checking Liveness Properties of Presburger Counter Systems Using Reachability Analysis

K. Vasanta Lakshmi, Aravind Acharya, and Raghavan Komondoor

Indian Institute of Science, Bangalore
{kvasanta,aravind.acharya,raghavan}@csa.iisc.ernet.in

Abstract. Counter systems are a well-known and powerful modeling notation for specifying infinite-state systems. In this paper we target the problem of checking liveness properties in counter systems. We propose two semi decision techniques towards this, both of which return a formula that encodes the set of reachable states of the system that satisfy a given liveness property. A novel aspect of our techniques is that they use reachability analysis techniques, which are well studied in the literature, as black boxes, and are hence able to compute precise answers on a much wider class of systems than previous approaches for the same problem. Secondly, they compute their results by iterative expansion or contraction, and hence permit an approximate solution to be obtained at any point. We state the formal properties of our techniques, and also provide experimental results using standard benchmarks to show the usefulness of our approaches. Finally, we sketch an extension of our liveness checking approach to check general CTL properties.

1 Introduction

Counter systems are a class of infinite state systems that are equivalent to simple looping programs that use integer variables, without arrays and pointers. A counter system has a finite set of *control states* and a finite set of *counters*, with each counter taking values from the infinite domain of integers. There are transitions between control states, with each transition being *guarded* by a predicate on the counters, and having an *action*, which indicates the updated values of the counters in terms of the old values. *Presburger* logic is the decidable first-order theory of natural numbers. Presburger formulas use variables, constants, addition and subtraction, comparisons, and quantification. The class of counter systems where the guards as well as actions are represented using Presburger formulas are called Presburger counter systems. Presburger counter systems have been shown to be applicable in various settings [1], such as the analysis of the TTP protocol, different broadcast protocols, as well as cache coherence protocols. In the rest of this paper we use "counter system" or even just "system" to refer to Presburger counter systems.

Verification of properties of counter systems has been an important topic in the research literature. While problems such as reachability analysis and temporal property checking are decidable for infinite systems such as pushdown systems and petri-nets [2, 3], these problems are in general undecidable on counter systems because of their greater expressive power. This said, various interesting subclasses of counter systems have been identified on which reachability analysis is decidable [4–8]. When it comes to *CTL* [9] temporal property checking, researchers have shown decidability of this problem on significantly narrower classes [10, 11]. We seek to bridge this gap somewhat, by proposing a novel CTL property checking technique that uses reachability queries as subroutines.

1.1 Our Approach

Although our technique addresses the full CTL, the focus in this paper is mainly on checking *liveness* properties, which are a fragment of the full CTL. Intuitively, a *state* $\boldsymbol{s}$ (i.e., a *vector* of actual counter values) of a counter system is said to satisfy liveness with respect to a given "good" property (which is expressed as a Presburger formula on the counter values) if no matter what trace is taken from $\boldsymbol{s}$ a state that satisfies the good property will eventually be reached [12]. A classic example of a liveness property is that an entity that requests a resource will eventually get the resource (i.e., will not *starve*). If there are no *stuck* states in the system (i.e., states with no outgoing transitions)[1], then a state $\boldsymbol{s}$ satisfies the liveness property iff there *does not exist* an infinite trace starting from $\boldsymbol{s}$ along which none of the states satisfy the good property. That is, using CTL notation, $\boldsymbol{s}$ ought to satisfy the temporal property $\neg\mathbf{EG}\phi$, where ϕ is the negation of the given good property. (**E** represents *existential* quantification over traces, while **G** indicates an infinite trace along which the property ϕ holds *globally*, i.e., on all states.) Therefore, we address the following problem: given a counter system M and a temporal property $\mathbf{EG}\phi$, where ϕ is a formula representing a set of states, return a Presburger formula that encodes the set of reachable states of the system that satisfy this temporal property.

It is easy to see that fix-point computations are required to analyze properties of infinite traces. Our key idea in this paper is to use the fix-point computation capabilities of reachability analysis techniques to solve **EG** properties. Our approach is to perform certain transformations on the given counter system, and then to perform reachability analysis iteratively, hence computing a progressively more precise approximation of the set of states that satisfy the **EG** property. We actually provide two alternative approaches for the same problem: one that computes a growing under-approximation of the solution, and the other that computes a shrinking over-approximation. Both are guaranteed to return a precise answer upon termination; however, termination is not always guaranteed, even on systems that are within the subclasses of systems on which reachability is decidable. In cases where guaranteed termination is not clear, the user can select one of the two approaches based on their desired direction of approximation, and forcibly stop the analysis at any point to obtain an approximate result.

[1] Any counter system with stuck states can be transformed into one without stuck states, by adding a "dead" control state that has an unconditional self-loop with identity action, and by adding transitions that take control from erstwhile stuck states to this control state.

1.2 Contributions

- The key novelty of our approach, over previous CTL model-checking approaches [10, 11] is the use of reachability analysis black-boxes as subroutines. In particular, as a result, we are able to show that the subclass of systems on which each of our approaches is guaranteed to terminate (with precise solutions) is arguably wider than the subclass addressed by Demri et al. [10] (and potentially incomparable with that of Bozzelli et al.).
- We support approximations in cases where termination is not guaranteed. This is a useful feature that is not a part of previous approaches.
- We also introduce an algorithm that can return the set of states that satisfy any given CTL property. Previous approaches do not address arbitrarily nested properties. (Our approaches for **EG** properties are in fact used as subroutines in this algorithm.) Due to lack of space we only provide a sketch of this algorithm.
- For each of our two **EG** approaches we state claims of precision (after termination), approximation in the appropriate direction (before termination), and guaranteed termination on a certain subclass of systems. We provide proofs of these claims in an associated technical report [13].
- We implement both our approaches, and provide experimental results on a large set of real-life counter systems provided by the FAST [1] toolkit that are outside the subclass of systems addressed by the approach of Demri et al. [10].

The remainder of this paper is organized as follows: In Section 2 we introduce some of the preliminary notions and terminology that underlies our approaches. In Sections 3 and 4 we describe our under-approximation and over-approximation approaches to answer **EG** properties, respectively. In Section 5 we sketch our algorithm for answering CTL properties. Section 6 contains the discussion on our implementation and experimental results, while Section 7 discusses related work.

2 Preliminaries

Definition 1 (Counter System). *A counter system M is represented by the tuple $M = \langle Q, C, \Sigma, \phi^{init}, G, F\rangle$ where Q is a finite set of natural numbers that encode the control states, C is a finite set of m counters, ϕ^{init} is a Presburger formula that represents the initial states of the system, Σ is a finite alphabet representing the set of transitions in M, such that for each $b \in \Sigma$ there exists a Presburger formula $G(b)$ and a Presburger formula $F(b)$ that are the guard and action of the transition b, respectively.*

Throughout this paper we use the notation g_b and f_b to represent $G(b)$ and $F(b)$.

Figure 1(a) shows a counter system, which also serves as our running example. Here $Q = \{q_0\}$ (encoded as the natural number zero), $C = \{x\}$, $\Sigma = \{t_0, t_1\}$,

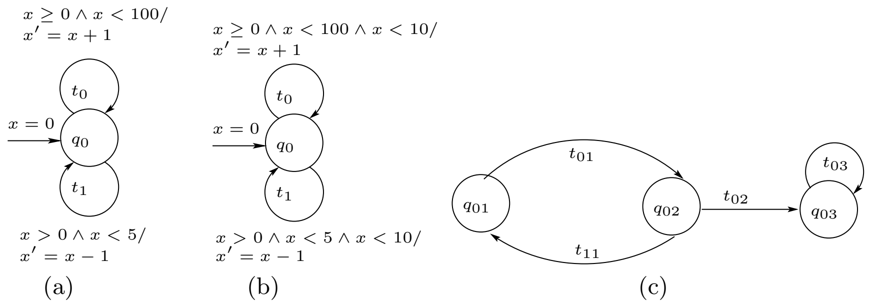


Fig. 1. (a) A counter system M. (b) Refinement M_1 of M w.r.t $(x < 10)$. (c) A flattening N of M_1. Each transition t_{ij} of N has the same guard and action as transition t_i of M_1.

$\phi^{init} = (x = 0)$ (shown as the incoming arrow into the system), $g_{t_0} \equiv (x \geq 0) \wedge (x < 100)$, $g_{t_1} \equiv (x > 0) \wedge (x < 5)$, $f_{t_0} \equiv (x' = x + 1)$, $f_{t_1} \equiv (x' = x - 1)$. In our figures we separate the guard and action of a transition using a "/".

A *state* (denoted by $\boldsymbol{s}$, $\boldsymbol{s_0}$, $\boldsymbol{s'}$ etc.) in a system is a column vector $\boldsymbol{v} \in \mathbb{N}^{m+1}$. The first element v_0 represents the control state, while the values of rest of the elements $v_1, \ldots, v_m$ represent the values of the counters C. We sometimes use the term *concrete state* to refer to a state.

Our Presburger formulas use the names of the counters, as well the *control-state variable* q (which refers to the first element v_0 of a state as mentioned above), as free variables. Any formula can be seen either as a property to be satisfied by a state (e.g., when used in a guard), or as a set of states (e.g., in the context of input to our algorithm or output from it). Throughout this paper we use ϕ, ϕ_i, etc., to denote Presburger formulas. Since the example systems we use for illustrations have only a single control-state, we omit the control-state variable q from the guards, actions, and formulas that we show (it will always be constrained to be zero). Also, sometimes we wish to use extra free variables (on top of the counter names and q) in a formula. Our notation in this case is as follows: $\phi_{(k)}$ is a Presburger formula with an additional free variable k. (There is a another kind of Presburger formula, too, which is used to represent actions of transitions, and uses unprimed and primed versions of the free variables mentioned above.)

A state $\boldsymbol{s}$ is said to *satisfy* a formula ϕ, denoted as $\boldsymbol{s} \models \phi$, if the formula ϕ evaluates to true when the free variables in ϕ are substituted by the corresponding values in $\boldsymbol{s}$. For this reason, we often refer to a formula as a "set of states", by which we mean the states that satisfy the formula.

The semantics of a counter system is as follows. A *concrete transition* $\boldsymbol{s} \xrightarrow{b} \boldsymbol{s'}$ is possible (due to transition b) if $\boldsymbol{s}$ satisfies g_b and $(\boldsymbol{s}, \boldsymbol{s'})$ satisfies f_b. In this case we say that $\boldsymbol{s}$ ($\boldsymbol{s'}$) is an immediate predecessor (successor) of $\boldsymbol{s'}$ ($\boldsymbol{s}$). A counter system can be *non-deterministic*; i.e., a state could have multiple successor states, either by the action of a single transition itself, or due to different transitions out of a control-state with overlapping guards. However, we assume that systems exhibit *finite branching*; i.e., every state has a finite number of immediate successors.

Given a counter system M, a *trace* t "in" M *starting from* a state $\boldsymbol{s_0}$ is any sequence of states $\boldsymbol{s_0}, \boldsymbol{s_1}, \ldots, \boldsymbol{s_n}$, $n \geq 0$, such that there is a concrete transition in M from each state in the sequence to the immediate successor (if any) in the sequence. This definition also generalizes in a natural way to infinite traces. If t is a trace in M we also say that M *exhibits* t. $\mathit{traces}(M, \phi)$ is the set of all traces in M from states that satisfy ϕ. A state $\boldsymbol{s_0}$ in a system M is said to satisfy a temporal formula $\mathbf{EG}\phi$, written as $\boldsymbol{s_0} \models \mathbf{EG}\phi$, iff there exists an infinite trace $\boldsymbol{s_0}, \boldsymbol{s_1}, \ldots$ in the system such that $\forall i \geq 0.\, \boldsymbol{s_i} \models \phi$.

Other Definitions. Given a counter system M and Presburger formula ϕ, we use the formula $\mathit{pre}(M, \phi)$ (which is also a Presburger formula in the counter variables and in q) to represent the set of all states that have a successor that satisfies ϕ. For the counter system M shown in Figure 1(a), $\mathit{pre}(M, (x \leq 2)) \equiv (x \geq 0) \wedge (x \leq 3)$.

An extension of the above definition is the formula $\mathit{pre}^k(M, \phi)_{(k)}$. This represents the set of all states from which some state that satisfies ϕ can be reached in exactly k steps (i.e., k concrete transitions). Note that k is an extra free variable in the formula $\mathit{pre}^k(M, \phi)_{(k)}$. For our example system M, $\mathit{pre}^k(M, x = 4)_{(k)} \equiv (x \leq 4) \wedge (x \geq (4 - k)) \wedge (\mathit{even}(k) \Rightarrow \mathit{even}(x)) \wedge (\mathit{odd}(k) \Rightarrow \mathit{odd}(x))$.

The backward reachability set for a set of states ϕ, namely $\mathit{pre}^*(M, \phi)$, represents the set of all states from which a state in ϕ can be reached in zero or more steps. For our example system M, $\mathit{pre}^*(M, x \leq 4) \equiv x \leq 4$.

A system M_1 is said to be a *refinement* of a system M with respect to a formula ϕ, written as $M_1 \equiv \mathit{refineSystem}(M, \phi)$, if M_1 is identical to M in every way except that the guard of each transition in M_1 is the corresponding guard in M *conjuncted* with ϕ. For instance, the system M_1 in Figure 1(b) is a refinement of the system M in part (a) of the same figure with respect to the formula '$x < 10$'. Intuitively, M_1 exhibits exactly those traces in M that do not go through a concrete transition from a state that does not satisfy ϕ.

A *flat* counter system is one in which no two distinct cycles among its control states overlap. That is, all cycles among its control states are simple cycles. A flat system N is said to be a *flattening* of a system M if, intuitively, (a) the two systems use the same set of counters, (b) each control-state q_i of M occurs zero or more times in N, with each of these copies encoded by the *same* natural number as q_i, and (c) any transition in N from a control-state q_{ij} (a copy of q_i in M) to a control-state q_{kl} (a copy of q_k in M) has the same guard and action as some transition from q_i to q_k in M. It is easy to see that in general, for *any* set of states ϕ, $\mathit{traces}(N, \phi) \subseteq \mathit{traces}(M, \phi)$. We say that N is a *trace flattening* of M with respect to a specific set of states ϕ if $\mathit{traces}(N, \phi) = \mathit{traces}(M, \phi)$.

For instance, Figure 1(c) shows a flattening N of the (non-flat) system M_1 in part (b) of the figure. Control state q_0 in M_1 has three copies in N; also, each transition t_{ij} in N corresponds to transition t_i in M_1. N is a trace flattening of M_1 with respect to the set of states '$x \geq 5$'. On the other hand, any trace that involves taking transition t_1 twice in a row, such as '$x = 4, x = 3, x = 2$' is missing in N.

3 Under-Approximation Approach for EG Properties

In this section we describe our under-approximation approach for solving **EG** properties, implemented as Algorithm *computeGlobalUnder*. The input to the algorithm is a counter system M and a temporal property $\mathbf{EG}\phi$. We first present the key ideas behind our approach, and then finally present our entire approach in pseudo-code form.

3.1 Our Approach

Using Refinement and Reachability. Let M_1 be the refinement of the given system M with respect to the given ϕ; i.e., $M_1 \equiv \mathit{refineSystem}(M, \phi)$. Clearly, a state satisfies $\mathbf{EG}\phi$ in M iff it satisfies $\mathbf{EG}\phi$ in M_1, which in turn is true iff there is at least one infinite trace from this state in M_1; this is because *every* concrete transition in M_1 starts from a state that satisfies ϕ. Our objective now is to find a Presburger formula, somehow using reachability analysis, that represents the set of states in M_1 that have an infinite trace starting from them. Two key insights that make this possible are: (a) In a finite-branching system, as per Köenig's Lemma, there is an infinite trace from a state iff there are traces starting from it of *all possible* lengths k, for $k \geq 0$. (b) A state has a trace of length k from it iff it satisfies the formula $\mathit{pre}^k(M_1, \phi)_{(k)}$, which can be computed by reachability analysis. Therefore, with this formula in hand, one only needs to eliminate k as a free variable from it using universal quantification, as in $\forall k \geq 0.\ \mathit{pre}^k(M_1, \phi)_{(k)}$, to obtain the precise set of states that satisfy $\mathbf{EG}\phi$ in M.

Computing $\mathit{pre}^k(M_1, \phi)_{(k)}$. Existing reachability analysis that are based on "accelerations" [4, 6–8] can be used as black-boxes for computing the formula $\mathit{pre}^k(M_1, \phi)_{(k)}$. However, a key limitation of all these techniques is that although they can compute the formula $\mathit{pre}^*(M_1, \phi)$ for interesting subclasses of systems, on the more difficult problem of computing the formula $\mathit{pre}^k(M_1, \phi)_{(k)}$ their applicability is restricted to the narrow class of flat systems. Whereas, most practical systems, such as those provided by the Fast toolkit [1] are not flat, and are not even trace-flattable with respect to large subsets of states in the system. A way out of this quandary is to obtain any flattening N of M_1, and to compute the formula $\mathit{pre}^k(N, \phi)_{(k)}$. The presence of an infinite trace in N from any state $\boldsymbol{s}$ implies the presence of the same trace in M_1. Therefore, the set of states that satisfy $\mathbf{EG}\phi$ in N (as represented by the formula $\forall k \geq 0.\ \mathit{pre}^k(N, \phi)_{(k)}$) is guaranteed to be a subset (i.e., an under-approximation) of the set of states that satisfy $\mathbf{EG}\phi$ in M_1.

We now build upon the idea above by systematically enumerating various flattenings of M_1, and by accumulating the sets of states that satisfy $\mathbf{EG}\phi$ in these

```
Require: A system $M$ and a set of states $\phi$.
Ensure: Returns a set of states, and a label *approx* which indicates whether the
    returned set is precise or is an under-approximation of $\mathbf{EG}\phi$ in $M$.
 1: $M_1 \leftarrow \mathit{refineSystem}(M, \phi)$. $k \leftarrow 1$. $X \leftarrow \emptyset$.
 2: while *not forced to stop* do
 3:    $\mathit{FLAT} \leftarrow$ *All flattenings of* $M_1$ *of length* $k$
 4:    for all $N \in \mathit{FLAT}$ do
 5:       $X \leftarrow X \vee \mathit{pre}^*(N, X)$
 6:       $X \leftarrow X \vee \forall k \geq 0.\mathit{pre}^k(N, \phi)_{(k)}$
 7:       if $\mathit{isTraceFlattening}(M_1, N, \phi - X)$ then
 8:          return $(X, \mathit{precise})$
 9:    $k \leftarrow k + 1$
10: return $(X, \mathit{under})$
```

Fig. 2. Algorithm *computeGlobalUnder*

flattenings. Therefore, this accumulated set, which we call X, is a monotonically non-decreasing under-approximation of the set of states that satisfy $\mathbf{EG}\phi$ in M_1. In order to be systematic, we enumerate flattenings of M_1 in increasing order of *length*, where the length of a flattening is the number of transitions it possesses.

Termination condition. There is no obvious way to decide to stop enumerating flattenings of M_1 based just on whether the set X is still growing or has stopped growing. Therefore, the termination condition that we actually use is as follows: when we come across a flattening N of M_1 such that $\mathit{traces}(M_1, \phi - X) = \mathit{traces}(N, \phi - X)$[2], we stop, and return the current set X as the precise solution. Our termination condition is correct for the following reason: X contains all states that satisfy $\mathbf{EG}\phi$ in N (in addition to states that satisfied $\mathbf{EG}\phi$ in other flattenings enumerated prior to N). Therefore, $\phi - X$ describes states that do not satisfy $\mathbf{EG}\phi$ in N, but could potentially satisfy $\mathbf{EG}\phi$ in M_1. However, since every trace in M_1 starting from states in $\phi - X$ is also present in N (as per the termination check) these states do not satisfy $\mathbf{EG}\phi$ in M_1, either. Therefore X represents precisely the set of states that satisfy $\mathbf{EG}\phi$ in M_1.

Figure 2 shows the pseudo-code for Algorithm *computeGlobalUnder*. We have already discussed all the details of this algorithm. One point to note is line 5; this makes sense because any state from which a state in $\mathbf{EG}\phi$ is reachable itself satisfies $\mathbf{EG}\phi$.

Illustration. Say we want to solve the property $\mathbf{EG}\phi$, where $\phi \equiv x < 10$, for the system M in Figure 1(a). The refined system M_1 is shown in part (b) of the figure. Part (c) of the figure shows a flattening N, wherein the set of states that satisfy $\mathbf{EG}\phi$ is $x < 5$. Ignoring other flattenings that might have been enumerated before N, let us treat X as being equal to $x < 5$. It can be observed that $\mathit{traces}(M_1, \phi - X) = \mathit{traces}(N, \phi - X)$. Therefore the algorithm will terminate on this input with answer $(x \geq 0) \wedge (x < 5)$.

[2] This check is decidable provided pre^* can be computed on the flattening N [10].

3.2 Theoretical Claims

It is not very difficult to see that the set X maintained by the algorithm is a monotonically non-decreasing under-approximation of the set of states that satisfy $\mathbf{EG}\phi$ in M. Also, that upon termination the accumulated set X contains the precise solution. However, it is *not* necessarily true that in all cases where X becomes equal to the precise solution the algorithm will detect this situation and terminate.

A sufficient condition for the termination of the algorithm on a system M is that (a) pre^k and pre^* queries terminate on flattenings of the refined system M_1, and (b) the system $M_1 \equiv \mathit{refineSystem}(M, \phi)$ has a flattening N such that $\mathit{traces}(N, \phi - X) = \mathit{traces}(M_1, \phi - X)$, where X is the set of states that satisfy $\mathbf{EG}\phi$ in N.

While this is a simple condition to state, this characterization describes a class that is strictly broader than the class addressed by the approach of Demri et al. [10], which targets only the class of systems M that are trace-flattable with respect to ϕ^{init}; i.e., M needs to have a flattening N such that N exhibits *all* traces that M exhibits from *all* states that are reachable in M.

We provide formal statements and proofs of all our claims in the associated technical report [13].

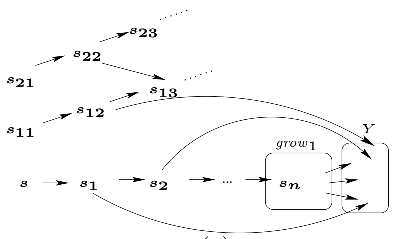
4 Over-Approximation Approach for EG Properties

Given a counter system M and a temporal formula $\mathbf{EG}\phi$ this algorithm first computes the refined system $M_1 \equiv \mathit{refineSystem}(M, \phi)$, and then iteratively accumulates in a set Y a growing set of states that definitely *do not satisfy* $\mathbf{EG}\phi$. Upon termination it returns $\phi^{reach} - Y$ as the precise set of states that satisfy $\mathbf{EG}\phi$ in M, whereas upon a forced stop it returns $\phi^{reach} - Y$ as an over-approximation. ϕ^{reach} is a Presburger formula representing the set of reachable states in M. This approach basically resembles the classical approach for solving $\mathbf{EG}$ properties for finite-state systems [9], but uses reachability analysis as a black-box to accelerate the process of adding states to Y.

4.1 Details of the Approach

Recall that a state does not satisfy $\mathbf{EG}\phi$ in M_1 iff all traces starting from it are finite. Therefore, the algorithm starts by initializing the set Y to the set of states that don't satisfy ϕ or are "stuck" (i.e., have no outgoing transition) in M_1, since these states trivially do not satisfy $\mathbf{EG}\phi$ (M_1 could have stuck states even if the original system M did not). Subsequently, in each iteration, the algorithm identifies states that do not satisfy $\mathbf{EG}\phi$ in M_1, using two different conditions as described below, and adds them to Y.

Condition 1: If all successors of a state $\boldsymbol{s}$ are in Y then $\boldsymbol{s}$ can be added to Y. The states that satisfy this property can be identified using the following Presburger formula:



$$\forall i,j \in \Sigma.\, ((g_i \wedge f_i) \wedge (g_j \wedge f_j[s''/s']) \implies (s' = s'') \vee (Y[s'/s] \wedge Y[s''/s]) \vee (Y[s'/s] \wedge \neg Y[s''/s]) \vee (\neg Y[s'/s] \wedge Y[s''/s]))$$

(a) (b)

Fig. 3. (a) Illustration of formula $grow_2$. (b) Formula for $\phi_{atmost\_one\_succ\_outside\_Y}$.

$$grow_1 \;\equiv\; (\forall s'.\,((g_1 \wedge f_1) \vee (g_2 \wedge f_2) \vee \ldots (g_n \wedge f_n) \implies Y[s'/s])) - Y$$

Assuming Y is a Presburger formula in the counter variables and q, $grow_1$ is also a Presburger formula in these same variables. $Y[s'/s]$ represents the variant of Y where each variable is substituted with its primed version.

Condition 2: Ignoring all concrete transitions whose target state is already in Y, if a state $\boldsymbol{s}$ is such that (a) there is only one trace t in M_1 starting from $\boldsymbol{s}$ (not counting prefixes of this trace t), and (b) t reaches a state that satisfies $grow_1$ after a finite number of steps, then $\boldsymbol{s}$ can be added to Y. In the illustration in Figure 3(a), states $\boldsymbol{s}, \boldsymbol{s_1}, \boldsymbol{s_2}$, etc., satisfy both sub-conditions (a) and (b) mentioned above; state $\boldsymbol{s_{11}}$ satisfies only sub-condition (a), while state $\boldsymbol{s_{21}}$ satisfies neither of the two sub-conditions.

The states that satisfy sub-condition (a) can be identified using the following Presburger formula:

$$grow_{2a} \equiv \neg(pre^*(M_1, \neg\phi_{atmost\_one\_succ\_outside\_Y}))$$

where $\phi_{atmost\_one\_succ\_outside\_Y}$ represents the states that have at most one successor state that is not already in Y. Therefore, $\neg\phi_{atmost\_one\_succ\_outside\_Y}$ represents states that have two or more successors outside Y. Therefore, the transitive predecessors of these states are the ones that *don't* belong to $grow_{2a}$.

The formula for $\phi_{atmost\_one\_succ\_outside\_Y}$ is shown in Figure 3(b). Intuitively, the part before the '$\implies$' identifies pairs of successor states (s', s'') of the state $\boldsymbol{s}$ under consideration, while the part after the '$\implies$' requires that s and s' be the same state, or that at least one of them be already in Y. g_i, f_i are the guard and action of transition i, respectively.

Now, sub-condition (b) above is captured by the following formula: $grow_{2b} \equiv pre^*(M_1, grow_1)$. Therefore, the states to be in added to Y by *Condition 2* are described by the formula $grow_2 \equiv grow_{2a} \wedge grow_{2b}$.

Figure 4 shows the pseudo-code for the entire algorithm. Note that the termination condition is that $grow_1$ and $grow_2$ are both unsatisfiable (i.e., empty).

Illustration. Consider the example system M given in Figure 1(a) and the property $\mathbf{EG}\phi$, where $\phi \equiv x < 10$. The over-approximation algorithm initializes the set Y to $x \geq 10$. In the first iteration of the loop state $(x = 9)$ has its only

```
Require: A system $M$ and a set of states $\phi$.
Ensure: Returns a set of states, and a label approx which indicates whether the
    returned set is precise or is an over-approximation of $\mathbf{EG}\phi$ in $M$.
1: $M_1 = \mathit{refineSystem}(M, \phi)$
2: /* Initialize $Y$ to states that have no successors or don't satisfy $\phi$. */
3: $Y = \neg(g_1 \vee g_2 \vee \cdots \vee g_n) \vee \neg\phi$
4: while ($(\mathit{grow}_1 \vee \mathit{grow}_2)$ is satisfiable) $\wedge$ not forced to stop do
5:     $Y = Y \vee (\mathit{grow}_1 \vee \mathit{grow}_2)$
6: return ($(\mathit{grow}_1 \vee \mathit{grow}_2)$ is satisfiable) ? $(\phi^{reach} - Y,\ \mathit{over})$ : $(\phi^{reach} - Y,\ \mathit{precise})$
```

Fig. 4. Algorithm *computeGlobalOver*

successor $(x = 10)$ in Y and hence will satisfy grow_1. Also, the states $(x \geq 5) \wedge (x < 9)$ have only one outgoing trace starting from them and every such trace ends in state in $x = 9$. Hence states $(x \geq 5) \wedge (x < 9)$ satisfy grow_2. Hence, Y gets expanded to $x \geq 5$. In the next iteration no states satisfy grow_1 or grow_2, and hence the algorithm terminates. It returns the set of reachable states that are not in Y, namely $(x \geq 0) \wedge (x < 5)$.

4.2 Theoretical Claims

We have already argued informally that the algorithm (a) maintains a growing under-approximation Y of the set of states in M_1 that do not satisfy $\mathbf{EG}\phi$, and (b) terminates iff Y becomes precisely equal to this set.

In order to make an intuitive argument about termination we argue termination of our algorithm on three successive classes, each one wider than the previous one. The first class is the class of systems M such that the refined system M_1 is flat and such that pre^* queries on it terminate. Any flat system can be seen as a directed acyclic graph (DAG), whose elements are simple cycles or transitions that are not part of any cycle. We argue that the algorithm "processes" any element e, i.e., identifies all states "in" the control-states in e that need to be added to Y, in the immediate subsequent iteration after all successor elements of e in the DAG have been processed. Intuitively, grow_1 is responsible for processing elements that are transitions, and grow_2 for simple cycles.

The next class is the class of systems M such that the refined system M_1 has a *trace flattening* with respect to ϕ^{init} and such that pre^* queries on M_1 terminate. This is a generalization of the class on which the approach of Demri et al. [10] terminates. Our argument for this class is a simple extension of our argument for flat systems that is based on the structure of the trace flattening of the system M_1 rather than on the structure of M_1 itself.

Our final class is of systems M such that (a) pre^* queries on the refined system M_1 terminate, (b) there exists an integer bound k, and a (finite or infinite) set of flattenings of M_1 such that each flattening N in the set contains at most k simple cycles (each one involving an arbitrary number of control states), and such that each trace in M_1 that starts from a state that does not satisfy $\mathbf{EG}\phi$ is exhibited by at least one of the flattenings mentioned above. (As it was with the under-approximation algorithm, this characterization is a sufficient condition, and does not exhaustively cover all cases on which our algorithm terminates.)

We provide a proof sketch of the final claim above, and full proofs of all other claims in the associated technical report [13].

An interesting question that is left to future work is to determine how the classes of systems on which our under- and over-approximation techniques terminate compare.

5 Algorithm for Full CTL

In this section we sketch our algorithm for computing the set of states in a counter system that satisfy any given *CTL* property. The algorithm takes a counter system M, a *CTL* temporal property ψ, and an *approximation label* as input. The CTL property is assumed to be in *existential normal form*, where the main operators are *EG*, *EX* ("exists next"), and *EU* ("exists until"). The label, which is from the set $\{$*over*, *under*, *precise*$\}$, specifies the allowed direction of approximation in case the set of states that satisfy ψ in M cannot be computed precisely. The algorithm works in two passes. The first pass is a top-down pass, where the objective is to identify the allowed direction of approximation for each sub-property of ψ. An interesting aspect here is that '$\neg$' operators cause the allowed direction of approximation to get reversed. In the second pass the set of states that satisfy each sub-property is computed in a bottom-up manner. We use the notation ϕ_i to denote the solution (set of states) computed for a sub-property ψ_i of ψ. For a sub-property $\mathbf{EG}\psi_1$, the solution is obtained by invoking $computeGlobalUnder(M, \phi_1)$ or $computeGlobalOver(M, \phi_1)$, depending on the label assigned to this sub-property in the top-down pass. A sub-property $EX\psi_i$ can be solved simply as $pre(M, \phi_i)$. A sub-property $E(\psi_i \, U \, \psi_j)$ can be solved as $pre^*(refineSystem(M, \phi_i), \phi_j)$. In case the underlying pre^* black-box is not able to terminate then the approximation label assigned to this sub-property can be used to perform an approximated pre^* computation. We provide a detailed discussion of the above algorithm in the associated technical report [13].

6 Implementation and Results

We have implemented our two algorithms *computeGlobalUnder* and *computeGlobalOver*. We use the reachability analysis black-boxes provided by the Fast toolkit [1] in our implementations. Fast is applicable on counter systems whose guards and actions satisfy certain constraints [6]. Fast provides a routine for computing pre^* formulas on systems, which necessarily terminates on flat systems as well as on systems that have a trace flattening with respect to ϕ^{init}, but also terminates on many other systems that do not have these properties. Fast also provides (an always terminating) routine to compute pre^k formulas on simple cycles, which we extended in a straightforward way to work on flat systems. We implemented the routine *isTraceFlattening*, which is

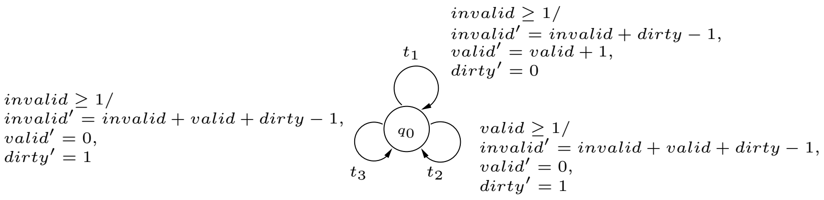


Fig. 5. MSI cache coherence protocol

required by Algorithm *computeGlobalUnder*, using the trace-flattening check formula referred to by Demri et al. [10] and shared with us by them via private communication.

Benchmarks Selection. The Fast toolkit comes bundled with a number of example counter systems, which model cache coherence protocols, client-server interactions, control systems for lifts and trains, producer-consumer systems, etc. For instance, the counter system shown in Figure 5 is from this toolkit, and models the MSI cache coherence protocol for a single cache line. The counters *invalid*, *valid* and *dirty* represent the number of processors in the respective states for the modeled cache line.

From the 45 example systems in the bundle, we chose, using a simple sufficient condition that we designed, 17 systems that are guaranteed to *not* have a trace-flattening with respect to ϕ^{init}. We chose such systems because they are *outside* the class of systems addressed by the previous approach of Demri et al. [10] and on which our approaches are known to definitely terminate. In other words, they are the more challenging systems.

These 17 systems (and the remaining 28 in the toolkit, also) were analyzed previously only with reachability queries; the toolkit as such does not contain any sample temporal properties for these systems. Therefore, after studying these systems we manually identified CTL temporal properties for these systems which we believe would be satisfied by all the reachable states of these systems, such that each CTL property contains an **EG** sub-property. We identified two properties each for two of the systems, and one each for the 15 remaining systems, thus resulting in 19 properties. For instance, for the MSI system shown in Figure 5, the temporal property we identified is $valid \geq 1 \implies \mathbf{A}\big((valid \geq 1)\mathbf{U}(dirty = 1)\big)$. This property states that if at some point the cache line is in the valid state in some processor then this remains true in subsequent steps and eventually some processor moves into the dirty state wrt this line. This property holds at all reachable states, intuitively, because transition t_1, which is the only transition that prevents any processor from entering into the dirty state, cannot be taken indefinitely often (due to the bound on the number of processors in any instance of the protocol). The above property can be written in existential normal form as $(valid < 1) \vee \neg((\mathbf{EG}(dirty \neq 1)) \vee \mathbf{E}\big((dirty \neq 1)\ \mathbf{U}\ (dirty \neq 1 \wedge valid < 1)\big))$.

Note that the counter systems in the Fast toolkit are actually *abstractions* of the underlying protocols or mechanisms modeled by them. Therefore, in some

			Under-Approximation				Over Approximation		
Sys	#counters	#transitions	RT(ms)	FL	NFE	Term	RT (ms)	NI	Term
syn	3	3	12	1	1	yes	20	2	yes
moe	5	4	18	1	1	yes	23	2	yes
ill	4	9	120	1	9	yes	140	3	yes
mes	4	4	101	2	19	yes	135	3	yes
cen	12	8	4985	3	96	yes	1600	4	yes
tic	6	6	(TO)	12	1055943	no	480	5	yes
lift	4	5	(TO)	16	1481555	no	720	3	yes
efm	6	5	(TO)	14	1117737	no	1200	3	yes
rea	12	9	(TO)	4	1702	no	9520	7	yes
con	11	8	(TO)	3	184	no	132700	5	yes

Fig. 6. Experimental results for both algorithms. Sys - System name (short), RT - running time in milliseconds, (TO)-*timed out*, FL - max. length of flattenings explored, NFE - number of flattenings explored, Term - termination of algorithm, NI - number of iterations.

cases, it is possible that a temporal property that we identified holds in the actual protocol or mechanism at all reachable states, but does not hold in the abstraction.

Since our implementation targets only **EG** sub-properties, in the rest of this section we restrict our attention to the **EG** sub-property inside each of the CTL properties that we identified. We call each of the 19 system-property pairs under consideration an *input pair*. We provide details of each input pair, such as an English-language description of the system and a specification of the corresponding CTL property in an associated technical report [13].

Results. We ran both our algorithms on the 19 input pairs, with a uniform 1-hour timeout. Algorithm *computeGlobalOver* terminated on 10 pairs, while algorithm *computeGlobalUnder* terminated on 5 of these 10 pairs. Neither algorithm terminated on the remaining 9 pairs within 1 hour.

We summarize the results on which at least one of our algorithms terminated in Figure 6. Each row in the table corresponds to results from both algorithms on an input pair. The first column in this table is the name of a system, shortened to its first three letters (the first one, “syn”, is the MSI system). The next two columns give information about the system. Columns $4-7$ of the table correspond to results from algorithm *computeGlobalUnder*, while columns $8-10$ correspond to results from algorithm *computeGlobalOver*. The meanings of these columns have been explained in the caption of the figure.

Discussion. The first five rows in the table in Figure 6 describe input pairs on which both our algorithms terminated. We observe that the algorithm *computeGlobalOver* takes more time than the algorithm *computeGlobalUnder* for smaller systems. This is mainly because of the large number of *pre** queries issued by algorithm *computeGlobalOver*. But for system centralserver, shown in the fifth row of the table, *computeGlobalUnder* takes a longer time. This is because it has to explore 96 flattenings; this involves a large number of pre^k and pre^* queries to the reachability engine when compared to the number of queries posed by algorithm *computeGlobalOver* in 4 iterations.

Rows 5-10 in the table are about systems on which only the over-approximation approach terminated within the time-out. There are multiple possible reasons for this, such as the set X not becoming precise within the time out, or the set becoming precise but the termination condition not becoming true. Due to the large sizes of the systems it was not possible for us to manually determine whether the algorithm would eventually terminate on the input pairs in these rows. Also, due to the large sizes of the computed formulas, we could not determine how "close" the approximate solutions were to the respective precise solutions when the timeout happened.

The 9 input pairs on which neither of our algorithms terminated within the time-out are not discussed in Figure 6. These pairs are from the following systems: ttp2, swimmingpool, dragon, futurbus (two properties), firefly (two properties), csm, and train. One reason for non-termination of both algorithms is the large size of some of these counter systems (e.g., ttp2 has 9 counters and 17 transitions), causing individual reachability queries to take more time, and also more iterations to be required by the algorithms. In fact, for ttp and swimmingpool systems, the approximations computed by our under-approximation algorithm were continuing to improve even after one hour. Another possible reason of non-termination of the two algorithms is the worst-case scenario wherein none of the reachable states of the system satisfy the given **EG** property. Both of our algorithms are more likely to take a long time or go into non-termination in this scenario. Again, due to the size and complexity of the systems, we were not able to determine manually for any of these 9 input pairs whether the scenario mentioned above held, and whether our algorithms would have eventually terminated if given more time. We observe empirically that the over-approximation algorithm terminates on a superset of inputs as the under-approximation algorithm. However, as mentioned in Section 4.2, we do not have a theoretical proof that this holds in general. However, both algorithms are useful per-se, because there are inputs where neither of them terminates. For instance, if one wishes to conservatively under-approximate the set of states that are live with respect to some "good" property, they would need an over-approximation of the property $\mathbf{EG}\phi$, where ϕ is the negation of the "good" property. However, if one wishes to check conservatively whether all states that satisfy property ϕ_1 also satisfy a property $\mathbf{EG}\phi_2$, they would need to check if ϕ_1 implies an under-approximation of $\mathbf{EG}\phi_2$. In summary, our empirical results show the value of our techniques in the context of analyzing natural **EG** properties of real pre-existing system models. The over-approximation approach terminated on 10 out of 19 input pairs; both algorithms take reasonable time (from a fraction of a second to a few seconds) on the vast majority of inputs on which they did not hit the 1-hour timeout. They provide approximate results upon timeout. In comparison with pre-existing approaches [10, 11] we are the first to report empirical evidence on real examples using an implementation.

7 Related Work

Research work on model-checking *CTL* properties in counter systems has progressed along side the developments in techniques to answer reachability on these systems. The approach of Bultan et al. [14] is an early approach; it does not use accelerations [4, 6–8] to traverse sequences of concrete transitions at one go, and is subsumed by subsequently proposed approaches [10, 11] that do use accelerations. These approaches both build a summary of all possible traces in the given counter system using accelerations. This summary is then checked against the given temporal property. The two key technical differences of our approach over these are: (a) Rather than attempting to summarize *all* the traces in the system, we use refinement and then accelerations to characterize only the traces that satisfy the *given property.* (b) We use *repeated* reachability queries, and not a single phase of applying accelerations. The consequences of these differences are as follows. Due to features (a) and (b) above, as discussed in Sections 3.2 and 4.2, we target systems beyond trace flattable systems, and terminate with precise results on a wider class of systems than the approach of Demri et al. [10]. The practical importance of this is borne out by our empirical studies. Feature (a) also enables us to solve arbitrarily nested CTL properties, while feature (b) enables us to compute approximated solutions in cases where a precise computation may not be possible, which is very useful in practice. The previous approaches do not possess these advantages. Finally, the previous approaches did not provide empirical results using implementations.

There are a few other noteworthy points about the previous approaches mentioned above. The approach of Bozelli et al. [11] does not have the finite-branching restriction. Also, although neither previous approach addresses arbitrarily nested CTL properties, they address certain operators of CTL* that we do not address.

Cook et al. [15, 16] proposed a technique to model check arbitrarily nested temporal properties in a restricted class of C programs. The major difference is that we address the "global" model-checking problem, wherein we return a formula that encodes *all* states that satisfy a property. In their case they check whether a *given* set of states satisfies a property. Also, they do not have capabilities for approximations. Nevertheless, an interesting investigation for future work would be to compare the classes of systems targeted by them and by us.

Acknowledgments. We thank A. Finkel and J. Leroux for their suggestions and for their help with the Fast tool. We also acknowledge Indian Space Reseach Organisation (ISRO) for providing partial financial support for this work.

References

1. FASTer, `http://altarica.labri.fr/forge/projects/faster/wiki/`
2. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Application to model-checking. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 135–150. Springer, Heidelberg (1997)
3. Esparza, J.: Decidability and complexity of petri net problems– An introduction. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 374–428. Springer, Heidelberg (1998)
4. Comon, H., Jurski, Y.: Multiple counters automata, safety analysis and presburger arithmetic. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 268–279. Springer, Heidelberg (1998)
5. Ibarra, O.H., Su, J., Dang, Z., Bultan, T., Kemmerer, R.A.: Counter machines and verification problems. Theoret. Comp. Sc. 289(1), 165–189 (2002)
6. Finkel, A., Leroux, J.: How to compose presburger-accelerations: Applications to broadcast protocols. Technical Report, Labor. Specif. et Verif. LSV (2002)
7. Darlot, C., Finkel, A., Van Begin, L.: About fast and trex accelerations. Electronic Notes in Theoretical Computer Science 128, 87–103 (2005)
8. Bozga, M., Gîrlea, C., Iosif, R.: Iterating octagons. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 337–351. Springer, Heidelberg (2009), doi:10.1007/978-3-642-00768-2_29
9. Clarke, J.E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press (1999)
10. Demri, S., Finkel, A., Goranko, V., van Drimmelen, G.: Towards a model-checker for counter systems. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218, pp. 493–507. Springer, Heidelberg (2006)
11. Bozzelli, L., Pinchinat, S.: Verification of gap-order constraint abstractions of counter systems. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI 2012. LNCS, vol. 7148, pp. 88–103. Springer, Heidelberg (2012)
12. Alpern, B., Schneider, F.B.: Defining liveness. Information Processing Letters 21(4), 181–185 (1985)
13. Lakshmi, K.V., Acharya, A., Komondoor, R.: Checking temporal properties of presburger counter systems using reachability analysis. CoRR (2013), `http://arxiv.org/abs/1312.1070`
14. Bultan, T., Gerber, R., Pugh, W.: Symbolic model checking of infinite state programs using presburger arithmetic. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 400–411. Springer, Heidelberg (1996)
15. Cook, B., Koskinen, E., Vardi, M.: Temporal property verification as a program analysis task. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 333–348. Springer, Heidelberg (2011)
16. Cook, B., Koskinen, E.: Reasoning about nondeterminism in programs. In: Proc. Conf. on Progr. Lang. Design and Impl. (PLDI), pp. 219–230 (2013)