Model-Checking in Presburger Counter Systems using Accelerations

A Thesis

Submitted For the Degree of

Master of Science (Engineering)

in the Computer Science and Engineering

by

Aravind Acharya N

Computer Science and Automation

Indian Institute of Science

BANGALORE – 560 012

August 2013

TO

My Parents

and

His Ingenuity

Acknowledgements

First of all, I would like to convey my deepest gratitude to my adviser Dr. K. V. Raghavan for his continuous guidance and persistent help without which this dissertation would not have been possible. Working with him was a pleasure and it taught me lot of things of which the most important is the essence of research. He has always been patient in listening to new ideas, and discussions with him has frequently led to key insights that contributed immensely to this thesis. I would like to thank Dr. Aditya Kanade for an excellent course on Automated Verification which gave me the necessary basics and the enthusiasm to work in model-checking and verification aspects. I also thank both Dr. K. V. Raghavan and Dr. Deepak D'Souza for offering a course on Program analysis and Verification. I am greatly thankful to K Vasanta Lakshmi in helping me understand the problem statement and sharing her valuable experience of working on this problem.

I would like to thank Dr Alien Finkel, Dr Stephane Demri, Dr Jerome Leurox, Dr Narayan Kumar and Gerald Point for timely clarification of doubts, invaluable suggestions and feedback. I would like to take this opportunity to thank all my friends, lab mates especially Anirudh Santhiar and Pranavadutta Devaki for the all the technical discussions that helped me to understand the subject better.

I extend my deepest gratitude to the chairman of the Department, Dr Y Narahari, for his constant support and motivations which enabled me to work harder. I would like to thank Mrs Lalitha, Mrs Suguna and Mrs Meenakshi for taking care of the administrative tasks smoothly. I would also like to thank all the non-technical staff of the department for making my stay in the department comfortable and unforgettable. I am greatly indebted to IISc, Bangalore and Department of Computer Science and Automation for

all the facilities it provides to the students.

I would also like to thank N R Prashanth, who motivated me to pursue higher education. I am also grateful to my friends Amogh, Jay, Vinayak, Prasanna, Ninad, Roshan, Thejas, Chandan, Irshad, Lakshmi, Apoorva, Ranjani, Pallavi, Malavika and Prachi for ensuring a very joyful stay at IISc. I would also thank my tennis partners Aniruddha, Aditya, Varun, Chandrashekar, Nitin, Shandilya, Prashanth, Prof. Ghosh and Prof. Narasimhan for providing a memorable time at the tennis court, Gymkhana, IISc. I would also like to thank my parents and my brother who have stood by me during the times of joy and sorrow.

Finally, I want to thank the Ministry of Human Resource Development, Department of Education, UGC-CAS and ISRO-IISc funding program for granting scholarship to support me during this course.

Abstract

Model checking is a powerful technique for analyzing reachability and temporal properties of finite state systems. Model-checking of finite state systems has been well-studied and there are well known efficient algorithms for this problem. However these algorithms may not terminate when applied directly to infinite state systems. Counter systems are a class of infinite state systems where the domain of counter values is possibly infinite. Many practical systems like cache coherence protocols, broadcast protocols etc, can naturally be modeled as counter systems. In this thesis we identify a class of counter systems, and propose a new technique to check whether a system from this class satisfies a given *CTL* formula. The key novelty of our approach is a way to use existing reachability analysis techniques to answer both "until" and "global" properties; also our technique for "global" properties is different from previous techniques that work on other classes of counter systems, as well as other classes of infinite state systems. We also provide some results by applying our approach to several natural examples, which illustrates the scope of our approach.

Contents

List of Tables

List of Figures

Keywords

Model-Checking, Counter Systems, Accelerations, Presburger Arithmetic, Computational Tree Logic(CTL)

Chapter 1

Introduction

Infinite state systems include programs, as well as other automata theoretic structures like petri-nets, pushdown systems and counter systems. An aspect of verification of these systems is to determine the set of reachable states starting from the initial states (i.e., *reachability*), on which there is extensive literature. For instance, reachability analysis in programs, although undecidable in general, has been addressed using a variety of approximation algorithms, as surveyed by Jhala et al. [1]. In the space of automata theoretic structures, reachability in infinite-state systems like pushdown-systems and petri-nets is decidable [2, 3].

Model-checking of *temporal properties* is another aspect of verification. Verification of *CTL* [4] properties for pushdown systems is decidable [2, 5]. This result has been extended for analysis of other kinds of structures, e.g., in the context of analysis of inter-procedural programs [6, 7]. *Counter systems* are a class of infinite state systems that are equivalent to simple looping programs without arrays, pointers and dynamic memory allocation. Counter systems are a richer class of infinite state systems than pushdown systems. Therefore, the algorithms to model-check temporal properties in pushdown systems cannot be extended in a straightforward way to answer temporal properties in counter systems. Counter systems have a finite set of *control states* and finite set of *counters*, with each counter taking values from the infinite domain of integers. Counter systems have transitions between control states. Each transition has a *guard* and an

action. Typically the guards are formulas on the counters and actions are formulas on the primed and unprimed versions of the counters where the primed versions represents the values of the counters after the transition. The class of counter systems where the guards and actions of transitions are given by Presburger formulas are called Presburger counter systems. Various subclasses of Presburger counter systems have been proposed in the literature [8, 9, 10, 11], with the focus predominantly being on reachability analysis. Presburger counter systems have been shown to be applicable in settings such as the analysis of the TTP protocol, broadcast protocols, cache coherence protocols, etc. in [12]. *CTL* properties involve the use of *next*, *until* and *global* operators, that are universally or existentially quantified over paths of the counter system. The basic propositions in the properties that we consider are Presburger formulas (i.e., predicates) over the counter values of the counter system. We use $\psi, \psi_1, \psi_2 \ldots$ to represent temporal properties and $\phi, \phi_1, \phi_2, \ldots$ to represent Presburger formulas over counter variables.

1.1 Our approach

The focus of this thesis is on *global* model-checking a given *CTL* property in a given Presburger counter system (which we will simply refer to as "counter system" from hereon). That is, we find the set of states that satisfy the given temporal property in the counter system. A state in a counter system is a vector which contains the control state and the values of the counters. We give a generic algorithm for this problem, which is structurally similar to the standard *CTL* model checking algorithm for finite state Kripke structures [4]. The algorithm computes inductively, the set of states that satisfy each sub-property of the given temporal property. In our approach, we use Presburger formulas whose free variables are the names of the counters, to represent the set of states that satisfy each sub-property (and finally the full *CTL* property). In some cases we either under or over-approximate the set of states that satisfy a sub-property. The approximation of the sub-property changes the precision of the solution of the properties at the higher level. Every approximation of a sub-property results in the same direction

of approximation of properties at higher levels except for the negation operator which inverts the direction of approximation. In cases where we approximate we indicate the direction of approximation of computed set of states that satisfy the given temporal property.

Our Algorithm for model-checking *CTL* properties in Counter Systems:

Model-checking of *CTL* properties include finding the set of states that satisfy *next*, *until* and *global* properties. The *next* state properties are easy to compute. Given a counter system M and a *CTL* property $\mathbf{EX}\psi_1$, we first inductively solve ψ_1 and get the set of states ϕ_1 that satisfy ψ_1. The set of immediate predecessors of ϕ_1 represented by $pre(M, \phi_1)$, can be computed easily. This is because the immediate successor states are defined by the transitions of the system.

Model-checking *until* and *global* properties involve fix point computations and is non-trivial. These computations may not always terminate but there are classes of counter systems where these computations can be done precisely.

Until Properties:

Given a counter system M and a CTL property $\mathbf{E}(\psi_1\ \mathbf{U}\ \psi_2)$ we find the set of states that satisfy $\mathbf{E}(\psi_1\ \mathbf{U}\ \psi_2)$ as follows:

- Inductively solve ψ_1 and ψ_2 yielding ϕ_1 and ϕ_2 as the set of states that satisfy ψ_1 and ψ_2 respectively.

- We then conjunct the guards of the transitions of M with ϕ_1 to get a new system M_1. We call M_1 as the refinement of M with respect to ϕ_1. Any state in M_1 from which any transition can be fired must satisfy ψ_1.

- All states from which a state in ϕ_2 can be reached in M_1 will satisfy the property $\mathbf{E}(\psi_1\ \mathbf{U}\ \psi_2)$. Existing reachability analysis techniques provide the routine $pre^*(M_1, \phi_2)$, that returns the set of predecessors of ϕ_2 in the counter system M_1.

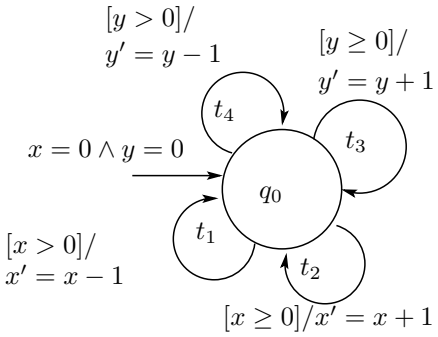

Figure 1.1: A counter system M

We use this routine in our algorithm as a black-box. The formula returned precisely represents the set of states that satisfy $\mathbf{E}(\psi_1\ \mathbf{U}\ \psi_2)$ in the machine M.

For example, consider the counter system shown in Figure 1.1. The formula shown in "[ ]" for each transition represents the guard of the transition and the formula after the "/" represents the action of the transition. Consider the property $\mathbf{E}((x \geq 10)\ \mathbf{U}\ (x = 10))$. First the system is refined with $x \geq 10$, meaning the guards of each transition are strengthened with $x \geq 10$. This is shown in Figure 1.2. Now we compute the set of states from which we can reach $x = 10$ in this refined system, using a reachability analysis technique as the black-box. The black-box returns $x \geq 10$ as the solution, which precisely represents the set of states that satisfy $\mathbf{E}(x \geq 10\ \mathbf{U}\ x = 10)$ in the system shown in Figure 1.1.

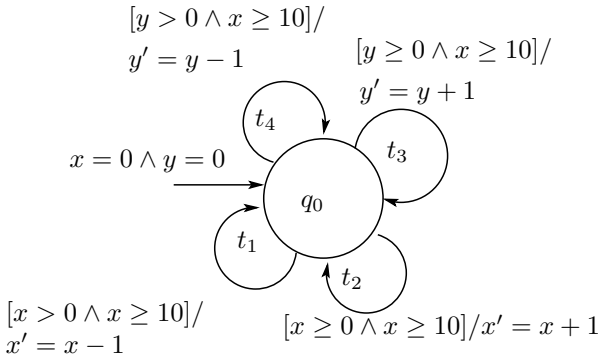

Figure 1.2: Refinement of counter system M in Figure 1.1 with $x \geq 10$

As noted above, our approach uses reachability analysis techniques as a black-box subroutine. These reachability analysis techniques typically use *accelerations* to reason about states that may be visited within loops, and hence help in termination of fix point computations. Our approach, at a high-level, resembles the idea of Bouajjani et al. [2] for answering "next" and "until" properties in pushdown systems. However, they have a built-in reachability analysis, and do not use a provided black box for this. There are different classes of counter systems on which reachability is decidable, each having their own reachability analysis routine. Hence using reachability analysis as a black box makes our approach more generic and applicable to more classes of counter systems.

Global Properties:

We handle "global" (i.e., **EG**) properties in a very different way than Bouajjani et al., because in counter systems these properties cannot be solved using simple reachability queries. Given a counter system M and a temporal property $\mathbf{EG}\psi$, we propose two different techniques to answer these global properties.

A naive computation:

The first technique we propose is guaranteed to terminate in cases where the underlying reachability black-box terminates. But in general it may under-approximate the set of states that satisfy the property $\mathbf{EG}\psi$. It uses the underlying reachability analysis black-box to compute the set of states that satisfy $\mathbf{EG}\psi$ as follows

- Compute inductively the set of states that satisfy ψ. Let ϕ_1 be the set of states that satisfy ψ.
- Refine the system M with respect to to ϕ_1 to get a system M_1.
- Every state from which there is an outgoing transition in M_1 satisfies ϕ_1 because the guard of each transition in M is conjuncted with ϕ_1. Hence the set of states from which there are paths of infinite length in M_1 must satisfy $\mathbf{EG}\psi$.

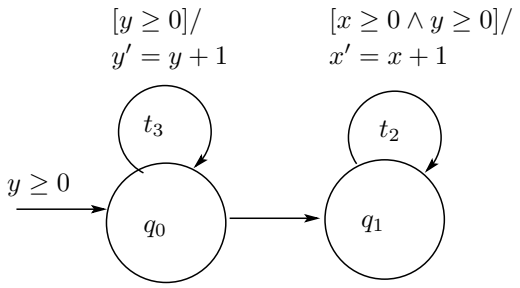

Figure 1.3: Counter system N, which is a flattening of M shown in Figure 1.1.

- Generate a system N which has no nested cycles among its control states. Such systems with no nested cycles are called *flat* systems. The reachability analysis tools generate such flat systems of a given system by unrolling the loops. The system N is called a flattening of the system M_1.
- In flat systems the set of that satisfy $\mathbf{EG}\phi$ states can be computed with existing techniques [13] [1].

For example consider model-checking the property $\mathbf{EG}(y \geq 0)$ on the counter system shown in Figure 1.1. Generate a flat system N from M, as shown in Figure 1.3. We then refine this new flat system N with respect to $y \geq 0$. We compute the set of states that have infinitely long paths in this system. The algorithm returns $y \geq 0$ as the solution.

Though the result returned by the algorithm was precise in the previous case, it will, in general, under-approximate the set of states that satisfy the property $\mathbf{EG}\psi$. This is because flattening the system might remove some transitions and hence traces (sequences of states) in the system.

Approach 2:

The previous approach, in general under-approximated the set of states that satisfy $\mathbf{EG}\psi$. This is because it generated a flattening and computed the set of states that satisfy $\mathbf{EG}\psi$ in the flattening and returned it as the solution. The approach did not check

[1] For any class of flat systems for which a terminating black-box is available, model-checking CTL has shown to be decidable. Hence all our contributions are towards addressing non-flat systems.

whether the computed set of states was the precise solution. In our second approach we iteratively compute the set of states that satisfy the property $\mathbf{EG}\psi$ by generating multiple flattenings. For each one compute the set of states that have infinite length paths along which ψ holds and then check whether the computed set of states precisely represent the set of states that satisfy $\mathbf{EG}\psi$. We iteratively produce different flattenings till we produce the one which $\mathbf{EG}\psi$ can be computed precisely as follows:

1. Inductively compute the set of states ϕ_1 that satisfy ψ.
2. Refine the system M with ϕ_1 to generate the new system M_1.
3. Generate a flattening N of M_1 and compute the set of states X that have infinite paths in N.
4. Check whether every trace from the remaining states, i.e, $\phi_1 - X$, that is present in M_1 is also present in N. Checking whether the traces from a given set of states are preserved in the flattening is decidable as shown by Demri et. al [13].
5. If *true* then return X as the precise solution.
6. Else go back Step 3.

Note that the approach may not always terminate, but whenever it terminates it returns the precise set of states that satisfy $\mathbf{EG}\psi$. We can also terminate the algorithm prematurely which will give an under-approximation of the set of states that satisfy the property.

For example consider model-checking the property $\mathbf{EG}(y \geq 0)$ on the counter system shown in Figure 1.1. The set of states that satisfy $y \geq 0$ is given by the formula $\phi_1 \equiv y \geq 0$ which is computed inductively. Now the system M is refined with respect to ϕ_1. We generate a flat system N, from this refined system as shown in Figure 1.4. We compute the set of states X that satisfy $\mathbf{EG}(y \geq 0)$ in N. The formula $y \geq 0$ precisely represents the set of states that satisfy the above property in N which is computed by the algorithm. algorithm returns X as the solution. $y \geq 0$ precisely represents the set

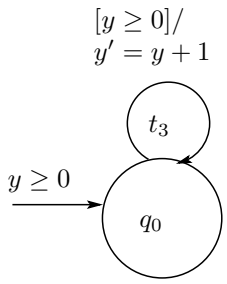

Figure 1.4: Another flattening N of the counter system M shown in Figure 1.1

of states that satisfy $\mathbf{EG}(y \geq 0)$ in the counter system M. The iterative nature of the algorithm and procedure to check whether the computed set of states is precise, which is the main difference of this approach when compared to the previous one is discussed in detail in Chapter 5.

1.2 Contributions

In this section we briefly mention our contributions and discuss them in detail in Chapter 8.

Technical novelty:

- To our knowledge we are the first to propose an algorithm for *CTL* model checking in counter systems that is structurally similar to the standard (finite-state) *CTL* model-checking algorithm, in that it inductively solves sub-properties of the given temporal property.

- We are the first to use the idea of refinement in counter systems which enables us to answer temporal properties in a larger class of counter systems.

- We use reachability analysis tools that use accelerations as a black-box which makes our algorithm generic and applicable on different classes of counter systems for which accelration black boxes are available. We also have an assumption that the systems that we consider are *finite branching*, that is, a given state will have a finite number of successor states.

- We also incorporate approximations to our algorithm. In this thesis we focus on under-approximation routines for answering "global" and "until" properties. We also emit the direction of approximation of the set of states that satisfy a temporal property ψ in case we approximate. Note that in this thesis we only give the routines that can under-approximate the *global* properties. Routines that over-approximate the set of states that satisfy the global properties are given by Lakshmi et al [14]. The over-approximation routine stated in [14] is not the contribution of the author of this thesis.

Correctness, Precision and Termination: Our algorithm uses different routines for answering until and global properties. The routine for until property terminates whenever the underlying reachability black-box terminates. We give two different approaches for answering global properties. The first approach terminates in all cases but in general, under-approximates the set of states that satisfy the property. The second approach precisely computes the set of states that satisfy the property if it terminates. Otherwise it produces an under-approximation of the set of states that satisfy the property.

Usefulness: We identify different natural temporal properties on several systems [12] and we show that our approaches terminates on these systems. These systems are not handled by the existing techniques.

The rest of the thesis is organized as follows: In Chapter 2 we give the basic terminology, followed by our generic algorithm in Chapter 3. In Chapter 4 we describe our core technique to answer *until* properties using existing reachability analysis techniques. We discuss our approaches to handle *global* properties in Chapter 5 along with stating some theorems that prove the correctness and precision of our approaches. In Chapter 6 we describe our experience with several realistic examples. We discuss the existing approaches in literature in Chapter 7. We discuss the key aspects of our work in Chapter 8 and finally conclude in Chapter 9.

Chapter 2

Notation and Terminology

In this chapter, we formally define counter systems, *CTL* temporal properties and their semantics along with the terminology that we use.

Definition 1 (Presburger Logic)**.** *Consider a finite set of variables X. A Presburger formula over X is defined by the following grammar*

$$\phi ::= t < t \mid t = t \mid \neg\phi \mid \phi \vee \phi \mid \exists x.\phi \mid \forall x.\phi \mid \text{true}$$
$$t ::= 0 \mid 1 \mid y \mid t + t, \text{ where } y \in X \text{ is a variable}$$

Presburger formulas are interpreted over natural numbers with $+, <, =, 0, 1$ have their usual meaning. These formulas have the same semantics as formulas in standard first order logic. Presburger arithmetic is a decidable theory; i.e, there exists an algorithm which says whether a given statement in Presburger arithmetic is true or not. Presburger arithmetic does not involve multiplication, although multiplication by constants can be incorporated by repeated addition.

Definition 2 (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 represented as a vector, 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 $G(b) = g_b$ and an $F(b) = f_b$ that are the guard and action of the transition b, respectively.*

The counter system shown in Figure 1.1 is given by the following tuple $M = \langle \{q_0\}, \{x, y\}, \{t_1, t_2, t_3, t_4\}, (x = 0 \wedge y = 0), \{g_{t_1}, g_{t_2}, g_{t_3}, g_{t_4}\}, \{f_{t_1}, f_{t_2}, f_{t_3}, f_{t_4}\} \rangle$ where $q_0 = 0$ which represents the control state, the guards $g_{t_1} \equiv x > 0$, $g_{t_2} \equiv x \geq 0$ which respectively represent the positive and non-negative values of the counter x, $g_{t_3} \equiv y \geq 0$ and $g_{t_4} \equiv y > 0$ which respectively represents the non-negative and positive values of the counter y and the actions $f_{t_2} \equiv x' = x + 1$, $f_{t_1} \equiv x' = x - 1$ increment and decrement the value of counter x respectively without changing the value of counter y, $f_{t_3} \equiv y' = y + 1$ and $f_{t_4} \equiv y' = y - 1$ increment and decrement the values of the counter y respectively without affecting the values of counter x.

A *state* (denoted by $\boldsymbol{s}$, $\boldsymbol{s'}$ etc.) in a system is a column vector $\boldsymbol{v} \in \mathbb{I}^{m+1}$. The first element v_0 represents the control state and the values of rest of the elements $v_1, \ldots, v_n$ represent the values of the counters C. We sometimes use the term *concrete state* to refer to a state. For example $\{0, 0, 0\}$ represents the state where the control state is encoded by the number 0 and the counters x and y have the value 0.

In our setting we use Presburger formulas of two kinds. The first one uses the counter variables and the control state variable q, as free variables. In general such formula represents a set of the states in the system. We use these formulas

- to represent the initial states ϕ^{init},
- as guards of transitions in the system
- as basic propositions of temporal properties and
- to represent the return value of our algorithm i.e, the set of states that satisfy any sub-property of the the given temporal property.

In these formulas we use the variable name q whose value encodes the control state. Note that in a formula that is used as the guard of a transition, the control state variable is typically constrained to be equal to the source control state of the transition. The systems that we show in this thesis have a single control state and whenever we do not show the variable q in the formula, we mean that its value is constrained to be equal to

0 (which is the number corresponding to the control state). Therefore, for convenience, we drop this constraint from the everywhere in this thesis even in the formulas that we return. For example, the guard of transition t_1 in Figure 1.1 is $x \geq 0$

The second kind of Presburger formulas is used specifically to represent the actions of transitions. The actions are Presburger formulas on primed and unprimed versions of the counter variables, where unprimed and primed versions of the variables represent the values before and after the transition respectively. Note that the action of a transition a implicitly sets the new control state to the destination of a; hence, we omit q from the formulas. In the actions we do not show the values of the counters that remain unaffected. For example, the action of transition t_1 in Figure 1.1 is $x' = x - 1$. It indicates that the value of the counter x will be decremented after the transition and the value of y will remain unchanged.

We now turn into the semantics of counter machines. Informally a transition a in a counter system M can be taken in a state $\boldsymbol{s}$ only if it satisfies the guard g_a of a. The values of the counters are modified according to the action f_a of a. Hence a transition in general defines a *partial function* on the set of states induced by M. We use the notation $\boldsymbol{s} \xrightarrow{a} \boldsymbol{s'}$ to represent a transition from $\boldsymbol{s}$ to $\boldsymbol{s'}$ on $a \in \Sigma$. We say $\boldsymbol{s'}$ to be the immediate successor of $\boldsymbol{s}$ and $\boldsymbol{s}$ to be the immediate predecessor of $\boldsymbol{s'}$.

The systems that we consider may be *non-deterministic*, i.e, there could be two transitions $t_i, t_j \in \Sigma$ and a state $\boldsymbol{s}$ such that $\boldsymbol{s} \models g_{t_i}$ and $\boldsymbol{s} \models g_{t_j}$. For example, consider the state $\{0, 0, 0\}$ in Figure 1.1. It satisfies the guards of transitions t_2 and t_3 and hence the system is non-deterministic. We assume that the action f_a of a transition a may be non-deterministic, but is *finitely branching*. That is, any given state has a finite number of successor states under f_a. For example, we do not allow actions of the form $x' \geq 0$ in our counter system.

Throughout the thesis we use ϕ, ϕ_i, etc., to denote Presburger formulas with counter variables and the control state variable as free variables. In case we have other free variables, we denote them is subscripts. For example, we use $\phi_{(k)}$ to represent a Presburger formula with the k and the state variables as free variables. 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 corresponding elements in $\boldsymbol{s}$.

We say that a counter machine $M_1 = \langle Q_1, C_1, \Sigma_1, \phi_1^{init}, G_1, F_1 \rangle$ is a *refinement* of machine $M = \langle Q, C, \Sigma, \phi^{init}, G, F \rangle$ with respect to Presburger formula ϕ if and only if $Q_1 = Q$, $C_1 = C$, $\Sigma_1 = \Sigma$, $\phi_1^{init} = \phi^{init}$ and for each $a \in \Sigma$, $g_{1_a} = g_a \wedge \phi$ and $f_{1_a} = f_a$. The function *refineSystem*(M, ϕ) refines the counter system M and returns a counter system M_1 which is a refinement of M with respect to ϕ. For example the counter system shown in Figure 1.2 is a refinement of the counter system shown in Figure 1.1 with respect to $x \geq 10$.

Given a counter system M and Presburger formula ϕ, representing the set of states, we define the successor states of ϕ in M as

$$post(M, \phi) \stackrel{def}{=} \{\boldsymbol{s} \mid \exists \boldsymbol{s_1} \in \mathbb{I}^{m+1}.\ \boldsymbol{s_1} \models \phi \wedge (\exists a \in \Sigma. \boldsymbol{s_1} \xrightarrow{a} \boldsymbol{s})\}$$

Similarly we define the set of predecessor states of ϕ in M as follows

$$pre(M, \phi) \stackrel{def}{=} \{\boldsymbol{s} \mid \exists \boldsymbol{s_1} \in \mathbb{I}^{m+1}.\ \boldsymbol{s_1} \models \phi \wedge (\exists a \in \Sigma. \boldsymbol{s} \xrightarrow{a} \boldsymbol{s_1})\}$$

For example consider the set of states $(x = 0 \wedge y = 0)$ for the example shown in Figure 1.1. The set of immediate successors, $post(x = 0 \wedge y = 0)$, is the formula $(x = 0 \ \wedge \ y = 1) \vee (y = 0 \ \wedge \ x = 1)$. Similarly, $pre(x = 0 \wedge y = 0) \equiv (x = 0 \wedge y = 1) \vee (x = 1 \wedge y = 0)$. The definition of successors of a set of states ϕ can be extended to the k^{th} successors of ϕ as follows

$$post^k(M, \phi)_{(k)} \stackrel{def}{=} \{\boldsymbol{s} \mid \exists \boldsymbol{s_1} \ldots \boldsymbol{s_k}.\ \boldsymbol{s_1} \models \phi \bigwedge_{i=1}^{k} (\exists a \in \Sigma. \boldsymbol{s_i} \xrightarrow{a} \boldsymbol{s_{i+1}} \wedge \exists a \in \Sigma. \boldsymbol{s_k} \xrightarrow{a} \boldsymbol{s}\}$$

Note that the subscript (k) indicates that $post^k(M, \phi)_{(k)}$ is a formula in which C (which includes the counter that encodes the control state) and k occur as free variables. For example $post^k(M, x < 0 \wedge y = 0)_{(k)}$, where M is the counter system shown in Figure 1.1,

is given by the formula $\exists k_1, k_2 \geq 0.\ k = k_1 + k_2 \wedge y + (k_2 - k_1) = 0 \wedge k_2 \leq k_1 \wedge x < 0$. Similarly, $pre^k(M, \phi)_{(k)}$ is a formula in s and k that represents the set of states from which some state in ϕ can be reached in k steps.

The set of reachable states from a given set of states $\phi_{\boldsymbol{s}}$, is given by the formula

$$post^*(M, \phi) \stackrel{def}{=} \exists k \geq 0.\ post^k(M, \phi_{\boldsymbol{s}})_{(k)}$$

For example $post^*(M, x = 0)$, where M is the system shown in Figure 1.1 are the set of states given by the formula $\exists k \geq 0.\ x - k = 0$ which simplifies to $x \geq 0$. Similarly the backward reachability set, for a set of states ϕ, namely $pre^*(M, \phi)$, can be defined as follows

$$pre^*(M, \phi) \stackrel{def}{=} \exists k \geq 0.\ pre^k(M, \phi)_{(k)}$$

For example $pre^*(M, x = 0)$ for the system shown in Figure 1.1 are the set of states where $x \geq 0$ which is given by the formula $\exists k \geq 0.\ x + k = 0$. Given a system M we use the formula ϕ^{reach} to denote the set of states that are reachable from the set of initial states ϕ^{init}.

Temporal properties are used to formally specify properties on sequences of states (i.e, paths) in a transition system. *CTL* properties allow us to quantify over paths in a transition system. Following the idea of Demri et al.[13] we extend the *CTL* grammar to let Presburger formulas of the form ϕ, be the basic propositions in the temporal properties. Temporal properties in the *existential normal form* of CTL can be defined by the following grammar

$$\psi \ \equiv \ \phi \,|\, \neg\psi \,|\, \psi \vee \psi \,|\, \mathbf{EX}\psi \,|\, \mathbf{E}(\psi \ \mathbf{U} \ \psi) \,|\, \mathbf{EG}\psi \tag{2.1}$$

Throughout the thesis we use ψ, ψ_i, etc., to denote temporal properties.

A trace from a state $\boldsymbol{s_0}$ in M is a (possibly infinite) sequence of states $\boldsymbol{s_0}, \boldsymbol{s_1} \ldots$ such that $\forall i \geq 0$: $(\exists a_i \in \Sigma.\ \boldsymbol{s_i} \xrightarrow{a} \boldsymbol{s_{i+1}})$. A state $\boldsymbol{s_0}$ satisfies a temporal formula ψ, written as $\boldsymbol{s_0} \models \psi$, as per the following definition

- $\boldsymbol{s_0} \models \phi \iff \boldsymbol{s_0} \models \phi$
- $\boldsymbol{s_0} \models \neg\psi \iff \boldsymbol{s_0} \not\models \psi$
- $\boldsymbol{s_0} \models \psi_1 \vee \psi_2 \iff \boldsymbol{s_0} \models \psi_1$ or $\boldsymbol{s_0} \models \psi_2$
- $\boldsymbol{s_0} \models \mathbf{EX}\psi_1 \iff$ there exists a state $\boldsymbol{s_1}$ such that $\boldsymbol{s_1} \models \psi_1$ and $\boldsymbol{s_1}$ is a successor state of $\boldsymbol{s_0}$.
- $\boldsymbol{s_0} \models \mathbf{E}(\psi_1\ \mathbf{U}\ \psi_2) \iff$ there exists a trace $\boldsymbol{s_0}, \boldsymbol{s_1}, \ldots, \boldsymbol{s_k}$ in the system such that $\boldsymbol{s_k} \models \psi_2$ and $\forall i, 0 \leq i < k.\, \boldsymbol{s_i} \models \psi_1 \wedge k \geq 0$.
- $\boldsymbol{s_0} \models \mathbf{EG}\psi \iff$ there exists an infinite trace $\boldsymbol{s_0}, \boldsymbol{s_1}, \ldots$ in the system such that $\forall i \geq 0.\, \boldsymbol{s_i} \models \psi$.

Any *CTL* formula involving universal path quantifiers has an equivalent *CTL* formula in existential normal form. Therefore, we restrict ourselves to formulas in existential normal form.

The problem of *global* model-checking (which we simply refer as model-checking) is to find the set of states ϕ in the counter system M such that a state $\boldsymbol{s}$ satisfies the given temporal property ψ if and only if $\boldsymbol{s} \models \phi$. A more specific variant of this problem is the problem of *local* model-checking. Given a temporal property ψ and a set of states ϕ in the counter system M, the problem of local model-checking returns *true* if and only if every state $\boldsymbol{s}$ that satisfies ϕ also satisfies the temporal property ψ. In this thesis we focus on the problem of *global* model-checking a given CTL property in a given counter system.

Chapter 3

Generic Algorithm

Let $M = \langle Q, C, \Sigma, \phi^{init}, G, F \rangle$ be a given counter system and let ψ be a given temporal property in existential normal form. We assume that the reachable state space of the system M, represented by formula ϕ^{reach}, has been pre-computed, and that the guard of each transition in M has been refined (i.e., conjuncted) with this formula. In this thesis we say that the set of states that satisfy the temporal property ψ is computed precisely if the formula that we return precisely represents the set of reachable states of M that satisfy ψ. Hence the assumption of the guard of each transition being conjuncted with ϕ^{reach} is essential.

3.1 The main algorithm

Algorithm 1 takes the counter system M, the temporal property ψ and an enumerator *label* (we will discuss the significance of this argument later) as arguments, and returns a pair (ϕ, *approx*), where ϕ is a formula that represents the set of states of M that satisfy ψ, and *approx* is an enumeration whose value is "*precise*" if the returned formula is precise, "*under*" if it is an under-approximation, and "*over*" if it is an over-approximation of the set of states that satisfy ψ. The routines *computeUntil* and *computeGlobal* in the algorithm (Line 17 and Line 21) use a reachability analysis technique as a black box to model-check *until* and *global* properties, respectively. We describe these routines in

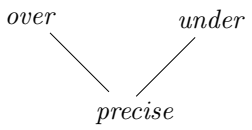

Figure 3.1: The Lattice *Approx*

detail in Sections 4 and 5; it is these routines that make the algorithm generic. There are many reachability analysis techniques for different classes of counter systems. Hence the algorithm can be applied to a variety of classes of counter systems which have finite branching and have a black box to compute the reachable states of the given counter system M. We treat the enumeration set $\{over, under, precise\}$ as a *meet semi-lattice*, which we call the lattice *Approx*, with both *over* and *under* dominating *precise* as shown in Figure 3.1. The input argument *label* and the returned enumerator *approx* are elements from this lattice.

The join operation on this lattice is denoted using "$\sqcup$" (with join of *over* and *under* being undefined); also, we define a negation operator "$\neg$" on this lattice, as follows: $\neg approx \equiv (approx = under)\ ?\ over\ :(approx = over)\ ?\ under\ :\ precise$.

The algorithm has a case structure, dependent on the root operator of ψ. If ψ is a basic proposition ϕ_i or any boolean combinations of temporal properties, then the set of reachable states that satisfy the formula ψ can be computed in a straightforward way. In case of negations we invert the enumerator *approx*. Model-checking "next" state properties involves a computing the immediate predecessors of a set of states(Line 13): to model-check $\mathbf{EX}\psi$ we first find the set of states ϕ_1 that satisfy ψ. $pre(M, \phi_1)$ gives the set of states that satisfies $\mathbf{EX}\psi$. $pre(M, \phi_1)$ can be computed easily because the transitions give the *pre-post* relation between the states of the counter system.

For until property $\mathbf{E}(\psi_1\ \mathbf{U}\ \psi_2)$, we first recursively find the set of states ϕ_1 and ϕ_2 that satisfy ψ_1 and ψ_2 respectively. Now our problem reduces to finding the set of states that satisfy the property $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$. This problem can not be solved using the normal standard *CTL* algorithm because of the following reason: The CTL algorithm maintains a set X which is initialized with set of states that satisfy ψ_2. X is iteratively populated

Algorithm 1 $SAT(M, \psi, label)$

Input: A counter system $M = \langle Q, C, \Sigma, \phi^{init}, G, F \rangle$ and a temporal property ψ and an enumerator *label* which indicates whether the set of states that satisfy ψ should be computed precisely or under-approximated or over-approximated.

Output: A pair $(\phi, approx)$, where ϕ is a Presburger formula representing the set of states that satisfy the formula ψ, and *approx* is one of the enum values from the lattice *Approx*, indicating whether ϕ is precise, under-approximated, or over-approximated.

1: **if** $\psi = \phi_i$ {ψ is a basic proposition.} **then**
2: **return** $(\phi^{reach} \wedge \phi_i, precise)$
3: **else if** $\psi = \neg\psi_1$ **then**
4: $(\phi_1, approx) \leftarrow SAT(M, \psi_1, \neg label)$
5: $approx_1 = \neg approx$
6: **return** $(\phi^{reach} \wedge \neg\phi_1, approx_1)$
7: **else if** $\psi = \psi_1 \vee \psi_2$ **then**
8: $(\phi_1, approx_1) \leftarrow SAT(M, \psi_1, label)$
9: $(\phi_2, approx_2) \leftarrow SAT(M, \psi_2, label)$
10: **return** $(\phi_1 \vee \phi_2, approx_1 \sqcup approx_2)$
11: **else if** $\psi = \mathbf{EX}\psi_1$ **then**
12: $(\phi_1, approx) \leftarrow SAT(M, \psi_1, label)$
13: **return** $(pre(M, \phi_1), approx)$
14: **else if** $\psi = \mathbf{E}(\psi_1\ \mathbf{U}\ \psi_2)$ **then**
15: $(\phi_1, approx_1) \leftarrow SAT(M, \psi_1, label)$
16: $(\phi_2, approx_2) \leftarrow SAT(M, \psi_2, label)$
17: $(\phi, approx_3) \leftarrow computeUntil(M, \phi_1, \phi_2, label)$
18: **return** $(\phi, approx_1 \sqcup approx_2 \sqcup approx_3)$
19: **else if** $\psi = \mathbf{EG}\psi_1$ **then**
20: $(\phi_1, approx_1) \leftarrow SAT(M, \psi_1, label)$
21: $(\phi, approx_2) \leftarrow computeGlobal(M, \phi, label)$
22: **return** $(\phi, approx_1 \sqcup approx_2)$
23: **end if**

with the set of immediate predecessors of X that satisfy ψ_1 till X reaches a fix-point. This fix-point computation need not terminate in general. Accelerations compute the set of reachable states of single loops in one step. Hence they help in computing a possibly infinite set of states in one go rather than multiple iterations and help in termination of until properties. The routine *computeUntil* computes the set of states that satisfy $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$ using reachability analysis tools, which use acceleration techniques, as black-box. It takes the the counter system M and a set of states ϕ as the input (the parameter *label* is discussed later) and returns the set of states ϕ and an enumerator *approx* indicating whether ϕ is precise or under-approximation or over-approximation of the set of states that satisfy $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$. Implementing this routine is discussed in detail in Section 4.

To model-check the formula $\psi \equiv \mathbf{EG}\psi_1$ (Line 19), we first compute the set of states ϕ_1 that satisfy ψ_1. We then call the routine *computeGlobal* which takes the the counter system M and a set of states ϕ (the parameter *label* is discussed later) as the arguments. The routine *computeGlobal* uses acceleration techniques to compute the set of states that satisfy $\mathbf{EG}\phi$ in the counter system M. It returns the set of states ϕ and an enumerator *approx* indicating whether ϕ is precise or under-approximation or over-approximation of the set of states that satisfy $\mathbf{EG}\phi$.

3.2 Ensuring sound approximations

Note that routines *computeUntil* and *computeGlobal* may, in general, under- or over-approximate the formulas they return. However, certain constraints are required on these behaviors, as otherwise the algorithm as a whole would not be able to soundly determine for each sub-property whether it is being solved precisely, or in an over- or under-approximated manner. For instance consider the property $\psi \equiv \psi_1 \vee \psi_2$; if one of ψ_1, ψ_2 is under-approximated *and* the other one is over-approximated, then we can say nothing about the approximation direction of ψ itself. In general, for the algorithm to return the direction of approximation of any temporal property ψ, it should not be

the case that one of the subproperties of ψ is under-approximated and the other is over-approximated. This can happen only when $\psi \equiv \psi_1 \vee \psi_2$ or $\psi \equiv \mathbf{E}(\psi_1 \ \mathbf{U} \ \psi_2)$. To ensure that Algoritm 1 returns the direction of approximation, we take an enumerator *label* as an input, which is an element from the lattice *Approx*. Algorithm 1 computes the direction of approximation of each each sub-property of ψ in the top-down phase. In the top down phase of Algorithm 1, every subproperty other than negation is assigned the same *label* as the root property. That is, if a property $\psi \equiv \psi_1 \vee \psi_2$ has to be under-approximated, the the algorithm enforces its sub-properties ψ_1 and ψ_2 to be under-approximated and thus ensuring that the join in lattice *Approx* exists (Line 10 Algorithm 1). In case of negation, the label is also negated with the "$\neg$" operator. The idea is that if *label* is *precise* in a call to Algorithm 1 then we *necessarily* need to compute a precise formula for ψ_i; if *label* is *over* (*under*) then we are free to compute either a precise or an over-approximated (under-approximated) formula for ψ_i. In the bottom up phase of the algorithm it computes the set of states that satisfies every sub-property of ψ in such a way that the returned set of states for each call, *respects* the computed direction of approximation of the particular sub-property.

In this thesis we provide a version of the routine *computeGlobal* that under-approximates the set of states that satisfy $\mathbf{EG}\phi$. The over-approximating version of the above routine is provided in [14] which is not the contribution of the author of this thesis.

The routines *computeUntil* and *computeGlobal* that we provide return the direction of approximation. This returned direction may not be the same as the one provided in the argument. For example, if *label* for the property $\mathbf{EG}\psi_1$ is *under*, the routine $computeGlobal(M, \phi_1, label)$ may return *precise* if the set of states that satisfy the formula $\mathbf{EG}\psi_1$ is computed precisely. Algorithm 1 would return *precise* as the direction of approximation. We assume that the enumerator *approx* returned by the routines *computeUntil* and *computeGlobal* is dominated by the enumerator *label*. In general the direction of approximation returned by Algorithm 1 will be $\sqsubseteq$ (dominated) by the enumerator in the argument.

Theorem 1. *Given a temporal property ψ and a counter system M, let the Algorithm*

$SAT(M, \psi, label)$ return the enumerator approx along with the formula ϕ that represents the set of states that satisfy the property ψ. Then approx $\sqsubseteq$ label and ϕ approximates the set of states that satisfy ψ in the direction given by the returned enumerator approx provided the routines computeGlobal and computeUntil are implemented correctly. Each routine is said to be correct if the enumerator and set of states returned by them is such that it approximates the precise set of states that satisfy the property given to the routine in the direction given by the enumerator returned by the routine.

Proof. We prove the above theorem by induction on height of the temporal property ψ. *Induction Hypothesis:* Let ψ be a formula of height k. Then $approx \sqsubseteq label$ and $\phi = SAT(M, \psi, label)$ approximates the set of states that satisfy ψ in the direction given by the returned enumerator *approx* provided the routines *computeGlobal* and *computeUntil* are implemented correctly.

Base case: ψ is a basic proposition ϕ_i. Then $SAT(M, \psi, label)$ returns $(\psi \equiv \phi_i \wedge \phi^{reach}, precise)$ which precisely represents the set of reachable states that satisfy ψ. Also the returned enumerator $precise \sqsubseteq label$ for any $label \in Approx$.

Induction Step: Let ψ be a formula of height $k+1$. Then we will prove the hypothesis case by case.

- If $\psi \equiv \mathbf{EX}\psi_1$ or $\psi \equiv \psi_1 \vee \psi_2$ then it is easy to see that the hypothesis holds.

- Let $\psi = \neg\psi_1$ and $(\phi_1, approx) = SAT(M, \psi_1, \neg label)$. It is easy to see that the set of states that satisfy ψ is given by the formula $\neg\phi_1 \wedge \phi^{reach}$. By Induction Hypothesis, $approx \sqsubseteq \neg label$. By the semantics of the negation operator in which was mentioned earlier in this section, $\neg approx \sqsubseteq label$.

- Let $\psi \equiv \mathbf{E}(\psi_1\ \mathbf{U}\ \psi_2)$, $(\phi_1, approx_1) = SAT(M, \psi_1, label)$ and $(\phi_1, approx_1) = SAT(M, \psi_2, label)$. Then by induction hypothesis $approx_1 \sqsubseteq label$ and $approx_2 \sqsubseteq label$. Since the routine *computeUntil* is correct, the enumerator $approx_3$ which was returned by *computeUntil* $approx_3 \sqsubseteq label$. Therefore $(approx_1 \sqcup approx_2 \sqcup approx_3) \sqsubseteq label$.

- Let $\psi \equiv \mathbf{EG}\psi_1$ and $(\phi_1, approx_1) = SAT(M, \psi_1, label)$. $approx_1 \sqsubseteq label$. Since the same enumerator *label* is passed as an argument to the routine *computeGlobal* and by the assumption that the routine *computeGlobal* is correct, the enumerator $approx_2$ returned by *computeGlobal* is such that $approx_2 \sqsubseteq label$. Therefore the returned enumerator $approx = approx_1 \sqcup approx_2$ is dominated by *label*.

□

Chapter 4

Until Properties

Given a counter system M and the set of states ϕ_1 and ϕ_2, in this section, we give an algorithm that computes the set of states that satisfy the property $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$. Note that Algorithm 1 requires the implementation of the routine $computeUntil(M, \phi_1, \phi_2, label)$, that computes the set of states that satisfy $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$. The inputs to the routine are the counter system M, Presburger formulas ϕ_1 and ϕ_2 and an enumerator *label* which is an element from the lattice *Approx* shown in Figure 3.1. The enumerator *label* indicates whether the set of states that satisfy $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$ have to be computed precisely or under-approximated. The routine returns a pair $(\phi, approx)$ where ϕ represents the set of states that satisfy the property $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$ in the counter system M and *approx* represents whether ϕ is precise or is an under-approximation of the set of states that satisfy $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$ in the counter system M.

4.1 Our Approach

Note that a state satisfies the property $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$ if it satisfies ϕ_2 or it satisfies ϕ_1 and it has a path to a state that satisfies ϕ_2 and all states along that path satisfy ϕ_1. To ensure the latter, we refine the system M with the formula ϕ_1, to obtain a refined system M_1. We use the function $refineSystem(M, \phi_1)$ for this, which basically strengthens the guard of each transition by conjuncting it with ϕ_1. Now any state from which a transition

can be fired in M_1 must satisfy ϕ_1. It is easy to see that the states such that there is a path in M_1 (of length ≥ 1) from the state to some state in ϕ_2 are exactly the ones that satisfy the property $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$ in M. Hence our problem reduces to finding the set of states from which we can reach ϕ_2 in the counter system M_1. There are many reachability analysis tools which provide the routines for the same.

Algorithm 2 gives the implementation of the routine *computeUntil*. The algorithm takes the system M and the set of states ϕ_1 and ϕ_2 (we will discuss the use of *label* later) and returns the set of states ϕ that satisfy $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$ in the counter system M along with the enumerator *approx* indicating whether ϕ is precise or under-approximation or over-approximation of the set of states that satisfy $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$. We first refine the input system M with respect to ϕ_1 (Line 1) to get the new system M_1. The routine *preStar* (Line 2) takes a counter system M_1 as the input and a set of states ϕ_2 and returns the set of states ϕ, that are predecessors of ϕ_2 in the machine M_1, along with an enumerator indicating whether ϕ under-approximates or precisely represents the set of states that are predecessors of ϕ_2 in the machine M_1. The routine *preStar* uses reachability analysis tools (discussed in Sec 5.1) as a black-box. These black-boxes provide the routine $pre^*(M_1, \phi_2)$. These black-boxes use accelerations to compute the set of states that are reachable by any number of iterations of a single loop in one step. They compute a growing under-approximation of the reachable set of states and when no new reachable states are found, they terminate and return the set of states ϕ that are predecessors of ϕ_2 in the counter system M_1. The routine *preStar* returns these set of states along with the enumerator *precise* indicating ϕ precisely represents the predecessors of the ϕ_2 in the counter system M_1.

The routine *preStar* returns precise results whenever it terminates and returns an under-approximation whenever termination is forced. Terminating the routine forcefully involves premature termination of the black-box which would have computed an under-approximation of the reachable states. Therefore, if *label* is *under*, we can choose to forcibly terminate the black-box which computes $pre^*(M_1, \phi_2)$ at any time, and make the routine *preStar* return its under-approximated formula along with the enumerator *under*.

Algorithm 2 *computeUntil*$(M, \phi_1, \phi_2, label)$

Input: A counter system M, set of states ϕ_1 and ϕ_2 and an enumerator indciating whether the formula $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$ has to be computed precisely or over-approximated or under-approximated.

Output: A set of states ϕ and the enumerator *approx* indicating whether ϕ is pricise or under-approximation or over-approximation of the set of states that satisfy $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$

1: $M_1 \leftarrow$ *refineSystem*(M, ϕ_1)

2: $(\phi, approx) \leftarrow$ *preStar*$(M, \phi_2, label)$

3: **return** $(\phi, approx)$

The returned formula represents an under-approximation of the set of states that satisfy the property $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$. We prove our precision results in Theorem 2 and Lemma 1 later in this section. If *label* is *over* then, we can run the routine *preStar*(M, ϕ_1, ϕ_2) till a predetermined timeout with the intention of computing a precise solution. If the routine does not terminate in the specified timeout, then we an return $(\phi_1 \vee \phi_2, over)$ as an over-approximation of the set of states that satisfy $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$. On the other hand, if *label* is *precise*, we have no choice but to wait for *preStar* to terminate.

Illustration

Consider the example system M shown in Figure 1.1 in Section 1, and the property $\mathbf{E}((x \geq 10)\ \mathbf{U}\ (x = 10))$. The recursive calls to process the two sub-properties of this property simply return the Presburger formulas $x \geq 10 \wedge y \geq 0$ and $x = 10 \wedge y \geq 0$, respectively. This is because any value of y less than 0 is not reachable in the system. We then refine the system M with $x \geq 10 \wedge y \geq 0$ to obtain a refined system M_1 (i.e., conjunct $x \geq 10 \wedge y \geq 0$ with the guards of all transitions), and compute $pre^*(M_1, (x = 10 \wedge y \geq 0))$. The routine *preStar* terminates and returns $(x \geq 10 \wedge y \geq 0, precise)$, which is precisely the set of reachable states that satisfy $\mathbf{E}((x \geq 10)\ \mathbf{U}\ (x = 0))$ in the system M.

4.2 Correctness of the routine *computeUntil*

In this section we prove that Algorithm 2 precisely returns the set of states that satisfy $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$ in the counter system M whenever the black-box *preStar* terminates. We also prove in Lemma 1 that whenever the black-box is terminated prematurely we get an under-approximation of the set of states that satisfy the property $\mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$.

Theorem 2. *Given a counter system M and a set of states ϕ_1 and ϕ_2, Algorithm 2 precisely returns the set of states that satisfy $\boldsymbol{E(\phi_1\ U\ \phi_2)}$ provided the routine preStar terminates.*

Proof. **Claim A:** If a state $\boldsymbol{s} \models \mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$ in the counter system M then $\boldsymbol{s} \in pre^*(M_1, \phi_2)$ where M_1 is the refinement of M with respect to ϕ_2.
Let $\psi \equiv \mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$ and let $\phi \equiv pre^*(M_1, \phi_2)$. Note that the guards of transitions in the refined system M_1 are conjuncted with ϕ_1 and during the pre-pass it is conjuncted with ϕ^{reach}, which defines the reachable set of states. Hence only reachable set of states that satisfy ϕ_1 satisfy the guards in the refined system M_1. Let $\boldsymbol{s} \models \psi$, and let $\boldsymbol{s}$ be reachable in the counter machine M. Let $g_a = G(a)$ and $f_a = F(a)$ be the guards and actions for a given transition a in the counter system M

$$
\begin{aligned}
& \boldsymbol{s} \models \mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2) \\
\Longrightarrow\ & \exists k \geq 0.\, \exists \boldsymbol{s_0}, \ldots, \boldsymbol{s_k}.\, \boldsymbol{s_k} \models \phi_2 \wedge (\forall i, 0 \leq i < k.\, \exists a \in \Sigma.\, \boldsymbol{s_i} \models g_a \wedge \\
& f_a \wedge \boldsymbol{s_i} \models \phi_1) \wedge \boldsymbol{s} = \boldsymbol{s_0} \\
\Longrightarrow\ & \exists k \geq 0.\, \exists \boldsymbol{s_0}, \ldots, \boldsymbol{s_k}.\, \boldsymbol{s_k} \models \phi_2 \wedge (\forall i, 0 \leq i < k.\, \exists a \in \Sigma.\, \boldsymbol{s_i} \models (g_a \wedge \phi_1) \wedge \\
& f_a) \wedge \boldsymbol{s} = \boldsymbol{s_0} \\
\Longrightarrow\ & \exists k \geq 0.\, \exists \boldsymbol{s_0}, \ldots, \boldsymbol{s_k}.\, \boldsymbol{s_k} \models \phi_2 \wedge (\forall i, 0 \leq i < k.\, \exists a_1 \in \Sigma_1.\, \boldsymbol{s_i} \models g_{a_1} \wedge \\
& f_{a_1}) \wedge \boldsymbol{s} = \boldsymbol{s_0} \\
\Longrightarrow\ & \boldsymbol{s} \models pre^*(M_1, \phi_2) \qquad (4.1)
\end{aligned}
$$

Proof of the converse is as follows:

Claim B: If a state $\boldsymbol{s}$ satisfies $pre^*(M_1, \phi_2)$ then $\boldsymbol{s} \models \mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2)$ in the counter system M where M_1 is the refinement of M wrt to ϕ_1.

$$
\begin{aligned}
& \boldsymbol{s} \models pre^*(M_1, \phi_2) \\
\implies & \exists k \geq 0.\, \exists \boldsymbol{s_0}, \ldots, \boldsymbol{s_k}.\, \boldsymbol{s_k} \models \phi_2 \wedge (\forall i, 0 \leq i < k.\, \exists a_1 \in \Sigma_1.\, \boldsymbol{s_i} \models g_{a_1} \wedge \\
& f_{a_1}) \wedge \boldsymbol{s} = \boldsymbol{s_0} \\
\implies & \exists k \geq 0.\, \exists \boldsymbol{s_0}, \ldots, \boldsymbol{s_k}.\, \boldsymbol{s_k} \models \phi_2 \wedge (\forall i, 0 \leq i < k.\, \exists a \in \Sigma.\, \boldsymbol{s_i} \models (g_a \wedge \phi_1) \wedge \\
& f_a) \wedge \boldsymbol{s} = \boldsymbol{s_0} \\
\implies & \boldsymbol{s} \models \mathbf{E}(\phi_1\ \mathbf{U}\ \phi_2) \implies \boldsymbol{s} \models \psi
\end{aligned} \tag{4.2}
$$

Note that in Equations 4.1 and 4.2, $g_{a_1} = g_a \wedge \phi_1$ and $f_{a_1} = f_a$ are the guards and actions in the refined system M_1 which is a refinement of M with respect to ϕ_1.

□

Lemma 1. *If routine computeUntil returns a set of states ϕ and the enumeration 'under' when termination is forced, then ϕ is an under-approximation of the set of states that satisfy $\boldsymbol{E(\phi_1\ U\ \phi_2)}$ in the counter system M.*

Proof. The routine *computeUntil* returns the enumerator *under* whenever the routine *preStar* is terminated forcefully. The proof of the theorem is straight froward from Claim B in proof of Theorem 2.

□

Chapter 5

Computing "Global" properties

Let $M = \langle Q, C, \Sigma, \phi^{init}, G, F \rangle$ be the input counter system and $\mathbf{EG}\phi$ be a temporal property to be verified. In this section we discuss two approaches that compute an under-approximation of the set of states of M that satisfy the property $\mathbf{EG}\phi$. We begin this section by introducing some terminology that is required to discuss these approaches.

A given counter system $M = \langle Q, C, \Sigma, \phi^{init}, G, F \rangle$ is said to be *flat* if for every control state $q_0 \in Q$, there is at most one control state sequence $q_0, q_1, \ldots q_i, q_0$ such that $\forall j,\ 0 \leq j < i.\ (\exists a \in \Sigma.\ q_j \xrightarrow{a} q_{j+1}) \wedge (\exists b \in \Sigma.\ q_i \xrightarrow{b} q_0) \wedge (\nexists j, k.\ 0 \leq j \leq i \wedge 0 \leq k \leq i \wedge (j \neq k \implies q_j \neq q_k))$. Informally, in a flat system all the cycles among its control states are simple cycles.

A trace from a set of states ϕ (represented as $traces(M, \phi)$) is a sequence of states $\boldsymbol{s_0}, \boldsymbol{s_1}, \boldsymbol{s_2}, \ldots$ such that $\boldsymbol{s_0}$ satisfies ϕ and $\forall i \geq 0. \exists a \in \Sigma. \boldsymbol{s_i} \xrightarrow{a} \boldsymbol{s_{i+1}}$. $traces(M, \phi)$ is the set of all traces from every state in ϕ.

A system $N = \langle Q_1, C, \Sigma_1, \phi_1^{init}, G_1, F_1 \rangle$ is said to be a p^Q-*flattening* (flattening, in short) of $M = \langle Q, C, \Sigma, \phi^{init}, G, F \rangle$, if

- N is flat.
- There exists a mapping function $p^Q : Q_1 \to Q$. That is for every q_1, q_1 and $p^Q(q_1)$ have the same natural number encoding.
- For every $a_1 \in \Sigma_1$ there exists a unique $a \in \Sigma$. such that $(g_{a_1} \equiv g_a \wedge f_{a_1} \equiv f_a)$.

That is, the guards and actions of the transitions in the flattening N are same as the guards and actions, as the transitions in the system M.

Note that every transition a_1 in the system N has a corresponding transition a in the system M whose guards and actions are the same. Hence it is easy to see that given a set of states ϕ in M, $traces(N, \phi) \subseteq traces(M, \phi)$. The flattening N *may* not preserve the reachable states of M either. We say that the flattening is of size k if it has k transitions.

A system $N = \langle Q_1, C, \Sigma_1, \phi_1^{init}, G_1, F_1 \rangle$ is said to be a p^Q-*trace-flattening* (trace flattening in short) of $M = \langle Q, C, \Sigma, \phi^{init}, G, F \rangle$ *with respect to* a set of states ϕ if

- N is a p^Q-flattening of M and
- $traces(N, \phi) = traces(M, \phi)$

The system M is said to be *trace-flattable* with respect to a set of states ϕ, if there exists a system N which is a trace-flattening of M with respect to to ϕ.

5.1 Introduction to some reachability analysis tools

In this section, we first define the external sub-routines that we need and also introduce some reachability analysis tools that provide the implementation of these routines. Any black box that provides these routines can be used in the the implementation of our routines *computeUntil* and *computeGlobal*.

- *preStar*: Given a counter system M and a set of states ϕ the routine *computeUntil* requires to compute the set of predecessors of ϕ (Line 2 in Algorithm 2) i.e, $pre^*(M, \phi)$.
- $compute\_pre^k(N, \phi)$: We need a black-box $compute\_pre^k$ which takes a counter system N and a set of states ϕ and returns a Presburger formula for $pre^k(N, \phi)_{(k)}$. $pre^k(N, \phi)_{(k)}$ represents the set of states that have paths of length k to some state that satisfies ϕ. This black-box requires the input counter system N to be flat. This is closely related to the black-box that computes $pre^*(N, \phi)$ because $pre^*(M, \phi) \equiv \exists k \geq 0.\ pre^k(N, \phi)_{(k)}$.

- *isTraceFlattening*(M, N, ϕ): The routine *computeGlobal* requires to check whether a system N is a trace flattening of M with respect to a set of states ϕ, that is, to check whether $traces(M, \phi) = traces(N, \phi)$.

We will now discuss the tools that provide the implementation of the above routines. The tool FAST [12, 10] is one such tool. FAST provides the implementation of the routine $pre^*(M, \phi)$. Fast terminates whenever (a) the counter system M is flat (b) for every transition $a \in \Sigma$, g_a is a Presburger formula and f_a is an *affine* function of the form $\boldsymbol{s'} = A\boldsymbol{s} + \boldsymbol{b}$ where A is a vector of size $m \times m$, and $\boldsymbol{b}$ is a column vector of size m and m is the number of counters in the system (c) multiplicative monoid of A is *finite*. If the system is not flat, then FAST generates a flattening N of M then computes $pre^*(M, \phi)$. Since a flattening need not preserve the reachable states of the system, FAST explores different flattenings until it finds a flattening N_1 such that $pre^*(N_1, \phi) \equiv pre^*(M, \phi)$, that is, the states from which ϕ can be reached in both M and N are same. FAST may go into non-termination in the process of exploring different flattenings. A premature termination of FAST will give an under-approximation of the set of states that satisfy $pre^*(M, \phi)$. The routine to compute $pre^*(M, \phi)$ which is incorporated in FAST necessarily terminates on flat systems and also 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, provided the guards and actions of M adhere to the restrictions stated above. But $pre^k(N, \phi)_{(k)}$ can be computed using FAST if M is flat. Given a flat system M, during the computation of $pre^*(M, \phi)$ FAST computes the formula for $pre^k(M, \phi)_{(k)}$ and then existentially quantifies over k to compute $pre^*(M, \phi)$. If the input system M is flat then $pre^k(M, \phi)_{(k)}$ can be computed precisely. Hence the tool FAST can be used as a black-box for computing $compute\_pre^k$ in flat systems.

The tool TREX another reachability analysis tool which computes the set of reachable states on a different class of counter systems. The restrictions on the guards and actions of the transitions of the input counter system are shown in Table 5.1. TREX provides an implementation for the routine $pre^*(M, \phi)$. It can also be used to compute $pre^k(M, \phi)_{(k)}$ provided M is flat. But the formula returned by TREX may not be in Presburger Logic

	FAST	TREX	LASH
Accelerations	yes	yes	yes
Guards	Presburger	$x_i \leq x_j + c, x_i \leq c, x_i \geq c$	convex sets
Actions	$\boldsymbol{s'} = A\boldsymbol{s} + \boldsymbol{b}$	$x_i = x_j + c_i, x_i = c_i$	$\boldsymbol{s'} = A\boldsymbol{s} + \boldsymbol{b}$
Automatic cycle search	yes	yes	no

Table 5.1: Comparison of various reachability analysis tools

in all cases.

The tool LASH addresses a class of counter systems similar to FAST as shown in Table 5.1. But it does not provide automatic cycle detection schemes which is necessary to accelerate cycles in the input counter system. LASH provides the routine to compute $pre^*(M, \phi)$, provided the cycles in M that need to accelerated are specified manually. It can also be used to compute $pre^k(M, \phi)_{(k)}$ of manually specified simple cycles in the counter system M. More detailed comparisons of these tools is available in [15, 16, 17].

The third black-box that we need is $isTraceFlattening(M, N, \phi)$. Demri et al [13] give a decision procedure to check whether a given flattening N of M is a trace-flattening of M with respect to a set of states ϕ. Their decision procedure returns *true* if $traces(M, \phi) = traces(N, \phi)$ else they return *false*. We use the same decision procedure as the black-box $isTraceFlattening(M, N, \phi)$. The implementation involves a pre^* query on the counter system N which is a flattening of the counter system M. Therefore an reachability analysis tool like FAST or TREX can be used to implement this back-box provided it is able to compute pre^* of some set of states in the counter system N.

5.2 Implementing the routine $computeGlobal(M, \phi, label)$

The routine $computeGlobal(M, \phi, label)$ is expected to compute the set of states that satisfy $\mathbf{EG}\phi$ in the counter system M. Note that the third parameter *label* is an element from the lattice *Approx*(introduced in Section 3) that indicates the expected direction of approximation of the set of states that satisfy the property $\mathbf{EG}\phi$. We provide two

routines with different approximation capabilities and the parameter *label* decides which of these routines to be chosen (discussed later). A state $\boldsymbol{s}$ in M satisfies $\mathbf{EG}\phi$ if and only if there is an infinite trace $T \in \mathit{traces}(M, \boldsymbol{s})$ such that all states in T satisfy ϕ. Let $M_1 = \mathit{refineSystem}(M, \phi)$ be the refinement of M with respect to ϕ. A transition a which could be fired from a state $\boldsymbol{s}$ in the system M can be fired in M_1 if and only if $\boldsymbol{s} \models \phi$. Hence every trace $T_1 \in \mathit{traces}(M_1, \boldsymbol{s})$ will be such that each state $\boldsymbol{s_i}$ which is not *stuck* (from which there are no outgoing transitions) in T_1 will satisfy ϕ along with the guard of the transition. Hence any state $\boldsymbol{s}$ which satisfies $\mathbf{EG}\phi$ in M will have atleast one infinite path from it in M_1. Hence our problem reduces to finding the set of states that have atleast one infinite path in M_1.

Recall that $\mathit{pre}^k(M, \phi)_{(k)}$ is a Presburger formula that represents the set of states from which there are paths of length k to some state that satisfies ϕ in the counter system M. Since M_1 is a refinement of M with respect to ϕ, $\mathit{pre}^k(M_1, \phi)_{(k)}$ represents the set of states which have paths of length k to a state that satisfies ϕ along which all states also satisfy ϕ. Any state $\boldsymbol{s}$ that satisfies $\mathbf{EG}\phi$ should have a path of infinite length in the counter system M_1. The set of states that have a path from from itself to some state that satisfies ϕ for all values of $k \geq 0$, are given by the formula $\forall k \geq 0.\ \mathit{pre}^k(M_1, \phi)_{(k)}$. Any state $\boldsymbol{s}$ satisfies this formula if and only if $\boldsymbol{s}$ satisfies the property $\mathbf{EG}\phi$. We formally prove this in Theorem 3. Hence our strategy will be to use the black-box $\mathit{compute\_pre}^k$ (discussed in Section 5.1) to generate a Presburger formula (in $\boldsymbol{s}, k$) for $\mathit{pre}^k(M_1, \phi)_{(k)}$, and then prefix the formula with "$\forall k \geq 0$". This formula will precisely represent the set of states that satisfy the property $\mathbf{EG}\phi$.

Theorem 3. *Given a counter system $M = \langle Q, C, \Sigma, \phi^{init}, G, F \rangle$, the set of states that satisfy the formula $\boldsymbol{EG}\phi$ in M is precisely given by the set of states that satisfy the formula $\forall k \geq 0.\ \mathit{pre}^k(M_1, \phi)_{(k)}$, where $M_1 = \mathit{refineSystem}(M, \phi)$.*

Proof. Let M_1 be the refined system returned by $\mathit{refineSystem}(M, \phi)$. Let $\boldsymbol{s}$ be a state in M which is reachable from ϕ^{init}. Let $\boldsymbol{s} \models \mathbf{EG}\phi$.

$$
\begin{aligned}
& \boldsymbol{s} \models \mathbf{EG}\phi \\
\Longrightarrow\ & \exists \boldsymbol{s_1}, \boldsymbol{s_2}, \ldots . \forall i \geq 1. \exists a \in \Sigma. \boldsymbol{s_i} \models g_a \wedge f_a \wedge \boldsymbol{s_i} \models \phi \wedge \boldsymbol{s} = \boldsymbol{s_1} \\
\Longrightarrow\ & \forall k \geq 0. \exists \boldsymbol{s_1}, \boldsymbol{s_2}, \ldots, \boldsymbol{s_k}. \forall i, 1 \leq i < k. (\exists a \in \Sigma. \boldsymbol{s_i} \models (g_a \wedge \phi) \wedge \\
& f_a) \wedge \boldsymbol{s_k} \models \phi \wedge \boldsymbol{s} = \boldsymbol{s_1} \qquad (5.1)
\end{aligned}
$$

Since M_1 is a refinement of M with respect to ϕ $g_{a_1} = g_a \wedge \phi$ and $f_{a_1} = f_a$ are the guards and actions of the refined system, Equation 5.1 can be rewritten as

$$
\begin{aligned}
& \forall k \geq 0. \exists \boldsymbol{s_1}, \boldsymbol{s_2}, \ldots, \boldsymbol{s_k}. \forall i, 1 \leq i < k. (\exists a_1 \in \Sigma_1. \boldsymbol{s_i} \models g_{a_1} \wedge \\
& f_{a_1}) \wedge \boldsymbol{s_k} \models \phi \wedge \boldsymbol{s} = \boldsymbol{s_1} \\
\Longrightarrow\ & \boldsymbol{s} \models \forall k \geq 0. \mathit{pre}^k(M_1, \phi)_{(k)}
\end{aligned}
$$

The converse of the theorem can be proved as follows: Let $\boldsymbol{s}$ satisfy $\phi \equiv \forall k \geq 0.\ \mathit{pre}^k(M_1, \phi)_{(k)}$. Let $\boldsymbol{s}$ be reachable.

$$
\begin{aligned}
\Longrightarrow\ & \forall k \geq 0.\ \exists \boldsymbol{s_1}, \boldsymbol{s_2}, \ldots, \boldsymbol{s_k}. \forall i, 1 \leq i < k. (\exists a_1 \in \Sigma_1. \boldsymbol{s_i} \models g_{a_1} \wedge \\
& f_{a_1}) \wedge \boldsymbol{s_k} \models \phi \wedge \boldsymbol{s} = \boldsymbol{s_1} \\
\Longrightarrow\ & \exists \boldsymbol{s_1}, \boldsymbol{s_2}, \ldots . \forall i \geq 1. (\exists a_1 \in \Sigma_1. \boldsymbol{s_i} \models g_{a_1} \wedge f_{a_1} \wedge \boldsymbol{s_i} \models \phi) \wedge \boldsymbol{s} = \boldsymbol{s_1} \\
& [\text{ By König's Lemma}^2 \text{ }] \\
\Longrightarrow\ & \exists \boldsymbol{s_1}, \boldsymbol{s_2}, \ldots . \forall i \geq 1. (\exists a \in \Sigma. \boldsymbol{s_i} \models g_a \wedge f_a) \wedge \boldsymbol{s_i} \models \phi) \wedge \boldsymbol{s} = \boldsymbol{s_1} \\
\Longrightarrow\ & \boldsymbol{s} \models \mathbf{EG}\phi
\end{aligned}
$$

□

[2] If G is a connected graph with infinitely many vertices such that every vertex has finite degree, then G contains an infinitely long simple path.

The two different techniques that we propose in this section generate a flattening N of the system M_1, and then compute $pre^k(N, \phi)_{(k)}$[1]. The first approach non-deterministically chooses a flattening but the second approach systematically generates flattenings till it finds a flattening which satisfies a certain property.

Approach 1: A simple under-approximation technique.

Let M be the given system and M_1 be the refinement of M with respect to ϕ. If M_1 is flat we can compute $pre^k(M_1, \phi)_{(k)}$ precisely by using the black-box *compute_pre*^{k} described in Section 5.1 and we are done. If M_1 is not flat then we need to generate a flattening N of M_1 and then use the black-box to compute $pre^k(M, \phi)_{(k)}$. The reason we need a flattening is because the *compute_pre*^{k} computes $pre^k(M, \phi)_{(k)}$ precisely on flat systems. Throughout this thesis, when we say compute $pre^k(M, \phi)_{(k)}$, we mean to that we invoke the black-box *compute_pre*^{k} to compute $pre^k(M, \phi)_{(k)}$. Note that the counter system that is passed as an argument to the black box is always flat.

In case the input counter system M is not flat, we first refine the counter system M_1 with respect to ϕ_1. We then generate a flattening of M_1. Note that there exists many different flattenings of M and hence in both the cases we non-deterministically choose a flattening. Let N be a flattening that was chosen non-deterministically as shown in Algorithm 3. Since N is flat we compute $pre^k(N, \phi)_{(k)}$ precisely and then append a $\forall k \geq 0$ (Line 3) in front of the computed formula. The set of states that satisfy the formula $\forall k \geq 0$. $pre^k(N, \phi)_{(k)}$ are precisely the set of states that satisfy the property $\mathbf{EG}\phi$ in the counter system N_1. However this is an under-approximation of the set of states that satisfy the formula $\mathbf{EG}\psi$ in the counter system M. This is because the chosen flattening N might not contain all the traces that are present in the counter system M_1 from the set of states ϕ. That is, N might be a flattening of M_1 such that $traces(N, \phi) \subset traces(M_1, \phi)$.

The flattening in Line 2 of Algorithm 3 can be chosen non deterministically or any

[1]Instead of computing $\forall k pre^k(N, \phi)_{(k)}$, any approach that computes a Presurger formula representing the set if states that satisfy $\mathbf{EG}\phi$ in N can be used

Algorithm 3 *computeGlobal*$(M, \phi, label)$

Input: A counter system M, Presburger formula ϕ and an enumerator *label* from the lattice *Approx* and $label = under$.

Output: A Presburger formula ϕ and the enumerator *under*.

1: $M_1 \leftarrow refineSystem(M, \phi)$.

2: Choose a flattening N of M_1

3: $X \equiv \forall k \geq 0.\ pre^k(N, \phi)_{(k)}$.

4: **return** $(X, under)$

heuristics can be used. Also some of the black-boxes that provide routines to compute pre^* generate a flattening of the input counter system. Hence some of the black-boxes that we describe in Section 5.1 can also be used to generate flattenings of the system M.

To check whether the set of states that is returned by Algorithm 3 is precise, we can check if $traces(M_1, \phi - X) = traces(N, \phi - X)$ (correctness of this check is discussed later) using the black-box $isTraceFlattening(M_1, N, \phi - X)$. If the black-box returns *true*, then we say that the X precisely represents the set of states that satisfies the property $\mathbf{EG}\phi$. Otherwise we do not have any idea whether X is precise or an under-approximation of the set of states that satisfy $\mathbf{EG}\phi$. We do not incorporate this check in this naive algorithm and we always return *under*. Note that the routine *computeGlobal* is required to return a pair: a Presburger formula and an enumerator indicating whether the returned formula is precise or over/under-approximation of the set of states that satisfies $\mathbf{EG}\phi$. We always return the enumerator *under* along with the computed formula for $\forall k \geq 0.\ pre^k(N, \phi)_{(k)}$.

This approach is always guaranteed to terminate and it always returns the enumerator *under*. The routine *computeGlobal* has an argument *label* which is passed by Algorithm 1 in Line 21. This argument can be either *precise* or *under*. Therefore this approach is applicable if and only if the label that was passed is *under*. Therefore the routine satisfies the assumption mentioned in Section 3 that the returned enumerator is always dominated by the one given as the argument.

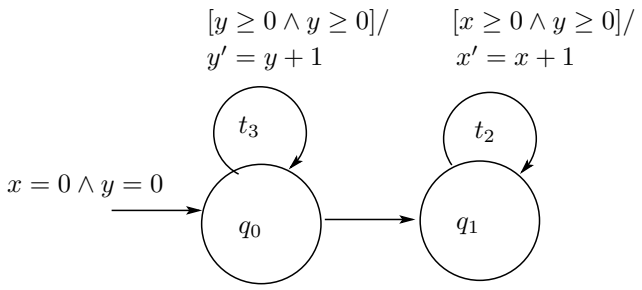

Figure 5.1: Refinement N_1 of the system in Figure 1.3 with respect to $y \geq 0$

Illustration: Consider the example counter system M shown in Figure 1.1 in Section 1. Let $\mathbf{EG}(y \geq 0)$ be the property to be verified. We refine the system M with respect to $y \geq 0$ and choose a flattening N as shown in Figure 5.1.

Since this system is flat, we can compute $pre^k(N, y \geq 0)_{(k)}$. $pre^k(N, y \geq 0)_{(k)}$ is given below in Formula 5.2

$$\begin{aligned}
\exists k_1, k_2.\ \exists x', y'.\ (& \\
& y' = y + k_1 \wedge x' = x \wedge k_1 \geq 0 \\
\wedge\ & \forall i.\ (0 \leq i < k_1 \implies y + i \geq 0) \\
\wedge\ & x'' = x' + k_2 \wedge y'' = y' \wedge k_2 \geq 0 \\
\wedge\ & \forall i.\ (0 \leq i < k_2 \implies (x' + i \geq 0 \wedge y' \geq 0)) \\
\wedge\ & k = k_1 + k_2 \wedge y'' \geq 0.)
\end{aligned} \tag{5.2}$$

Formula 5.2 can be simplified to get a formula with only x, y and k as free variables. We then conjuct the Formula 5.2 with $k \geq 0$ and then universally quantify over k to get the formula for $\forall k \geq 0.\ pre^k(N, y \geq 0)_{(k)}$. The set of states that satisfy $\forall k \geq 0.\ pre^k(N, y \geq 0)_{(k)}$ is $y \geq 0$. Note that this precisely represents the set of states that satisfy the property $\mathbf{EG}(y \geq 0)$ in the counter system M (though the algorithm returns *under*).

As another example consider the property $\mathbf{EG}(x < 0 \wedge y \geq 0)$ that is to be verified in the counter system shown in Figure 5.2. The counter system has two counters x and

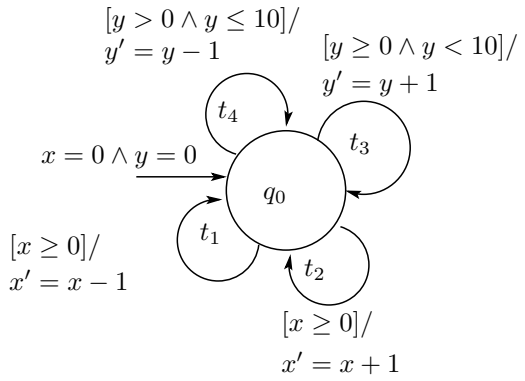

Figure 5.2: Counter system M

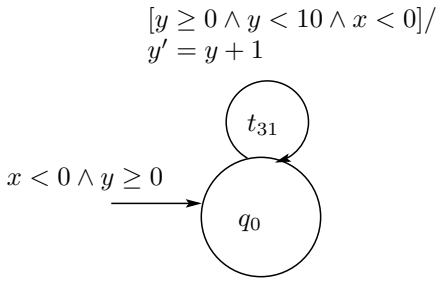

Figure 5.3: Flattening of the counter system shown in Figure 5.2

y. This counter system differs from the one shown in Figure 1.1 in terms of values that the counters x and y can take. In the current example, the counter x can take a value -1 in addition to positive values of x and counter y can take values between 0 and 10 only. To model-check the counter system M shown in Figure 5.2, we first produce a flattening N and then refine it with $x < 0 \wedge y \geq 0$ as shown in Figure 5.3. We then compute the set of states that satisfy $\forall k \geq 0$ $pre^k(N, x < 0 \wedge y \geq 0)_{(k)}$. None of the states in N will satisfy the formula and the algorithm returns *false*. This is because every trace from a state with a counter value $y \geq 0$ ends at $y = 10$ and traces with $y > 10$ does not satisfy the guard of any of the transitions that change the value of counter y in N. The only negative value of x which can be taken is -1 after which the value of counter x can not be changed. Hence every trace in the counter system N will be at most of length 11. But every state which satisfies $(y \geq 0 \wedge y \leq 10 \wedge x < 0)$ actually satisfies the

formula $\mathbf{EG}x < 0 \wedge y \geq 0$ in the counter system M. Hence the formula returned by the approach is an under-approximation of the set of states that satisfy $\mathbf{EG}(x < 0 \wedge y \geq 0)$. This is mainly because N does not preserve certain important traces present in M_1 (and hence in M) .

Approach 2: Iterative method to implement the routine *computeGlobal*

In this section we give an implementation of *computeGlobal*$(M, \phi, label)$, which precisely computes the set of states that satisfy the property $\mathbf{EG}\phi$ whenever it terminates. Note that Approach 1 that we proposed suffered from the limitation that we were not able to compute precisely the set of states that satisfy $\mathbf{EG}\phi$ in the given system M in systems as shown in Figure 5.2. This was because of the fact that Approach 1 generates a single flattening which may or may not preserve traces in the system. Hence we iteratively produce different flattenings until we find the one that preserves a certain important traces that M exhibits.

Let us say that we refine the system M with respect to ϕ to get a new system M_1. A naive property would ask M_1 to be trace-flattable with respect to ϕ. That is, there exists a flattening N of M_1 such that $traces(M_1, \phi) = traces(N, \phi)$ Since N is a trace-flattening of M_1 every trace in M_1 is preserved in N and hence $pre^k(M_1, \phi)_{(k)} \equiv pre^k(N, \phi)_{(k)}$. Because N is flat, we can compute $pre^k(N, \phi)_{(k)}$ precisely. This trace flattening N of M_1 with respect to ϕ can be generated by unrolling the loops in M_1 as described by Demri et al, [13].

By generating a trace-flattening of M_1 with respect to ϕ, all the traces in the system M_1 are equivalent to the traces in the flattening N of M_1. However, by the semantics of $\mathbf{EG}\phi$, we need to look for the states that have at least one infinite path in M_1. Hence it is unnecessary to preserve all traces. Hence the naive property of generating a flattening N of M_1 such that $traces(M_1, \phi) = traces(N, \phi)$ can be weakened. To find a state $\boldsymbol{s}$ that satisfies $\mathbf{EG}\phi$, we need to generate a flattening that has at least one infinite trace from a state $\boldsymbol{s}$ if it exists and the flattening need not have any other trace from the state $\boldsymbol{s}$ which was originally present in the flattening. In other words, once we generate

a flattening N of M_1 and check if all traces from the set of states that do not satisfy $\mathbf{EG}\phi$ in the system N are present in the counter system M_1. That is we check whether $traces(M_1, \phi - X) = traces(N, \phi - X)$ where X represents the set of states that satisfy $\mathbf{EG}\phi$ in the flattening. This is because there might be a state $\boldsymbol{s_1}$ such that there is a transition $\boldsymbol{s_1} \xrightarrow{a} \boldsymbol{s}$ in M_1 and this may be missing in N. In such a case we would miss an infinite trace from $\boldsymbol{s_1}$ if there existed an infinite trace from the state $\boldsymbol{s}$. Hence we need to check whether traces from all states that do not have an infinite path in M_1 are present in N. We iteratively generate flattenings N of M_1, and compute the set of states X that satisfy $\forall k \geq 0.\ pre^k(N, \phi)_{(k)}$. We then check if $traces(M_1, \phi - X) = traces(N, \phi - X)$. If so then we return X as the precise set of states that satisfy $\mathbf{EG}\phi$ in the counter system M as shown in Algorithm 4. We prove the correctness of this algorithm in Theorem 4 in Section 5.4.

The algorithm first refines the input system M with respect to ϕ (Line 1 in Algorithm 4) to get the new system M_1. The algorithm iteratively produces bigger flattenings of M_1 (Line 5). After producing each flattening N of M_1, we compute $X \equiv \forall k \geq 0.\ pre^k(N, \phi)_{(k)}$ in each case in Line 7. Since N is flat $\forall k \geq 0.\ pre^k(N, \phi)_{(k)}$ can be computed precisely by using the black-box to compute the formula for $pre^k(M, \phi)_{(k)}$ and then universally quantifying over k. We then check if $isTraceFlattening(M, N, \phi - X)$ holds (Line 8). The routine $isTraceFlattening(M, N, \phi - X)$ returns *true* if $traces(M_1, \phi - X) = traces(N, \phi - X)$. This is a black-box (discussed in Section 5.1) for us to check $traces(M_1, \phi - X) = traces(N, \phi - X)$. If yes we return X as the set of states that satisfy $\mathbf{EG}\phi$ along with the enumerator *precise* in Line 9.

Note that the way in which we enumerate the flattenings in Line 5 is not arbitrary. We enumerate them in increasing order of length i.e, in a breadth first way. This is to ensure the completeness property (the details of which we provide later).

The routine *computeGlobal* need not terminate in all cases. At any point of time the algorithm may be forced to terminate and return X along with the enumerator *under* saying that X represents the under-approximation of the set of states that satisfy $\mathbf{EG}\phi$. Note that this routine can be forced to terminate only if *label* is *under*. Else we will

Algorithm 4 *computeGlobal*$(M, \phi, label)$

Input: A system M and the set of states ϕ and an enumerator $label \in under, precise$.

Output: A set of states X and an enumerator *approx* which indicates the direction of approximation of X with respect to the set of states that satisfy $\mathbf{EG}\phi$ in M. The returned enumerator $approx \sqsubseteq label$ in the Lattice *Approx*

1: $M_1 \leftarrow refineSystem(M, \phi)$
2: $k \leftarrow 1$
3: $X \leftarrow \emptyset$
4: **while** *not forced to terminate* **do**
5: $FLAT \leftarrow$ *All flattenings of* M_1 *of length* k
6: **for all** $N \in FLAT$ **do**
7: $X \leftarrow \forall k \geq 0.pre^k(N, \phi)_{(k)}$
8: **if** $isTraceFlattening(M_1, N, \phi - X)$ **then**
9: **return** $(X, precise)$
10: **end if**
11: **end for**
12: $k \leftarrow k + 1$
13: **end while**
14: **return** $(X, under)$

have to wait for the routine to terminate and if it terminates we return X along with the enumerator *precise*. As described in Section 3 the algorithm *computeGlobal* is expected to return an enumerator *approx* such that, $approx \sqsubseteq label$, where *label* is the last argument to *computeGlobal*. When the method is invoked with the $label = under$, the algorithm returns *under* or *precise*, and when $label = precise$ we wait for the algorithm to terminate and return *precise*. Hence in both cases the value that we return is dominated by the one given in the argument.

We discuss in detail a class of systems on which Algorithm 4 necessarily terminates in Section 5.3. We formally prove that Algorithm 4 gives precisely the set of states that satisfy the property $\mathbf{EG}\phi$ whenever it terminates and under-approximates the set

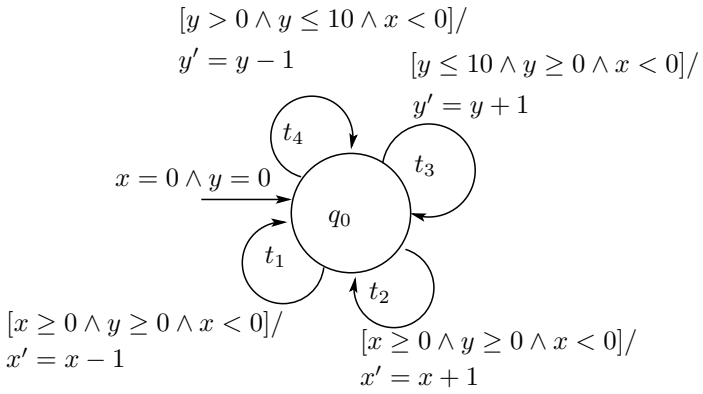

Figure 5.4: Refinement of counter system shown in Figure 5.2 with respect to $(x < 0) \wedge (y \geq 0)$

of states that satisfy the property when it is forced to terminate in Section 5.4.

Illustration: Consider the counter system shown in Figure 5.2. Suppose we want to model-check the property $\mathbf{EG}(x < 0 \wedge y \geq 0)$. Note that Approach 1 returned *false* which is an under-approximation of the set of states that satisfied above the property.

The set of states that satisfy the property $(x < 0 \wedge y \geq 0)$ is given by $\phi \equiv (x < 0 \wedge y \geq 0)$ is passed as the argument to the routine *computeGlobal* along with the machine M. Let the enumerator *label* passed to the routine be *precise*. We first refine the system M with ϕ to get the refined system M_1 which is shown in Figure 5.4.

We then generate a flattening N of M_1 as shown in the Figure 5.3. The flattening shown in the figure is a flattening of length 1. Note that this was the same flattening produced by Approach 1. None of the states in N would satisfy $\forall k \geq 0$. $pre^k(N, (x < 0 \wedge y \geq 0))_{(k)}$. The set X would remain empty after Line 7 during the first iteration of Algorithm 4. Now we call the routine *isTraceFlattening*$(M, N, \phi - X)$. $\phi - X$ is the set of states given by the formula $(x < 0 \wedge y \geq 0)$. Now in the counter system N there is no transition that decrements the value of the counter y which is present in the counter system M_1. For example the transition from a state $y = 1$ to the state $y = 0$ that was present in the counter system M_1 is missing in N. In general, every state with value of $y \geq 0 \wedge y \leq 10$ has a missing transition and hence a missing trace which was present in the

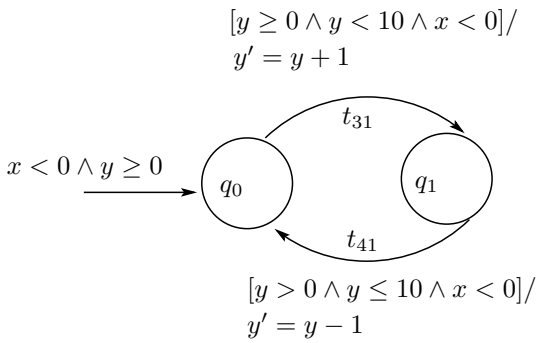

Figure 5.5: A Flattening of counter system shown in Figure 5.2

counter system M_1 but not in N. Therefore the routine *isTraceFlattening*$(M_1, N, \phi - X)$ would return the value *false* indicating *traces*$(M_1, \phi - X) \neq$ *traces*$(N, \phi - X)$.

We therefore generate a bigger flattening of the counter system M_1 shown in Figure 5.4 during the second iteration of Algorithm 4. Let us say that we generate a new flattening N of M_1, with two control states, as shown in Figure 5.5. The transitions t_{31} and t_{41} increment and decrement the values of counter y in the system N respectively. We compute $\forall k \geq 0$. $pre^k(N, \phi)_{(k)}$ where $\phi \equiv (x < 0 \wedge y \geq 0)$. The set of states that satisfy the formula is given by $x < 0 \wedge y \geq 0 \wedge y \leq 10$. This is because from every value of y between 0 and 10 you can increment and decrement infinitely. Now the routine *isTraceFlattening*$(M, N, \phi - X)$ will return *true*. This is because, the set of states $\phi - X$ is $(y \geq 11) \wedge (x < 0)$ and none of these states satisfy the guard of any transition in the counter system M_1 shown in Figure 5.4. Therefore *traces*$(M_1, \phi - X) =$ *traces*$(N, \phi - X)$. We then return $(x < 0 \wedge y \geq 0 \wedge y \leq 10)$ along with the enumerator *precise* as the set of states that satisfy the property $\mathbf{EG}(x < 0 \wedge y \geq 0)$ in the counter system M.

Note that the traces in the flattening N showed in Figure 5.5. The flattening does not have have all the traces in the counter system M_1. For example the trace from a state $(x = -1 \wedge y = 0) \xrightarrow{t_3} (x = -1 \wedge y = 1), (x = -1 \wedge y = 1) \xrightarrow{t_3} (x = -1 \wedge y = 2) \ldots$ is missing the system shown in Figure 5.5. But the state $(x = -1 \wedge y = 0)$ has the trace $(x = -1 \wedge y = 0) \xrightarrow{t_{31}} (x = -1 \wedge y = 1), (x = -1 \wedge y = 1) \xrightarrow{t_{31}} (x = -1 \wedge y = 0) \ldots$ in

N, which is infinite. This trace is sufficent to prove that $(x = -1 \wedge y = 0)$ satisfies $\mathbf{EG}(x < 0 \wedge y \geq 0)$. This is the same case for every state with $x = -1$ and $y \geq 0 \wedge y \leq 10$. Therefore the flattening N is not completely trace equivalent with the counter system M_1. This is our main optimization which enables to answer global properties on a larger class of system than trace-flattable systems.

5.3 Termination and precision characteristics of the two approaches

In this section we give the termination characteristics of the two approaches described in Section 5.2. Note that both approaches provide the implementation of the routine *computeGlobal*.

The Approach 1 that we proposed will necessarily terminate in all cases. But it is not clear how to determine whether the computed set of states is precise or an under-approximation of the set of states that satisfy the property $\mathbf{EG}\phi$. The second approach tries to overcome this drawback. It gives precise results whenever it terminates. But the algorithm may go into non-termination. The algorithm may be made to terminate forcefully and hence obtain an under-approximation of the set of states that satisfy the property $\mathbf{EG}\phi$. In the rest of this section we give a comparison between the two approaches regarding their precision.

We always return the enumerator *under* in Approach 1 because it is not clear how to determine whether the set of states returned is precise or an under-approximation of the set of states that satisfies $\mathbf{EG}\phi$. Let us assume that there exists some way to say whether the returned solution by Approach 1 is precise (though the algorithm returns *under*). Then, in such a case we would like to answer the question whether Approach 2 will necessarily terminate. We now give an example of a system on which Approach 1 happens to give a precise result (although it returns the enumerator *under*) but Approach 2 does not terminate.

Therefore if Approach 2 is forced to terminate then it will give an under-approximated

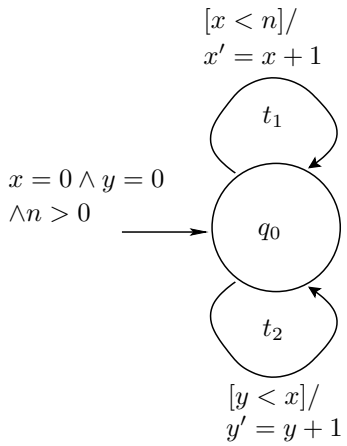

Figure 5.6: Counter system M

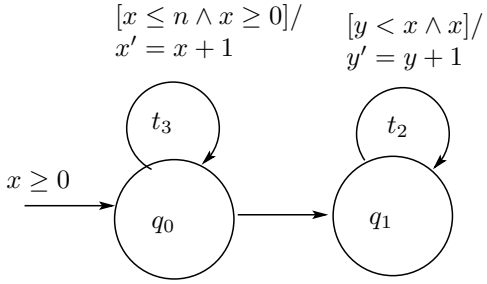

Figure 5.7: A flattening N of the counter system M shown in Figure 5.6

solution. The second illustration in Section 5.2 gave the example of a system on which Approach 1 computed an under-approximated result but Approach 2 computed a precise solution. This indicates the fact that the two approaches are incomparable. Consider the counter system shown in Figure 5.6. "n" is a parameter to the system. Note that there is a bound on the lengths of all possible traces in the system namely, $2n$. Therefore for any set of states ϕ, $\mathbf{EG}\phi$ is *false*. Consider the property $\mathbf{EG}(x \geq 0)$. Approach 1 produces a flattening N and then refine it with respect to $x \geq 0$, as shown in Figure 5.7. We then compute the set of states that satisfy $\forall k \geq 0.pre^k(N, (x \geq 0))_{(k)}$. None of the states in N will satisfy the above formula and Approach 1 returns $(false, under)$ as the solution. Note that *false* happens to be the precise answer (even though the returned approximation enumerator does not capture this).

Now consider Approach 2. We first refine the system M with respect to to $(x \geq 0)$ to get the system M_1. Algorithm 4 iteratively generates flattenigns till it finds a flattening N of M_1 such that $traces(M_1, \phi - X) = traces(N, \phi - X)$, where X is the set of states that satisfy $\mathbf{EG}(x \geq 0)$ in N. During each such iteration the set X is computed, and in each of those iterations X remains *false*. Now for the algorithm to terminate, the generated flattening N should be such that $traces(N, x \geq 0) = traces(M_1, x \geq 0)$. But such a flattening N of the system M_1 does not exist because the transitions t_1 and t_2 can be nested arbitrarily. Thus any generated flattening N of M_1 will be trace equivalent with M_1 with respect to to the set of states $x \geq 0$. That is $traces(N, x \geq 0) \neq traces(M_1, x \geq 0)$ for any flattening N of M_1 and hence the algorithm goes into nontermination. Therefore Approach 2 need not always terminate whenever Approach 1 gives precise results.

5.4 Theorems on correctness and termination of Appraoch 2

We now state and prove the various claims that we made earlier in this section. All the theorems assume that the input counter system M is finite-branching. We also assume the existance of black-box API's that were described in Section 5.1.

In the following theorem, we prove that the routine *computeGlobal* precisely computes the set of states that satisfy $\forall k \geq 0$ $pre^k(M_1, \phi)_{(k)}$ and therefore the set of states that satisfy the property $\mathbf{EG}\phi$ in the counter system M.

Theorem 4. *Given a counter system M and a set of states ϕ, Algorithm 4 returns precisely the set of states that satisfy $\boldsymbol{EG}\phi$ in M whenever it terminates.*

Proof. Let X be a Presburger formula representing a set of states in the counter system M_1, returned by Algorithm 4 upon termination.

Claim A: Consider a state $\boldsymbol{s} \in X$. We will prove that $\boldsymbol{s} \models \forall k \geq 0.\ pre^k(M_1, \phi)_{(k)}$ where M_1 is a refinement of M with respect to to ϕ. Then by Theorem 3, it follows that $\boldsymbol{s} \models \mathbf{EG}\phi$ in the counter system M.

Let N be the last flattening of M_1 that was produced by Algorithm 4 prior to termination. Then, $\boldsymbol{s} \in X$
$\implies \boldsymbol{s} \in \forall k \geq 0.\ pre^k(N, \phi)_{(k)}$. Since N is a flattening of M_1, $traces(N, \phi) \subseteq traces(M_1, \phi)$. Therefore every state which satisfies $\forall k \geq 0.\ pre^k(N, \phi)_{(k)}$ will satisfy $\forall k \geq 0.\ pre^k(M_1, \phi)_{(k)}$.
$\implies \boldsymbol{s} \models \forall k \geq 0.\ pre^k(M_1, \phi)_{(k)}$.
Then by Theorem 3, $\boldsymbol{s} \models \mathbf{EG}\phi$ in the counter system M.

Claim B: If a state $\boldsymbol{s} \notin X$ then $\boldsymbol{s} \not\models \mathbf{EG}\phi$ in M.
Let N be the final flattening of M_1 using which the set X was computed. $\boldsymbol{s} \notin X$
$\implies \boldsymbol{s} \notin \forall k \geq 0.\ pre^k(N, \phi)_{(k)}$. Therefore by definition of $\forall k \geq 0.\ pre^k(N, \phi)_{(k)}$ there is no infinite length path in N starting from $\boldsymbol{s}$. Now since $\boldsymbol{s} \notin X$ and since we terminate only when $traces(M_1, \phi - X) = traces(N, \phi - X)$ it follows that there is no infinite length path in M_1 from the state $\boldsymbol{s}$. Then by theorem 3 $\boldsymbol{s} \not\models \mathbf{EG}\phi$ in the counter system M. □

Theorem 4 proves that Algorithm 4 gives precisely the set of states that satisfy the property $\mathbf{EG}\phi$. The following lemma states that Algorithm 4 under-approximates the set of states that satisfy the property $\mathbf{EG}\phi$ in the input counter system M when forced to terminate.

Lemma 2. *Given a counter system M and a set of states ϕ, Algorithm 4 returns an under-approximation of the set of states that satisfy $\boldsymbol{EG}\phi$ in the counter system M when prematurely terminated.*

Proof. The proof follows directly from Claim A in the proof of Theorem 4. □

Theorem 4 and Lemma 2 give the characteristics of correctness and precision of Algorithm 4, both when it terminates and when forced to terminate.

Termination characteristics of Approach 2: For the rest of this section, we give a characterization of the class of systems on which Approach 2 necessarily terminates[1].

[1]K Vasanta Lakshmi, who is a PhD student with Dr. K V Raghavan, has come up with this characterization and I'm including it for the sake of completeness.

Though the characterization is not syntactic, it helps to prove that trace-flattable systems are a strict subset of the class of systems that satisfy the characterization.

Note that the implementation of the routine *computeGlobal*$(M, \phi, label)$ provided in Approach 2 computes X, of the set of states that satisfies $\mathbf{EG}\phi$ in the counter system M. The algorithm terminates whenever the traces in the flattened system N from the set of states $\phi - X$ are the same as in the system M_1, which is a refinement of M with respect to to ϕ. That is, $traces(M_1, \phi - X) = traces(N, \phi - X)$. That is, by definition of X, every state in X satisfies $\mathbf{EG}\phi$ and every trace from a state which does not satisfy $\mathbf{EG}\phi$ must be present in the system M_1 and its flattening N. When there exists such a flattening N of M then Algorithm 4 will terminate. This is formally stated and proved in Theorem 5. We then prove in Lemma 3 that trace flattable systems, that were addressed by Demri et al [13], are a strict subset of the class of systems that are defined by the characterization.

Theorem 5. *Given a counter system M, let M_1 be the refinement of M with respect to to ϕ. Let X be the set of states that satisfy $\boldsymbol{EG}\phi$ in M. Algorithm 4 terminates if and only if (a) there exists a flattening N of M_1 such that every state that satisfies $\boldsymbol{EG}\phi$ in M has at least one infinite trace in N (b) the systems M_1 and N are trace equivalent with respect to the set of states that do not satisfy $\boldsymbol{EG}\phi$ that is $traces(M_1, \phi - X) = traces(N, \phi - X)$ and (c)pre^k and pre^* queries terminates on flattenings of M_1.*

Proof. Let N be a flattening of M_1 such that (a) every state that satisfies $\mathbf{EG}\phi$ in M, has at least one of in N and (b) the systems M_1 and N are such that $traces(M_1, \phi - X) = traces(N, \phi - X)$. Since there is at least one infinite trace from every state $\boldsymbol{s}$ in X in the counter system N, $\boldsymbol{s} \in \forall k \geq 0.\ pre^k(N, \phi)_{(k)}$. Also $traces(N, \phi - X) = traces(M_1, \phi - X)$. Therefore the trace equivalence check in Line 8 in Algorithm 4 will return *true* and hence algorithm will terminate. The only thing that remains to prove is that we will definitely find such a flattening N if it exists. Note that we explore different flattenings of increasing lengths in a breadth first manner. Since N is a flattening of M_1, N is of finite length and hence breadth first generation of flattenings (in Line 5) will find it. □

In the following lemma, we prove that we terminate on all trace-flattable systems.

Lemma 3. *Given a counter system M, if M is trace flattable then the routine computeGlobal definitely terminates for any given ϕ, i.e, the class of systems that the routine computeGlobal terminates is a strict superset of trace-flattable systems.*

Proof. If the system M is trace-flattable then there exists a flattening N of M such that $traces(M, \phi^{init}) = traces(N, \phi^{init})$. Let X be the set of reachable states that satisfy $\mathbf{EG}\phi$ in the counter system M. Due to trace equivalence, it follows that X precisely represents the set of states that satisfy $\mathbf{EG}\phi$ in M. Also from trace equivalence it follows that $traces(M, \phi - X) = traces(N, \phi - X)$. Therefore Algorithm 4 will definitely terminate when it generates the flattening N with the precise answer X. The generation of X is guaranteed as per Theorem 5 (Note that the algorithm may terminate even before the system N is generated).

Also all the systems that we show in Chapter 6 including the running example shown in Figure 1.1 are not trace-flattable and Algorithm 4 terminates on these systems. Therefore Algorithm 4 terminates on the class of counter systems that is a strict superset of trace-flattable systems. □

Note that the characterization that we state is on the system M_1. M_1 is the refinement of M with respect to to ϕ. The formula ϕ is which represents a set of states is dependent on the temporal property that is provided as the input to Algorithm 1(Section 3). Therefore the characterization we state in on the system and the temporal property. But if the system M is trace-flattable then we terminate on *any* given property.

Chapter 6

Empirical Work

Many practical systems such as broadcast protocols, cache coherence protocols and synchronization protocols can be modeled as counter systems. We give some practical examples of counter systems that have natural temporal properties that can be expressed in *CTL*, and checked by Algorithm 1. We discuss the implementation issues and the black-boxes used in implementing the routines *computeUntil* and *computeGlobal*.

6.1 Details of our Implementation

We have a partial implementation of Algorithm 1. The implementation is intended only to do precise computations and hence the third argument *label* has to be precise in every recursive call. We therefore omit this parameter from the input as well as the enumerator that indicates the direction of approximation from the output. That is if the algorithm terminates, then we will have precise results. The routine *computeUntil* requires black-boxes to compute $pre^*(M, \phi)$ for a given counter system M. We use the tool FAST [12] for implementing the black-box *preStar*. The implementation of this routine is complete and is used in the evaluating all *until* the properties mentioned in this section.

We use Approach 2 for the implementation of the routine *computeGlobal*. The routine *computeGlobal* requires the implementation of the black-boxes $compute\_pre^k$ and $isTraceFlattening(M, N, \phi)$, where N is a flattening of the counter system M. The black

box *compute_pre*^{k} computes a Presburger formula $pre^k(N,\phi)_{(k)}$ for the given flat counter system N. We use the tool FAST to compute this for flat systems. For a given input counter system M, FAST was modified to iteratively generate flattenings N of M with increased sizes. During each iteration, we modified the acceleration engine in FAST to return the formula for $pre^k(N,\phi)_{(k)}$ for a given set of states ϕ. We then universally quantify with $\forall k \geq 0$ using the API's available in the FAST toolkit. Then we use the black-box $isTraceFlattening(M,N,\phi)$ proposed by Demri et al, [13] to check whether $traces(M,\phi - X) = traces(N,\phi - X)$ where X is the set of states that satisfy $\mathbf{EG}\phi$ in M. They do not provide this implementation and therefore we have implemented that in the tool FAST and have modified the termination condition of FAST based on the result of the above check. This implementation which involves computation of $\forall k \geq 0$. $pre^k(N,\phi)_{(k)}$ and $isTraceFlattening(M,N,\phi)$ is in the testing phase. Hence the results for global properties that we mention in this chapter are manually simulated and we are yet to obtain the results from the implementation.

6.2 Benchmark Examples

The systems that we consider for evaluation are mostly cache coherence protocols, broadcast protocols, etc. These protocols have been modeled as counter systems and are available with the FAST tool. These protocols have been studied extensively for analyzing the safety properties (i.e, reachability). All these systems are not trace-flattable and they do not fall in the class of systems addressed by any of the existing approaches [13, 2, 18]. We identify several natural temporal properties on these examples and we discuss each class of these examples in detail. Though the systems were available as a part of the fast tool, we identified the temporal properties ourself on these systems based on the domain knowledge of the systems.

Cache Coherence Protocols: A coherency protocol is a protocol which maintains the consistency between all the caches in a system of distributed shared memory. The protocol maintains memory coherence according to a specific consistency model. Many

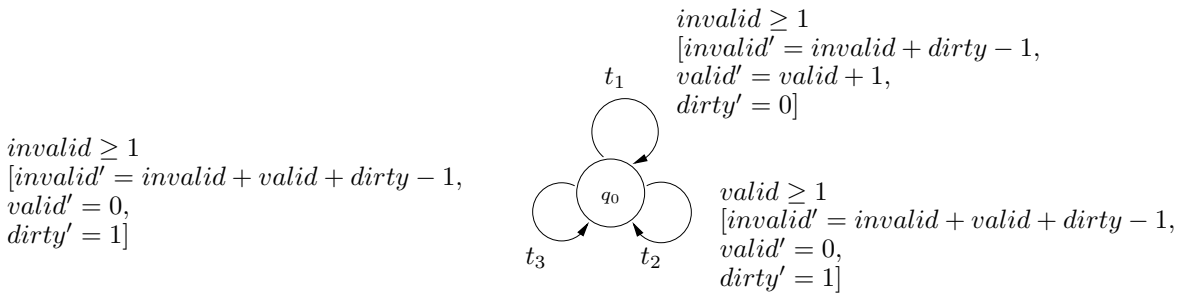

Figure 6.1: MSI Cache Coherence protocol

different cache coherence protocols can be modeled as counter systems. Each of these protocols can be modeled as a counter system. Each cache line in a given protocol can be in different states. The counters in the system represent number of processors in a given state for a particular cache line. The transitions in the system represent reads and writes by a processor whose cache line is in a specific state. For example consider the MSI protocol shown in Figure 6.1. Each counter in this automaton *Modified*, *Shared* and *Invalid* represents the number of processors in modified, shared and invalid states for a given cache line. Transition t_1 represents a read miss by a processor in invalid state. The actions show that all processors that previously had a modified copy write to the memory and move to an invalid state $((invalid' = invalid + dirty - 1) \wedge (dirty' = 0))$, and the requesting processor gets a copy from memory and moves to shared state $(valid' = valid + 1)$.

We briefly discuss each cache coherence protocol and a sample temporal property on each of these protocols. Note that all these example systems are modeled as counter automata and are available as a part of the FAST tool.

1. SYNAPSE.fst: This automaton models the MSI cache coherence protocol. The protocol requires that if a cache line of a processor is in a shared state then it remains in the shared state until there is a write to the cache line along all paths. This is a property that involves both *global* and *until* queries.

2. MESI.fst: This protocol has a counter that models the number of processors in the exclusive state for a given cache line in addition to the ones that were described by the MSI protocol. The protocol requires that whenever a cache line of a processor is in the exclusive state, no other processors have the same cache line in the modified state along all paths. It involves a *reachability* query which translates to an *until* query in existential normal form.

3. MOESI.fst: In addition to the four common MESI protocol states, this protocol has a fifth "owned" state representing a cache line that is both modified and shared. A property that holds in this system is that a cache line in this protocol remains in the exclusive state until a processor writes to the cache line. This is posed as an *until* query.

4. DRAGON.fst: The dragon protocol is same as the MOESI cache coherence protocol. The shared dirty state has the same semantics as that of the owned state in the MOESI protocol. The dirty state is similar to modified state of the MOESI protocol. A property of this protocol is that when the cache line of every processor is there in the shared state, it remains in the shared state until the processor in the shared dirty state writes to the cache cache line. This is an *until* query in the system.

5. Berkeley.fst: The protocol has four cache states: invalid, unowned, exclusive, and non-exclusively owned. The invalid and the exclusive cache states have the same meaning as in MESI protocol. The unowned state is similar to shared state of the MESI protocol. In this protocol, a cache block in the non-exclusive owned state may be updated only after informing other caches which result in invalidation of cache blocks of other processors. This is expressed as a query on the *next* states.

6. FIREFLY.fst: The firefly protocol is similar to the MESI protocol. It differs from MESI in the fact that a cache line is never invalidated. Therefore, in this system, when there are no processors which have a cache line in the invalid state, the

number of processors with their cache lines in the invalid state continues to be zero across all execution sequences infinitely. This is a *reachability* query in the system.

Other systems: We now focus on the next set of examples which are specific systems that are modeled as counter systems.

7. centralserver.fst: This is an automaton that models the client and server objects in a client server system. There are two resources 'C' and 'D' and each of these resources have respective queues and multiple processing units. The central server puts every request from the the client to respective queues and allocates the resource when there are pending requests in the wait queue of each resource. The values of different counters in the system model the number of pending requests in the queue and also the number of requests that are currently being served for each resource. There are counters that save the state of other counters when the server is stopped and the state is restored when the server starts. A property of this client server model is that every request for a resource from the client is be granted. If we consider the queues and the processing units of the resource D then along all execution paths, the processing units must be serving at least one request till there are no more requests in the waiting queue for D. Our algorithm says that every reachable state in this system does not satisfy this property. This is because of two major reasons: a) The server might keep allocating the requests to the resource C from the clients without granting the resource D even when there are pending requests in the waiting queue of D. b) The server might be in its loop of processing stopping and restarting without servicing both the requests. This involves both *until* and *global* queries.

8. Lift.fst: This system models the behavior of a lift. The counters in the system model the current floor of the lift, the floor from or to which a request was made and the direction in which the lift is supposed to at each floor. Let us assume that the request was from a floor that was higher than the current floor. Then the counter which indicates the direction of movement of the lift at each floor

should contain the value which enables the lift to move upwards till the floor from which the request was made is reached along every possible path the lift can take. Model-checking this property involves both *until* and *global* queries.

6.3 Results

Sl No	Temporal Property	Satisfying states	flattening size	time taken (in ms)
1	$valid \geq 1 \implies \mathbf{A}((valid \geq 1)\mathbf{U}(dirty = 1))$	ϕ^{reach}	3	12
2	$\mathbf{AG}\begin{pmatrix} exclusive \geq 1 \implies \\ modified = 0 \end{pmatrix}$	ϕ^{reach}	–	9
3	$\mathbf{E}((exclusive = 1)\ \mathbf{U}\ (modified = 1))$	$((exclusive = 1) \vee (modified = 1)) \wedge \phi^{reach}$	–	9
4	$\mathbf{E}(\phi_1\ \mathbf{U}\ (\phi_2\ \vee \phi_3))^{\dagger}$	$(\phi_1\ \vee \phi_2 \vee \phi_3)\ \wedge \phi^{reach}$	–	1
5	$\mathbf{AX}\begin{pmatrix} invalid \geq 1 \wedge \\ exclusive = 1 \end{pmatrix}$	$(invalid = 0) \wedge \phi^{reach} \wedge (unowned + nonexclusive \geq 1)$	–	–
6	$\mathbf{AG}(invalid = 0)$	$invalid = 0\ \wedge \phi^{reach}$	–	4
7	$\mathbf{A}((busyD \geq 1)\mathbf{U}(waitD \geq 1))$	$(waitD \geq 1) \wedge \phi^{reach}$	5	4985
8	$\mathbf{A}((a \geq 1)\mathbf{U}(c = g))$	$((c = g) \vee ((a \geq 1)\ \wedge (c < g))) \wedge \phi^{reach}$	4	12

$^{\dagger}\phi_1 = \begin{bmatrix} invalid = 0 \wedge \\ shared\_dirty = 0 \wedge \\ shared \geq 1 \wedge \\ exclusive = 0 \wedge \\ dirty = 0 \end{bmatrix}$ $\phi_2 = \begin{bmatrix} invalid = 1 \wedge \\ shared\_dirty = 1 \wedge \\ shared \geq 0 \wedge \\ exclusive = 0 \wedge \\ dirty = 0 \end{bmatrix}$ $\phi_3 = \begin{bmatrix} invalid = 1 \wedge \\ shared\_dirty = 0 \wedge \\ shared \geq 1 \wedge \\ exclusive = 0 \wedge \\ dirty = 0 \end{bmatrix}$

Table 6.1: Non-Trace Flattable Counter Systems

Table 6.1 gives more information about the systems and the natural temporal properties that introduced in Section 6.2 along with certain aspects of performance of our approach. The number in the first column represents the corresponding counter system that was discussed in Section 6.2. The second column formally describes the property

Counter System	Temporal Property
train.fst	$\mathbf{EG}((b \geq s + 9) \wedge (d \geq 9))$
Dragon.fst	$\mathbf{EG}(dirty = 0)$
Illinois.fst	$\mathbf{EG}(valid >= 1)$
Futurebus.fst	$\mathbf{EG}(pendingR \geq 1)$
Futurebus.fst	$\mathbf{EG}(pendingW \geq 1)$
ttp2.fst	$\mathbf{EG}(df = cf)$

Table 6.2: Counter systems on which the properties are yet to be verified

these systems were expected to satisfy. The property was informally defined in Section 6.2 along with the existing benchmark. Some properties are written in a form other than existential normal form, mainly for the purpose of readability. These properties were converted to properties in existential normal form before they were being evaluated. The systems $1, 7$ and 8 have both until and global properties. The systems 3 and 4 have until properties and system 5 has a next state property. The properties in systems 2 and 6 are reachability queries. The third column gives the set of states that satisfy the temporal property. The fourth column indicates the flattening size. We mention it only for global properties. The flattening size is the size of the flattening that provided the precise result when the routine *computeGlobal* terminated during the evaluation of underlying global property. Note that flattening size is the number of transitions that are present in the flattening. For until and next state properties this was not evaluated and hence the entry in the table is marked with a $-$. The final column is the time taken by the routines *computeUntil* and *computeGlobal* measured in milliseconds.

We have identified several other systems and examples of global properties on these systems. These examples have not been solved or using our algorithm and we will be using these examples to evaluate the implementation. The systems are listed in Table 6.2. The properties were taken based on the behavior of the input counter system.

We describe every system and the corresponding property of each system shown in

Table 6.2. The first column gives the name of the counter system (as in the FAST toolkit) and the second column gives the formal representation of the property to be verified on these systems. All the properties involve only *global* queries.

1. train.fst: The counter system models the running status of a train. It indicates control states indicate whether the train is running, stopped or is on time or is running late. The counters together indicate whether train is speeding up, slowing down or has stopped. One of the requirement of this systems is that should be no path along which the train speeds up indefinitely.

2. Dragon.fst: This protocol was discussed earlier in this Section 6.2. We identify another temporal property that there exists no processor whose cache line is in the dirty state if there are no writes to the cache line.

3. Illinois.fst: This is same as the MESI cache coherence protocol. We want to prove the property that the cache line remains in a shared state infinitely if there are no writes to the cache line.

4. futurebus.fst: This is a coherence protocol for single or multibus multiprocessor system. In this protocol, the reads and writes are not performed as and when a read or write instruction is issued. All local read or write misses are given a higher priority over remote read or write misses. The remote reads and writes are queued. Therefore a requirement would be that every read and write in the queue is eventually serviced. Therefore such a system should falsify the property $\mathbf{EG}(pendingR \geq 1)$ and $\mathbf{EG}(pendingW \geq 1)$, where $pendingR$ and $pendingW$ are the counters that represent the number of pending reads and writes. Each of these is represented as a separate property in Table 6.2.

5. ttp2.fst: It models the Time Triggered Protocol (TTP) which was primarily designed for vehicular and industrial applications. The detailed working and modeling of the protocol is described in [19]. The protocol has a number of stati Therefore we are waiting for the implementation to be completed to evaluate these systems.

ons out of which some are faulty. In a situation where there are faulty stations, the working stations form a clique among themselves and communicate among other stations in the same clique. One of the requirements of the protocol is that the counters df and cf, that represent number of faulty systems, should be equal infinitely.

We have around 5 other systems and properties, from fast toolkit, which we would like to verify and have not been mentioned here. These properties (along with the ones mentioned in Table 6.2) were difficult to simulate manually. We do not have the solutions that are expected to be returned by Algorithm 1 on these systems.

Chapter 7

Related Work

The seminal work on temporal property checking in infinite-state systems is the one by Bouajjani et al. [2], for push-down systems. Given a CTL property $\mathbf{E}(\psi_1\ \mathbf{U}\ \psi_2)$ they propose to refine the system with ψ_1 and compute a $pre^*(\psi_2)$ on this refined system. For a global CTL property of the form $\mathbf{EG}\psi_1$, they refine the system with ψ_1 and compute the set of states that satisfy $\psi_1 \wedge pre^*(pre(\psi_1))$. The structural properties of pushdown systems ensure that these computations return the precise set of states that satisfy the corresponding CTL property. Our refinement idea for answering "until" properties using refinement is similar to theirs, although we compute pre^* using a provided reachability black box. Using an existing reachability black-box helps us to answer temporal properties in a larger class of counter systems. However, their approach for "global" properties, when applied to counter systems directly will only return the set of states in the refined system which are part of a concrete cycle or are in pre^* of a concrete cycle. This in general gives an under-approximation. The solution would not contain the set of states that have infinitely long paths along which no state repeats. We provide two approaches to answer "global" properties that specifically address counter systems.

The initial work in model-checking for counter systems was done by Bultan et al. [20]. They answer safety and liveness properties in counter systems. This approach is based on symbolically encoding the transition relations and the states of counter system using Presburger formulas. Using this representation, they answer the "next" state properties

by doing a *pre* operation. For the fix-point computations of "global" and "future" (which is a simpler case of "until" property) properties, they repeatedly apply *pre* and *post* operations, without using any accelerations (i.e., pre^* or $post^*$). These fix point computations need not terminate in general. For instance, their algorithm will not terminate on the example given in Section 1. They do not answer the "until" properties in general and hence they do not address the entire fragment of CTL, which we do. Our approach uses accelerations, which help us to terminate on a much wider class of systems than theirs.

The closest work to ours is by Demri et al. [13]. They propose an algorithm that takes a flat counter system M and a temporal property ψ and returns a Presburger formula that represents the set of states in M that satisfy ψ. Their approach is as follows: first they compute a Presburger formula ϕ_1 which precisely represents the set of traces in M. This, in general, is possible only with flat systems whose cycles can be accelerated. Using ϕ_1 they construct a Presburger formula ϕ_2 that represents the temporal property ψ. Finally they construct a Presburger formula ϕ to check if along all valid paths from a state, ϕ_2 holds. The set of states that satisfy ϕ satisfy the temporal property ψ. They can check precisely CTL^* properties for flat systems whose cycles can be accelerated.

They also give a semi decision procedure to model-check a fragment of CTL^* with no nested path quantifiers for non-flat but trace-flattable systems. They use the heuristics in FAST to generate a flattenings of the given system M by unrolling the nested loops in M. Once they generate a flattening N of M, they check whether $traces(M, \phi^{init}) = traces(N, \phi^{init})$. They provide a decision procedure to check whether check whether a given system M and a flattening N of M have the same set of traces from a set of input states ϕ_i. If $traces(N, \phi^{init}) = traces(N, \phi^{init})$ then they model-check the system N. Otherwise they keep on generating a new flattenings. The way FAST explores different flattenings is breadth first. Hence they are guaranteed to find a trace flattening of the system if one exists.

The main difference of our approach from theirs is that they are not inductive; that is they try to answer the entire property in one go rather then solving each sub-property

of the given temporal property on the given system M at a time. The major advantage of this approach is that finding the set of states that satisfy the sub-property of the given temporal property in a system M requires only that the subset of the traces in M that are pertinent to the sub-property be preserved in N. For example if a property $\mathbf{EG}\phi$ has to be verified in a counter system M it is not necessary to generate the flattening which preserves traces from the set of states that do not satisfy ϕ. Preserving a subset of traces enables us to check precisely CTL on certain systems which only have flattenings do not preserve all traces. Also, we necessarily terminate on all flat and trace flattable systems, which is the class of systems addressed by Demri et. al [13] We are also able to handle approximations whenever necessary, which is something they do not address. The example discussed in Section 1 is a non-trace flattable system, on which we can solve various properties precisely. So are the examples that we show in Section 6.

Most recently, Cook at al [21] gave an approach to model-check CTL in programs. They first give a model-checking procedure to verify the universal fragment of CTL (fragment of CTL with only "forall" path quantifiers) in C programs. They take a program, a set of initial states and a temporal property ψ in the universal fragment of CTL as input and return a boolean value. The returned value is *true* if all initial states satisfy the property. Given a program P whose transition relation is R and a set of initial states I, they first generate a program P', which returns *false* in case the initial states of the program do not satisfy the property. They can use the program analysis tools to infer that the program never returns *false*. They then extend their work in [22] to answer the existential fragment of CTL (and hence the entire CTL). The basic idea behind this extension is their observation that answering existential quantification can be reduced to answering universal quantification if the state space of the system is restricted appropriately. When an existential query is asked, instead of a universal query, the counter example returned may be used to repeatedly prune the state space of the system thereby finally reduce to answer the universal quantification on a restricted state space. Their approaches for answering CTL properties have a restriction that the transition relation has a finite number of ranking functions, i.e, the transition relations

are disjunctively well founded. One major difference of our work from their's is that they address the problem of *local* model-checking: that is, given a set of initial states in a transition system and a temporal property ψ, they say whether the initial states satisfy ψ. In our case we address a more general problem of *global* model-checking. That is, given a counter system M and a temporal property ψ, we find the set of states in the counter system that satisfy the property ψ. We are currently unclear on the class of systems on which their approach terminates and hence it is difficult for us to compare our class of systems with theirs.

Another approach for model-checking of counter systems was proposed by Bozelli et al. [18]. They consider temporal properties in $GCCTL^*$, which is the logic similar to CTL^*, whose basic propositions in the properties are gap order constraints. These constraints can represent the lower and upper bounds on the values of counters and the equality or difference between pairs of counters. They prove that model-checking the entire fragment of $GCCTL^*$, is undecidable. They consider temporal formulas in $E\text{-}GCCTL^*$ and $A\text{-}GCCTL^*$ where the basic propositions are conjunctions of *gap order constraints* on counters. They prove that it is decidable to model-check these formulas on counter systems where guards and actions of transitions are conjunctions on gap-order constraints. The logics CTL and $E\text{-}GCCTL^*$ as well as $A\text{-}GCCTL^*$ are incomparable. The major difference about our approach and theirs is that the ability to incorporate black-boxes. We answer CTL properties on systems for which black boxes are available and the systems have a finite branching property. There are some gap-order systems on which we terminate and there are gap order systems that we do not handle because they violate the finite branching property. Our class of systems is incomparable with theirs as gap order constraints cannot model actions of the form $x' = x + y + c$. Most of the example systems shown in Section 6 cannot be analyzed by their approach because they use non-gap-order constraints to model their transitions.

Ball et al. [23] give a framework to create abstractions of systems such that a property on the system can be proved to be *false*. They give sufficient conditions under which various forms of temporal properties can be falsified. They do not provide an algorithm

for model-checking.

The work on computing loop bounds by Gulwani et al. [24] proposed a loop flattening algorithm which is close to our algorithm for finding trace flattenings of a given system. The focus of their paper is mostly on finding precise transitive closure of programs and they do not provide any algorithm for model-checking CTL properties.

We now discuss some literature around reachability analysis of counter automata, which can potentially be used as black-boxes in our algorithms. Finkel et al[10], give a class of counter systems called Finite Linear Systems, where every transition defines a Presburger linear function on the set of counters and the multiplicative monoid generated by the actions of the counters is finite. This class is a superset of the class of systems considered by Boigelot [25]. They give a procedure to compute a Presburger formula representing the transitive closure of a loop as a pre-post relation on counter values. The acceleration techniques described by Finkel at al [10] can be used for both forward and backward reachability analysis. These techniques are implemented in the tool FAST [26].

Comon et al [8] show that reachability is decidable for flat counter systems whose actions are conjuncts of the form $x\#y+c$ where $x, y \in C \cup C'$, $c \in \mathbb{Z}$ and $\# \in \{=, \geq, \leq\}$. The class of systems that is considered by Comon et al is incomparable with ours since they cannot express actions of the form $x + y + z = c$ where x, y, z are counters in the system. On the other hand the actions that they consider are relations while the actions of the transition that we consider here are functions. Hence we cannot express an action of the form $x' \leq x+5$. We cannot directly extend our algorithm to these class of systems since there can be infinite branching at a node. But our algorithms can be applied to a subclass of these systems that have finite branching property.

Ibarra et al [27, 9]. give a class of multi counter systems called reversal bounded multi-counter machines where every counter alternates between increasing and decreasing modes finite number of times. Reachability is decidable in reversal bounded multi counter systems. We are yet to explore and check whether Algorithm 1 can be applied to such systems.

Chapter 8

Key Aspects of our Approach

In this section we compare and contrast our approach with the existing techniques in literature along with bringing out the key aspects of our approach that enable us to perform better than existing techniques. We also describe in detail our contributions.

The inductive nature of our algorithm allows us to answer temporal properties in systems that are not addressed by the existing techniques. All the example systems that we consider in the thesis are neither flat nor trace-flattable systems. The approach by Demri et al [13] needs a single "global" flattening which preserves all traces in the input system. On the other hand we have the flexibility to explore different flattenings of M for different sub-properties of the input temporal property each of which preserves a possibly different subset of the traces from the input system M. This feature is possible because of *refinement*. When a non-trace-flattable system is refined, the system could essentially become flat or trace-flattable. Therefore we do better than the previous approaches which do not use refinement even on non-nested CTL properties. Thus this is an orthogonal contribution in addition to inductiveness. However refinement is possible only because of the fact that the algorithm is inductive. This is because each sub-property of the input temporal property might require a different refinement and therefore a different set of traces which are necessary for solving a sub-property are preserved. Therefore flat and trace-flattable systems, where reachability is decidable, are a special case for us, wherein we terminate with precise results (like them).

We also incorporate approximations to our algorithm. In this thesis we focus on under-approximation routines for answering "global" and "until" properties. The existing approaches try to answer the property precisely and with that objective they might go into non-termination. On the other hand, we try to incorporate approximations and output the direction of approximation of the set of states that satisfy the temporal property.

Precision, Correctness and Termination. We have proved various aspects of correctness, precision and termination of our approach. For "until" properties we algorithm gives precise results as long as the underlying reachability technique terminates and gives precise results. However on forced termination we get under-approximated results even in cases where black-boxes do not terminate. We give a characterization of the set of systems on which our algorithm terminates. The set of systems that fall under this characterization are a strict super-set of flat and trace-flattable systems which were targeted by Demri et al [13].

For "global" properties, our two different approaches have different characteristics. The first approach terminates in all cases and in general, gives an under-approximation of the set set of states that satisfy the global property. The second approach need not terminate in all cases, but when ever it terminates, it returns precisely the set of states that satisfy the given global property. During each iteration, the algorithm computes a the set of states that definitely satisfy the global property in the given system. Hence forced termination of the algorithm would result in an under-approximated solution to the set of states that satisfy the global property.

The existing techniques have the ability to give only precise results and they do not incorporate approximations. This is because they try to answer the temporal property at one go along the paths in the system, with the ability to give only precise answers.

Usefulness: We show that several example systems provided by other researchers [12] to demonstrate applicability of counter systems to modeling realistic protocols are amenable to our model-checking technique. These examples are addressed by previous reachability

techniques [10], but are not in the class addressed by previous model-checking techniques for *CTL* properties. The properties that we identify on these systems are natural temporal properties. Our approaches terminate on these systems and properties, and we precisely answer the properties in each case using FAST [12] as the black-box for reachability analysis.

Chapter 9

Conclusions

In this thesis we give an approach to model-check CTL temporal properties on counter systems where a reachability-analysis technique is available as a black box. Like previous techniques, we are precise on flat and trace-flattable systems. We also terminate on systems that are not trace-flattable. However, unlike previous techniques, we provide approximation techniques to address systems on which we do not terminate. Note that we provide only the under-approximation versions of the routines. The routine that over-approximates the set of states that satisfy a *global* property is given by Lakshmi et al in [14]. We also give an empirical validation of our algorithm using real world examples and some natural temporal properties on these systems where our approach is able to answer precisely. Existing approaches cannot answer these temporal properties on these systems.

We would like to mention a few directions for future work. We would like to extend our approach such that whenever the underlying reachability-analysis technique terminates (with full precision) on a system M, then our approach also provably terminates on it (with full precision), either for all CTL properties, or at least a significant fragment of these properties. Secondly, we would like to expand the class of temporal properties that we address to include LTL and CTL^*. Finally, we would also like to explore the suitability of using other reachability analysis techniques [9, 11, 8] as a black box. The major reason why we need to study these classes separately is that the refinements in

our algorithm can make the refined counter system system fall out of the class of counter system eventhough the input counter system falls within the class addressed by these black boxes. It might be necessary to impose certain restrictions on the basic propositions in the temporal properties and coming up with these restrictions will be a challenging task.

Bibliography

[1] R. Jhala and R. Majumdar, "Software model checking," *ACM Comput. Surv.*, vol. 41, pp. 21:1–21:54, Oct. 2009.

[2] A. Bouajjani, J. Esparza, and O. Maler, "Reachability analysis of pushdown automata: Application to model-checking," in *Proceedings of the 8th International Conference on Concurrency Theory*, CONCUR '97, (London, UK, UK), pp. 135–150, Springer-Verlag, 1997.

[3] J. Esparza, "Decidability of model checking for infinite-state concurrent systems," *Acta Informatica*, vol. 34, pp. 85–107, 1997. 10.1007/s002360050074.

[4] J. E. M. Clarke, O. Grumberg, and D. A. Peled, *Model checking*. Cambridge, MA, USA: MIT Press, 1999.

[5] A. Finkel, B. Willems, and P. Wolper, "A direct symbolic approach to model checking pushdown systems (extended abstract)," *Electronic Notes in Theoretical Computer Science*, vol. 9, no. 0, pp. 27–37, 1997.

[6] T. Reps, S. Schwoon, S. Jha, and D. Melski, "Weighted pushdown systems and their application to interprocedural dataflow analysis," *Science of Computer Programming*, vol. 58, no. 1, pp. 206–263, 2005. 10th International Static Analysis Symposium.

[7] R. Alur, K. Etessami, and P. Madhusudan, "A temporal logic for nested calls and returns," in *TACAS*, vol. 2988, pp. 467–481, 2005.

[8] H. Comon and Y. Jurski, "Multiple counters automata, safety analysis and presburger arithmetic," in *Proceedings of Computer Aided Verification (CAV)*, pp. 268–279, 1998.

[9] O. H. Ibarra, J. Su, Z. Dang, T. Bultan, and R. A. Kemmerer, "Counter machines and verification problems," *Theoretical Computer Science*, vol. 289, no. 1, pp. 165–189, 2002.

[10] A. Finkel and J. Leroux, "How to compose presburger-accelerations: Applications to broadcast protocols," *Technical Report, Laborataire Specification et Verification*, 2002.

[11] M. Bozga, C. Gîrlea, and R. Iosif, "Iterating octagons," in *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 337–351, 2009.

[12] "FASTer." http://altarica.labri.fr/forge/projects/faster/wiki/.

[13] S. Demri, A. Finkel, V. Goranko, and G. V. Drimmelen, "Towards a model-checker for counter systems," in *Proceedings of Automated Technology for Verification and Analysis (ATVA)*, pp. 493–507, 2006.

[14] K. V. Lakshmi, A. Acharya, and R. Komondoor, "Checking temporal properties of presburger counter systems using reachability analysis," *CoRR*, 2013. http://arxiv.org/abs/1312.1070.

[15] C. Darlot, A. Finkel, and L. Van Begin, "About fast and trex accelerations," *Electronic Notes in Theoretical Computer Science*, vol. 128, pp. 87–103, May 2005.

[16] S. Bardin, A. Finkel, J. Leroux, and P. Schnoebelen, "Flat acceleration in symbolic model checking," in *Proceedings of the Third international conference on Automated Technology for Verification and Analysis*, ATVA'05, (Berlin, Heidelberg), pp. 474–488, Springer-Verlag, 2005.

[17] "Fast." http://www.lsv.ens-cachan.fr/Software/fast/comparison.php.

[18] L. Bozzelli and S. Pinchinat, "Verification of gap-order constraint abstractions of counter systems," in *Proceedings of Verification, Model Checking, and Abstract Interpretation (VMCAI)*, pp. 88–103, 2012.

[19] S. Bardin, A. Finkel, and J. Leroux, "Faster acceleration of counter automata in practice," in *TACAS*, pp. 576–590, 2004.

[20] T. Bultan, R. Gerber, and W. Pugh, "Symbolic model checking of infinite state programs using presburger arithmetic," in *Proceedings of Computer Aided Verification (CAV)*, pp. 400–411, 1996.

[21] B. Cook, E. Koskinen, and M. Vardi, "Temporal property verification as a program analysis task," in *Proceedings of the 23rd international conference on Computer aided verification*, CAV'11, (Berlin, Heidelberg), pp. 333–348, Springer-Verlag, 2011.

[22] B. Cook and E. Koskinen, "Reasoning about nondeterminism in programs," in *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, PLDI '13, (New York, NY, USA), pp. 219–230, ACM, 2013.

[23] T. Ball, O. Kupferman, and G. Yorsh, "Abstraction for falsification," in *Proceedings of CAV*, pp. 67–81, 2005.

[24] S. Gulwani, S. Jain, and E. Koskinen, "Control-flow refinement and progress invariants for bound analysis," in *Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation*, PLDI '09, (New York, NY, USA), pp. 375–385, ACM, 2009.

[25] B. Boigelot, "Symbolic methods for exploring infinite state spaces," 1998.

[26] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci, "Fast: acceleration from theory to practice," *International Journal on Software Tools for Technology Transfer*, vol. 10, pp. 401–424, Sept. 2008.

[27] O. H. Ibarra, "Reversal-bounded multicounter machines and their decision problems," *J. ACM*, vol. 25, pp. 116–133, Jan. 1978.