Safety Verification for Linear Systems

Parasara Sridhar Duggirala[*]
University of Illinois at Urbana Champaign
duggira3@illinois.edu

Ashish Tiwari[*†]
SRI International
tiwari@csl.sri.com

ABSTRACT

An embedded software controller is safe if the composition of the controller and the plant does not reach any unsafe state starting from legal initial states (in an unbounded time horizon). Linear systems – specified using linear ordinary differential or difference equations – form an important class of models for such control systems. We present a new decidability result for safety verification of linear systems. Our decidability result assumes that the set of initial states and the set of unsafe states satisfy some conditions. When the set of initial and unsafe states do not satisfy these conditions, they can be overapproximated by sets that do satisfy the conditions. We thus get a counterexample guided abstraction refinement (CEGAR) procedure for the unconstrained safety verification of linear systems. Our new procedure performs abstraction-refinement on the initial and unsafe region, and not on the system itself. We present the new procedure and describe experimental results that demonstrate its effectiveness.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems; D.2.4 [**Software verification**]: Formal methods

General Terms

Design, Verification

Keywords

Hybrid Systems, Formal Methods, Verification, Abstraction

[*]Supported in part by NSF grants CSR-0917398 and SHF:CSR-1017483.
[†]Supported in part by DARPA under contract FA8750-12-C-0284 and under subcontract VA-DSR 21806-S4 under prime contract FA8650-10-C-7075.

1. INTRODUCTION

Safety is one of the central requirements for engineered systems. A system is deemed safe if it can never reach an undesirable (bad) state. Safety verification is one of the core problems that define the field of formal methods. It is a challenging problem and a wide variety of techniques exist for automatically proving safety.

A system's dynamics can be specified in continuous-time, discrete-time, or on a combination (hybrid time). Embedded control systems are often modeled as either continuous or hybrid dynamical systems. For hybrid dynamical systems, the classical approach for safety verification is based on iterative computation of the set of *all* reachable states of the system, but the procedure is typically exponential in the dimension of the state space. An alternate approach for safety verification, explicitly designed to deal with the large state space, is based on constructing *abstractions* of the system. An abstraction is created by defining an equivalence relation on the state space and merging states in one equivalence class to define one *abstract* state, and the dynamics are lifted to the abstract space. If the abstraction fails to prove safety, then it has to be refined into a finer abstraction. Analyzing the cause of failure usually suggests a possible refinement step. The refinement steps can be repeated until either we find an abstraction sufficient to prove safety, or we find a counterexample. This methodology is nicknamed counter-example guided abstraction refinement (CEGAR).

A safety verification problem has three components: a system defined by its state space and dynamics, an initial set, and an unsafe set. Existing abstraction-refinement techniques have focused only on the first component, namely the system. In this paper, we pursue a new technique that focuses on abstracting and refining only the second and third components of the safety verification problem. In our approach, the state space of the system is not abstracted. Instead, the initial set and the unsafe set are abstracted, and iteratively refined.

Let $\mathit{Reach}(\mathit{Init}, \mathit{Sys}, [0,\infty))$ denote the set of states reachable from the set Init of initial states following the dynamics of the system Sys in time interval $[0,\infty)$. The safety verification problem is about checking if

$$\mathit{Reach}(\mathit{Init}, \mathit{Sys}, [0,\infty)) \cap \mathit{Unsafe} = \emptyset \qquad (1)$$

Our approach for safety verification starts with constructing a coarse abstraction of the initial set and the unsafe set (Figure 1). We then check if the overapproximation of the unsafe set is unreachable from the overapproximation of the

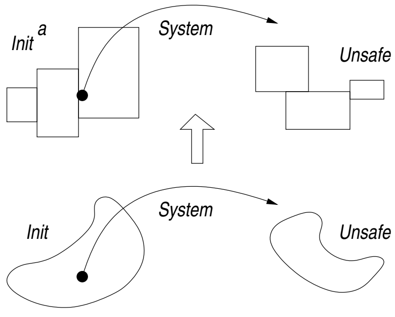

Figure 1: Our approach is based on over-approximating the initial and unsafe set in a careful way so that the safety verification problem for the abstract initial and unsafe set becomes decidable (for any linear system).

initial set; that is, if

$$Reach(Init^a, Sys, [0,\infty)) \ \cap \ Unsafe^a \quad = \quad \emptyset \qquad (2)$$

If we fail, then we analyze the cause of failure. We use that information to refine either the overapproximation of the unsafe set, or the overapproximation of the initial set. We repeat the process until either the safety property is proved, or it is disproved, or we run out of resources.

There are two key insights in our procedure that are crucial for its success. First, the over-approximation for the initial and unsafe sets are very carefully chosen (Figure 1). In fact, they are chosen to guarantee that the safety verification problem on the abstract initial set and the abstract unsafe set, namely Equation 2, becomes *efficiently decidable* for any *linear system*. In the CEGAR loop, when we have to refine the abstraction, we refine the abstractions of the initial and unsafe sets.

The second key insight, which is used in the decision procedure for the abstract problem, is that we compute the intersection of the reachable states and the unsafe region on the *time axis*, and not on the original state space. That is, we compute a time interval $[\mathtt{tmin}, \mathtt{tmax}]$ such that

$$\begin{array}{rcll} Reach(Init^a, Sys, [0,\infty)) \ \cap \ Unsafe^a & = & \emptyset & \text{iff} \\ Reach(Init^a, Sys, [\mathtt{tmin}, \mathtt{tmax}]) \ \cap \ Unsafe^a & = & \emptyset & \end{array}$$

Why is the computation of $[\mathtt{tmin}, \mathtt{tmax}]$ important? The reason is that, whereas the set

$$Reach(Init, Sys, [0,\infty))$$

is difficult to compute (it can be nonconvex), the set

$$Reach(Init, Sys, T),$$

for a fixed T, is easy to describe symbolically. We use the abstraction to identify a promising fixed value for T, and for this value of T, we compute reachability at time T on the concrete (un-abstracted) system. This enables us to detect if abstract counter-examples are genuine or spurious. Concretization of abstract counter-examples is one of the main challenges when implementing CEGAR for continuous and hybrid systems.

In this paper, we illustrate our new approach for safety verification on *linear dynamical systems*. We realize that this is a significant restriction – since the path from linear systems to hybrid systems is nontrivial. However, our investigation here is motivated by the following reasons. (i) Safety verification of linear systems is not a trivial problem. It has been studied before, and there are even some decidability results for classes of linear systems [1]. But, very few of the linear dynamics we come across in practice ever fall in the known decidable classes. (ii) A particularly important class of hybrid systems are those that contain linear dynamics in each mode. Before we can develop improved techniques for safety verification of such hybrid systems, it is important to fully understand the base case – when there is just one mode in the system. (iii) The idea of performing abstraction-refinement on the initial and unsafe set, and not on the system itself, is new. Hence, it is useful to first evaluate its effectiveness, and identify the challenges and issues, on a simple class of systems.

The main contributions of this work are as follows:

- We present a new decidability result. Specifically, we present a set of (sufficient) conditions under which the safety verification problem for linear (affine) systems becomes decidable (Section 4).
- We present a new approach for safety verification of arbitrary linear systems. When the (sufficient) conditions for decidability are violated, we abstract the original safety verification problem into a problem that falls in the decidable class identified above. Furthermore, if the abstraction is too coarse, we iteratively refine the abstraction, guided by counter-examples. We argue that our CEGAR approach is effective for solving certain "robust" instances of the safety verification problem (Section 5).
- We implemented a tool that implements the above procedure. We also present some performance results (Section 6).

2. AN EXAMPLE

We introduce a small and simple example here to illustrate the technical results in this paper. We will illustrate the decision procedure, and the abstraction-refinement procedure on this example.

Consider a system over two variables x, y given by:

dynamics	:	$\frac{dx}{dt} = x; \frac{dy}{dt} = 1;$
initial states *Init*	:	$2 \le x \le 3$ and $y = 0$
safe set $Safe_1$	:	$x > 5$ or $y < 1$
safe set $Safe_2$	:	$x > 7$ or $y < 1$

The variable x is exponentially increasing, and the variable y is a clock that keeps track of the time elapsed. Initially, x is in the interval $[2,3]$ and y is 0. The safety sets define limits on the value of x when at least one time unit has elapsed. The set $Safe_1$ says that when $y \ge 1$, then $x > 5$. The set $Safe_2$ says that when $y \ge 1$, then $x > 7$.

The example is chosen to be this simple to enable the reader to immediately observe that, if the system starts in the initial region *Init* specified above, it will remain inside the safety region $Safe_1$ always in the future; however, it will *not* always remain inside the $Safe_2$ set.

Our goal is to automatically prove (or disprove) that the system always remains inside some given *safe* set when starting from the given initial region.

One way to check safety is as follows: Consider the initial set $2 \le x \le 3 \wedge 0 \le y \le 0$, and the first unsafe set $Unsafe_1 :=$

$\neg$*Safe*, which is $x \leq 5 \wedge y \geq 1$. Consider the value of y. Initially, it is 0, and it is atleast 1 in the unsafe region. Since $dy/dt = 1$, it follows $y(t) = y(0) + t$. Hence, if the system reaches the unsafe region at time T, then T should lie in the interval $[tlb_y, tub_y]$, where

$$tlb_y \;:=\; \tfrac{1-0}{1} = 1 \qquad tub_y \;:=\; \tfrac{+\infty-0}{1} = +\infty$$

Hence, T must be in the interval $[1, +\infty]$. Now, consider the value of x. Initially, $2 \leq x \leq 3$ and in the unsafe region $-\infty \leq x \leq 5$. Since $dx/dt = x$, it follows that $x(t) = x(0) * e^t$. Hence, if the system reaches the unsafe region at time T, then T should lie in the interval $[tlb_x, tub_x]$, where

$$tlb_x \;:=\; 0 \qquad tub_x \;:=\; \ln(\tfrac{5}{2})/1 = 0.91629$$

Hence, T must be in the interval $[0, 0.91629]$. Therefore, T must be in the intersection of the two time intervals. But, the intersection is empty. Therefore, there can be no such T. Hence, the system is proved safe.

If we repeat the procedure on the second unsafe set $Unsafe_2 := (x \leq 7 \wedge y \geq 1)$, then we get the following interval for the time T of possible intersection with unsafe region:

$$[1, +\infty] \cap [0, \ln(\frac{7}{2})/1] = [1, 1.25]$$

Hence, the reach set intersects the unsafe region at time T in $[1, 1.25]$, and we declare the system unsafe.

In the above example, the linear forms being constrained in the *Init* and *Unsafe* formula, namely x and y, have a special property that makes it easy for us to compute the time interval (for intersection with unsafe region) above. Specifically, the variables x and y are monotonically changing. When the *Init* and *Unsafe* set are specified using such forms, we call the verification problem *aligned.* Aligned safety verification for linear system turns out to be decidable and the procedure used above is a decision procedure.

In case the initial set *Init* and the unsafe set *Unsafe* are not aligned, then we cannot use the procedure above: however, we can over-approximate (abstract) to an aligned problem and use the above procedure.

3. SAFETY FOR LINEAR SYSTEMS

In this section, we will formally define the safety verification problem for linear systems.

A *linear system* is a tuple $\langle X, f\rangle$ where X is a finite set of n real-valued variables whose dynamics are given by $\frac{dX}{dt} = f(X)$, where f is a n-dimensional vector of affine functions from $\Re^n$ to $\Re$. A *trajectory* is a mapping $\vec{x}$ from time $[0, \infty)$ to the state space $\Re^n$ that satisfies the system of differential equations $\frac{d\vec{x}(t)}{dt} = f(\vec{x}(t))$ for all $t \geq 0$.

Definition 1. [Safety Verification Problem] Given a linear system $\langle X, f\rangle$, an initial set $Init \subseteq \Re^n$, and a safe set $Safe \subseteq \Re^n$, the *safety verification problem* seeks to determine if there is a trajectory $\vec{x}(t)$ of the system and a time instance $T \geq 0$ such that $\vec{x}(0) \in Init$ and $\vec{x}(T) \notin Safe$.

4. A DECIDABILITY RESULT

In this section, we present a sufficient condition under which the safety verification problem becomes decidable. A key novelty here is that the sufficient condition will put restrictions on the initial and unsafe sets, but not on the (dynamics of the) linear system.

Let $\frac{dX}{dt} = AX + b$ denote the dynamics of the linear system. An *eigenform* of such a system is a mapping from the state space $\Re^X$ to $\Re$ that changes monotonically along the trajectories of the system.

Definition 2. [Eigenform] An *eigenform* for a linear system $\frac{dX}{dt} = AX + b$ is a mapping $V : \Re^X \mapsto \Re$ such that either

- $\frac{dV}{dt}$ is a constant λ in $\Re$, or
- $\frac{dV}{dt}$ is equal to λV for some $\lambda \in \Re$

where $\frac{dV}{dt}$ is $\sum_{x_i \in X} \frac{\partial V}{\partial x_i}\frac{dx_i}{dt}$. In both cases, we call the constant λ an *eigenvalue* for the eigenform V.

An eigenform V will be written as an arithmetic expression over the state variables X. For the example in Section 2, $V(X) := x$ is an eigenform and so is $V(X) := y$.

A linear system $dX/dt = AX + b$ has several eigenforms. Every left eigenvector of A corresponding to a nonzero real eigenvalue of A gives an eigenform: let $\vec{c}$ be a left eigenvector of A, that is,

$$\vec{c}^T A \;=\; \lambda \vec{c}^T$$

and now consider the function $V(X) := \vec{c}^T X + (\vec{c}^T b)/\lambda$. We have,

$$\frac{dV}{dt} \;=\; \vec{c}^T(AX + b) = \lambda\vec{c}^T X + \vec{c}^T b = \lambda V$$

Similarly, left eigenvectors of A corresponding to 0 eigenvalue also generate an eigenform: let $\vec{c}$ be a left eigenvector of A corresponding to eigenvalue 0, and consider the function $V(X) := \vec{c}^T X$. We have,

$$\frac{dV}{dt} \;=\; \vec{c}^T(AX + b) = \vec{c}^T b \in \Re$$

Note that the eigenforms $V(X)$ defined above are all linear functions. A linear system also has *quadratic* eigenforms. For simplicity, consider the linear system $dX/dt = AX$. If $\vec{c} + \iota\vec{d}$ is a left eigenvector of A corresponding to the complex eigenvalue $\lambda + \iota\omega$, then consider the function $V(X) := (\vec{c}^T X)^2 + (\vec{d}^T X)^2$. We have,

$$\begin{aligned}
\frac{dV}{dt} &= 2\vec{c}^T X\vec{c}^T(AX) + 2\vec{d}^T X\vec{d}^T(AX) \\
&= 2\vec{c}^T X(\lambda c^T - \omega d^T)X + 2\vec{d}^T X(\lambda d^T + \omega c^T)X \\
&= 2\lambda(\vec{c}^T X)^2 + 2\lambda(\vec{d}^T X)^2 = 2\lambda V
\end{aligned}$$

Thus, there exist several linear and quadratic eigenforms for a given linear system.

We next introduce a sufficient condition for decidability of the safety verification problem. The condition restricts the sets *Init* and *Safe*. The restriction is as follows: the sets *Init* and *Safe* are assumed to be specified as a Boolean combination of atomic predicates of the form $V \leq 0$ or $V \geq 0$, where V is an eigenform of the linear system. Instances of the safety verification problem that satisfy this restriction are called aligned safety verification problems.

Definition 3. [Aligned safety verification problem] A safety verification problem is *aligned* if there exists a finite set $\{V_1, \ldots, V_m\}$ of eigenforms of the linear system such that the initial set *Init* and the unsafe set *Unsafe* (unsafe set is

```
 1: procedure CHECKSAFETY($X, f, \vec{V}, \vec{\lambda}, \vec{l}_0, \vec{u}_0, \vec{l}_1, \vec{u}_1$)
 2:     Input: Linear system $\langle X, f \rangle$
 3:     Input: Vector $\vec{V}$ of eigenforms with eigenvalues $\vec{\lambda}$
 4:     Input: Init set $\vec{l}_0 \leq \vec{V} \leq \vec{u}_0$
 5:     Input: Unsafe set $\vec{l}_1 \leq \vec{V} \leq \vec{u}_1$
 6:     Output: "Valid" if Unsafe is not reachable from Init
 7:     Output: "Invalid" if Unsafe is reachable from Init
 8:     tmin = 0, tmax = $+\infty$
 9:     for each eigenform $V$ in $\vec{V}$ do
10:         let $l_0 \leq V \leq u_0$ be the constraint in Init
11:         let $l_1 \leq V \leq u_1$ be the constraint in Unsafe
12:         if $\frac{dV}{dt} = \lambda$ then
13:             [tlb,tub] = $([l_1, u_1] - [l_0, u_0])/\lambda$
14:         else if $\frac{dV}{dt} = \lambda V$ then
15:             [tlb,tub] = $(\ln \frac{[l_1,u_1]}{[l_0,u_0]})/\lambda$
16:         end if
17:         tmax = min( tmax, tub )
18:         tmin = max( tmin, tlb )
19:     end for
20:     if tmax < tmin then
21:         return "Valid"
22:     else
23:         return "Invalid", tmin, tmax
24:     end if
25: end procedure
```

Figure 2: Procedure used to solve the aligned safety verification problem.

the complement of the safe set *Safe*) can be written as a (finite) formula of the form

$$\bigvee_i \bigwedge_{j=1}^{m} (l_{ij} \leq V_j(X) \leq u_{ij}),$$

where l_{ij} and u_{ij} are either fixed bounds (in $\Re$) or $\pm\infty$.

The set of states that satisfy a constraint of the form $\bigwedge_j (l_{ij} \leq V_j(X) \leq u_{ij})$ will be called a "box". (Geometrically, it is a box that is aligned with the eigenforms. Aligned sets have been used before for controller synthesis of linear systems [2].) Since the initial region *Init* and the unsafe region *Unsafe* are both finite unions of boxes, we can solve the aligned safety verification problem by separately solving it for each choice of the initial box and the unsafe box. Using vector notation, we will compactly write the constraint $\bigwedge_j (l_j \leq V_j(X) \leq u_j)$ simply as $\vec{l} \leq \vec{V} \leq \vec{u}$.

EXAMPLE 1 (ALIGNED SAFETY VERIFICATION PROBLEM). *The safety verification problem defined in Section 2 is an instance of the aligned safety verification problem. This is because both the initial set and the unsafe set is specified by bounds on x and y, and both x and y are eigenforms.*

We now present a procedure to solve the aligned safety verification problem for the case when the initial set is a single box, and the unsafe set is also a single box. Subsequently, we will state and prove the correctness of the procedure, and thus obtain our decidability result.

The procedure for solving the aligned safety verification when the initial and unsafe set are both single boxes is given in Figure 2. The procedure uses computation over real numbers. We will discuss related computability issues later; for now, view the procedure in Figure 2 as a conceptual procedure.

The key idea in Procedure `checkSafety` is to compute the *time interval* when the system can potentially reach the unsafe region. Each eigenform will provide one estimate for when the system reaches the unsafe region. In detail, Procedure `checkSafety` works as follows: separately for each eigenform V, if the initial box has the bounds $l_0 \leq V \leq u_0$ for V, and the unsafe box has the bounds $l_1 \leq V \leq u_1$ for V, then we compute the time interval required for the value of V to change from $[l_0, u_0]$ to $[l_1, u_1]$. Since V_i changes monotonically, we can easily compute this time interval, say $[tlb, tub]$, using interval arithmetic (Line 13 and Line 15). We do not describe here all the details about how the interval arithmetic computation on Line 13 and Line 15 is performed, since it is all well known and (in any case) easily worked out. Just for illustration, for example, when $\lambda > 0$ and $dV/dt = \lambda$, the computation on Line 13 is performed as follows:

$$tub = (u_1 - l_0)/\lambda \quad (3)$$

$$tlb = (l_1 - u_0)/\lambda \quad (4)$$

For each eigenform, we get a different time interval $[tlb, tub]$. We finally intersect all these time intervals to get the final interval $[tmin, tmax]$ (Lines 17 and 18). If this final interval is nonempty, then we know that there is a time instant when the unsafe box is reached.

We next state and prove the correctness of the procedure `checkSafety`. The soundness of Procedure `checkSafety` follows from the soundness of interval arithmetic computation. Hence, we first state the latter as a lemma.

LEMMA 1. *Let V be an eigenform and $\vec{x}(t)$ be a trajectory such that $V(\vec{x}(0)) \in [l_0, u_0]$ and $V(\vec{x}(T)) \in [l_1, u_1]$. Then,*

- *if $[tlb, tub] = \frac{[l_1,u_1]-[l_0,u_0]}{\lambda}$ is computed using interval arithmetic (as on Line 13), and if $dV/dt = \lambda$, then $T \in [tlb, tub]$.*
- *if $[tlb, tub] = \frac{\ln(\frac{[l_1,u_1]}{[l_0,u_0]})}{\lambda}$ is computed using interval arithmetic (as on Line 15), and if $dV/dt = \lambda V$, then $T \in [tlb, tub]$.*

PROOF. We present the proof of one special case: when $dV/dt = \lambda$ and $\lambda > 0$. For this case, the interval arithmetic computation is worked out in Equations 3 and 4. Let Y be a fresh copy of the variables X. We prove that $T \in [tlb, tub]$ by observing that the formula

$$\begin{aligned} \forall X, Y: \quad & V(X) \in [l_0, u_0] \;\wedge\; V(Y) \in [l_1, u_1] \;\wedge \\ & V(Y) = V(X) + \lambda * T \;\wedge\; \lambda > 0 \implies \\ & \frac{l_1 - u_0}{\lambda} \leq T \leq \frac{u_1 - l_0}{\lambda} \end{aligned}$$

is valid in the theory of reals. The proof can be similarly worked out for all other cases. $\square$

We can use Lemma 1 to now state and prove soundness of Procedure `checkSafety`.

LEMMA 2. *[Soundness] If the procedure* `checkSafety` *returns "Valid", then the Unsafe box is not reachable from the Init box.*

PROOF. Suppose that the procedure `checkSafety` returns "Valid", but the system is unsafe. Since the system is unsafe, there is a time instance T and a trajectory $\vec{x}(t)$ s.t. $\vec{x}(0) \in \mathit{Init}$ and $\vec{x}(T) \in \mathit{Unsafe}$. We claim that T will always be in the interval $[\mathit{tmin}, \mathit{tmax}]$. This is clearly true initially. Now, consider an execution of the loop (Line 9) and let V be the eigenform being considered. Since $\vec{x}(0) \in \mathit{Init}$, it follows that $V(\vec{x}(0)) \in [l_0, u_0]$. Similarly, since $\vec{x}(T) \in \mathit{Unsafe}$, it follows that $V(\vec{x}(T)) \in [l_1, u_1]$. If $\mathit{tlb}, \mathit{tub}$ are the bounds for time interval computed (on Line 13 or on Line 15), then we need to prove that $T \in [\mathit{tlb}, \mathit{tub}]$. This, however, follows from Lemma 1.

Hence, after each iteration T will continue to remain in the interval $[\mathit{tmin}, \mathit{tmax}]$. Hence, the procedure `checkSafety` will return "Invalid", contradicting our assumption that is returns "Valid". □

Having proved that procedure always returns sound answers, we now move to completeness. We need the converse of the soundness lemma for interval arithmetic for proving completeness.

LEMMA 3. *If $[\mathit{tlb}, \mathit{tub}] = \frac{[l_1,u_1]-[l_0,u_0]}{\lambda}$ is computed using interval arithmetic (as on Line 13), then for any $T \in [\mathit{tlb}, \mathit{tub}]$, there is a value $v0$ and a value $v1$ such that $v0 \in [l_0, u_0]$, $v1 \in [l_1, u_1]$, and $v1 = v0 + \lambda * T$.*

Similarly, if $[\mathit{tlb}, \mathit{tub}] = \frac{\ln(\frac{[l_1,u_1]}{[l_0,u_0]})}{\lambda}$ is computed using interval arithmetic (as on Line 15), then for any $T \in [\mathit{tlb}, \mathit{tub}]$, there is a value $v0$ and a value $v1$ such that $v0 \in [l_0, u_0]$, $v1 \in [l_1, u_1]$, and $v1 = v0 * e^{\lambda * T}$.

PROOF. We present a proof for the second claim, since the proof of the first claim follows similar reasoning. Let $[\mathit{tlb}, \mathit{tub}] = \frac{\ln(\frac{[l_1,u_1]}{[l_0,u_0]})}{\lambda}$. The interval computation above is performed differently depending on the sign of l_0, u_0, l_1, u_1 and λ. Here, we present the proof for just one case: suppose l_0, u_0, l_1, u_1 and λ are all positive. For this case, the above interval computation is carried out as follows:

$$\mathit{tlb} = \ln(\frac{l_1}{u_0})/\lambda \quad (5)$$

$$\mathit{tub} = \ln(\frac{u_1}{l_0})/\lambda \quad (6)$$

Since $T \in [\mathit{tlb}, \mathit{tub}]$, using monotonicity of the exponential function, we conclude that $e^{\lambda * T}$ is in the interval $[e^{\lambda \mathit{tlb}}, e^{\lambda \mathit{tub}}]$. Using the definition of $\mathit{tlb}, \mathit{tub}$ in Equation 5 and 6 above, we conclude that the interval $[e^{\lambda \mathit{tlb}}, e^{\lambda \mathit{tub}}]$ is contained in the interval $[l_1/u_0, u_1/l_0]$. Together, we get $e^{\lambda T} \in [l_1/u_0, u_1/l_0]$. Hence, we can find $v_0 \in [l_0, u_0]$ and $v_1 \in [l_1, u_1]$ such that $v_0 * e^{\lambda T} = v_1$. This completes the proof for this case. We prove the result similarly for all other cases. □

Before we state and prove completeness, we need an additional technical assumption on the set *Init*. We say a box $\wedge_i(l_i \leq V_i \leq u_i)$ is *completely feasible* if the constraint $\wedge_i V_i = a_i$ is feasible (that is, has a solution) for every possible $a_i \in [l_i, u_i]$. For example, the box $(1 \leq x \leq 2) \wedge (2 \leq x + y \leq 3)$ is completely feasible, whereas the box $(1 \leq x \leq 2) \wedge (2 \leq x \leq 3)$ is not completely feasible (since $x = 1 \wedge x = 2$ is not feasible).

Now we state and prove the completeness of Procedure `checkSafety`.

LEMMA 4. *[Completeness] If the Unsafe box is not reachable from the Init box, then the procedure `checkSafety` returns "Valid", assuming Init is completely feasible.*

PROOF. Suppose not. Then, the system is safe, but the procedure returns "Invalid". Since the procedure returns "Invalid", there must be a T in the final time interval $[\mathit{tmin}, \mathit{tmax}]$ computed by the procedure. Using the assumption that *Init* is completely feasible, we will find an initial point $\vec{x}(0) \in \mathit{Init}$ s.t. starting from $\vec{x}(0)$, the state reached at time T, namely the state $\vec{x}(T)$, is in *Unsafe*. This will yield a contradiction.

To find $\vec{x}(0)$, we first find values for $V(\vec{x}(0))$ for each eigenform V. Consider an eigenform V. Let $l_0 \leq V \leq u_0$ be the constraint in *Init* and let $l_1 \leq V \leq u_1$ be the constraint in *Unsafe* corresponding to this eigenform.

Now, let $[\mathit{tlb}, \mathit{tub}]$ be the time interval computed in the procedure `checkSafety` for this eigenform (on Line 13 or Line 15). Since $T \in [\mathit{tmin}, \mathit{tmax}]$, and since $[\mathit{tmin}, \mathit{tmax}]$ is contained in the interval $[\mathit{tlb}, \mathit{tub}]$, we have that $T \in [\mathit{tlb}, \mathit{tub}]$.

If $[\mathit{tlb}, \mathit{tub}]$ is computed on Line 15, then using Lemma 3, we conclude that there is a $v0 \in [l_0, u_0]$ and a $v1 \in [l_1, u_1]$ s.t. $v1 = v0 * e^{\lambda * T}$.

If $[\mathit{tlb}, \mathit{tub}]$ is computed on Line 13, then using Lemma 3, we conclude that there is a $v0 \in [l_0, u_0]$ and a $v1 \in [l_1, u_1]$ s.t. $v1 = v0 + \lambda * T$.

Thus, we can find a value $v0 \in [l_0, u_0]$ and a value $v1 \in [l_1, u_1]$ for each eigenform V. Let V_i be the i-th eigenform, and let $v0_i, v1_i$ be the two values we find for V_i. Using the assumption that the initial set of states *Init* is completely feasible, we know there exists a concrete point, say $\vec{x}_0$, s.t. $V_i(\vec{x}_0) = v0_i$ for all i.

The point $\vec{x}_0$ will serve as our initial point. Consider the trajectory starting from this point. (All linear ordinary differential equations satisfy the Lipschitz condition, and hence, there exists a unique trajectory starting from any concrete initial point). At time T, this trajectory will reach some state, say $\vec{x}_1$. It is easy to see that the value of the eigenform V_i at the state $\vec{x}_1$ will be $v1_i$. Hence, $\vec{x}_1$ will be a state in the set *Unsafe*. Hence, we conclude that the system is not safe, contradicting our original assumption. □

Combining the results in Lemma 2 and Lemma 4, we get the following decidability result for safety verification.

THEOREM 1. *The aligned safety verification problem is decidable under the following assumptions:*

- *Init is a finite union of completely feasible aligned boxes*
- *The bounds $\vec{l}_0, \vec{u}_0$ that specify the set Init are rational (or more generally computable reals)*
- *The bounds $\vec{l}_1, \vec{u}_1$ that specify the set Unsafe are rational (or more generally computable reals)*
- *The eigenvalues corresponding to the eigenforms are rational (or more generally algebraic numbers, or just computable reals)*

PROOF. Putting together Lemma 2 and Lemma 4, we conclude that the Procedure `checkSafety` in Figure 2 correctly solves the aligned safety verification problem. The only thing remaining to prove is that each step of Procedure `checkSafety` can be effectively computed. Note that the checks (in the conditions) on Line 12 and Line 14 are not performed: we assume we are given the type of each eigenform. We only need to argue that the computation over real numbers being performed on Line 13, Line 15, Line 17, Line 18, and Line 20 can be effectively performed. Since computable real numbers form a real closed field, it follows that the result of Line 13 is computable. For Line 15, we additionally note that the result of taking a natural logarithm of a computable real results in a computable real. Lines 17,18 and 20 require comparison on computable real numbers. Thus, all operations on reals in Procedure `checkSafety` can be effectively performed. This completes the proof of the theorem. □

We make a few remarks about the procedure `checkSafety`. First, the procedure can be easily modified to handle the case when some bounds are $-\infty$ or ∞. Second, the assumption that the initial set *Init* be completely feasible is typically true: if the eigenforms are left eigenvectors, then since eigenvectors are linearly independent, it follows that the *Init* set specified using these eigenforms will be completely feasible.

The decidability result above is an important and significant observation for many reasons.

- Recall that there are seminal results on decidability of safety for linear systems [1], but they apply to systems $dX/dt = AX + b$ where the matrix A satisfies one of the following three conditions; either (a) all eigenvalues of A are rational, or (b) all eigenvalues of A are imaginary rationals, or (c) A is nilpotent. However, these are strong assumptions. Our result does not put any restriction on the dynamics of the linear system; though, the effectiveness of our procedure improves as the number of (rational/real) eigenvalues increases.
- Another popular method for safety verification is based on the use of so-called *barrier certificates*. It proves safety by showing that the initial set of states can be separated from the unsafe set by a "barrier" that the dynamics of the system cannot cut across. A barrier certificate is an inductive invariant (in computer science terminology). In our procedure, we do not compute a single polynomial function as a barrier, but more generally a conjunction of functions as a barrier. Moreover, using the CEGAR procedure described later, we get a CEGAR version of barrier certificate based methods.
- For timed automata, the forms that describe the faces of zones/regions are eigenforms. Linear eigenforms also exist for dynamics used in (each mode of a) linear hybrid automata. Even for more complicated dynamics, our result clearly suggests that one should use eigenforms of the dynamics in each mode to perform analysis, such as symbolic model checking [3].
- The fact that aligned safety verification problem is efficiently decidable is not very useful by itself. This is because it will *almost never* be the case that the initial set and the unsafe set are aligned to eigenforms. But, the decidability result can be used in different ways. One promising approach for the *general* safety verification problem is based on abstracting the general problem to the aligned problem, and then using the decision procedure in possibly a CEGAR loop to efficiently answer the general problem. We discuss this approach next.

5. A CEGAR ALGORITHM

In this section, using the decision procedure for the aligned safety verification problem, we present a counter-example guided abstraction-refinement (CEGAR [4]) procedure for the general safety verification problem.

The key idea behind solving the general safety verification problem is that arbitrary initial set *Init* and unsafe set *Unsafe* can be over-approximated using boxes over eigenforms. Furthermore, these over-approximations can be refined iteratively "as-needed". Figure 3 explains the complete procedure using illustrations.

The first step in solving the general safety verification problem consists of finding as many eigenforms of the linear system as possible. In Section 4, we had mentioned that a linear system has lots of eigenforms. Recall that these eigenforms can be obtained by computing left eigenvectors and eigenvalues. Assuming that the A matrix contains only *rational* values, the left eigenvectors and eigenvalues of A will have only *algebraic* reals, which are computable. Figure 3 assumes that there are two eigenforms, namely x and y.

The second step consists of computing over-approximations of the set of initial states, *Init*, and the set of unsafe states, *Unsafe*, using lower and upper bounds along the computed eigenforms. In Figure 3 (top-middle), the over-approximations are shown using dotted lines.

Having computed an over-approximation of *Init* and *Unsafe*, we now have an instance of the aligned safety verification problem. We use the decision procedure presented in Section 4 to solve the abstract problem. If the decision procedure returns "Valid" (indicating that the system is safe), then we can conclude that the original problem is also safe. If the decision procedure returns "Invalid" (indicating that the abstraction is not safe), then we cannot conclude anything yet.

When the decision procedure `checkSafety` returns "Invalid", we know there exists a trajectory from an initial state to an unsafe state. This trajectory is a *potential counter-example* to the safety claim. We can, in fact, compute this potential counter-example. We compute it as a pair of states: an initial state and a final state.

There are two possible cases now: either the potential counter-example is also a *valid* counter-example for the original (concrete) problem, or it is *spurious*. There are two ways in which the counter-example can be spurious.

1. The final state of the counter-example is in the over-approximation of *Unsafe*, but it is not in *Unsafe*. This is illustrated in Figure 3 (top-right).
2. The initial state of the counter-example is in the over-approximation of *Init*, but it is not in *Init*. This is illustrated in Figure 3 (bottom-middle).

In the first case, we refine the over-approximation of *Unsafe* by removing the final state. A very important point here is that we remove not just one state, but a large subset (aligned box) of states. Figure 3 (bottom-left) illustrates this process. As a result of refinement, the over-approximation of *Unsafe* becomes a finite union of smaller aligned boxes. In Figure 3 (bottom-left), the over-approximation of *Unsafe* is now a union of two box regions (marked 1 and 2).

In the second case, we similarly refine the overapproximation of *Init*. As illustrated in Figure 3 (bottom-right), the over-approximation of *Init* is now a union of two box regions (marked 1 and 2).

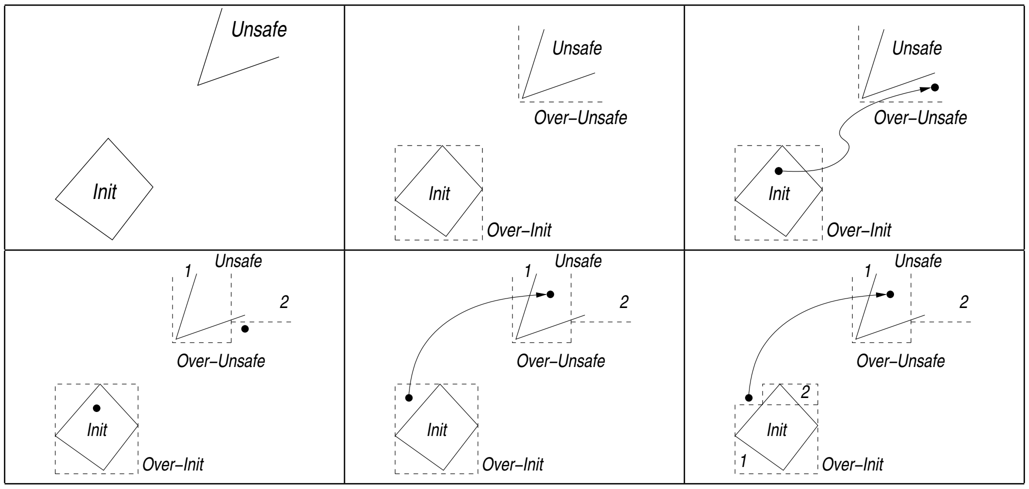

Figure 3: Illustrating the CEGAR procedure: (Top-left) An example initial set *Init* and an unsafe set *Unsafe*. (Top-middle) Abstract *Init* and *Unsafe* using boxes over eigenforms, say, x, y. (Top-right) Counter-example obtained using the decision procedure on the aligned instance, which is spurious since the end state is not in *Unsafe*. (Bottom-left) Refine the over-approximation of *Unsafe* by eliminating the spurious end state. The refined over-approximation of *Unsafe* is a disjoint union of two aligned boxes. (Bottom-middle) New counter-example obtained by the decision procedure over the refined problem, which is again spurious since it starts from a state that is not in *Init*. (Bottom-right) Refine over-approximation of *Init* into a disjoint union of two aligned boxes by removing the spurious initial state.

Once we have refined the over-approximation of the initial set *Init* or the over-approximation of the unsafe set *Unsafe*, we repeat the whole process again: for all possible choices of the initial box and for all possible choices of the unsafe box (boxes marked 1 and 2 in Figure 3(bottom-right)). The process is repeated until either safety is proved, or a valid counter-example is found, or refinement fails (i.e., the result of refinement is the same as its input). The procedure fails in the last case.

There are three important subtasks that are required to implement the CEGAR procedure described above:

1. Over-approximating the sets *Init* and *Unsafe*
2. Checking if an abstract counterexample is spurious
3. Refining an over-approximation

5.1 Over-approximating *Init* and *Unsafe*

Let ϕ be a finite intersection (conjunction) of linear half-spaces (or convex spaces). Given a set $\vec{V}$ of eigenforms, we compute an aligned overapproximation of ϕ as

$$\bigwedge_i (l_i \leq V_i \leq u_i)$$

where for each $V_i \in \vec{V}$, l_i is found by minimizing V_i subject to the constraint ϕ; and u_i is found by maximizing V_i subject to the constraint ϕ using a linear programming (LP) solver (or a convex optimization solver).

5.2 Checking if abstract counterexample is spurious

We check if an abstract counterexample is spurious by generating a constraint that will be satisfiable if the abstract counterexample is not spurious.

Recall that the procedure `checkSafety`, which is used to solve the abstract problem, returns a time interval [`tmin`, `tmax`] where an initial state can potentially reach an unsafe state. Let us assume that the initial set is ϕ_0, but restricted to the box ϕ_{box0} given as $\vec{l}_0 \leq \vec{V} \leq \vec{u}_0$. Similarly, the unsafe set is ϕ_1, but restricted to the box ϕ_{box1} given as $\vec{l}_1 \leq \vec{V} \leq \vec{u}_1$.

To determine if there is a concrete trajectory from the initial set to the unsafe set with time duration in [`tmin`,`tmax`], we pick a fixed time instant T from the interval [`tmin`, `tmax`].

We then compute a relation ψ_{tra} on variables X, X', where X' is a primed copy of X, that relates an initial state (value of X) to the state reached at time T (value of X') from that initial state. This relation has been called *timed relational abstraction* before [5, 6]. The relation ψ_{tra} is computed as a conjunction of equality constraints. Specifically, for each eigenform V, we generate one equation that relates the value of V at time T to the value of V at time 0. For eigenform V s.t. $dV/dt = \lambda V$, the equation is $V(X') = V(X) * e^{\lambda * T}$, whereas for eigenform V s.t. $dV/dt = \lambda$, the equation is $V(X') = V(X) + \lambda * T$ (note that λ and T are fixed constants). Let ψ_{tra} denote this conjunction of equations over X, X'. If there are $n = |X|$ linearly independent eigenforms, then we can use ψ_{tra} as the timed relational abstraction. If not, then we have to find the relationship between X' and X using matrix exponentiation. The key observation is that for a fixed T, the relationship between X and X' can be computed, and the procedure `checkSafety` returns an interval that helps in fixing T.

Finally, we check if an unsafe state is reachable from an initial state in time T by checking feasibility of the following constraint ψ over variables X, X':

$$\psi := (\psi_{tra} \ \wedge \ \phi_0 \wedge \phi_{box0} \ \wedge \ \phi_1[X \mapsto X'] \wedge \phi_{box1}[X \mapsto X'])$$

where the notation $\phi[X \mapsto X']$ denotes the formula obtained by replacing X by X' in ϕ. If the constraint ψ is feasible, then we get a concrete counter-example.

If the constraint ψ is infeasible, then we drop ϕ_1 from it and recheck feasibility. If the modified constraint is feasible, then its solution is a state in the unsafe box ϕ_{box1}, which is not present in the unsafe region ϕ_1, but which is reachable from some initial state. This indicates that the unsafe set needs refinement.

Similarly, by solving another constraint, we can determine if the initial set needs to be refined.

For proving the correctness of the above procedure for checking if an abstract counterexample is spurious, we have to assume that the relationship ψ_{tra} computed between X and X' is complete. This is required to establish the claim that the computed counter-examples indeed correspond to actual trajectories of the system.

5.3 Refining an over-approximation

The refinement procedure improves the over-approximation by removing spurious points from it. The key observation here is that rather than removing a single (given) point, we remove a large subspace (with nonzero volume). The reason is to expedite convergence of the iterative refinement procedure.

The refinement procedure is given a formula ϕ representing some region of the state space, a box $\psi := (\vec{l} \leq \vec{V} \leq \vec{u})$, and a point $\vec{x}$. The point lies inside the box ψ, but it lies outside ϕ. The goal is to construct an improved over-approximation of the region $\phi \wedge \psi$ as a union of boxes by removing (a small box around the) point $\vec{x}$ from it.

The refinement procedure works by first "fattening" the point $\vec{x}$ into a box φ that is guaranteed to remain outside the region $\phi \wedge \psi$. Initially, the box φ is initialized to the degenerate box $\bigwedge_{V\in\vec{V}}(V = V(\vec{x}))$, where $V(\vec{x})$ denotes the real number resulting from evaluating V at $\vec{x}$. In each iteration, an eigenform V from $\vec{V}$ is picked and the constraint $V = V(\vec{x})$ in φ is replaced by a weaker constraint $l \leq V \leq u$, where l, u are computed by solving two different optimization problems. The lower-bound l for V is computed as the result of following optimization problem:
`maximize` V `s.t.` $\phi \wedge \psi \wedge \varphi[V = V(\vec{x}) \mapsto V \leq V(\vec{x})]$
The notation $\varphi[V = V(\vec{x}) \mapsto V \geq V\vec{x}]$ denotes the constraint obtained from φ by replacing in φ the constraint before $\mapsto$ by the constraint after $\mapsto$. The calculation of the upper bound u is done similarly.

The box φ is updated by replacing the equality constraint $V = V(\vec{x})$ by the constraint $(l + V(\vec{x}))/2 \leq V \leq (u + V(\vec{x}))/2$. We use the average of l and $V(\vec{x})$ in place of l as a heuristic to ensure that the final box φ has nonzero volume.

Once the fattened box φ has been computed, we return (disjoint) boxes that make up the (topological closure of the) space $\psi - \varphi$, where $-$ denotes set difference. We note here that if the fattened box φ is of lower dimension than the box ψ (i.e., if $l = u = V(\vec{x})$ is true, but $l_0 = u_0$ is not true, where $l_0 \leq V \leq u_0$ is the constraint in ψ), then the refinement procedure fails. This can happen, for example, when there are not enough eigenforms available to truly refine an over-approximation.

The correctness of Procedure `refine` follows from the observation that the box φ computed in the procedure is always *outside* the region $\phi \wedge \psi$.

Due to space constraints, we have described the overall CEGAR procedure only informally.

Note that the CEGAR procedure can fail in two ways: either the refinement step can fail because there are fewer eigenforms than needed to refine an overapproximation, or the number of refinements can become unbounded and cause nontermination. However, under some assumptions, the CEGAR approach is sound and complete for solving *robust* instances of the safety verification problem.

5.4 Robust Instances of the Safety Verification Problem

We first define robust instances of the safety verification problem.

Definition 4. [Robust Safety Verification Problems] An instance of the safety verification problem – consisting of a linear system *Sys*, an initial set *Init*, and an unsafe set *Unsafe* is *robust* if, it is safe

$$Reach(Init, Sys, [0,\infty)) \ \cap \ Unsafe \ = \ \emptyset$$

iff there exists an $\epsilon > 0$ s.t.

$$Reach(B_\epsilon(Init), Sys, [0,\infty)) \ \cap \ B_\epsilon(Unsafe) \ = \ \emptyset$$

where the epsilon-ball $B_\epsilon(S)$ around a set S is defined as $\{\vec{x} \in \Re^n \mid \|\vec{x} - \vec{y}\| \leq \epsilon \text{ for some } \vec{y} \in S\}$.

A set $\{V_1, \ldots, V_m\}$ of eigenforms is complete for a set $S \subseteq \Re^n$ if for every point $\vec{x} \in S$, the only solution of the constraint $\wedge_i V_i = V_i(\vec{x})$ is the point $\vec{x}$. Intuitively, a set of eigenforms is complete for a set if the value of all the eigenforms at any point in the set uniquely identify that point. If we have n eigenforms that are linearly independent, then they will be complete for any set S.

THEOREM 2. *Given a robust instance of the safety verification problem, if Init, Unsafe are compact, and if there exists a set $\{V_1, \ldots, V_m\}$ of eigenforms that is complete for Init and Unsafe, then there exists a (finite) execution of the CEGAR procedure that correctly solves the safety verification problem.*

PROOF. Let $\epsilon > 0$ be the constant that defines the robust instance of the safety verification problem. By definition of completeness of the set of eigenforms for *Init*, we have

$$Init \quad = \quad \bigvee_{\vec{x}\in Init} (\bigwedge_i V_i = V_i(\vec{x}))$$

Let $\delta > 0$. We can now cover *Init* by open sets,

$$Init \quad \subset \quad \bigvee_{\vec{x}\in Init} (\bigwedge_i V_i(\vec{x}) - \delta < V_i < V_i(\vec{x}) + \delta)$$

Moreover, by picking δ small enough, we can ensure that the right-hand side is contained in $B_\epsilon(Init)$. Since *Init* is compact, there is a finite cover; hence there is a finite subset $\widetilde{Init} \subset Init$ s.t.

$$Init \ \subset \ \bigvee_{\vec{x}\in \widetilde{Init}} (\bigwedge_i V_i(\vec{x}) - \delta < V_i < V_i(\vec{x}) + \delta) \ \subset \ B_\epsilon(Init)$$

Thus, we can overapproximate *Init* by finitely many aligned boxes, and the overapproximation is still contained in $B_\epsilon(Init)$. We can do the same for the *Unsafe* set. We use the CEGAR procedure to create exactly these finitely many aligned boxes as an overapproximation for *Init* and as an overapproximation for *Unsafe*. Due to robustness assumption, the answer of the safety verification problem for this overapproximation is the same as the answer for the original problem. This concludes the proof. $\square$

6. EXPERIMENTS

We implemented a prototype of the counter-example guided iterative refinement procedure for checking safety of linear systems. HybridSal [8] is used as the language for specifying the system and the safety property. We use finite precision floating point numbers in our implementation. The implementation is in Python and uses GNU Linear Programming Kit (GLPK) as the underlying solver, and the `numpy,scipy` python packages for performing matrix functions.

Our tool takes two required arguments – a filename containing the system description and a name of the property in that file that needs to be verified. There are optional arguments for setting values of various parameters. There is a parameter to set a limit on the number of refinements the tool considers. Since we use floating point arithmetic, there is also a parameter to set the numerical tolerance. Needless to say, values of these parameters influence the running time of the tool.

Experimental evaluation of the implementation is ongoing. We performed some preliminary evaluation on several small and medium-sized examples, including an 8-dimensional model of insulin metabolism [9], and a 28-dimensional helicopter model from the SpaceEx website [10, 11]. Since our tool is not a reachability tool, but a verification tool, it requires a safety property, and its running time depends crucially on the safety property. Table 1 presents sample experimental results. For a fixed problem, running time (last column) is proportional to the number of refinement steps performed (column `#r`). The number of refinements depends on the property. For the 28-dimension helicopter example, a simple false property (called p2 in Table 1) is identified as invalid in 2.6s, whereas a nontrivial valid property (called p1) is proved valid after 5.7s. In all examples, the properties describe bounds on the state variables.

Column `params` mentions two important parameters used in the implementation. The first parameter under column `params` is a user-provided budget for depth of the refinement tree, and the program aborts (fails) if it ever exceeds the provided budget. The second parameter under column `params` sets the numerical tolerance of the procedure and is crucial in determining the running time: a lower value, say 0.001, improves accuracy and trust in the outcome, but increases running time (Heli,p1 first two rows) – except in cases when the user-defined budget causes the tool to abort early (Heli,p1 third row).

The examples labeled Ex1-Ex6 in Table 1 are simple 2–6 dimensional problems created by hand to check correctness of the implementation. In all cases, the output of the tool (column `result` in Table 1) is sound.

Table 1: Performance of CEGAR implementation

Prob	n	params	result	#r	d	time(s)
Insulin	8	20,0.1	Proved	517	14	4.02
		20,0.01	Abort	15089	20	104
		24,0.001	Abort	532	24	5.12
Heli,p1	28	20,0.1	Proved	70	9	5.7
		16,0.01	Abort	374	16	19.5
		20,0.001	Abort	215	20	10.9
Heli,p2	28	20,0.1	CE	0	0	2.6
		20,0.01	CE	0	0	2.6
		20,0.001	CE	0	0	2.6
Ex1,p1	2	10,0.001	Proved	0	0	0.33
Ex2,p1	4	10,0.001	Proved	0	0	0.33
Ex3,p1	4	10,0.001	Proved	1	1	0.36
Ex4,p1	2	10,0.001	Proved	0	0	0.33
Ex5,p1	2	10,0.001	Proved	0	0	0.33
Ex6,p2	6	10,0.001	CE	0	0	0.36

Column `Prob` lists the problem instance (model,property pair), column `n` contains the dimension, column `params` contains the parameters used for running our tool, column `result` shows the result produced by the tool, column `#r` contains the total number of refinements performed by the tool in that run, column `d` contains the maximum depth of the refinement tree, column `time` shows the running (real) time of the tool in seconds.

7. DISCUSSION AND RELATED WORK

Reachability tools, such as SpaceEx [10], focus their effort on finding very high quality (bounded time) reach sets. They completely ignore the safety property that the user may be interested in proving. The CEGAR procedure for safety verification is attractive since it is goal directed: it determines what (refinement steps) to do based on the given initial set and the unsafe set. However, there is no a-priori bound on the number of refinements the program may need. In fact, it can easily need unbounded number of refinements: if $dx/dt = 1, dy/dt = 1$ and initially $x = 2, y = 1$, then using x, y as the two eigenforms, the CEGAR procedure needs an infinite number of refinements to prove that $x > y$ always. (Our implementation aborts after exhausting the user-provided budget for refinements.)

There are conditions under which the CEGAR procedures exhibits good termination behavior. As indicated by Theorem 2, the procedure performs better when the unsafe set is compact, and it performs poorly when the unsafe set is unbounded. Again, as suggested by Theorem 2, the procedure performs better when the unsafe region is "well-separated" from the reachable set, and it performs poorly when the reach set "almost touches" the unsafe region. Finally, the procedure performs better when the linear system has many different eigenforms. Note here that, in the limit, when the linear system has n linearly independent linear eigenforms (corresponding to rational eigenvalues), then the reachability problem is known to be decidable [1].

The fact that our tool works better on compact unsafe sets suggests several interesting strategies for proving safety. For example, using our tool, rather than prove "`always B`", it is always easier to prove the equivalent conjunction "`always A implies B`" and "`always not(A) implies B`". This is because, while $\texttt{not}(B)$ may not be compact, the sets $A \wedge \texttt{not}(B)$ and $\texttt{not}(A) \wedge \texttt{not}(B)$ could be made bounded by choosing A appropriately.

We believe the CEGAR procedure for safety verification of linear systems is an important first step toward devising a procedure for safety verification of hybrid systems. We plan to extend our implementation to hybrid systems by handling intersection with guards. In the future, we also plan to re-place the linear programming solver in the implementation by a complex optimization solver [12]. This will enable us to handle quadratic eigenforms directly.

The idea of using iterative abstraction and refinement for safety verification of hybrid systems has been used before by several authors [13–19]. The first set of techniques [13–15, 17, 20] use finite discrete transition systems as abstractions: the unbounded *state space* is mapped (abstracted) into a finite state space by defining an equivalence relation on the concrete states, and the dynamics is mapped into transitions on equivalence classes of states. The partitioning of the state space is often based on predicates (predicate abstraction). Constructing such abstractions is difficult since it requires mapping of continuous dynamics to discrete transitions. Performing refinements (CEGAR) is challenging since it requires discovering new predicates and concretizing abstract counterexamples. Termination of the CEGAR loop is also an issue in these approaches. An alternative approach for defining abstractions is based on hiding variables or merging locations to produce abstract timed and hybrid systems [16, 18, 19]. However, this approach has been limited to hybrid systems with very simple continuous dynamics, such as in timed and rectangular automata.

Relational abstraction is a new form of abstraction that does *not* abstract the state space, but only abstracts (over-approximates) the dynamics (the transition relation). It has been used to perform safety verification [5, 7], but there is no refinement step and no CEGAR loop in [5]. Relational abstractions, constructed using Lyapunov-like functions, have been used to prove liveness properties [21, 22], where the appropriate well-founded relational abstraction (i.e., an over-approximation of the transition relation) is generated using a CEGAR procedure. In our work, it is the initial states and the unsafe region that are abstracted and then iteratively refined.

8. CONCLUSION

Safety verification of continuous and hybrid systems is an important problem. We presented a new counterexample guided abstraction refinement procedure for safety verification of linear systems. The main insight behind the procedure is that of eigenforms: linear or quadratic functionals that change monotonically along the trajectory of the system. Even if the exact details of using them might vary, we believe that all procedures for safety verification will benefit by incorporating eigenforms in their computation. We proved decidability of the aligned safety verification problem for linear systems, and also showed that the CEGAR procedure can be effective for solving robust instances of the safety verification problem.

9. REFERENCES

[1] G. Lafferriere, G. J. Pappas, and S. Yovine, “Symbolic reachability computations for families of linear vector fields,” *J. Symbolic Computation*, vol. 32, no. 3, pp. 231–253, 2001.

[2] P. Tabuada, *Verification and control of hybrid systems.* Springer, 2009.

[3] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, “Symbolic model checking for real-time systems,” in *Proc. LICS*, 1992, pp. 394–406.

[4] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, “Counterexample-guided abstraction refinement,” in *Proc. CAV*, 2000, pp. 154–169.

[5] S. Sankaranarayanan and A. Tiwari, “Relational abstractions for continuous and hybrid systems,” in *Proc. CAV*, ser. LNCS, vol. 6806, 2011, pp. 686–702.

[6] A. Zutshi, S. Sankaranarayanan, and A. Tiwari, “Timed relational abstractions for sampled data control systems,” in *Proc. CAV*, ser. LNCS, vol. 7358, 2012, pp. 343–361.

[7] S. Ratschan and Z. She, “Constraints for continuous reachability in the verification of hybrid systems,” in *Proc. AISC*, ser. LNCS, vol. 4120, 2006, pp. 196–210.

[8] A. Tiwari, “Hybridsal relational abstracter,” in *Proc. CAV*, ser. LNCS, vol. 7358, 2012, pp. 725–731.

[9] S. Gulwani and A. Tiwari, “Constraint-based approach for analysis of hybrid systems,” in *Proc. CAV*, ser. LNCS, vol. 5123. Springer, 2008, pp. 190–203.

[10] G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, “Spaceex: Scalable verification of hybrid systems,” in *CAV*, ser. LNCS, vol. 6806, 2011, pp. 379–395.

[11] “Spaceex: State Space Explorer,” 2010, http://spaceex.imag.fr/.

[12] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge University Press, 2004.

[13] R. Alur, T. Dang, and F. Ivancic, “Progress on reachability analysis of hybrid systems using predicate abstraction,” in *Proc. HSCC*, ser. LNCS, vol. 2623, 2003, pp. 4–19.

[14] E. M. Clarke, A. Fehnker, Z. Han, B. H. Krogh, O. Stursberg, and M. Theobald, “Verification of hybrid systems based on counterexample-guided abstraction refinement,” in *Proc. TACAS*, ser. LNCS, vol. 2619, 2003, pp. 192–207.

[15] M. Sorea, “Lazy approximation for dense real-time systems,” in *Proc. FORMATS/FTRTFT*, 2004, pp. 363–378.

[16] S. K. Jha, B. H. Krogh, J. E. Weimer, and E. M. Clarke, “Reachability for linear hybrid automata using iterative relaxation abstraction,” in *Proc. HSCC.* Springer-Verlag, 2007.

[17] M. Segelken, “Abstraction and counterexample-guided construction of omega -automata for model checking of step-discrete linear hybrid models.” in *Proc. CAV.* Springer, 2007, pp. 433–448.

[18] H. Dierks, S. Kupferschmid, and K. G. Larsen, “Automatic abstraction refinement for timed automata,” in *Proc. FORMATS.* Springer, 2007.

[19] P. Prabhakar, P. S. Duggirala, S. Mitra, and M. Viswanathan, “Hybrid automata-based cegar for rectangular hybrid systems,” in *Proc. VMCAI*, 2013.

[20] A. Tiwari and G. Khanna, “Series of abstractions for hybrid automata,” in *Proc. HSCC*, ser. LNCS, vol. 2289. Springer, 2002, pp. 465–478.

[21] P. S. Duggirala and S. Mitra, “Lyapunov abstractions for inevitability of hybrid systems,” in *Proc. HSCC*, 2012, pp. 115–124.

[22] ——, “Abstraction refinement for stability,” in *Proc. ICCPS*, 2011, pp. 22–31.