Abstraction Refinement for Stability

Parasara Sridhar Duggirala and Sayan Mitra
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
{duggira3, mitras}@illinois.edu

***Abstract*—The paper presents a counterexample-guided abstraction refinement procedure for verifying stability (region stabillity) of CPS modeled as hybrid automata. It relies on a characterization of the blocking property of hybrid automata in terms of well-foundedness of a relation that combines the discrete transitions and continuous trajectories and a key assumption about the switching speed of the system in terms of average dwell time, but does not require the individual modes to be stable. This characterization enables the adaptation of program analysis techniques to the domain of hybrid systems. It is shown that the procedure is complete for rectangular initialized hybrid automata. Several illustrative examples are verified using a prototype tool that implements the methodology.**

I. Introduction

In this paper, we will say that a system $\mathcal{A}$ with state space $\mathcal{X}$ is *stable* with respect to a set of states $\mathcal{S} \subseteq \mathcal{X}$ if starting from any state $x \in \mathcal{X}$, every execution of $\mathcal{A}$ eventually enters and remains in $\mathcal{S}$. This property has also been called *region stability* [26] and practical stability [17] in the hybrid and dynamical systems literature, *self-stabilization* in the distributed computing literature [12], and is related to asymptotic stability which is widely studied in control theory. Stability is important in many cyber-physical systems as it captures the desirable liveness property of a system returning to an acceptable state after failures or disturbances deflect it to an arbitrary state. The ability of a traffic control protocol to reconfigure routes in the face of congestion and disruptions, the ability of an autonomous vehicle to track a new target, the ability of a real-time system to recover from failures are all examples of stability. In this paper, we study the problem of verifying stability of CPS modeled as hybrid automata [1].

Computing the reach set is at best difficult and often impossible, for hybrid models of complex CPS. Automatic invariance and safety verification of such systems therefore invariably rely on overapproximating the reach set (see for example [5], [4], [7], [15], [16]). Recently, several abstraction-refinement based techniques have been developed for automatically constructing abstractions of hybrid automata especially for proving given safety properties [27]. A general scheme for abstraction-refinement called counterexample guided abstraction refinement (CEGAR) [9], [2] works as follows: The process starts with a coarse abstraction $\mathcal{B}_0$ of the given system $\mathcal{A}$ and a safety property P. In each iteration, the abstraction $\mathcal{B}_i$ is model-checked with respect to P. If it is proven to be safe then $\mathcal{A}$ can be inferred to be safe. Otherwise, the model checker returns a counterexample β_i illustrating that the abstraction $\mathcal{B}_i$ is not safe. Then β_i is *validated* to determine if there is indeed a real counterexample of $\mathcal{A}$ that corresponds to it. Otherwise, the spurious abstract counterexample β is used to obtain a new finer abstraction $\mathcal{B}_{i+1}$ which (at least) eliminates the spurious counter example β_i. Under some restrictions, the above procedure can be shown to be complete. That is, the procedure terminates with either an abstract automaton which establishes safety or a real counterexample which illustrates safety violation. In [3], [9], [27], [28], [8], it has been shown how variations of such abstraction-refinement based algorithms can dramatically improve the efficiency of safety verification. The verification procedure we propose in this paper relies on an analogous CEGAR-based approach, but for proving stability.

Our work stems from the simple observation, that, roughly speaking, a hybrid automaton $\mathcal{A}$ is stable with respect to a set $\mathcal{S}$ if and only if the automaton $\mathcal{A}'$, obtained by restricting $\mathcal{A}$ to $\mathcal{S}^c$, is blocking. This is useful because verifying blocking property is closely related to termination analysis of (infinite state) programs, and enable us to employ powerful results and recently developed tools from software verification and program analysis [11], [25], [10] for verifying stability of hybrid systems.

The standard technique for proving that a program is terminating (or blocking) is to establish that the transition relation for the program is contained in a well-founded relation. Of course, finding such a well-founded relation automatically is impossible in general [30]. Using Ramsey's theorem, in [25], it is shown that a program is terminating if and only if the *transitive closure* of the transition relation is contained in a disjunctive union of a finite collection $\mathcal{R}$ of well-founded relations. This result makes it possible to systematically search for $\mathcal{R}$ by considering one loop of the program at a time [24]. Based on [25], in [10] the authors present a (necessarily incomplete) abstraction-refinement based procedure for verifying termination of programs.

Programs evolve in discrete steps while CPS involve continuous trajectories in addition to discrete steps. A trajectory models the evolution of the state variables of a hybrid automaton over an interval in $\mathbb{R}_{\geq 0}$. Thus,

any trajectory can be broken down into infinitely many tiny steps and consequently the relation defined by such steps cannot be well-founded. This poses as a technical barrier in adapting the above results to the context of hybrid systems. In addressing this issue, we define *hybrid step relation* that relates the pre-state of a trajectory to a post-state obtained by applying a transition at the end of the trajectory. The hybrid step relation together with a mild assumption about the switching speed of the system in terms of average dwell time [13], enable us to characterize blocking in terms of well-founded relations. With the help of tools for synthesizing ranking functions [25] and results from termination analysis [10] we then proceed to develop a CEGAR-based scheme for proving the blocking and hence stability. We apply the technique to several examples using a prototype tool. We also show that the procedure is complete for certain classes of initialized hybrid automata (e.g., rectangular) for which the individual steps of the procedure can be effectively computed.

II. Related Work

Analyzing the asymptotic stability[1] of hybrid automata is challenging because the stability of the continuous dynamics of each individual mode does not necessarily imply the stability of the whole automaton. The basic techniques rely on finding a *Common Lyapunov function*, whose derivative along the trajectories of all the modes must satisfy suitable inequalities. Approaches for finding common Lyapunov functions for hybrid automata have been presented in [23]. By reasoning about the graph structure of the automaton and solving a set of semi-definite programs, a family of local Lyapunov functions are generated which can be combined to obtain a common Lyapunov function.

When such a function cannot be found or does not exist, *Multiple Lyapunov* functions [6] are useful for proving stability of a chosen execution. These and many other stability related results which use Lyapunov functions for the individual modes are discussed in [18], [31]. For example, if the individual modes of the automaton are asymptotically stable, then the notion of *dwell time* [22] and the more general *average dwell time (ADT)* [13] provide sufficient conditions for system stability. These conditions have been used to develop (in some cases automatic) verification procedures in [20].

In [26], stability of a linear hybrid system is characterized as *finiteness* of certain sequences called *snapshot sequences*. Three kinds of snapshot sequences are identified and a new automaton which represents the evolution through these three kinds of snapshot sequences is created. By analyzing the unary reachability property of the new automaton, the stability of the given automaton is verified.

[1]The relationship between asymptotic stability and stability as defined at the beginning of this paper, is discussed in Section III-B.

III. Preliminaries

We will use the Hybrid Input/Output Automaton (HIOA) framework for modeling cyber-physical systems. We begin this section by introducing some key concepts in this framework and refer the reader to [14], [19] for a more detailed development.

Let V be a set of variables. Each variable $v \in V$ is associated with a *type*, denoted by $type(v)$, which defines the set of values v can take. A valuation $\mathbf{v}$ for V maps each $v \in V$ to a value in $type(v)$. We use the standard $\mathbf{v}.x$ notation to refer to the valuation of a variable $x \in V$ at $\mathbf{v}$. The set of all valuations of V is denoted by $val(V)$. A *trajectory* for a set of variables V models continuous evolution of the values of the variables over an interval of time. Formally, a trajectory τ is a map from a left-closed interval of $\mathbb{R}_{\geq 0}$ with left endpoint 0 to $val(V)$. The domain of τ is denoted by $\tau.dom$. The *first state* of τ, $\tau.\mathsf{fstate}$, is $\tau(0)$. A trajectory τ is *closed* if the domain of τ is a closed interval $[0, t]$ for some $t \in \mathbb{R}_{\geq 0}$, and in that case we define $\tau.\mathsf{ltime} \triangleq t$ and the last state in τ, $\tau.\mathsf{lstate} \triangleq \tau(t)$. If the domain is an open interval, then $\tau.\mathsf{ltime}$ is defined as the supremum of $\tau.dom$.

A. Hybrid Automata

A *hybrid automaton* is a state machine for which a state is defined by the valuation of a collection of variables and the state changes either instantaneously through discrete transitions or over an interval of time following a trajectory.

Definition 1. *A* Hybrid Automaton (HA) $\mathcal{A}$ *is a tuple* $\langle V, A, \mathcal{D}, \mathcal{T} \rangle$ *where*

(a) $V = X \cup \{loc\}$ *is a set of variables, where* loc *is a discrete variable of finite type* L *called the set of* locations, *and each* $x \in X$ *is a continuous variable of type* $\mathbb{R}$*; the elements of* $val(V)$ *are called* states

(c) A *is a finite set of actions*

(d) $\mathcal{D} \subseteq val(V) \times A \times val(V)$ *is a set of* transitions *A transition* $(\mathbf{v}, a, \mathbf{v}') \in \mathcal{D}$ *is written in short as* $\mathbf{v} \xrightarrow{a}_{\mathcal{A}} \mathbf{v}'$ *or as* $\mathbf{v} \xrightarrow{a} \mathbf{v}'$ *when* $\mathcal{A}$ *is clear from the context*

(e) And $\mathcal{T}$ *is set of trajectories for* V *which is closed under prefix, suffix, and concatenation (see [14] and [19] for details). Over any trajectory* $\tau \in \mathcal{T}$*,* loc *remains constant and each* $x \in X$ *evolves according to certain differential-algebraic equations. For* $l \in L$, $\mathcal{T}_l \triangleq \{\tau \in \mathcal{T} \mid \tau(0).loc = l\}$ *is the set of trajectories in location* l

A hybrid automaton usually also specifies a set of initial states but for this paper that information is unnecessary.

Notation and Syntax. Transitions are specified using *guards* and *reset maps*. For every action $a \in A$, a guard $G_a \triangleq \{\mathbf{v} \mid \exists \mathbf{v}', \mathbf{v} \xrightarrow{a} \mathbf{v}'\}$, specifies the set of states at which some discrete transition labeled by a *can* occur. The set G_a is defined by an expression, denoted by $G_a(V)$, involving the variables in V. A reset map for a

is a function $R_a : val(V) \to 2^{val(V)}$ which specifies how the state changes when a does occur. It is specified by an expression, $R_a(V, V')$, involving V and their primed versions V'.

Trajectories are specified using *location invariants* and differential-algebraic equations (and inequalities). For every $l \in L$, $I_l \subseteq val(V)$, defined by an expression $I_l(V)$, is called the *location invariant*. E_l is a collection of differential-algebraic equations and inequalities. A trajectory τ with $\tau(0).loc = l$ is in $\mathcal{T}_l \subseteq \mathcal{T}$ iff (a) τ satisfies E_l, and (b) for all $t \in \tau.dom$, $\tau(t) \in I_l$ and $\tau(t).loc = l$. In this paper, we assume that the solutions of the differential-algebraic equations are available to us in the form of an expression $E_l(V, V, t)$ which describes the relationship between τ.fstate and τ.lstate for any trajectory $\tau \in \mathcal{T}_l$ with τ.ltime $= t$. For example, for $\dot{x} \leq -a; y = bx$, the expression $E(V, V', t)$ is $x' \leq x - at; y' = bx'$.

B. Executions and Stability

An execution of $\mathcal{A}$ records all the information associated with a particular run of $\mathcal{A}$. Formally, an *execution fragment* is a (possibly infinite) alternating sequence $\tau_0 a_1 \tau_1 \ldots$ where for each τ_i in the sequence, except possibly the last, τ_i is closed and $\tau_i.\mathsf{lstate} \xrightarrow{a_{i+1}} \tau_{i+1}.fstate$.

The first state of an execution fragment α is denoted by $\alpha.\mathsf{fstate} \triangleq \tau_0.\mathsf{fstate}$. The *duration* of an execution fragment α is defined as $\alpha.\mathsf{ltime} \triangleq \sum_i \tau_i.\mathsf{ltime}$, where the summation is over all the trajectories in α. An execution is *finite* if it is finite sequence, otherwise it is *infinite*. A finite execution is *closed* if its last trajectory τ_n is closed. In this case, the last state of α, is defined as $\alpha.\mathsf{lstate} \triangleq \tau_n.\mathsf{lstate}$. An execution is said to be *admissible* if its duration is infinite, and it is said to be *Zeno* if its neither admissible nor finite.

A location $l \in L$ is closed if time cannot diverge in it. That is, for every $\tau \in \mathcal{T}_l$, $\tau.dom$ is bounded. The automaton $\mathcal{A}$ is *locally closed* if all its locations are closed.

A hybrid automaton $\mathcal{A}$ is *nonblocking* if for every state $\mathbf{v}$, time diverges eventually for all executions starting from $\mathbf{v}$. It is *blocking* if all its executions are closed. Thus, there are hybrid automata which are neither blocking nor nonblocking. For example, a Zeno hybrid automaton falls in this class because time does not diverge in Zeno executions and nor are they closed. A hybrid automaton for which some of the executions are closed while others are Zeno or diverging, is also neither blocking nor nonblocking.

Definition 2. *For hybrid automaton $\mathcal{A}$, a set of states $S \subseteq val(V)$ is said to be* closed *if (a) for every $\mathbf{v} \xrightarrow{a} \mathbf{v}'$ with $\mathbf{v} \in S$, $\mathbf{v}'$ is also in S, and (b) for every $\tau \in \mathcal{T}$, with $\tau.\mathsf{fstate} \in S$, $\tau.\mathsf{lstate} \in S$. A hybrid automaton $\mathcal{A}$ is said to be* stable with respect to S, *$S \subseteq val(V)$, if S is closed and from every state $\mathbf{v}$ every execution starting from $\mathbf{v}$ reaches S.*

Definition 3. *A hybrid automaton $\mathcal{A}$ is said to be* globally asymptotically stable at the origin *if for every $\epsilon > 0$, $\mathcal{A}$ is region stable with respect to $val(loc) \times B_\epsilon$, where $B_\epsilon \subseteq val(X)$ is the set of valuations corresponding to the ball of radius ϵ around the origin.*

Stability is also called practical stability and is identical to the *self-stabilization* property that is commonly used as a requirement in fault-tolerance and distributed systems literature (see, for example [12]). We note that if the set S is the singleton set with the origin, then asymptotic stability is weaker than stability with respect to S. On the other hand, if S is a superset of the origin, then asymptotic stability implies stability with respect to S.

The notion of dwell time and the more general average dwell time were introduced in [13] (see [18] for more details), to obtain sufficient conditions for proving asymptotic stability of hybrid systems.

Definition 4. *A hybrid automaton $\mathcal{A}$ is said to have an* average dwell time (ADT) *$\Delta > 0$, if there exists a constant $N_0 > 0$ such that for any execution α the number of discrete transitions (mode switches) over $N(\alpha)$ is bounded by:*

$$N(\alpha) \leq N_0 + \alpha.ltime/\Delta. \tag{1}$$

Roughly, $\mathcal{A}$ has ADT Δ if on an average it performs at most one mode witch or transition every Δ time, plus a constant number N_0 of extra switches. The sufficient condition in [13] requires that (a) each of the individual modes of the hybrid automaton is asymptotically stable, i.e., they have exponentially decaying Lyapunov functions, and (b) that the ADT is large enough with respect to decay rate of the Lyapunov functions of the individual modes.

Throughout this paper we assume that the hybrid automata in question have some positive ADT.

Assumption 1. *There exist $\Delta \in \mathbb{R}_+$ and $N_0 \in \mathbb{Z}_+$ such that every execution of $\mathcal{A}$ satisfies Equation (1).*

Though our results use the existence of a ADT, they do not require the individual modes to be stable and only require the ADT Δ to be positive. Typical models of CPS systems, particularly those which periodically sense inputs and make decisions, satisfy this assumption (see, for example. [33], [32]).

IV. Verifying Stability

In Section IV-D we will identify the problem of verifying stability of a hybrid automaton to the problem of verifying the blocking property of another related automaton. In the next section we are developing a methodology for verifying the blocking property.

A. Verifying Blocking Property

The transitions and the trajectories of a hybrid automaton defines a *hybrid step relation* $\Gamma_{\mathcal{A}}$ on the state space, defined as $\mathbf{v}\Gamma\mathbf{v}'$ iff there exists $\mathbf{v}'' \in val(V), a \in A, \tau \in \mathcal{T}$ such that $\mathbf{v} = \tau.\mathsf{fstate}$, $\mathbf{v}'' = \tau.\mathsf{lstate}$, and $\mathbf{v}'' \xrightarrow{a} \mathbf{v}'$. That is, there exists a trajectory and a transition which take $\mathbf{v}$

to $\mathbf{v}'$. When the automaton $\mathcal{A}$ is clear from the context, we drop the suffix and write $\Gamma_{\mathcal{A}}$ as Γ. A relation is *well-founded* if it permits no infinite decreasing chains. Next, we state a key theorem which characterizes blocking in terms of well-foundedness of the hybrid step relation.

Theorem 1. *A locally closed HA $\mathcal{A}$ with positive ADT is blocking iff Γ is well-founded.*

Proof: Suppose that $\mathcal{A}$ is not blocking, i.e., there exists an execution α in which time diverges. Since $\mathcal{A}$ is locally closed, α must be of the form $\tau_0, a_1, \tau_1, \ldots$. Let $\beta = \mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \ldots$, be a sequence of states such that for each i, $\mathbf{v}_i = \tau_i.\mathsf{fstate}$. Thus, for each i, there exists $\mathbf{v}'_i$ such that $\tau_i.\mathsf{fstate} = \mathbf{v}_i$, $\tau_i.\mathsf{lstate} = \mathbf{v}'_i$ and $\mathbf{v}'_i \xrightarrow{a_i} \mathbf{v}_{i+1}$, and it follows that $\mathbf{v}_i \Gamma \mathbf{v}_{i+1}$. Since β is an infinite sequence, we conclude that Γ is not well-founded.

Suppose that the hybrid transition relation Γ is not well founded. There exists a sequence $\beta = \mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \ldots$, such that for each i in the sequence $\mathbf{v}_i \Gamma \mathbf{v}_{i+1}$. From β and the definition of Γ, we can construct a sequence $\beta' = \mathbf{v}_0, \mathbf{v}'_0, \mathbf{v}_1, \mathbf{v}'_1, \mathbf{v}_2, \mathbf{v}'_2 \ldots$ such that for each i, there exists τ_i, a_i such that $\mathbf{v}_i = \tau_i.\mathsf{fstate}, \mathbf{v}'_i = \tau_i.\mathsf{lstate}$ and $\mathbf{v}'_i \xrightarrow{a_i} \mathbf{v}_{i+1}$. It follows that $\tau_0 a_1 \tau_1 a_2 \ldots$ is an infinite execution fragment of $\mathcal{A}$. Recall (from Definition 4) that every infinite execution fragment of any hybrid automaton with positive ADT is admissible (i.e., time diverges), and therefore by Assumption 1, $\alpha.\mathsf{ltime} = \infty$, and $\mathcal{A}$ is not blocking. ∎

Theorem 1 which uses the assumptions about local closedness and ADT makes it possible to analyze blocking properties of hybrid automata using well-formed relations. Using Theorem 1 from [25], we obtain a (possibly) more practical condition for verifying the blocking nature of $\mathcal{A}$:

Theorem 2. *A locally closed $\mathcal{A}$ with positive ADT is blocking iff the transitive closure of Γ is contained in the disjunctive union of a finite collection of well-founded relations. That is, there exists a collection $\{R_1, \ldots, R_n\}$ of well founded relations such that $\Gamma^+ \subseteq \cup R_i$, where Γ^+ is the transitive closure of Γ.*

A pair of states $(\mathbf{v}, \mathbf{v}') \in \Gamma^+$ iff there exists an execution fragment α ending with a point trajectory with $\alpha.\mathsf{fstate} = \mathbf{v}$ and $\alpha.\mathsf{lstate} = \mathbf{v}'$. If $\mathbf{v}.loc = l$ and $\mathbf{v}'.loc = l'$ and $l \neq l'$ then the $(\mathbf{v}, \mathbf{v}')$ is contained in the simple well-founded relation $R \triangleq \{(\mathbf{v}, \mathbf{v}') \mid \mathbf{v}.loc = l \ \wedge \ \mathbf{v}'.loc = l'\}$.

Now for execution fragments that begin and end at the same location, instead of considering each such fragments individually, it is equivalent to consider a relation that captures all fragments that fit a particular pattern of alternating locations and actions and to show that this relation is contained in a well-founded relation.

We proceed by defining the relation corresponding to a *path* of $\mathcal{A}$ which is a sequence of locations and actions. Let $\sigma = l_0 a_1 l_1 \ldots l_n$ be a *path*. We define the relation $\rho_\sigma \subseteq val(V) \times val(V)$, as $(\mathbf{v}, \mathbf{v}') \in \rho_\sigma$ iff there exists an execution fragment $\alpha = \tau_0 a_1 \tau_1 \ldots \tau_n$ such that for each i in the sequence $\tau_i.\mathsf{fstate}.loc = l_i$. Under some additional assumptions which are discussed below, ρ_σ can be symbolically computed as:

$$\begin{aligned}
\rho_l(V, V') &= I_l(V) \ \wedge \ (\exists \ t \ E_l(V, V', t) \ \wedge \ I_l(V')), \\
&\quad \forall l \in L, && (2)\\
\rho_a(V, V') &= G_a(V) \ \wedge \ R_a(V, V'), \forall a \in A && (3)\\
\rho_{l,a}(V, V') &= \rho_l \circ \rho_a && (4)\\
\rho_\sigma(V, V') &= \rho_{l_0,a_1} \circ \rho_{l_1,a_2} \circ \rho_{l_2,a_3} \circ \ldots \circ \rho_{l_{n-1},a_n}. && (5)
\end{aligned}$$

Remark 1. (i) If σ is a cyclic path then the last action a_n must be a transition into location l_0 and the last location l_n contains not additional information. Hence, l_n does not appear in the above expression for ρ_σ. (ii) Clearly, the set of discrete transitions satisfying $\rho_a(V, V')$ are exactly those in $\mathcal{D}$ that are labeled by a. However, $\mathcal{T}_l$ is contained (and in general not equal to) in the set of trajectories satisfying $\rho_l(V, V')$: a trajectory τ that violates I_l at an intermediate point but not at the end-points satisfies $\rho_l(V, V')$. Equality holds under several conditions, for example, (a) I_l is convex and E_l is a linear function of time, and (b) I_l^c is closed under $\mathcal{T}_l$.

In view of the above remark, we have the following version of Theorem 3.

Theorem 3. *A locally closed $\mathcal{A}$ with positive ADT is blocking iff there exists a collection $\{R_1, \ldots, R_n\}$ of well-founded relations such that for every cyclic path σ, $\rho_\sigma \in R_i$ for some $i \in \{1, \ldots, n\}$.*

Example 1 The HA TwoTanks of Figure 1 is locally closed and has a positive dwell time. Its only cyclic path is $\sigma = l_1 \ a_1 \ l_0 \ a_0$. From the specification of the automaton we obtain:

$$\begin{aligned}
\rho_{l_0 a_0} &= 0 \leq x, y \leq 100 \wedge 40 \leq x' \leq 50 \wedge \\
&\quad y' \leq 10 \wedge x + 3y - 3y' \leq x' \leq 5y - 5y' + x \\
\rho_{l_1 a_1} &= 0 \leq x, y \leq 100 \wedge 20 \leq y' \leq 30 \wedge \\
&\quad y + 2/5(x - x') \leq y' \leq y + 4/5(x - x') \\
\rho_\sigma &= \exists x'', 0 \leq x', y' \leq 100 \wedge 20 \leq y'' \leq 30 \\
&\quad \wedge y' + 2/5(x' - x'') \leq y'' \leq y + 4/5(x' - x'') \\
&\quad \wedge 0 \leq x'', y'' \leq 100 \wedge 40 \leq x''' \leq 50 \wedge y''' \leq 10 \wedge \\
&\quad x'' + 3y'' - 3y''' \leq x''' \leq 5y'' - 5y''' + x''
\end{aligned}$$

Upon eliminating the quantifiers ρ_σ is simplified to:

$$\begin{aligned}
&0 \leq x \leq 100 \wedge 0 \leq y \leq 100 \wedge 40 \leq x' \leq 50 \ \wedge \ y' \leq 10 \\
&\wedge \ -25 \leq x' - x \leq -1 \ \wedge \ y' \geq y + 2,
\end{aligned}$$

which is a well-founded relation and can be automatically proved to be so with existing tools such as [24]. □

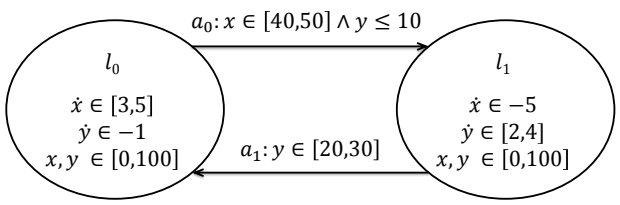

Fig. 1. Rectangular hybrid automaton TwoTanks.

B. Abstraction-Refinement Algorithm

For general hybrid automata, direct application of the proof rule given by Theorem 3 may yield step transition relations for which automatic quantifier elimination and checking of well-foundedness can be hard. This motivates the need for abstractions for verifying the blocking property.

Abstractions of hybrid automata have employed for making verification tractable from the beginning of this research area [3], [28], [27]. An abstraction for a hybrid automaton $\mathcal{A}$ is another (hybrid or finite state) automaton $\mathcal{B}$ such that every execution of $\mathcal{A}$ is simulated by some execution of $\mathcal{B}$. The abstraction $\mathcal{B}$ is defined in terms of an *abstraction function* $f_{abs} : val(V_A) \to val(V_\mathcal{B})$, or more generally a *simulation relation* $R_{sim} \subseteq val(V_A) \subseteq val(V_\mathcal{B})$, such that every transition and trajectory of $\mathcal{A}$ can be simulated by a transition (and trajectory in the case of hybrid abstractions) of $\mathcal{B}$ while preserving f_{abs} and R_{sim}.

Instead of abstracting the states and defining transitions and trajectories in this abstract state space, it is possible to abstract the hybrid step relation directly. The advantage of doing so in the context of termination analysis of programs has been illustrated in a sequence of papers by Cook, Podelski, and Rybalchenko [10].

An abstraction function abs maps a binary relation $\rho \subseteq val(V) \times val(V)$ to a superset $abs(\rho)$. In the spirit of ordinary predicate abstraction, we define the abstraction $abs_\mathcal{P}(\rho)$ with respect to a set of predicates $\mathcal{P}$, where each $p \in \mathcal{P}$ is a relation defining a subset of $val(V) \times val(V)$, as the smallest superset of ρ that can be constructed using conjunctions of the predicates in $\mathcal{P}$ that are weaker than ρ. Thus, if $\rho = p_1 \wedge p_2 \wedge \ldots\ p_k$ for some $p_1, p_2, \ldots, p_k \in \mathcal{P}$, then $abs(\rho) = \bigwedge_{i=1}^{k} p_i = \rho$. Abstraction function $\overline{abs}$ for a path $\sigma = l_0 a_1 l_1 \ldots l_n a_{n+1}$ is defined inductively as:

$$\begin{aligned} \overline{abs}_\mathcal{P}(\sigma) &= abs_\mathcal{P}(\rho_{l_0 a_1} \circ abs_\mathcal{P}(\rho)) \\ &\quad \text{where } \rho = abs_\mathcal{P}(l_1 a_2 l_2 \ldots l_n) \\ \overline{abs}_\mathcal{P}(l_n a_{n+1}) &= abs_\mathcal{P}(\rho_{l_n a_{n+1}}) \end{aligned}$$

From Theorem 3 and this notion of hybrid relation abstraction, a sufficient condition for blocking can be derived.

Corollary 4. *A locally closed $\mathcal{A}$ with positive ADT is blocking if there exists a collection of predicates $\mathcal{P}$ and a collection of well-founded relations $\mathcal{R}$ such that for every cyclic path σ, $abs_\mathcal{P}(\rho_\sigma) \in R_i$, for some $R_i \in \mathcal{R}$.*

A counter-example guided abstraction refinement (CEGAR) algorithm for verifying program termination has been presented in [10]. This algorithm can be adapted in a straightforward manner for proving the blocking properties of hybrid automata with abstraction and refinement. The key observations in [10] which lead to this algorithm are following: (a) A counterexample to the blocking property is a loop of the hybrid automaton, say σ, such that $abs(\rho_\sigma)$ is not well-founded. Thus, abstract counterexamples are found by systematically searching for loops that satisfy the above condition. (b) An abstract counterexample is actually a feasible counterexample of $\mathcal{A}$, and the corresponding concrete relation ρ_σ is not well-founded. (c) Otherwise, the abstract counterexample is spurious and this may be because of two reasons: (i) First, the abstraction function $abs_\mathcal{P}$ defined by the collection of predicates $\mathcal{P}$ is too coarse, that is, the abstract counterexample $abs_\mathcal{P}(\rho_\sigma) \nsubseteq R_i$ for any $R_i \in \mathcal{R}$, but $\rho_\sigma \subseteq R_i$ for some R_i. In this case a collection of predicates are added to $\mathcal{P}$ to eliminate the abstract counterexample $abs(\rho_\sigma)$. (ii) Second, the collection of disjunctive well founded relations $\mathcal{R}$ is not weak enough, that is, even the concrete relation ρ_σ is not included in any of the R_i's in $\mathcal{R}$. In this case, *some* well-founded relation containing ρ_σ is added to the collection $\mathcal{R}$.

$\mathcal{R} \leftarrow \emptyset$; $\mathcal{P} \leftarrow \emptyset$ 1
while 2
  if *exists* $\sigma = l_0 a_1 \ldots l_n a_{n+1}$ *s.t.* $\alpha_\mathcal{P}(\rho_\sigma) \nsubseteq R$ **for** *any* $R \in \mathcal{R}$ **then** 3
    if *exists* $R \in \mathcal{R}$ **such that** $\rho_\sigma \subseteq R$ **then** 4
      Refine Abstraction 5
      $\mathcal{P}_{path} \leftarrow \bigcup_{i \in 0..n} Preds(\rho_{l_i a_{i+1}} \circ \ldots \circ \rho_{l_n a_{n+1}})$ 6
      $\mathcal{P}_{loop} \leftarrow Preds(R) \cup \bigcup_{i \in 0..n} Preds(\rho_{l_i a_{i+1}} \circ \ldots \circ \rho_{l_n a_{n+1}} \circ R)$ 7
      $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}_{path} \cup \mathcal{P}_{loop}$ 8
    else 9
      if ρ_σ *is well-founded relation by a (new) ranking relation* R **then** 10
        Weaken disjunctive well-founded relation 11
        $\mathcal{R} \leftarrow \mathcal{R} \cup R$ 12
      else 13
        return "$\mathcal{A}$ *is not blocking,* $l_0 a_1 \ldots l_n a_{n+1}$" 14
  else 15
    return "$\mathcal{A}$ *is blocking*" 16
end 17

Fig. 2. Algorithm 1. Abstraction refinement based verification algorithm for blocking properties of hybrid automata.

Initially, the set of predicates and the well-founded ranking relations are empty. The algorithm explores the abstractions of the loops in the hybrid automaton in a depth-first manner as possible counterexamples to the blocking property. For each spurious counterexample, the algorithm (a) adds the corresponding ranking relations to $\mathcal{R}$, and (b) adds predicates to $\mathcal{P}$ for refining the abstraction. In doing so, in each refinement step there is progress in the sense that once a loop σ is analyzed and eliminated, it also eliminates an infinite collection of loop counterexamples of the form σ^i (the proof of

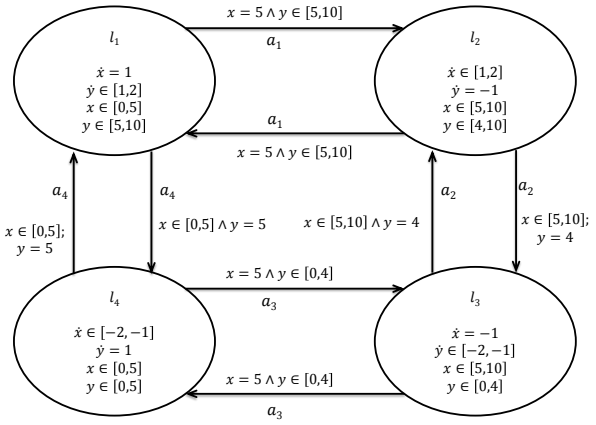

Fig. 3. Hybrid automaton Nav-1 modeling a 2-D Navigation Benchmark.

this is essentially the same as the proof of Theorem 3 in [10]). Furthermore, in Section IV-C we show that for initialized hybrid automata this procedure is actually complete. That is, if all the intermediate steps (computing ρ_σ and its abstraction, checking if it is well-founded, etc.) in the algorithm are effectively computable, the the procedure terminates with either a counterexample loop or establishes that $\mathcal{A}$ is not blocking.

1) Case Study : Nav-1*:* In this section, we illustrate the algorithm in Figure 2 by executing it step-by-step hybrid automaton Nav-1. For this purpose, we use the prototype tool that uses RankFinder [24].

We have a variant of 2 dimensional navigation example in Nav-1. The 2 dimensional plane is divided into 4 regions, and in each region, the direction of motion are given by the dynamics of the two variables x and y. Automaton Nav-1 consists of 4 locations $l_1, \ldots, l_4$; for each location, there are two incoming transitions and two outgoing transitions. As Nav-1 is a rectangular hybrid automaton, the hybrid step relations can be derived from the specification the guards, reset functions, the invariants and the differential equations. We first start by evaluating $\rho_{l_1 a_4}$. From the hybrid automaton, we have $\rho_{l_1 a_4} = \exists t > 0, x_1, y_1, 0 \leq x \leq 5 \wedge 5 \leq y \leq 10 \wedge x_1 = x + t \wedge y + t \leq y_1 \leq y + 2t \wedge 0 \leq x_1 \leq 5 \wedge y_1 = 5 \wedge x' = x_1 \wedge y' = y_1$. On simplification, we get that $\rho_{l_1 a_4} = \emptyset$. Similarly, we calculate the *hybrid step relation* for all the locations and actions as follows:

$$
\begin{aligned}
\rho_{l_1,a_4} &\equiv \emptyset \\
\rho_{l_1,a_1} &\equiv 0 \leq x \leq 5 \wedge 5 \leq y \leq 10 \wedge x' = 5 \\
&\quad \wedge 5 - x + y \leq y' \leq 10 - 2x + y \\
\rho_{l_2,a_1} &\equiv \emptyset \\
\rho_{l_1,a_2} &\equiv 5 \leq x \leq 10 \wedge 4 \leq y \leq 10 \wedge y' = 4 \\
&\quad \wedge x + y - 4 \leq x' \leq 2y + x - 8 \\
\rho_{l_3,a_2} &\equiv \emptyset \\
\rho_{l_3,a_3} &\equiv 5 \leq x \leq 10 \wedge 0 \leq y \leq 4 \wedge x' = 5 \\
&\quad \wedge 10 - 2x + y \leq y' \leq 5 + y - x \\
\rho_{l_4,a_3} &\equiv \emptyset \\
\rho_{l_4,a_4} &\equiv 0 \leq x \leq 5 \wedge 0 \leq y \leq 5 \wedge y' = 5 \\
&\quad \wedge x + 2y - 10 \leq x' \leq x + y - 5
\end{aligned}
$$

The algorithm considers all the loops in Nav-1. We start with the simple loops of length 2, such as, $\sigma = l_2 a_1 l_1 a_1$, and so on. For each of these loops σ the relation ρ_σ is proven to be well-founded, and then the algorithm moves on to loops of length 4, and so on. As we will observe, since all the loops of length 2 are not feasible, the only loops possible in this automaton are simple loops. We will prove that all these loops are well founded and hence the algorithm terminates after considering all the simple loops.

Step I/Line 1 Initially, the set of well-founded relations $\mathcal{R} = \emptyset$ and the set of transition predicates $\mathcal{P} = \emptyset$.

Step II/Lines 4, 5, 6, 7 and 8 We start enumerating simple loops and compute their abstractions. Because $\mathcal{R}$ is empty, we find that for the cyclic path $\sigma = l_2 a_1 l_1 a_1$. The abstract relation $\alpha_{\mathcal{P}}(\sigma)$ does not entail any relations in $\mathcal{R}$. This means that σ is a counter example. We now examine whether it is spurious or not. Now, we move to line 5 and since $\mathcal{R}$ is empty, there does not exist a relation R in $\mathcal{R}$ such that $\rho(\sigma) \subseteq R$. We hence move to line 10. We have $\rho(\sigma) = \rho_{l_2 a_1} \circ \rho_{l_1 a_1}$, we get

$$
\begin{aligned}
&= \exists x'', y'', \wedge((x', y'), (x'', y'')) \in \emptyset \\
&\quad \wedge 0 \leq x'' \leq 5 \wedge 5 \leq y'' \leq 10 \wedge x''' = 5 \\
&\quad \wedge 5 - x'' + y'' \leq y''' \leq 10 - 2x'' + y'' \\
&= false
\end{aligned}
$$

Since the relation $false$ is well founded, the counter example σ is spurious because the set of ranking functions is too strong, i.e. $\mathcal{R}$ is too restricted. The ranking function $\emptyset$ is the evidence that σ is well founded. Thus, we go to line 13 and add the empty relation $\emptyset$ to $\mathcal{R}$.

Step III/Lines 4, 5, 6, 7 and 8 We observe that the loop

$\sigma = l_2a_1l_1a_1$ is still a counter example because,

$$\begin{aligned}\alpha_{\mathcal{P}}(\sigma) &= \alpha_{\mathcal{P}}(\rho_{l_2a_1} \circ \alpha_{\mathcal{P}}(\rho_{l_1a_1})) \\ &= \alpha_{\mathcal{P}}(\rho_{l_2a_1} \circ True)) \\ &= True\end{aligned}$$

Now, since $True$ is not entailed in any of the well founded relations in $\mathcal{R}$, we go to line 5 and recall that this relation is well founded by the relation $\emptyset$. Thus, we add $Preds(\rho_{l_1a_1})$, $Preds(\rho_{l_2a_1} \circ \rho_{l_1a_1})$, $Preds(\rho_{l_1a_1} \circ R)$ and $Preds(\rho_{l_2a_1} \circ \rho_{l_1a_1} \circ R)$. Now, since $\rho_{l_2a_1} \circ \rho_{l_1a_1} = false$ and $R = \emptyset$, we add the following set of transition predicates

$$\begin{aligned}\mathcal{P} = \{&0 \leq x \leq 5, 5 \leq y \leq 10, \\ &x' = 5, 5 - x \leq y' - y \leq 10 - 2x\}.\end{aligned}$$

Step IV/Lines 4, 5, 6, 7 and 8 We note that after adding the set of predicates and the well founded relations, we can note that $\sigma = l_2a_1l_1a_1$ is no longer a counter example. In a similar manner, all the simple loops with two locations are not feasible and hence we can prove that all these loops are well founded and the set of predicates added after considering all these loops are

$$\begin{aligned}\mathcal{P} = \{&0 \leq x \leq 5, 5 \leq y \leq 10, \\ &x' = 5, 5 - x \leq y' - y \leq 10 - 2x, \\ &5 \leq x \leq 10, 4 \leq y \leq 10, y' = 4, \\ &y - 4 \leq x' - x \leq 2y - 8, \\ &5 \leq x \leq 10, 0 \leq y \leq 4, x' = 5, \\ &10 - 2x \leq y' - y \leq 5 - x, \\ &0 \leq x \leq 5, 0 \leq y \leq 5, y' = 5, \\ &2y - 10 \leq x' - x \leq y - 5\}\end{aligned}$$

Step V/Lines 4, 5, 6, 7 and 8 After eliminating all the simple loops of length 2, we now look at loops of length 4. Consider the loop $\sigma = l_1a_1l_2a_2l_3a_3l_4a_4$. We observe that $\alpha_{\mathcal{P}}(\sigma) \not\subseteq R$ for $R \in \mathcal{R}$. Hence, σ is a counter example. We now have to examine whether it is a spurious counter example or not. Now, we move to line 5 and since $\mathcal{R}$ has only one element namely $\emptyset$ as the set of well founded relations, there does not exist any relation R in $\mathcal{R}$ such that $\rho(\sigma) \subseteq R$. We construct $\rho(\sigma)$ as follows.

$$\begin{aligned}\rho(\sigma) = &\exists x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, \\ &0 \leq x \leq 5, y = 5, x_1 = x, y_1 = y, \\ &0 \leq x_1 \leq 5, 5 \leq y_1 \leq 10, x_2 = 5, \\ &5 - x_1 + y_1 \leq y_2 \leq 10 - 2x_1 + y_1, \\ &5 \leq x_2 \leq 10, 4 \leq y_2 \leq 10, y_3 = 4, \\ &x_2 + y_2 - 4 \leq x_3 \leq 2y_2 + x_2 - 8, \\ &5 \leq x_3 \leq 10, 0 \leq y_3 \leq 4, x_4 = 5, \\ &10 - 2x_3 + y_3 \leq y_4 \leq 5 + y_3 - x_3, \\ &0 \leq x_4 \leq 5, 0 \leq y_4 \leq 5, y_5 = 5, \\ &x_4 + 2y_4 - 10 \leq x_5 \leq x_4 + y_4 - 5.\end{aligned}$$

Simplification of the above relation will yield us the relation $0 \leq x \leq 5, x_5 \leq x - 2, 0 \leq x_5 \leq 5$. Hence we know that this loop will not run for infinite number of times. Thus, we go to line 13 and add the following ranking relation $R_1 = 0 \leq x \leq 5, 0 \leq x' \leq 5, x' \leq x - 2$ to $\mathcal{R}$.

Step VI/Lines 4, 5, 6, 7 and 8 We observe that the loop $\sigma = l_1a_1l_2a_2l_3a_3l_4a_4$ is still a counter example because we have $\alpha_{\mathcal{P}}(\sigma) = true$. However, we know that $\rho(\sigma) \subseteq R_1$. Thus, we just need to add the set of predicates given by $Preds(R_1)$, $\bigcup_{i \in 1..4} Preds(\rho(l_ia_i \ldots l_4a_4))$ and $\bigcup_{i \in 1..4} Preds(\rho(l_ia_i \ldots l_4a_4) \circ R_1)$. These predicates are as follows,

$$\begin{aligned}Preds = \{&0 \leq x \leq 5, 0 \leq x' \leq 5, x' \leq x - 2, \\ &0 \leq y \leq 4, x = 5, x' \leq 5 - y, y' = 5 \\ &5 \leq x \leq 10, y = 4, x' \leq 9 - x, y' = 5 \\ &x = 5, 5 \leq y \leq 10, x' \leq 8 - y, y' = 5 \\ &0 \leq x \leq 5, y = 5, x' \leq x - 2, y' = 5\}\end{aligned}$$

After adding these predicates, we can see that the loop $\sigma = l_1a_1l_2a_2l_3a_3l_4a_4$ is not a counter example. Hence all the feasible simple loops in the given hybrid automaton can be verified to be well founded and hence the hybrid automaton is blocking

2) Nav-2*:* Nav-2 is also a 2 dimensional navigation example which is very similar to Nav-1. We will precisely define Nav-2 by enumerating all the differences from Nav-1. The differences are *a)* The invariant for location l_3 is $5 \leq x \leq 10 \wedge 0 \leq y \leq 5$ *b)* The guard for transition a_2 is $5 \leq x \leq 10 \wedge y = 5$ *c)* The guard for transition a_2 is $5 \leq x \leq 10 \wedge y = 5$

Algorithm 1 returns that Nav-2 is not blocking and it finds an infinite execution of the system. As in the case of Nav-1, the Algorithm 1 eliminates all the simple loops of length 2 in Nav-2. However, when Algorithm 1 iterates over loops of length 4, it finds an actual counter example. Consider the loop $\sigma = l_1a_1l_2a_2l_3a_3l_4a_4$. After the abstraction, we get that σ is a counter example, because of the set of predicates. The algorithm, now

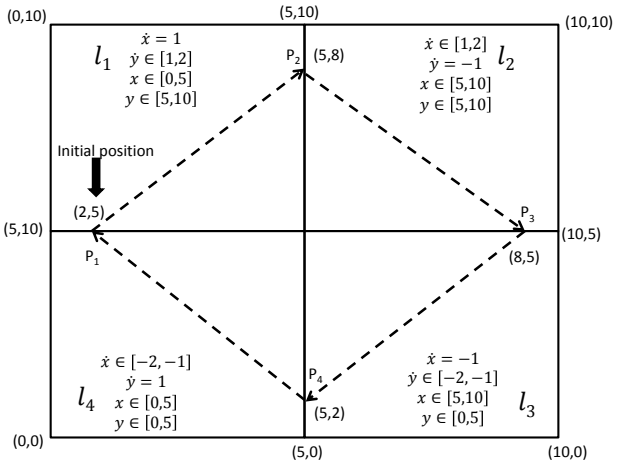

Fig. 4. Counter example in the Nav-2.

progresses to line 11, where ρ_σ is calculated similar to illustrated in Nav-1 and is examined whether it is well founded or not. If we calculate ρ_σ we get that $\rho_\sigma = 0 \leq x \leq 5, y = 5, 16x - 75 \leq x' \leq x, y' = 5$. When we give this relation to the ranking function synthesis tool, we are unable to give a well founded relation $R, \rho_\sigma \subseteq R$. Upon closer examination, we can see that there is an infinite execution of the hybrid automaton given as follows.

$$(x_0, y_0) = (2,5); loc_0 = l_1;$$
$$(2,5,l_1) \xrightarrow[3sec.]{} (5,8,l_1) \xrightarrow[a_1]{} (5,8,l_2) \xrightarrow[3sec.]{} (8,5,l_2)$$
$$\xrightarrow[a_2]{} (8,5,l_3) \xrightarrow[3sec.]{} (5,2,l_3) \xrightarrow[a_3]{} (5,2,l_3) \xrightarrow[3sec.]{} (2,5,l_3)$$
$$\xrightarrow[a_4]{} (2,5,l_1).$$

This counterexample is shown in the figure 4.

C. Completeness of Initialized Hybrid Automata

The abstraction refinement algorithm of Section IV-B, eliminates families of candidate counterexamples (to the blocking property) by simultaneously refining the abstraction and searching for disjoint well-founded relations that prove blocking. We have found that the algorithm terminates for several classes of hybrid automata in practice. In this section, we prove a simple completeness result which shows that the algorithm terminates for initialized hybrid automata.

A path σ is said to be an *action-cycle* if it is of the form $l_0 a \beta a l_n$, where $a \in A$ and β is a path. An action cycle σ is *simple* if no other actions repeat in σ except the first and the last actions. Theorem 5 implies that, at least for initialized hybrid automata, if all the intermediate steps of Algorithm 1 (e.g., finding and abstracting the hybrid step relations, checking their well-foundedness, etc.) are effectively computable, then the algorithm terminates because it suffices to search for counterexamples over simple action-cycles.

Theorem 5. *Consider a locally closed initialized hybrid automaton $\mathcal{A}$ with positive ADT. The following two statements are equivalent:*
(a) $\mathcal{A}$ is blocking.
(b) Every simple action-cycle of $\mathcal{A}$ is well-founded.

Proof: We prove $(a) \implies (b)$ by establishing its contrapositive. Suppose, there exists a simple action-cycle σ that is not well-founded. We can create an infinite path $\sigma' = \sigma \frown \sigma \frown \ldots$ by concatenating infinite number of copies of σ. Since ρ_σ not well-founded, it is non-empty and there exists an infinite execution fragment of $\mathcal{A}$ corresponding to σ'. The ADT property implies that such an infinite execution is actually admissible, i.e., time diverges.

We prove $(b) \implies (a)$ by establishing its contrapositive. Suppose α is an infinite execution of $\mathcal{A}$. Since α is an infinite sequence over finite set of actions A, there exists an action a which repeats infinitely many times in α. That is, α can be written as $\alpha = \alpha_0 a \alpha_1 a \alpha_2 a \ldots$, where each α_i is an execution fragment free of action a. Considering execution fragment α_1, we observe that α.fstate $\in R_a$ and α.lstate $\in G_a$, where R_a is the initialization set for action a and G_a is the guard set for action a. Thus, $\alpha' = \alpha_0 a \alpha_1 a \alpha_1 a \alpha_1 a \ldots$ is also an infinite execution of $\mathcal{A}$.

Case a. If α_1 does not have repeated actions then $a \sigma_1 a$ is a simple action-cycle which is not well-founded, where σ_1 is the path corresponding to the sequence of locations and actions in α. **Case b.** Suppose, α_1 has repeated actions, then we show that we can derive another execution fragment α_1', such that α_1'.fstate $\in R_a$ and α_1'.lstate $\in G_a$ and α' does not have repeated actions. Suppose, $\alpha_1 = \beta_1 b \beta_2 b \beta_3$ where $\beta_1, \beta_2, \beta_3$ are execution fragments and $b \in A$ is a repeated action in α_1. We define $\alpha_1' = \beta_1 b \beta_3$ and argue that α_1' is also a valid execution fragment of $\mathcal{A}$. Note that β_3.fstate, β_2.fstate are both in the set R_b. Since $\mathcal{A}$ is an initialized hybrid automaton, β_1.lstate $\xrightarrow{b}$ β_3.fstate is a valid transition, from which it follows that α_1' is a valid execution fragment of $\mathcal{A}$ with α_1'.fstate $\in R_a$ and α_1'.lstate $\in G_a$.

From α_1 we obtained α_1' by removing at least one pair of repeated actions, namely b. Since α_1 can only have a finite number of repeated actions, by performing the above operation repeatedly, we obtain an execution fragment α_1^* which does not have any repeated actions, and with α_1^*.fstate $\in R_a$ and α_1^*.lstate $\in G_a$. Then, $a \sigma_1^* a$ is a simple action-cycle which is not well-founded, where σ_1^* is the path corresponding to the sequence of locations and actions in α_1^*. ■

D. From Blocking to Stability

Having examined the blocking property and verification algorithms for it, we proceed to show how the

stability of a hybrid automaton $\mathcal{A}$ can be verified by proving that another automaton $\mathcal{B}$ is blocking. Informally, automaton $\mathcal{B}$ is obtained by removing the set S from $\mathcal{A}$.

Definition 5. *Given a hybrid automaton* $\mathcal{A} = \langle V_{\mathcal{B}}, A_{\mathcal{B}}, \mathcal{D}_{\mathcal{B}}, \mathcal{T}_{\mathcal{B}} \rangle$ *and a set of states* $S \subseteq val(V_{\mathcal{A}})$*, the* restriction of $\mathcal{A}$ to S*, denoted by* $\mathcal{A} \setminus S$*, is the hybrid automaton* $\mathcal{B} = \langle V_{\mathcal{B}}, A_{\mathcal{B}}, \mathcal{D}_{\mathcal{B}}, \mathcal{T}_{\mathcal{B}} \rangle$*, where:*

(a) $V_{\mathcal{B}} = \mathcal{V}_A$,
(b) $A_{\mathcal{B}} = A_{\mathcal{A}}$,
(c) $D_{\mathcal{B}} = \{(\mathbf{v}, a, \mathbf{v}') \in \mathcal{D}_{\mathcal{A}} \mid \mathbf{v} \notin S,\ \mathbf{v}' \notin S\}$, *and*
(d) $\mathcal{T}_{\mathcal{B}} = \{\tau \mid \forall\, t \in \tau.dom, t \neq \tau.\mathsf{ltime} \implies \tau(t) \notin S\}$.

Theorem 6. *If (1)* S *is closed, (2)* $\mathcal{A} \setminus S$ *is blocking, and (3)* $\mathcal{A}$ *is nonblocking, then* $\mathcal{A}$ *stable with respect to* S.

Proof: Suppose $\mathcal{A}$ is not stable with respect to S. By condition (1) in the hypothesis of the theorem, there exists an execution of α of $\mathcal{A}$ such that α is contained in S^c. By condition (3) there exists an extension α' of this execution α such that α' is $\alpha'.\mathsf{ltime} = \infty$. Since, every transition and trajectory in α' is a valid transition (respectively trajectory) of $\mathcal{A} \setminus S$, α' is a admissible execution of $\mathcal{A} \setminus S$, which contradicts (2). ■

Theorem 7. *If* $\mathcal{A}$ *is stable with respect to* S *implies that* $\mathcal{A} \lceil S$ *is blocking.*

Proof: Consider any execution α of $\mathcal{A} \setminus S$. Clearly it is also an execution of $\mathcal{A}$. Note that α is contained in S^c except possibly its last state. If α were admissible or zeno then, $\mathcal{A}$ would not be region stable with respect to S. Therefore, α must be closed, and it follows that $\mathcal{A} \setminus S$ is blocking. ■

Remark 2. Applying Theorem 6 for proving stability requires us to check that S is a closed set, blocking, and nonblocking properties. Previous section gives an algorithm for proving the blocking property. There are standard techniques for proving that a set is closed which are used in inductive safety verification (see, for example, [21], [33]). A sufficient condition for establishing that an automaton $\mathcal{A}$ is non-blocking is to prove that $\mathcal{A}$ has a positive dwell time or average dwell time and the algorithms in [20] can be used to that end.

V. Discussions and Future Work

We have presented a CEGAR-based approach for stability verification of cyber-physical systems. The procedure uses results from program analysis, specifically, abstraction-refinement with disjunctive union of well-founded relations. In order to adapt these results in the context of hybrid systems, we have characterized the blocking property of hybrid automata in terms of well-foundedness of a relation that combines the discrete transitions and the trajectories. This characterization requires the automaton to have a bounded switching speed, i.e., expressed in this paper as a positive average dwell time (though, unlike previous sufficient conditions involving ADT, our procedure does not require the individual modes of the hybrid system to be stable). We have also shown that the procedure is complete for certain classes of initialized hybrid automata (e.g., rectangular) for which the individual steps of the procedure can be effectively computed.

These suggest several directions of future research which would advance the area of CPS verification. First of all, we need to develop more powerful completeness results. One approach would be to find classes of CPS models where searching simple cycles are sufficient for finding counterexample to the blocking property. Secondly, an obvious direction is to generalize the CEGAR-based verification scheme to linear, affine, and polynomial hybrid systems. This direction is likely to involve have to use techniques of synthesizing non-linear ranking functions for validating counterexamples. A third direction of research is to explore how knowledge of Lyapunov functions for the individual modes can be used for stability verification. The ADT condition [13] and the multiple-Lyapunov function conditions [6] give sufficient conditions for stability based on the existence of Lyapunov functions. Can CEGAR-based techniques benefit from Lypunov functions in systems where the above results are not applicable? Finally, a lot of work remains to be done in developing software tools and in applying them to realistic CPS models.

Acknowledgment

We thank Andreas Podelski and Corina Mitrohin for providing valuable feedback on an earlier draft of this paper. This work is supported by the National Science Foundation (NSF) under contract number NSF CNS-1016791.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995. 1

[2] R. Alur, T. Dang, and F. Ivancic. Counter-Example Guided Predicate Abstraction of Hybrid Systems. In *TACAS 2003*, pages 208–223, 2003. 1

[3] R. Alur, T. Dang, and F. Ivančić. Counterexample-guided predicate abstraction of hybrid systems. *Theor. Comput. Sci.*, 354(2):250–271, 2006. 1, 5

[4] E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, volume 1790 of *LNCS*, pages 20–31. Hybrid Systems: Computation and Control, 2000. 1

[5] A. M. Bayen, E. Cruck, and C. Tomlin. Guaranteed overapproximations of unsafe sets for continuous and hybrid systems: solving the hamilton-jacobi equation using viability techniques. In Tomlin and Greenstreet [29], pages 90–104. 1

[6] M. Branicky. Multiple lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Transactions on Automatic Control*, 43:475–482, 1998. 2, 9

[7] A. Chutinan and B. H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. *LNCS*, 1569:76–91, 1999. 1

[8] E. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Abstraction and Counterexample-Guided Refinement in Model Checking of Hybrid Systems. *International Journal on Foundations of Computer Science*, 14(4):583–604, 2003. 1

[9] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement. In *CAV 2000*, pages 154–169, 2000. 1

[10] B. Cook, A. Podelski, and A. Rybalchenko. Abstraction refinement for termination. In *IN SAS2005: STATIC ANALYSIS SYMPOSIUM, VOLUME 3672 OF LNCS*, pages 87–101. Springer, 2005. 1, 2, 5, 6

[11] B. Cook, A. Podelski, and A. Rybalchenko. Termination proofs for systems code. In *PLDI '06: Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation*, pages 415–426, New York, NY, USA, 2006. ACM. 1

[12] S. Dolev. *Self-stabilization*. MIT Press, Cambridge, MA, USA, 2000. 1, 3

[13] J. Hespanha and A. Morse. Stability of switched systems with average dwell-time. In *Proceedings of 38th IEEE Conference on Decision and Control*, pages 2655–2660, 1999. 2, 3, 9

[14] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. *The Theory of Timed I/O Automata*. Synthesis Lectures on Computer Science. Morgan Claypool, November 2005. Also available as Technical Report MIT-LCS-TR-917. 2

[15] K.-D. Kim, S. Mitra, and P. R. Kumar. Bounded epsilon-reachability of linear hybrid automata with a deterministic and transversal discrete transition condition. In *Proceedings of The 49th IEEE Conference on Decision and Control (CDC)*. IEEE, 2010. 1

[16] A. B. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In *HSCC*, pages 202–214, 2000. 1

[17] S. M. A. Lakshmikantham, V. Leela. *Practical Stability of Nonlinear Systems*. World Scientific Pub. Co. Inc., 1990. 1

[18] D. Liberzon. *Switching in Systems and Control*. Systems and Control: Foundations and Applications. Birkhauser, Boston, June 2003. 2, 3

[19] S. Mitra. *A Verification Framework for Hybrid Systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, September 2007. 2

[20] S. Mitra, D. Liberzon, and N. Lynch. Verifying average dwell time of hybrid systems. *Trans. on Embedded Computing Sys.*, 8(1):1–37, 2008. 2, 9

[21] S. Mitra, Y. Wang, N. Lynch, and E. Feron. Safety verification of model helicopter controller using hybrid Input/Output automata. In O. Maler and A. Pnueli, editors, *HSCC*, volume 2623 of *LNCS*, pages 343–358. Springer, 2003. 9

[22] A. S. Morse. Supervisory control of families of linear set-point controllers, part 1: exact matching. *IEEE Transactions on Automatic Control*, 41:1413–1431, 1996. 2

[23] J. Oehlerking and O. Theel. Decompositional construction of lyapunov functions for hybrid systems. In R. Majumdar and P. Tabuada, editors, *Hybrid Systems: Computation and Control*, volume 5469 of *Lecture Notes in Computer Science*, pages 276–290. 2009. 2

[24] A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In *VMCAI*, pages 239–251, 2004. 1, 4, 6

[25] A. Podelski and A. Rybalchenko. Transition invariants. In *LICS '04: Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, pages 32–41, Washington, DC, USA, 2004. IEEE Computer Society. 1, 2, 4

[26] A. Podelski and S. Wagner. A sound and complete proof rule for region stability of hybrid systems. In *HSCC'07: Proceedings of the 10th international conference on Hybrid systems*, pages 750–753, Berlin, Heidelberg, 2007. Springer-Verlag. 1, 2

[27] P. Prabhakar, P. S. Duggirala, S. Mitra, and M. Viswanathan. Hybrid automata-based cegar for hybrid systems, 2010. In preparation: full version available from:users.crhc.uiuc.edu/mitras/research/2010/cegar-full.pdf. 1, 5

[28] A. Tiwari and G. Khanna. Series of abstractions for hybrid automata. In Tomlin and Greenstreet [29], pages 465–478. 1, 5

[29] C. Tomlin and M. R. Greenstreet, editors. *Hybrid Systems: Computation and Control, 5th International Workshop, HSCC 2002, Stanford, CA, USA, March 25-27, 2002, Proceedings*, volume 2289 of *LNCS*. Springer, 2002. 9, 10

[30] A. Turing. On computable numbers, with an application to the entscheidungsproblem. 2(42):230–265, 1936. 1

[31] A. van der Schaft and H. Schumacher. *An Introduction to Hybrid Dynamical Systems*. Springer, London, 2000. 2

[32] T. Wongpiromsarn, S. Mitra, R. Murray, and A. Lamperski. Verification of periodically controlled hybrid systems: Application to an autonomous vehicle. *Special Issue of ACM Transactions on Embedded Computing Sys. (TECS)*, 2010. To appear. 3

[33] T. Wongpiromsarn, S. Mitra, R. M. Murray, and A. G. Lamperski. Periodically controlled hybrid systems: Verifying a controller for an autonomous vehicle. In R. Majumdar and P. Tabuada, editors, *12th International Conference Hybrid Systems: Computation and Control (HSCC 2009)*, volume 5469 of *Lecture Notes in Computer Science*, pages 396–410. Springer, 2009. 3, 9