SPECIFICATION-GUIDED ANALYSIS OF HYBRID SYSTEMS USING A HIERARCHY OF VALIDATION METHODS

Olaf Stursberg [*] **Ansgar Fehnker** [**] **Zhi Han** [**]
Bruce H. Krogh [**]

[*] *University of Dortmund, Process Control Lab (CT-AST), 44221 Dortmund, Germany.*
Email: olaf.stursberg@uni-dortmund.de
[**] *Carnegie Mellon University, ECE Department 5000 Forbes Ave., Pittsburgh PA 15213-3890, USA.*
Email: {ansgar|krogh|zhih}@ece.cmu.edu

Abstract: As the key step in verifying hybrid systems, the computation of reachable sets largely determines the complexity and thus the applicability of a verification approach. Most existing methods compute the reachable set without considering the specification to be verified. This paper presents an approach that identifies for abstractions of the hybrid model those behaviors that potentially violate the specification. A tailor-made sequence of validation procedures then checks the existence of these behaviors for the original model. In many cases, the proposed iterative algorithm that combines model abstraction, behavior validation, and model refinement to verify the specification explores a considerably smaller part of the reachable set than standard techniques. The paper describes an implementation of the procedure and illustrates the approach for an automotive cruise control example. *Copyright © 2003 IFAC*

Keywords: Abstraction, Counterexample, Hybrid System, Model Refinement, Reachability Analysis, Verification.

1. INTRODUCTION

Verification, notably *model checking*, is currently viewed by the hybrid systems research community as a promising technique to prove certain model properties algorithmically. In the context of designing logic controllers for continuous or hybrid systems, model checking can determine if the controller design meets given specifications. Examples of approaches to verify logic controllers for hybrid systems with nonlinear continuous dynamics can be found, e. g., in (Puri and Varaiya, 1996; Henzinger and Wong-Toi, 1996; Chutinan and Krogh, 1999; Stursberg *et al.*, 1998). However, model checking is not yet a success story for real-world applications, mainly because the complexity of computation limits its applicability to rather small hybrid systems (Silva *et al.*, 2001).

The critical step with respect to complexity in hybrid system model checkers is the computation of reachable sets. Most existing approaches compute reachable sets without taking the specification into account, i.e. the reachable hybrid set is incrementally enlarged in all admissible directions until a state is encountered that violates the specification. This procedure often explores large sets of behaviors that are not relevant for the property to be investigated. In contrast, the approach presented here aims at first identifying the relevant behaviors by analyzing a discrete abstraction that can be constructed quickly. The costly computation of reachable hybrid states is then restricted to just those behaviors in the abstract system that possibly correspond to violations of the specification in the original hybrid system. These behaviors, called *counterexamples*, are then validated for the original model. If a counterexample cannot be validated for the latter, this result is used to refine the abstract model. The procedure of applying abstraction, model checking, and refinement iteratively was first introduced as *counterexample-guided verification* for finite transitions systems in (Clarke *et al.*, 2000). While the extension to general infinite state systems has recently been described in (Clarke *et al.*, 2003), this paper presents the following: a specific formulation for the class of hybrid automata with guards and invariants defined by polyhedral sets and nonlinear continuous dynamics; a hierarchy of validation methods for refuting counterexamples with the least costs possible; and a MATLAB-based implementation of the procedure. An automotive example is used to illustrate the approach.

The objective of reducing the computation of reachable hybrid sets can also be found in (Alur *et al.*, 2002). However, that approach uses different abstraction and refinement schemes and not a hierarchy of validation methods.

2. ANALYSIS OF HYBRID AUTOMATA

This section defines the type of model under consideration and the verification task to be solved.

Definition 1. Hybrid Automaton. A *hybrid automaton* $HA = (Z, z_0, X, X_0, inv, T, g, u, f)$ consists of:

- the finite set of *locations* $Z = \{z_1, \ldots, z_{n_z}\}$ with an *initial location* $z_0 \in Z$.
- the *continuous state space* $X \subseteq \mathbb{R}^n$ and the set of *initial continuous states* X_0 such that $X_0 \subseteq inv(z_0)$.
- the mapping $inv : Z \to 2^X$ which assigns an *invariant* of the form $inv(z) \subseteq X$ to each location $z \in Z$.
- the set of *discrete transitions* $T \subseteq Z \times Z$. A transition from $z_1 \in Z$ into $z_2 \in Z$ is denoted by (z_1, z_2).
- the function $g : T \to 2^X$ that associates a *guard* $g((z_1, z_2)) \subseteq X$ with each $(z_1, z_2) \in T$ such that $g((z_1, z_2)) \cap inv(z_1) \neq \emptyset$.
- the *jump* function $u : T \times X \to X$ which assigns an element of X to each $(z_1, z_2) \in T$ and $x \in g((z_1, z_2))$.
- the *flow function* $f : Z \times X \to \mathbb{R}^n$ that defines a continuous vector field $f(z, x)$ for each $z \in Z$. The evolution in z is governed by a differential equation $\dot{\chi}(t) = f(z, \chi(t))$, for which a unique solution for each $\chi(0) \in X_0$ is assumed to exist.

Let $s = (z, x)$ denote a *hybrid state* of HA. A sequence $\sigma = (s_0, s_1, s_2, \ldots)$ is then a feasible *run* of HA, iff $s_0 = (z_0, x_0)$ with $x_0 \in X_0$ and iff for each pair $(s_i, s_{i+1}) \in \sigma$ the state $s_{i+1} = (z_{i+1}, x_{i+1})$ results from $s_i = (z_i, x_i)$ by:

(a) a continuous evolution $\chi : [0, \tau] \to X$, $\tau \in \mathbb{R}^{>0}$ where: $\chi(0) = x_i$, $\dot{\chi}(t) = f(z_i, \chi(t))$, $\chi(t) \in inv(z_i)$ for $t \in [0, \tau]$,

(b) followed by a transition: $(z_i, z_{i+1}) \in T$, $\chi(\tau) \in g((z_i, z_{i+1}))$, and $x_{i+1} \in u((z_i, z_{i+1}), x_i) \cap inv(z_2)$. $\diamond$

Using arbitrary sets for X_0, the guards, and the invariants, as well as arbitrary jump functions make the computation of reachable sets intractable. Hence, the guards, invariants and the initial set are assumed to be bounded polyhedra [1]. Furthermore, an additional restriction is that the jump function u is defined for each transition as an affine mapping $x \to Ax + b$, with $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$. These restrictions are similar to that given in (Chutinan and Krogh, 1999).

We investigate the following property for HA:

Definition 2. Safety of HA. For HA as in Def. 1, let $z_b \in Z \setminus \{z_0\}$ denote an *unsafe* location. Then, HA is said to be *safe* with respect to z_b, iff all runs σ of HA do not contain a state $s_b = (z_b, x)$, $x \in X$. If this is true, we write $HA \models Spec$, otherwise $HA \not\models Spec$.

3. GUIDED VERIFICATION

The motivation of our approach to verifying safety of HA is the observation that existing algorithms do not use the specification to guide the search for a run that contains the unsafe state, i.e., s_b is solely used as a termination criterion but not to select runs which are explored first. The principle of our approach for *counterexample guided verification* is illustrated by the program flowchart in Fig. 1, which also represents the structure of an implementation in MATLAB. The system to be verified is modeled as HA – it is referred to as *concrete model*, denoted by C, from now on. The user additionally specifies an unsafe location z_b. A finite state transition system, called the *abstract model* A, is then derived from C by an abstraction step. The unsafe location z_b is represented by a state $\hat{s}_b$ of A. The abstraction ensures that each transition of the concrete model C is matched by a corresponding transition in the abstraction. Consequently, if a run that ends in location z_b exists for C, then a run that ends in $\hat{s}_b$ exists for A (see Sec. 3.1 for more details).

[1] A *half-space* is defined as the subset of $\mathbb{R}^n$ that satisfies a linear inequality $c \cdot x \leq d$, with $c \in \mathbb{R}^{1 \times n}, d \in \mathbb{R}$. The equality $c \cdot x = d$ is the *bounding hyperplane* of the half-space. A *polyhedron* is defined as the intersection of finitely many distinct half-spaces.

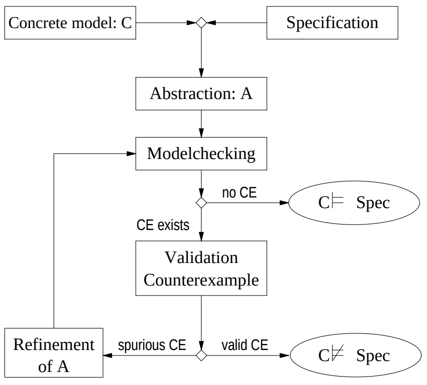

Fig. 1. Program flowchart.

The next step is then to apply model checking to A in order to analyze whether $\hat{s}_b$ is reachable in A. If this is not true, $C \models Spec$ can be concluded since A represents a superset of the behaviors of C. Otherwise a *counterexample* (CE) is obtained as a run of A from the initial state to $\hat{s}_b$. A *validation* step checks if the counterexample for A has a corresponding behavior in C. If so, the algorithm terminates with $C \not\models Spec$. If there is no corresponding behavior, information obtained from the counterexample (respectively a part of it) is used to refine the model A, such that this particular counterexample is excluded. The refined model is then examined for further existing counterexamples. The main steps of the approach are now described in detail.

3.1 Abstraction

To represent the abstract model, a transition system is defined as $TS = (\hat{S}, \hat{S}_0, \hat{E})$ with a finite state set $\hat{S}$, an initial set $\hat{S}_0 = \{s_0\} \subset \hat{S}$, and the transition set $\hat{E} \subseteq \hat{S} \times \hat{S}$. A *run* of TS is a sequence of states $\hat{\sigma} = (\hat{s}_0, \hat{s}_1, \hat{s}_2, \ldots)$ with $\hat{s}_i \in \hat{S}$ and $(\hat{s}_{i-1}, \hat{s}_i) \in \hat{E}$. A transition system $A = (\hat{S}, \hat{S}_0, \hat{E})$ is called an *abstract* model of C, if there exists an abstraction function $\alpha : Z \times X \to \hat{S}$, such that (i) $\hat{S}_0 = \alpha(\{z_0\} \times X_0)$, and (ii) a transition $(\hat{s}_1, \hat{s}_2) \in \hat{E}$ exists with $\alpha(s_1) = \hat{s}_1$ and $\alpha(s_2) = \hat{s}_2$ if C has a transition from $s_1 = (z_1, x_1)$ to $s_2 = (z_2, x_2)$. Given a run $\hat{\sigma} = (\hat{s}_0, \ldots, \hat{s}_n)$ of A, we call $\sigma = (s_0, s_1, \ldots, s_b)$ of C a *corresponding run* if for each pair (s_i, s_j) of consecutive states the following holds: $(\alpha(s_1), \alpha(s_2)) \in \hat{E}$.

For the procedure depicted in Fig. 1, we obtain an initial abstract model A from C as follows:

Definition 3. Initial abstract model of C. Given the concrete model C, the initial abstract model $A = (\hat{S}, \hat{S}_0, \hat{E})$ is obtained from an *abstraction function* $\alpha : Z \times X \to \hat{S}$ with:

$$\alpha(z, x) = \begin{cases} \hat{s}_0 & \text{if } z = z_0 \wedge x \in X_0 \\ \hat{s}'_0 & \text{if } z = z_0 \wedge x \in inv(z_0) \setminus X_0 \\ \hat{s}_b & \text{if } z = z_b \wedge x \in inv(z_b) \\ \hat{s}_i & \text{if } z_i \notin \{z_0, z_b\} \wedge x \in inv(z_i) \end{cases} \qquad (1)$$

and the transition set is given by: $\hat{E} = \{(\hat{s}_i, \hat{s}_j) | (z_i, z_j) \in T\} \cup \{(\hat{s}'_0, \hat{s}_j) | (z_0, z_j) \in T\} \cup \{(\hat{s}_i, \hat{s}'_0) | (z_i, z_0) \in T\}$ for $i, j \in \{1, \ldots, n_z\}$. ◇

All locations of C are mapped onto one abstract state, except for the initial location which is mapped onto two states: one to represent the initial set, and one to represent the remainder. Obviously, for each run σ of C a corresponding run $\hat{\sigma}$ of A exists, since (a) each state $s \in \sigma$ is represented by a state $\hat{s} \in \hat{S}$ and (b) each transition (s_i, s_{i+1}) with $s_i, s_{i+1} \in \sigma$ is mapped into a transition in $\hat{E}$. The converse is not true, however, i.e., a run $\hat{\sigma}$ of A might not have a corresponding run σ of C.

3.2 Model Checking

The safety property stated in Def. 2 for HA simply translates into the task of determining whether A has a path $\hat{\rho} = (\hat{s}_0, \ldots, \hat{s}_b)$, which is called *counterexample* of A. The determination of a counterexample can be accomplished by a simple explicit-state and breadth-first search through the transition structure of A. This search can be carried out very efficiently for those numbers of states in $\hat{S}$ (respectively locations of HA) that are usually encountered, i.e., the costs for this step are negligible compared to the costs required for the validation step.

3.3 Validation Strategy

Assume that model checking led to a counterexample $\hat{\sigma} = (\hat{s}_0, \hat{s}_1 \ldots, \hat{s}_p, \hat{s}_b)$. For each $\hat{s}_j \in \hat{\sigma}$, a *corresponding set of hybrid states* is defined by $S_j = \alpha^{-1}(\hat{s}_j)$. The validation routine checks by evaluating the hybrid dynamics of C whether a corresponding run $\sigma = (s_0, s_1, \ldots, s_b)$ of C exists. For each pair of consecutive states $(\hat{s}_i, \hat{s}_j)$ it is necessary to determine whether there exist states $s_i \in S_i$ and $s_j \in S_j$ that are connected by a continuous trajectory and a discrete transition. Different strategies are conceivable for the order in which the transitions of $\hat{\sigma}$ are validated. One strategy realized in the implementation is the validation along the counterexample starting with $\hat{s}_0$, and S_0 respectively.

Since the validation evaluates the dynamic behavior of C, it is by far the most expensive part of the verification procedure. We introduce a strategy that aims at reducing the effort as much as possible: Four different validation methods V_1, V_2, V_3, and V_4 are employed (as described in detail in Sec. 4). The methods check different (necessary and sufficient) conditions for the **non**-existence of transitions between states in S_i and S_j. The computational effort for the application of each method increases considerably from V_1 to V_4. The methods are hence applied from V_1 to V_4, in order to refute a transition with the least effort possible. The implementation offers two distinct options, the *sequential* and the *alternating* application of the methods. The sequential mode means that: (1.) V_1 is applied from transition $(\hat{s}_0, \hat{s}_1)$ up to transition $(\hat{s}_p, \hat{s}_b)$ of $\hat{\sigma}$, (2.) V_2 is applied to $(\hat{s}_0, \hat{s}_1)$ up to $(\hat{s}_p, \hat{s}_b)$, etc. The sequence of the alternating mode is in contrast: (1.) the transition $(\hat{s}_0, \hat{s}_1)$ is investigated by the methods V_1 up to V_4 in this order, (2.) $(\hat{s}_1, \hat{s}_2)$ is checked by V_1 up to V_4, etc. For both modes, the validation sequence is terminated as soon as a transition, and thus the counterexample $\hat{\sigma}$ is refuted.

However, to show that $(\hat{s}_0, \dots, \hat{s}_m)$ is spurious, it is sufficient to know that one of the transitions $(\hat{s}_i, \hat{s}_{i+1})$ is spurious. Also, the counterexample is spurious if one of the fragments $(\hat{s}_i, \hat{s}_{i+1}, \hat{s}_{i+2})$ is spurious, i.e., no corresponding trajectory (s_i, s_{i+1}, s_{i+2}) exists in C. Similarly fragments of any length $n \leq m$ can be defined. As shown for an example in Sec. 5, it can be advantageous to investigate fragments of increasing length to refute a counterexample as early as possible.

3.4 Refinement

The information obtained from the validation step is used to refine the abstract model in two distinct ways, the elimination of transitions and the splitting of states.

Definition 4. Refinement by Transition Elimination. For an abstract model A, let $(\hat{s}_i, \hat{s}_j) \in \hat{E}$ be a refuted transition of a counterexample. Then, a function ρ_{purge} maps A and $\alpha : S \to \hat{S}$ onto a *refined* abstract model $A' = (\hat{S}, \hat{s}_0, \hat{E}')$ with a new set of transitions $\hat{E}' = \hat{E} \setminus (\hat{s}_i, \hat{s}_j)$. ◇

The abstract model A is also refined if V_4 is applied during the validation process. In addition to checking the validity of a transition $(\hat{s}_i, \hat{s}_j)$ of $\hat{\sigma}$, this method also determines the corresponding reachable hybrid states in the following sense (see also Sec. 4): If $S_i = \alpha^{-1}(\hat{s}_i)$ and $S_j = \alpha^{-1}(\hat{s}_j)$, V_4 applies an operator $\mathit{Reach}(S_i) = S_j^r$ which returns S_j^r as the part of S_j that is reachable in C by trajectories corresponding to the transition $(\hat{s}_i, \hat{s}_j)$ of A. Once S_j^r is computed it is reasonable to use this information to update A by splitting $\hat{s}_j$ into two new abstract states (one of which represents the reachable part of S_j and the other one the remainder) and by re-arranging the transitions according to the reachability result:

Definition 5. Refinement based on Reachability. For models A and C, and an abstraction function $\alpha : S \to \hat{S}$, let $(\hat{s}_i, \hat{s}_j) \in \hat{E}$ be a transition of $\hat{\sigma}$, and $S_j^r = \mathit{Reach}(S_i)$ the result of V_4, where $S_i = \alpha^{-1}(\hat{s}_i)$, $S_j = \alpha^{-1}(\hat{s}_j)$, and $S_j^c = S_j \setminus S_j^r$. Then, a function $(A', \alpha', \alpha'') = \rho_{split}(A, \alpha, (\hat{s}_i, \hat{s}_j))$ leads to a *refined* abstract model $A' = \{\hat{S}', \hat{S}'_0, \hat{E}'\}$ with: $\hat{S}' = (\hat{S} \setminus \{\hat{s}_j\}) \cup \{\hat{s}_j^r, \hat{s}_j^c\}$ (iff S_j^r, S_j^c are non-empty), $\hat{S}'_0 = \{\hat{s}' \in \hat{S}' | \alpha''(\hat{s}') \in \hat{S}_0\}$, and $\hat{E}' = \{(\hat{s}'_i, \hat{s}'_j) | \ \exists\ (\hat{s}_i, \hat{s}_j) \in \hat{E} : \hat{s}_i = \alpha''(\hat{s}'_i) \wedge \hat{s}_j = \alpha''(\hat{s}'_j)\} \setminus (\hat{s}_i, \hat{s}_j^c)$. The *refined abstraction functions* $\alpha' : S \to \hat{S}'$ and $\alpha'' : \hat{S}' \to \hat{S}$ are:

- $\alpha'(s) = \begin{cases} \alpha(s) \text{ if } s \notin S_j \\ \hat{s}_j^r \quad \text{if } s \in S_j^r \ \wedge \ S_j^r \neq \emptyset \\ \hat{s}_j^c \quad \text{if } s \in S_j \setminus S_j^r \ \wedge \ S_j \setminus S_j^r \neq \emptyset \end{cases}$
- $\alpha''(\hat{s}') = \begin{cases} \hat{s}_j \quad \text{if } \hat{s}' \in \{\hat{s}_j^r, \hat{s}_j^c\} \\ \alpha(s) \text{ otherwise} \end{cases}$ ◇

4. VALIDATION METHODS

We now describe the validation methods V_1 to V_4 referred to in Sec. 3.3. In the following definitions, a concrete model C, an abstract model A, an abstraction function α, and a counterexample $\hat{\sigma}$ are given. Let $(\hat{s}_i, \hat{s}_j)$ denote a transition between two consecutive states in $\hat{\sigma}$, and the sets of hybrid states corresponding to $\hat{s}_i$ and $\hat{s}_j$ are: $S_i = \alpha^{-1}(\hat{s}_i)$, $S_j = \alpha^{-1}(\hat{s}_j)$. Furthermore, let $(z_i, z_j) \in T$ be a transition of C for which $s_i = (z_i, x)$ with $x \in \mathit{inv}(z_i)$ and $s_j = (z_j, x)$ with $x \in \mathit{inv}(z_j)$.

4.1 V1: Set Intersection

Definition 6. Validation V_1. The validation method V_1 identifies $(\hat{s}_i, \hat{s}_j)$ as refuted if: $\nexists\ (z_i, z_j) \in T$ such that $u((z_i, z_j), x) \in S_j$ for any $x \in g(z_i, z_j)$. ◇

The guard set is polyhedral and the update function linear, such that applying $u((z_i, z_j), x)$ to the bounding hyperplanes of $g(z_i, z_j)$ leads to a polyhedral set p^* again. Hence, $V1$ reduces to the simple check if $p^* \cap S_j$ is empty.

4.2 V2: Gradient Check

Definition 7. Validation V_2. Let $g(z_i, z_j)$ be the guard set of a transition $(z_i, z_j) \in T$. The validation method V_2 identifies $(\hat{s}_i, \hat{s}_j)$ as refuted if the following applies for each face q of the polyhedron $g(z_i, z_j)$: $\nexists\ x \in q$ with $c \cdot f(z_i, x) < 0$. (c denotes the normal vector of q and points out of $g(z_i, z_j)$.) ◇

This check can be implemented as an optimization problem $\min\limits_{x} c \cdot f(z_i, x)$ for each face. The solution is more costly than applying V_1.

4.3 V3: Trajectory Check

Definition 8. Validation V_3. Again, let $g(z_i, z_j)$ be the guard set of $(z_i, z_j) \in T$ and define $g'(z_i, z_j) = \{x \mid \exists\ u((z_i, z_j), x) \in S_j, x \in g(z_i, z_j)\}$. Let $\lambda(x) =$

$\|x - x_g\|_2$ define the distance between x and the nearest point x_g in $g'(z_i, z_j)$. The validation method V_3 declares $(\hat{s}_i, \hat{s}_j)$ as refuted if $\lambda_{min} > 0$ applies to the solution of:

$$\lambda_{min} = \min_{\chi(0)\in S_i,\ t\in[0,\tau(\chi(0))]} \lambda(\chi(t)) \qquad (2)$$

subject to: $\dot{\chi(t)} = f(z_i, \chi(t))$, $\chi(t) \in inv(z_i)$, and $\tau = (\exists\ \chi(0) \in S_i : \chi(\tau^+) \notin inv(z_i))$. $\diamond$

The implementation of V_3 comprises a nonlinear optimization with embedded numerical simulation to compute the trajectories. (An upper time bound $\tau \leq \tau_{max}$ is required for the simulation if a trajectory does never leave $inv(z_i)$.)

4.4 V4: Flowpipe Approximation

Let $Reach(S_i) = S_j^r$ denote the operator that returns $S_j^r = \{s_j \mid \exists\ \sigma = (\ldots, s_i, s_j, \ldots), s_i \in S_i, s_j \in S_j\}$. Applying this operator consists of the following three steps: (a) determining the reachable subset $g'(z_i, z_j) \subseteq g(z_i, z_j)\}$ of the transition guard; (b) computing a polyhedron p' that tightly envelops $g'(z_i, z_j)$, e.g., the convex hull; (c) applying the update function $u((z_i, z_j), x)$ to the faces of p' and intersecting the result with S_j to obtain S_j^r. The most difficult part of this procedure is the computation of $g'(z_i, z_j)$. Our implementation uses the principle of computing *polygonal flowpipe approximations* as described in (Chutinan and Krogh, 1999). The idea is to use nonlinear optimization to get a set of polyhedra that tightly enclose all trajectories inside of $inv(z_i)$ that start from S_i. Since the steps (a) and (b) of the above procedure yield conservative approximations, the result is an over-approximation $\tilde{S}_j^r \supseteq S_j^r$. Since the optimization is carried out for each face of a series of polyhedra, this method is considerably costlier than V_3. This is the reason for preferring the other methods whenever possible. However, the implementation also comprises less expensive methods to compute $\tilde{S}_j^r$ for the cases that $f(z, x)$ is linear or describes constant rates.

5. EXAMPLE

The approach is illustrated by an adaptive cruise control example that is part of a vehicle-to-vehicle coordination system (Girard *et al.*, 2001). A part is considered that comprises two modes: the cruise control (*cc*-mode) in which a car tries to keep a constant speed, and an adaptive cruise control (*acc*-mode) in which the car aims at keeping a safe distance to a preceding car. The system includes also an automatic transmission system with four gears. The controller switches into the *acc*-mode whenever the distance r between the car and a preceding vehicle is below a desired distance r_{des} minus a hysteresis parameter h. The controller

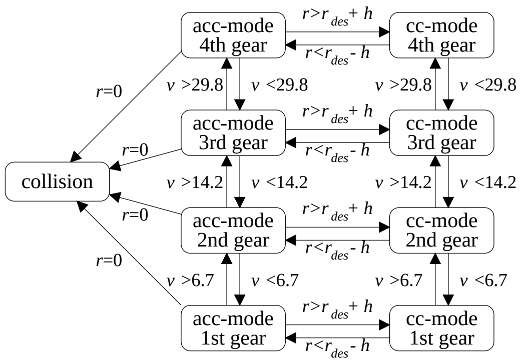

Fig. 2. The discrete part of the hybrid automaton.

switches back if $r > r_{des} + h$. The distance r_{des} depends linearly on the velocity v of the following car. The overall hybrid automaton has 8 locations for the normal operation and one additional unsafe location that is entered in the case of a collision (Fig. 2). Given the (constant) velocity of the leading car v_{lead}, the distance changes according to $\dot{r} = v_{lead} - v$ in all locations. The controller determines the desired acceleration a_{des} and velocity v_{des} of the follower according to this rule: the desired acceleration a_{des} is proportional to $v_{des} - v$ in the *cc*-mode, and equal to $k_1(v_{lead} - v) + k_2(r - r_{des})$ in the *acc*-mode (with constants k_1, k_2). This sliding mode controller ensures that the system is asymptotically stable in $v = v_{lead}$ and $r = r_{des}$ (if a_{des} can be realized). To assess the approach presented here, it is compared with the tool CheckMate (Chutinan and Krogh, 1999). The latter uses only the validation method V_4 and explores the hybrid state space according to a breadth-first like enlargement of the reachable set. CheckMate applies the method V_4 six times to verify that the system is safe. This takes 325 CPU-seconds on a Pentium 3, 600 MHz PC. The shaded area in Fig 3.a depicts the continuous states that are found to be reachable. All trajectories eventually reach the equilibrium point $v = v_{lead} = 25$ m/sec and $r = r_{des} = 42.5$ m.

First an instance of the procedure for counterexample-guided verification is applied, in which complete counterexamples are validated by applying V_1, V_2, and V_4 in sequential order. Eleven counterexamples are generated overall, the costliest method V_4 is applied five times, and 150 CPU-sec are required. As shown in Fig. 3.b, this procedure does not require to compute the reachable set in the location 3^{rd} gear and *acc*-mode. The method V_3 is sufficient to show that the transition into the unsafe state cannot occur. Another instance of the procedure uses the following scheme: First the methods V_1 and V_3 are applied to each single transition of a counterexample, then to each pair of successive transitions of the counterexample. In a third step, V_1, V_3, and V_4 are applied to pairs of transitions, and then to the complete counterexample. This scheme leads to 10 coun-

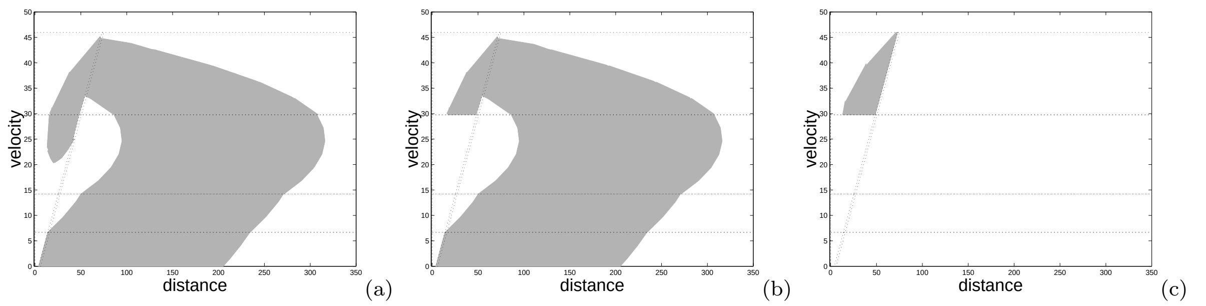

Fig. 3. The shaded areas mark the reached continuous states that were computed by V_4 with: (a) CheckMate, (b) guided verification by validating complete counterexamples, (c) guided verification using fragments of counterexamples. The diagonal dotted lines are the conditions for switching between the *acc*- and the *cc*-mode. The horizontal lines are the conditions for switching between gears. The system is initially in location 1st gear and the *cc*-mode (which corresponds to the lower right polyhedron), and the distance r is initially below 200 m.

terexamples, and the complete computation took only 60 seconds. Seven of the 10 counterexamples could be refuted by applying V_1 and/or V_3 to a single transition. V_4 was used only once to show that once the system switches from the 4th gear and the *cc*-mode to the *acc*-mode, it will inevitably switch to 3rd gear from which collision is not possible (Fig. 3.c).

6. CONCLUSIONS

This paper combines two new ideas to make hybrid system verification more efficient: the guidance of search by using abstract models to identify runs that potentially violate the specification, and the use of a hierarchy of validation methods to refute counterexamples with the least costs possible. The advantages of the approach can be summarized as follows: If C fulfills the specification, it can happen that no counterexample exists and the algorithm terminates immediately. A breadth-first approach with just the validation method V_4 (the procedure that exists in tools like CheckMate) would compute the complete reachable hybrid set to get the same result. Even if the specification-guided search has to refute a set of counterexamples, their validation corresponds in many cases to an exploration of only a small subset of the complete reachable hybrid set. Additionally, some of these counterexamples may be refutable by using a low-cost validation method. If $C| \neq Spec$, the first counterexample may be already one that violates the specification. Then, again only a small portion of the hybrid state space is explored. Also note that by the iterative refinement of the abstract model, the elimination of a transition can immediately remove a whole set of counterexamples.

ACKNOWLEDGEMENTS

The authors appreciate thorough discussions with E. Clarke, J. Ouaknine, and M. Theobald on various issues of the approach presented.

REFERENCES

Alur, R., T. Dang and F. Ivancic (2002). Reachability analysis of hybrid systems via predicate abstraction. In: *Hybrid Systems: Comp. and Control.* Vol. 2289 of *LNCS*. pp. 35–48.

Chutinan, A. and B.H. Krogh (1999). Verification of polyhedral-invariant hybrid automata using polygonal flowpipe-approximation. In: *Hybrid Systems: Comp. and Control.* LNCS 1569. Springer. pp. 76–90.

Clarke, E., A. Fehnker, Z. Han, B.H. Krogh, O. Stursberg and M. Theobald (2003). Verification of hybrid systems based on counterexample-guided abstraction refinement. In: *Proc. 9th Int. Conf. TACAS.* Vol. 2619 of *LNCS*. Springer. pp. 192–207.

Clarke, E., O. Grumberg, S. Jha, Y. Lu and H. Veith (2000). Counterexample-guided abstraction refinement. In: *Computer-Aided Verification.* Vol. 1855 of *LNCS*. Springer. pp. 154–169.

Girard, A.R., J.B. Souza, J.A. Misener and J.K. Hedrick (2001). A control architecture for integrated cooperative cruise control and collision warning systems. In: *Proc. 40th IEEE Conf. on Decision and Control.*

Henzinger, T. A. and H. Wong-Toi (1996). Linear phase-portrait approximations for nonlinear hybrid systems. In: *Hybrid Systems III.* Vol. 1066 of *LNCS*. Springer. pp. 377–388.

Puri, A. and P. Varaiya (1996). Verification of hybrid systems using abstractions. In: *Proc. 13th IFAC World Congress.* pp. 477–482.

Silva, B.I., O. Stursberg, B. Krogh and S. Engell (2001). An assessment of the current status of algorithmic approaches to the verification of hybrid systems. In: *Proc. 40th IEEE Conf. on Decision and Control.* pp. 2867–2874.

Stursberg, O., S. Kowalewski, J. Preussig and H. Treseler (1998). Block-diagram based modelling and analysis of hybrid processes under discrete control. *J. Europeen des Syst. Automatises* **32**(9-10), 1097–1118.