Journal of the Indian Institute of Science
A Multidisciplinary Reviews Journal
ISSN: 0970-4140 Coden-JIISAD

REVIEWS

Conflict-Tolerant Specifications for Hybrid Systems

Deepak D'Souza[1], Madhu Gopinathan[1], S. Ramesh[2] and Prahladavaradan Sampath[3]

Abstract | We propose a framework for developing and reasoning about hybrid systems that are comprised of a plant with multiple controllers, each of which controls the plant intermittently. The framework is based on the notion of a "conflict-tolerant" specification for a controller, and provides a modular way of developing and reasoning about such systems. We propose a novel mechanism of defining conflict-tolerant specifications for general hybrid systems, using "acceptor" and "advisor" components. We also give a decision procedure for verifying whether a controller satisfies its conflict-tolerant specification, in the special case when the components are modeled using initialized rectangular hybrid automata.

1 Introduction

Modern embedded systems, ranging from washing machines to automobiles, contain a large number of features, many of which are realised as software controllers which advise the base system (or plant) on how to conform to the feature's specification. In the automotive industry for example, advanced features such as adaptive cruise control, collision avoidance, and electronic stability control[15] are developed as part of a software product line, and a subset of these features are integrated into automotive products based on market needs and regulatory requirements. In such a setting, the controllers do not always agree on how the system should behave, and could reach a point of "conflict" at which they offer conflicting advice. Such conflicts are usually resolved by a supervisory controller which may for example shut down a controller of lower priority until the system reaches a state where the supervisor deems it safe to resume the feature. Controllers for each feature are typically thus only *intermittently* in control of the system during its execution.

One would like to reason about such systems in a *compositional* manner, beginning with the correctness of each feature with respect to the base system, and then inferring a property of the whole system. Unfortunately, traditional specification mechanisms such as a safety specification which prescribes a prefix-closed set of behaviours, are not adequate in this setting. Fig. 1(a) illustrates a classical safety specification for a feature, shown as a shaded cone growing to the right with time, and a system trajectory σ. Once the system behaviour leaves the specified safety cone, due to say the feature being suspended, the controller has *no* specification to adhere to once it is resumed at time t.

In this paper we provide a framework that addresses some of these problems in the setting of hybrid system models. Our solution is based on the notion of "conflict-tolerance" introduced in earlier work in a discrete and real-time setting.[6,7] The starting point of this framework is the notion of a *conflict-tolerant specification*. A conflict-tolerant specification is an *advice function* f that specifies for every plant behaviour σ, a prefix-closed set of all future behaviours that are considered safe. This is illustrated in Fig. 1(b), which shows the initial safety cone, and the safety cone specified after the system has exhibited behaviour σ. A controller for the plant satisfies a conflict-tolerant specification f if after every plant behaviour σ (possibly *not* according to the controller's advice), the *subsequent* behaviours of the plant that are according to the controller's advice, are all contained in $f(\sigma)$.

To illustrate how a conflict-tolerant specification can capture a specifier's intent more richly than a classical specification, consider a feature which requires the water level in a tank to be always between 2 and 4 cm. A classical specification for this feature is shown in Fig. 2(a). The invariant $2 \leq w \leq 4$ captures the desired requirement. If the controller for this feature is suspended due to a conflict, then the water level may rise above

[1]*Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India.*
deepakd@csa.iisc.ernet.in
gmadhu@csa.iisc.ernet.in
[2]*General Motors R&D, Warren.*
ramesh.s@gm.com
[3]*Mathworks India, Bangalore.*
Prahlad.sampath@ieee.org

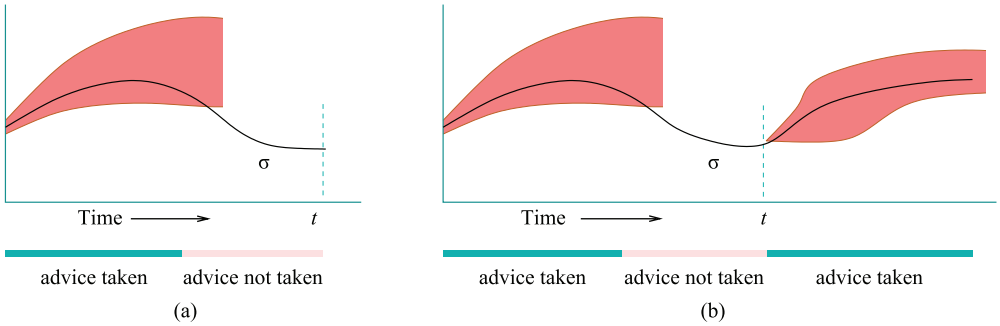

Figure 1: (a) A classical (safety) specification, and (b) a conflict-tolerant specification.

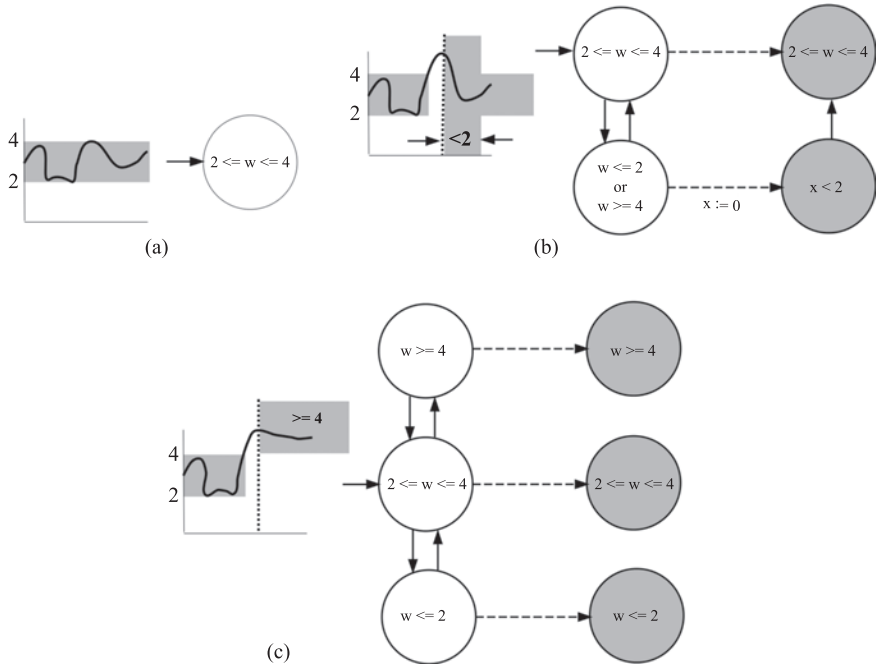

Figure 2: A classical specification (a), and conflict-tolerant specifications (b) and (c).

4 cm or fall below 2 cm. The specification does not specify the desired behaviour if the water level falls below 2 cm or rises above 4 cm. In contrast, the tolerant specification in Fig. 2(b) specifies that ideally the water level should be between 2 and 4 cm and if this is violated, then the water level must be brought between 2 and 4cm in less than 2 seconds after resuming the controller (the time of resumption is indicated by the dotted line). The hybrid automaton shown in Fig. 2(b) specifies an advice function in the following way. It has two components: an "acceptor" automaton shown on the left side, and an "advisor" automaton shown on the right side (with shaded states). The safety language advised for a behaviour σ is obtained by running the acceptor automaton on σ to reach a configuration c, then applying the function given by the dashed edges to obtain a configuration d of the advisor automaton, and then collecting all possible behaviours allowed by the advisor automaton starting from the configuration d. In a similar manner, the tolerant specification in Fig. 2(c) specifies that ideally the water level should be between 2 and 4 cm, and if this is violated, then the water level must be kept below 2 cm if it is currently below 2cm, and kept above 4 cm if it is currently above 4 cm.

We note that both the tolerant specifications of Fig. 2(b) and (c) induce the same classical specification shown in Fig. 2(a), in that the behaviours of the plant that are *always* according to their advice coincide with the safety language of Fig. 2(a). However, as tolerant specifications they are quite different.

Let us now consider a system that comprises a base system $\mathscr{B}$ with multiple conflict-tolerant controllers $\mathscr{C}_1, \ldots, \mathscr{C}$, each of which are known to satisfy their conflict-tolerant specifications

$\mathscr{S}_1,\ldots,\mathscr{S}_k$, respectively. Let these controllers be supervised by an arbitrary supervisory controller. Then we can give the following guarantee on the behaviour of the composed system: For each controller $\mathscr{C}_i$, in each interval of time during which it's advice is taken, the behaviour of the system in this interval is *according to* the tolerant-specification $\mathscr{S}_i$. Thus, in each of the intervals in which it is in control, each controller does "the right thing". This is illustrated in Fig. 1(b).

In this paper, our contributions are the following. We propose a novel mechanism to specify conflict-tolerant specifications for general hybrid systems, using acceptor and advisor components. We also show how we can algorithmically solve the controller verification problem—i.e. whether a controller satisfies its conflict-tolerant specification with respect to a given base system—when the plant, the controller, and the acceptor and advisor components of the tolerant specification are given as initialized rectangular hybrid automata.

We would like to emphasize that our framework is *modular*, in that each controller need only be specified, developed and verified *once*, regardless of which other controllers and supervisors it is integrated with.

In earlier work[6–8] we have proposed a conflict-tolerant framework in the discrete and timed settings. Apart from the fact that we deal with more general behaviours in the form of signals, the work here departs from our earlier work in several ways.

- In earlier work, our tolerant specifications comprised a single component with "advised" and "not-advised" transitions. The notion of acceptor and advisor components used in this paper have enabled us to create a more general specification mechanism, not limited to only "consistent" specifications.
- In the hybrid setting, the specification and control mechanisms are very different from each other. The specification speaks only about the plant behaviours, whereas the controller has to control the plant using only the input signals. This is in contrast to the earlier settings in which the tolerant specification is essentially itself a valid controller for the plant.

Related Work. In[10] it is argued that system verification must be decomposed by features as every feature naturally has an associated property to be verified. There are several approaches in the literature where features are specified as finite state automata and a conflict is detected by checking whether a state in which the features advise conflicting system actions, is reached.[14] The problem of conflict detection at the specification stage is addressed in,[9] where conflict between two feature specifications in temporal logic is detected automatically. Our approach of viewing features as controllers[17] follows that of[5,19] in the setting of discrete events. In both these works, the main issue addressed is that of resuming the advice of a controller once it has been suspended due to conflict with a higher priority controller. In,[5] specifications are designed to anticipate conflict, by having two kinds of states, *in-spec* and *out-of-spec*. When a controller's specification is violated it transitions to an out-of-spec state from where it passively observes the system behaviour, till it sees a specified event that brings it back to an in-spec state. Note that these controllers do not offer any useful advice in the out-of-spec states. In[12] a rule-based feature model and composition operators for resolving conflicts based on prioritization is presented. However, the notion of a conflict-tolerant *specification* (as against the feature implementation itself) is absent in their work.

In the hybrid setting, the work in[18] synthesizes controllers that satisfy safety specifications by computing a maximal safe set of states from which there exists a strategy to meet its specification. Outside this safe set, the synthesized controller allows any control input. In our setting this corresponds to synthesizing a controller (if one exists) for simple forms of tolerant specifications that comprise essentially the same safety cone for each possible behaviour.

The rest of the paper is structured as follows: After preliminary definitions, in Section 3 we introduce our control setting and illustrate conflict between two controllers. We then introduce the notion of conflict-tolerance in Section 4. In Section 5, we address the verification problem. We conclude with a discussion in Section 6.

2 Preliminaries

We model the behaviour of a hybrid system using the notion of a signal. Let W be a set of variables. A valuation for the variables W is a function $\mathbf{w} : W \to \mathbb{R}$. We denote by $\mathbf{W}$ the set of all valuations for the variables in W. For a valuation $\mathbf{w}$ over W and a subset X of W we denote by $\mathbf{w} \triangleright X$ the standard projection of the valuation to the variables in X.

A *signal* over a set of variables W is a function $\sigma : I \to \mathbf{W}$ where the domain I is an interval $[0,r)$ for some $r \geq 0$ and σ has only finitely many points of discontinuity. Thus, there is a strictly increasing sequence of time points $t_0 = 0 < t_1 < t_2 < \cdots < t_n = r$ such that for every $k \in 0,\ldots,n-1$, σ is continuous

in the interval $I_k = [t_k, t_{k+1})$. Note that in each interval I_k, σ is right continuous at t_k. Our definition of a signal is similar to that of temporal behaviour in.[4] We denote by $\mathscr{W}$ the set of all signals over the set of variables W. A *signal language* over W is simply a subset of $\mathscr{W}$. For a signal σ over W and a subset X of W we denote by $\sigma \trianglerighteq X$ the projection of the signal σ to the set of variables X.

Let $\varepsilon : [0,0) \to \mathbf{W}$ denote the empty signal, i.e. the signal whose domain is an empty set. Let $\sigma_1 : [0,t_1) \to \mathbf{W}$ and $\sigma_2 : [0,t_2) \to \mathbf{W}$ be signals in $\mathscr{W}$. We define the *concatenation* of σ_1 and σ_2, denoted $\sigma_1 \cdot \sigma_2$, to be the signal $\sigma : [0, t_1 + t_2) \to \mathbf{W}$ given by

$$\sigma(t) = \begin{cases} \sigma_1(t) & \text{if } t < t_1 \\ \sigma_2(t - t_1) & \text{if } t_1 \le t < t_1 + t_2 . \end{cases}$$

For signals $\sigma_1, \sigma_2 \in \mathscr{W}$, we say that σ_1 is a *prefix* of σ_2, denoted $\sigma_1 \preceq \sigma_2$, if there exists a signal $\sigma \in \mathscr{W}$ such that $\sigma_1 \cdot \sigma = \sigma_2$. We say that a signal language L over W is *prefix-closed* if whenever $\tau \in L$ and $\sigma \preceq \tau$, we have $\sigma \in L$. For a set of signals $L \subseteq \mathscr{W}$ and a signal σ, we denote by $ext_\sigma(L)$, the *set of extensions* of σ that are in L, i.e.

$$ext_\sigma(L) = \{\tau \in \mathscr{W} \mid \sigma \cdot \tau \in L\}.$$

Our definition of a hybrid automaton is adapted from the definitions of hybrid automata in the literature.[1,4,18] We consider "open" hybrid automata in which only some of the variables are controlled by the automaton. We first define the components of a hybrid automaton and give an informal description of what the components stand for. The way in which the components are used will be described formally when we define a trajectory of a hybrid automaton.

Definition 1 (Hybrid Automaton). *A* hybrid automaton *is a tuple*

$$\mathscr{H} = \langle Q, V, C, F, init, tcp, \to \rangle$$

where

- *Q is a finite set of discrete variables with a finite range of values. The finite set $\mathbf{Q}$ is the set of modes of $\mathscr{H}$.*
- *V is a finite set of state variables. The set $\mathbf{V}$ represents the states of $\mathscr{H}$. A configuration of $\mathscr{H}$ is a pair $(\mathbf{q},\mathbf{v}) \in \mathbf{Q} \times \mathbf{V}$. The set $C \subseteq V$ is the set of variables controlled by $\mathscr{H}$, i.e. $\mathscr{H}$ constrains the initial value, continuous flow and resets of variables in C. Let $\bar{C}$ denote the set of uncontrolled variables of $\mathscr{H}$, i.e. $\bar{C} = V \setminus C$.*
- *Let $\mathbf{D} = C \to 2^{\mathbb{R}}$ be the set of maps specifying a set of derivatives for variables in C. Then, $F : \mathbf{Q} \to (\mathbf{V} \to \mathbf{D})$ assigns to each mode a* flow condition *which constrains the time derivative of the continuous flow of the controlled variables. Thus, for $\mathbf{q} \in \mathbf{Q}$ and $\mathbf{v} \in \mathbf{V}$, $F(\mathbf{q})(\mathbf{v}) = \mathbf{d}$ says that when current state of the system is $\mathbf{v}$ in mode $\mathbf{q}$, the time derivative of the signal for each variable $c \in C$ must be in $\mathbf{d}(c)$. Alternatively, $\mathbf{D}$ can be looked at as defining a set-valued vector field $\vec{\boldsymbol{D}}$ which prescribes the laws of continuous flow for the controlled variables. We assume that for a discrete variable u, the signals of u are constant in every mode.*
- *$init \subseteq \mathbf{Q} \times \mathbf{C}$ specifies a set of initial configurations given by $\{(\mathbf{q},\mathbf{v}) \in \mathbf{Q} \times \mathbf{V} \mid (\mathbf{q},\mathbf{v} \triangleright C) \in init\}$. Thus, init does not constrain the initial values of the uncontrolled variables.*
- *$tcp : \mathbf{Q} \to 2^{\mathbf{V}}$ assigns to each mode a **time can progress** condition. If $v \in tcp(\mathbf{q})$, then it is possible for the automaton $\mathscr{H}$ to evolve continuously from the state $\mathbf{v}$.*
- *$\to \subseteq \mathbf{Q} \times 2^{\mathbf{V}} \times (\mathbf{V} \to 2^{\mathbf{C}}) \times \mathbf{Q}$ is a **jump relation**. Thus for a jump $e = (\mathbf{q}, \mathbf{g}, reset, \mathbf{q}') \in \to$, $\mathbf{q}$ is the source mode, $\mathbf{q}'$ is the target mode, $\mathbf{g}$ is the subset of states from which the jump e is enabled and reset : $\mathbf{V} \to 2^{\mathbf{C}}$ is a function which gives the possible values that the controlled variables can be reset to after the jump e.*

For a mode $\mathbf{q} \in \mathbf{Q}$, by $F_\mathbf{q}$ we mean the flow condition $F(\mathbf{q})$, and by $tcp_\mathbf{q}$ we mean the time can progress condition $tcp(\mathbf{q})$. Note that *init* does not constrain the initial valuations of the uncontrolled variables. Similarly, the flow condition $F_\mathbf{q}$ does not constrain the continuous flow of the uncontrolled variables. However, the set $tcp_\mathbf{q}$ can be used to express any assumptions on the valuations of the uncontrolled variables and the controlled variables in a given mode $\mathbf{q}$. It can also be used to force a change of mode and thereby change the laws of continuous flow prescribed by $F_\mathbf{q}$.

Let $(\mathbf{q},\mathbf{v})$, $(\mathbf{q},\mathbf{v}')$ be configurations of $\mathscr{H}$. Let $\tau : [a,b) \to \mathbf{V}$ be a function which is continuous in $[a,b)$ and differentiable in (a,b). We say that $\mathscr{H}$ *evolves continuously* from $(\mathbf{q},\mathbf{v})$ to $(\mathbf{q},\mathbf{v}')$ with signal τ, written $(\mathbf{q},\mathbf{v}) \xrightarrow{\tau} (\mathbf{q},\mathbf{v}')$, iff the following conditions are satisfied:

- $\tau(a) = \mathbf{v}$,
- $\tau(b^-) = \mathbf{v}'$ where $\tau(b^-)$ is the left limit of τ at b,
- for all $t \in [a,b)$, $\tau(t) \in tcp_\mathbf{q}$, and
- For $c \in C$, let τ_c denote the signal $\tau \trianglerighteq \{c\}$. Then for all $t \in (a,b)$ and for all controlled variables $c \in C$, $\dot{\tau}_c(t) \in F_q(\tau(t))(c)$, i.e. the signal for the controlled variables obeys the flow condition of mode $\mathbf{q}$.

We say that $\mathscr{H}$ *jumps* from $(\mathbf{q},\mathbf{v})$ to $(\mathbf{q}',\mathbf{v}')$, written $(\mathbf{q},\mathbf{v}) \to (\mathbf{q}',\mathbf{v}')$ iff there exists $(\mathbf{q},\mathbf{g}, reset, \mathbf{q}')$

$\in \rightarrow$ such that $\mathbf{v} \in \mathbf{g}$, $\mathbf{v}' \triangleright C \in reset(\mathbf{v})$, and $\mathbf{v}' \triangleright \bar{C} = \mathbf{v} \triangleright \bar{C}$.

Definition 2 (Trajectory of $\mathscr{H}$). A trajectory of $\mathscr{H}$ *starting from a configuration* $(\mathbf{q},\mathbf{v})$ *is a pair of signals* (v,σ) *where* $v : I \rightarrow \mathbf{Q}$ *and* $\sigma : I \rightarrow \mathbf{V}$, *satisfying the following conditions. Let* $I = [0,r)$ *with* $r \geq 0$, *and let* $t_0, \ldots, t_n$ *be the union of the points of discontinuity in the signals* v *and* σ. *Let* I_k *be the interval* $[t_k, t_{k+1})$. *Then*

- *Either* $v(0) = \mathbf{q}$ *and* $\sigma(0) = \mathbf{v}$, *or* $v(0) = \mathbf{q}'$ *and* $\sigma(0) = \mathbf{v}'$ *and* $(\mathbf{q},\mathbf{v}) \rightarrow (\mathbf{q}',\mathbf{v}')$. *Thus, the trajectory starts either with a continuous evolution from* $(\mathbf{q},\mathbf{v})$ *or with a jump from* $(\mathbf{q},\mathbf{v})$ *to* $(\mathbf{q}',\mathbf{v}')$.
- *For every* k *from* 0 *to* $n-1$, v *is a constant function in the interval* I_k, *and* σ *is continuous in the interval* I_k.
- *Let* $\mathbf{q}_k$ *be the value of* v *in the interval* I_k. *Then* $\mathscr{H}$ *can evolve continuously from* $(\mathbf{q}_k, \sigma(t_k))$ *to* $(\mathbf{q}_k, \sigma(t_{k+1}^-))$ *via the signal* σ *restricted to the interval* $[t_k, t_{k+1})$.
- *For every* k *from* 1 *to* $n-1$
  - *either* $(q_{k-1}, \sigma(t_k^-)) \rightarrow (q_k, \sigma(t_k))$, *i.e.* $\mathscr{H}$ *jumps*,
  - *or* $\mathbf{q}_{k-1} = \mathbf{q}_k$ *and* $(\sigma \trianglerighteq C)(t_k^-) = (\sigma \trianglerighteq C)(t_k)$, *i.e.* $\mathscr{H}$ *does not jump and the controlled variables do not change.*

Let $Traj_{(\mathbf{q},\mathbf{v})}(\mathscr{H})$ denote the set of trajectories of $\mathscr{H}$ starting from a configuration $(\mathbf{q},\mathbf{v})$. Let $Traj(\mathscr{H})$ denote the set of trajectories of $\mathscr{H}$ starting from any $(\mathbf{q},\mathbf{v})$ which satisfies *init*, i.e.

$$Traj(\mathscr{H}) = \{\tau \in Traj_{(\mathbf{q},\mathbf{v})}(\mathscr{H}) \mid (\mathbf{q},\mathbf{v} \triangleright C) \in init\}.$$

The *signal language* of $\mathscr{H}$ starting from a given configuration $(\mathbf{q},\mathbf{v})$, denoted $L_{(\mathbf{q},\mathbf{v})}(\mathscr{H})$ is defined as $L_{(\mathbf{q},\mathbf{v})}(\mathscr{H}) = \{\sigma \mid \exists v : (v, \sigma) \in Traj_{(\mathbf{q},\mathbf{v})}(\mathscr{H})\}$. The *signal language* of $\mathscr{H}$, denoted $L(\mathscr{H})$, is defined as $L(\mathscr{H}) = \{\sigma \mid \exists v : (v, \sigma) \in Traj(\mathscr{H})\}$ We say a hybrid automaton $\mathscr{H}$ is *deterministic* if for any signal $\sigma \in L(\mathscr{H})$, there is exactly one trajectory of the form $(v,\sigma) \in Traj(\mathscr{H})$.

Let $X \subseteq V$ and let $\tau : [0,r) \rightarrow \mathbf{X}$ be a signal over X. We define the *configurations* of $\mathscr{H}$ after τ, denoted $config_{\mathscr{H}}(\tau)$, to be

$$config_{\mathscr{H}}(\tau) = \{(\mathbf{q},\mathbf{v}) \mid \exists (v,\sigma) \in Traj(\mathscr{H}) \text{ s.t. } v(r^-) = \mathbf{q} \text{ and } \sigma(r^-) = v \text{ and } \sigma \trianglerighteq X = \tau\}.$$

We define the language of $\mathscr{H}$ *after* σ to be

$$L_\sigma(\mathscr{H}) = \bigcup_{(\mathbf{q},\mathbf{v}) \in config_{\mathscr{H}}(\sigma)} L_{(\mathbf{q},\mathbf{v})}(\mathscr{H}).$$

Let $\mathscr{H}_1 = (Q_1, V, C_1, F_1, init_1, tcp_1, \rightarrow_1)$ and $\mathscr{H}_2 = (Q_2, V, C_2, F_2, init_2, tcp_2, \rightarrow_2)$ be two hybrid automata over the same set of variables V. The automata $\mathscr{H}_1$ and $\mathscr{H}_2$ may represent two controllers that control the same input variables to a plant, and hence $C_1 \cap C_2$ may not be empty. The *synchronized product* of $\mathscr{H}_1$ and $\mathscr{H}_2$, denoted $\mathscr{H}_1 || \mathscr{H}_2$ is defined to be the hybrid automaton $\mathscr{H} = (Q,V,C,F,init,tcp,\rightarrow)$ where

- $Q = Q_1 \times Q_2$,
- $C = C_1 \cup C_2$,
- Let $\mathbf{D} = C \rightarrow 2^{\mathbb{R}}$ be a set of maps specifying a set of derivatives for variables in C. Then $F : \mathbf{Q} \rightarrow (\mathbf{V} \rightarrow \mathbf{D})$ is given by $\forall \mathbf{q}_1, \mathbf{q}_2 \in \mathbf{Q}, \forall v \in \mathbf{V}, \forall \mathbf{d} \in \mathbf{D}, \mathbf{d} \in F(\mathbf{q}_1,\mathbf{q}_2)(\mathbf{v})$ iff $\mathbf{d} \triangleright C_1 \in F_1(\mathbf{q}_1)(\mathbf{v})$ and $\mathbf{d} \triangleright C_2 \in F_2(\mathbf{q}_2)(\mathbf{v})$,
- $init \subseteq \mathbf{Q} \times \mathbf{C}$ where $((\mathbf{q}_1,\mathbf{q}_2),c) \in init$ iff $(\mathbf{q}_1, \mathbf{c} \triangleright C_1) \in init_1$ and $(\mathbf{q}_2, \mathbf{c} \triangleright C_2) \in init_2$,
- $tcp : \mathbf{Q} \rightarrow 2^{\mathbf{V}}$ where $tcp((\mathbf{q}_1,\mathbf{q}_2)) = tcp_1(\mathbf{q}_1) \cap tcp_2(\mathbf{q}_2)$,
- $((\mathbf{q}_1,\mathbf{q}_2), \mathbf{g}, reset, (\mathbf{q}_1',\mathbf{q}_2')) \in \rightarrow$ if one of the conditions below is satisfied:
  - $\mathbf{q}_1 = \mathbf{q}_1'$, $\exists (\mathbf{q}_2, \mathbf{g}_2, reset_2, \mathbf{q}_2') \in \rightarrow_2$ s.t. $\mathbf{g} = \mathbf{g}_2$ and $\mathbf{c} \in reset(\mathbf{v})$ iff $\mathbf{c} \triangleright C_2 \in reset_2(\mathbf{v})$ and $\mathbf{c} \triangleright C_1 = \mathbf{v} \triangleright C_1$.
  - $(\mathbf{q}_1, \mathbf{g}_1, reset_1, \mathbf{q}_1') \in \rightarrow_1$, $\mathbf{g} = \mathbf{g}_1$ and $\mathbf{c} \in reset(\mathbf{v})$ iff $\mathbf{c} \triangleright C_1 \in reset_1(\mathbf{v})$, $\mathbf{c} \triangleright C_2 = \mathbf{v} \triangleright C_2$, and $\mathbf{q}_2 = \mathbf{q}_2'$.
  - $(\mathbf{q}_1, \mathbf{g}_1, reset_1, \mathbf{q}_1') \in \rightarrow_1, (\mathbf{q}_2, \mathbf{g}_2, reset_2, \mathbf{q}_2') \in \rightarrow_2$, $\mathbf{g} = \mathbf{g}_1 \cap \mathbf{g}_2$ and $\mathbf{c} \in reset(\mathbf{v})$ iff $\mathbf{c} \triangleright C_1 \in reset_1(\mathbf{v})$ and $\mathbf{c} \triangleright C_2 \in reset_2(\mathbf{v})$.

Proposition 1. *Let* $\mathscr{H}_1$ *and* $\mathscr{H}_2$ *be two open hybrid automata. Then* $L(\mathscr{H}_1 || \mathscr{H}_2) = L(\mathscr{H}_1) \cap L(\mathscr{H}_2)$.

3 Controllers and Conflict

In this section we introduce our control setting, safety specifications, and conflict between controllers. We use a running example to illustrate these notions.

We consider a control setting that is based on a partitioned set of variables (X,U,Y). Here X is the set of variables controlled by the plant, U is the set of input variables of the plant used by the controller to control it and Y is the set of auxiliary variables used by the specification (for example clock variables to represent timing constraints).

Definition 3 (Plant) *Let* $V = X \cup U$. *A* ***plant*** *over* (X,U,Y) *is a deterministic hybrid automaton* $\mathscr{P}$ *over* V, *which controls the variables in* X. *We assume that the plant* $\mathscr{P}$ *is* ***non-blocking*** *in that if* $\sigma \in L(\mathscr{P})$, *then* $L_\sigma(\mathscr{P}) \neq \{\varepsilon\}$.

As a running example, we consider a car under the control of two features: *cruise control* and *electronic stability control*. For the car, we use a simple model in which it is assumed that the friction retarding the motion of the car is proportional to the car's speed. The equation of

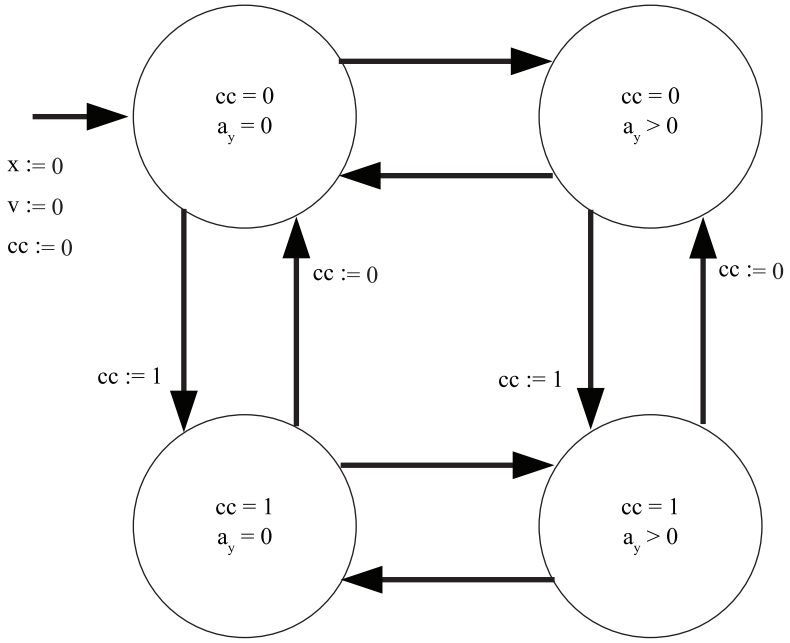

Figure 3: Plant model of car. Flow conditions (not shown in the diagram) are $\dot{x} = v$ and $\dot{v} = -\frac{b}{m}v + \frac{1}{m}u$ in all control states.

motion can then be written as $u - bv = m\dot{v}$ where u is the force imparted by the engine, v is the velocity, bv is the frictional force and m is the mass of the car.

A plant model of the car is shown in Fig. 3. The set of plant variables is $X = \{x, v, a_y, cc\}$ where x denotes the position of the car, v the velocity, a_y denotes the sensed lateral acceleration and cc is a discrete variable which denotes whether the cruise control feature is off ($cc = 0$) or on ($cc = 1$). The set of input variables is $U = \{u\}$ where u denotes the acceleration or braking force applied on the car. Initially, the position and velocity are zero, and cruise control is off. The flow conditions of the plant are $\dot{x} = v$ and $\dot{v} = -\frac{b}{m}v + \frac{1}{m}u$ in all control states. We use the convention that if a (controlled) variable not mentioned in the reset of an edge, then its value unchanged in the reset.

Definition 4 (Controller)**.** ***A controller*** *over (X,U,Y) is a deterministic hybrid automaton* $\mathscr{C} = (Q, V, U, F, init, tcp, \rightarrow)$ *over V which controls the variables in U. The controller* $\mathscr{C}$ *is valid with respect to a plant* $\mathscr{P}$ *over (X,U,Y) if* $\mathscr{C}$ *is* ***non-blocking*** *with respect to* $\mathscr{P}$*, i.e. if* $\sigma \in L(\mathscr{P}||\mathscr{C})$*, then* $L_\sigma(\mathscr{P}||\mathscr{C}) \neq \{\varepsilon\}$.

A classical *safety specification* over (X,U,Y), is a deterministic hybrid automaton $\mathscr{H}$ over $X \cup Y$ which prescibes a language $L(\mathscr{H})$ representing the set of behaviours it considers safe. We note that the language $L(\mathscr{H})$ is necessarily prefix-closed. Let $\mathscr{P}$ be a plant over (X,U,Y), $\mathscr{C}$ be a controller for $\mathscr{P}$ and $\mathscr{S}$ be a safety specification over (X,U,Y). We say that the controller $\mathscr{C}$ for $\mathscr{P}$ satisfies $\mathscr{S}$ if

$$L(\mathscr{P}||\mathscr{C}) \trianglerighteq X \subseteq L(\mathscr{S}) \trianglerighteq X.$$

As an example, we consider a cruise control feature which should satisfy the following requirements once the driver turns on cruise control and sets a reference speed V_r.

- There are two modes of operation: *gradual* mode and *rapid* mode.
- In the gradual mode, the rise time, i.e. the time it takes for the car to reach the reference speed, can be up to 20 seconds. In the rapid mode, the rise time must be less than 5 seconds.
- The steady state error must be less than a given threshold ε, that is $|V_r - v| < \varepsilon$.

Figure 4 shows a specification which captures the above requirements. If the difference between the current speed and the reference speed is greater than 10, then the car must rapidly accelerate so that the speed is equal to the reference speed within 5 seconds. Otherwise, the car can gradually accelerate to reach the reference speed within 20 seconds.

The controllers are implemented using proportional, integral control. The input u is of the form [16]

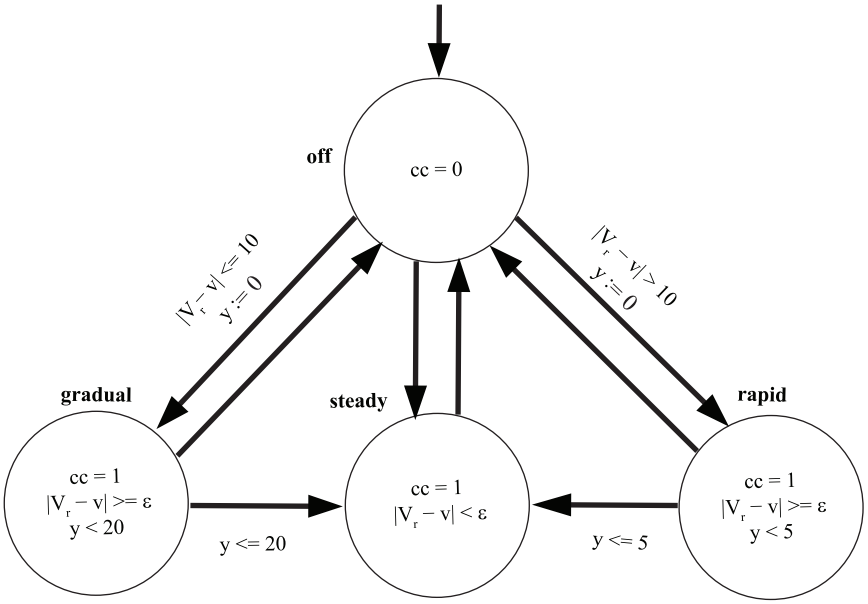

Figure 4: Cruise Control Specification $\mathscr{S}_{cc}$. Here y is a clock variable with $\dot{y} = 1$.

$$u(t) = K_p(V_r - v(t) + K_i(V_r \cdot t - x(t))$$

where K_p is the proportional gain and K_i is the integral gain. Figure 5 shows a controller that satisfies the given specification. For the rest of the paper, we fix the mass of the car $m = 1000$ kg, the coefficient of proportionality $b = 50$ N · s/m, the reference speed set by the driver $V_r = 30$ m/s and the steady state error threshold $\varepsilon = 5\%$ of V_r. Figures 6 and 7 show the response of the controlled system when cruise control is turned on with the reference speed $V_r = 30$ m/s and the car is running at 10 m/s and at 22 m/s respectively. The given implementation of the controller is valid with respect to the plant $\mathscr{P}$ and satisfies the cruise control specification $\mathscr{S}_{cc}$.

Now consider another feature called ele*ctronic stability control* which is used to improve the stability of the car. When the car is traveling on a curve of radius r, the lateral acceleration is given by $a_y = v^2/r$. The stability control feature requires the lateral acceleration to be under a certain threshold l_{th}.

Figure 8 shows a specification for this feature which requires that if the lateral acceleration exceeds the threshold l_{th}, then it must be brought under the threshold within 2 seconds.

Figure 9 shows the response of the controlled system when stability control intervenes and reduces the speed by applying a braking force, thus reducing lateral acceleration below the given threshold. In this example, we set the lateral acceleration threshold l_{th} to $0.7g = 6.86$ m/s^2 and assume that the car is traveling on a curve of radius 100 m.

We now illustrate the notion of conflict between controllers.

Definition 5 (Conflict). *Let $\mathscr{C}_1$ and $\mathscr{C}_2$ be valid controllers for a plant $\mathscr{P}$. The controllers $\mathscr{C}_1$ and $\mathscr{C}_2$ are in* ***conflict*** *with respect to $\mathscr{P}$, if there exists a behaviour τ in $L(\mathscr{P}\|\mathscr{C}_1\|\mathscr{C}_2)$ such that $L_\tau(\mathscr{P}\|\mathscr{C}_1\|\mathscr{C}_2) = \{\varepsilon\}$, i.e. after the plant behaviour τ, the controllers $\mathscr{C}_1$ and $\mathscr{C}_2$ do not agree on any extension of τ.*

Consider the example plant behaviour shown in Fig. 10. The driver turns on cruise control when the car is traveling at a speed of 22 m/s (79 km/hr). Later, the car travels on a curve and the lateral acceleration exceeds the safety threshold requiring stability control to intervene after 23 seconds. The cruise control specification requires the car to be within 5% of the target speed of 30 m/s (108 km/hr) whereas the stability control specification requires the speed to be reduced so that the lateral acceleration falls below the safety threshold.

Let us say the controllers are overseen by a supervisory controller, that intervenes in favour of stability control to continue with its advice while disregarding the advice of the cruise controller. If the car now reaches a stable state where the lateral acceleration is within the threshold, and the superviser seeks to resume the cruise control feature, the cruise controller has *no* meaningful specification to adhere to, since the plant's

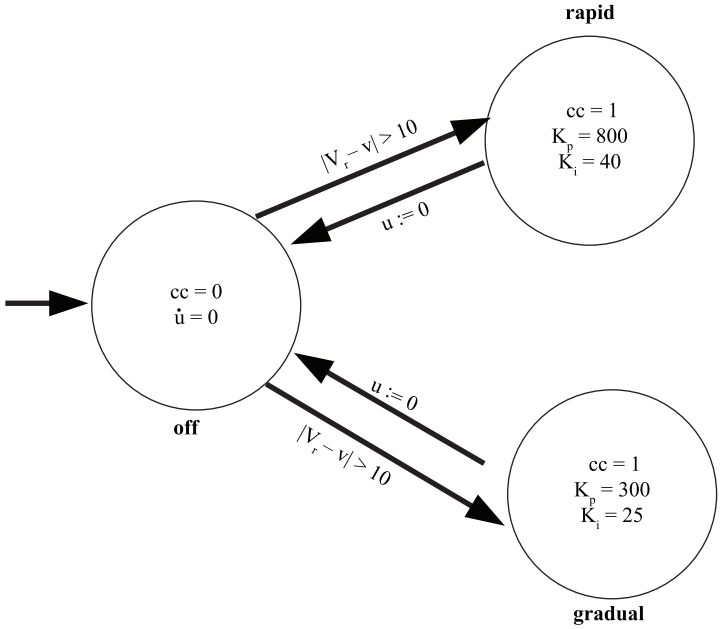

Figure 5: Cruise control implementation. In gradual and rapid modes, $\dot{u} = -K_p\dot{v} + K_i(V_r - \dot{x})$.

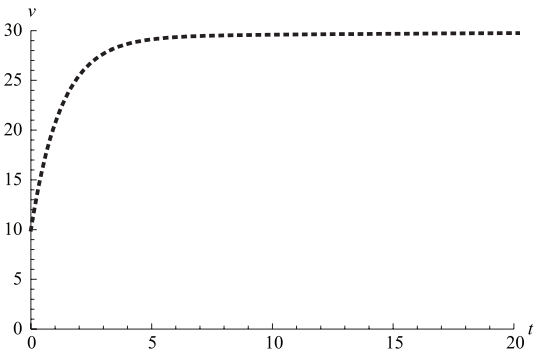

Figure 6: Cruise Control Rapid Mode ($V_r = 30$ m/s, $K_p = 800$, $K_i = 40$).

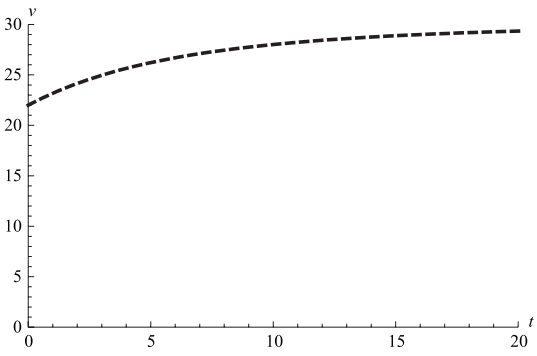

Figure 7: Cruise Control Gradual Mode ($V_r = 30$ m/s, $K_p = 300$, $K_i = 25$).

trajectory has already departed from its set of safe behaviours.

4 Conflict-Tolerant Specifications

We now introduce our notion of a conflict-tolerant specification that addresses some of the issues with classical specifications brought out in the last section.

Definition 6 (Advice Function). *An* ***advice function*** *over a set of variables W is a function $f : \mathcal{W} \to 2^{\mathcal{W}}$ such that for* ***every*** *signal $\sigma \in \mathcal{W}$, $f(\sigma)$ is a prefix-closed set of signals.*

Let (X,U,Y) be a partitioned set of variables. A *conflict-tolerant specification* over (X,U,Y) is simply an advice function over X.

Let $\mathscr{P}$ be a plant over (X,U,Y). Let $\mathscr{C}$ be a controller for $\mathscr{P}$. Let σ be a behaviour of the plant not necessarily generated under the control of the controller $\mathscr{C}$. Then we can define the behaviour of the plant $\mathscr{P}$ controlled by $\mathscr{C}$, *after* the plant behaviour σ, denoted $L^c_\sigma(\mathscr{P}\|\mathscr{C})$, as follows. Suppose the plant $\mathscr{P}$ has reached a configuration $(\mathbf{p}, (\mathbf{x},\mathbf{u}))$ after σ. Let the configuration reached by the controller $\mathscr{C}$ after observing $\sigma \trianglerighteq X$ be $(\mathbf{q}, (\mathbf{x},\mathbf{u}'))$. Then $L^c_\sigma(\mathscr{P}\|\mathscr{C})$ is defined to be the set of behaviours of $\mathscr{P}\|\mathscr{C}$ from the configuration $((\mathbf{p},\mathbf{q}), (\mathbf{x},\mathbf{u}'))$. Thus, the state of the variables controlled by the plant remains the same, but the input configuration is $\mathbf{u}'$ as advised by the controller $\mathscr{C}$.

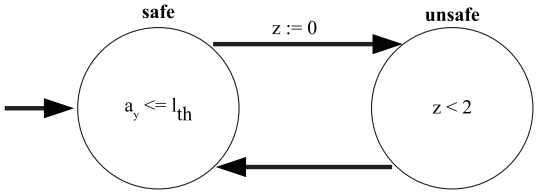

Figure 8: Electronic stability control specification. Here z is a clock variable with $\dot{z} = 1$.

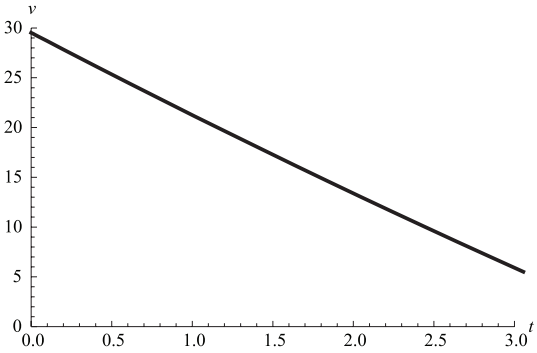

Figure 9: Electronic stability control implementation.

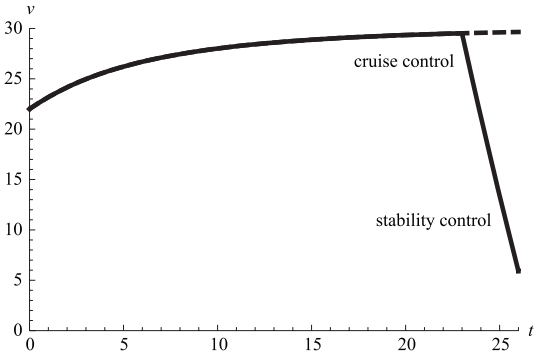

Figure 10: Conflict between cruise control and stability control.

Definition 7 ($\mathscr{C}$ satisfies f). *Let $\mathscr{P}$ be a plant over (X,U,Y) and let f be a conflicttolerant specification over (X,U,Y). A controller $\mathscr{C}$ for $\mathscr{P}$ satisfies f iff for each $\sigma \in L(\mathscr{P})$,*

$$L^c_\sigma(\mathscr{P}\|\mathscr{C}) \trianglerighteq X \subseteq f(\sigma \trianglerighteq X).$$

*Thus after **any** plant behaviour σ over X, if the plant follows the advice of $\mathscr{C}$, then the resulting behavior over X conforms to the safety language prescribed by f after observing σ restricted to X.*

We now describe a mechanism to define an advice function using hybrid automata. The mechanism will use an *acceptor* automaton and an *advisor* automaton as described below.

Definition 8 (Conflict-Tolerant Hybrid Automaton). *Let (X,U,Y) be a partitioned set of variables and let $W = X \cup Y$.* A conflict-tolerant hybrid automaton *over (X,U,Y) is a tuple $\mathscr{S}' = (Acc, Adv, E)$ where*

- *Acc is a hybrid automaton over W called the **acceptor**. Let*

$$Acc = (P, W, W, F_1, init_1, tcp_1, \rightarrow_1).$$

The acceptor automaton must be deterministic with respect to X, i.e. for all $\sigma \in L(Acc) \trianglerighteq X$, there is a unique trajectory τ of Acc such that $\tau \trianglerighteq X = \sigma$.
- *Adv is a deterministic hybrid automaton over W called the **advisor**. Let $Adv = (Q, W, W, F_2, init_2, tcp_2, \rightarrow_2)$.*
- *$E \subseteq \mathbf{P} \times 2^{\mathbf{W}} \times (\mathbf{W} \rightarrow 2^{\mathbf{W}}) \times \mathbf{Q}$ is the **advice relation** between the configurations of the acceptor Acc and the advisor Adv. For an edge $e = (\mathbf{p}, \mathbf{g}, reset, \mathbf{q}) \in E$, $\mathbf{p}$ is a mode of Acc, $\mathbf{g}$ is the subset of states of Acc from which e is enabled, $reset : \mathbf{W} \rightarrow 2^{\mathbf{W}}$ is a function which gives the states of the advisor Adv when the edge e is taken and $\mathbf{q}$ is a mode of the advisor Adv.*

In addition, as defined below, the advice relation must be deterministic and it must not reset the variables in X. Let $m_E : (\mathbf{P} \times \mathbf{W}) \rightarrow 2^{\mathbf{Q} \times \mathbf{W}}$ be the map induced by the advice relation E such that $(\mathbf{q},\mathbf{w}') \in m_E((\mathbf{p},\mathbf{w}))$ iff there exists $e = (\mathbf{p}, \mathbf{g}, reset, \mathbf{q}) \in E$, $\mathbf{w} \in \mathbf{g}$ and $\mathbf{w}' \in reset(\mathbf{w})$. For all reachable configurations $(\mathbf{p},\mathbf{w})$ of Acc, the map m_E must be such that $|m_E((\mathbf{p},\mathbf{w}))| = 1$ and if $m_E((\mathbf{p},\mathbf{w})) = \{(\mathbf{q},w')\}$, then $\mathbf{w}' \triangleright \boldsymbol{X} = \mathbf{w} \triangleright \boldsymbol{X}$. In the sequel we will treat m_E as a function (on the reachable states of Acc) that maps a reachable configuration $(\mathbf{p},\mathbf{w})$ of Acc to a configuration $(\mathbf{q},\mathbf{w}')$ of Adv.

The conflict-tolerant hybrid automaton $\mathscr{S}'$ above defines an advice function over the set of variables X as follows. Let σ be a signal in $L(Acc) \trianglerighteq X$. Then there is a unique configuration $(\mathbf{p},\mathbf{w})$ reached by σ in the acceptor automaton Acc. Let $m_E((\mathbf{p}, \mathbf{w}))$ be $(\mathbf{q},\mathbf{w}')$. Then we define the *constrained* signal language of $\mathscr{S}'$ after σ, denoted $L^c_\sigma(\mathscr{S}')$, to be $L_{(\mathbf{q},\mathbf{w}')}(Adv)$.

The advice function $f_{\mathscr{S}'}$ over X defined by the conflict-tolerant hybrid automaton $\mathscr{S}'$ can now be defined to be

$$f_{\mathscr{S}'}(\sigma) = \begin{cases} L^c_\sigma(\mathscr{S}') \trianglerighteq X & \text{if } \sigma \in L(Acc) \trianglerighteq X \\ \mathscr{X} & \text{otherwise.} \end{cases}$$

Given a plant $\mathscr{P}$, a controller $\mathscr{C}$, and a conflict-tolerant hybrid automaton $\mathscr{S}'$ over (X,U,Y), we say $\mathscr{C}$ satisfies $\mathscr{S}'$ with respect to $\mathscr{P}$ if $\mathscr{C}$ satisfies

the conflict-tolerant specification $f_{\mathscr{S}'}$ with respect to $\mathscr{P}$.

We now illustrate these definitions with a couple of tolerant specification for the cruise control feature discussed in the previous section. Figure 11 shows a tolerant specification for cruise control in which the automaton on the left side is the acceptor and the automaton on the right side (with shaded states) is the advisor. The transitions from the modes of the acceptor to the modes of the advisor denoting the advice relation, are shown with dotted arrows. If cruise control is turned on, then the advisor automaton advises to accelerate either in the gradual mode (when $|V_r - v| <= 10$) or in the rapid mode (when $|V_r - v| > 10$) unless $|V_r - v| < \varepsilon$ in which case the advice is to continue in the steady mode. If the cruise control is turned off, then it can continue to be off or it can be turned on. This tolerant specification is a natural extension of the classical specification given in Fig. 4.

Figure 12 shows another tolerant specification for the cruise control feature. This specification requires that (i) if cruise control has been turned on for a long time (more than 20 seconds), then the car must rapidly accelerate irrespective of the magnitude of the difference between v and V_r, and (ii) if cruise control has been turned on (possibly when $|V_r - v| \leq 10$) but $|V_r - v|$ is currently greater than 10, then the car must rapidly accelerate. The acceptor automaton uses clock w to measure the time since the driver has turned on cruise control.

Note that both these tolerant specifications induce the *same* classical specification shown in Fig. 4, in that the behaviours of the plant that are *always* according to their advice coincide with the safety language of Fig. 4. However, as tolerant specifications they are quite different.

Figure 13 shows a controller that satisfies the tolerant specification $\mathscr{S}'_1$ in Fig. 11 (but *not* the tolerant specification $\mathscr{S}'_2$).

Consider now the example plant under the control of the controller $\mathscr{C}_2$ of Fig. 13 and the controller for stability control. After 20 seconds, the cruise control specification in the gradual mode requires the speed to be within 5% of the target

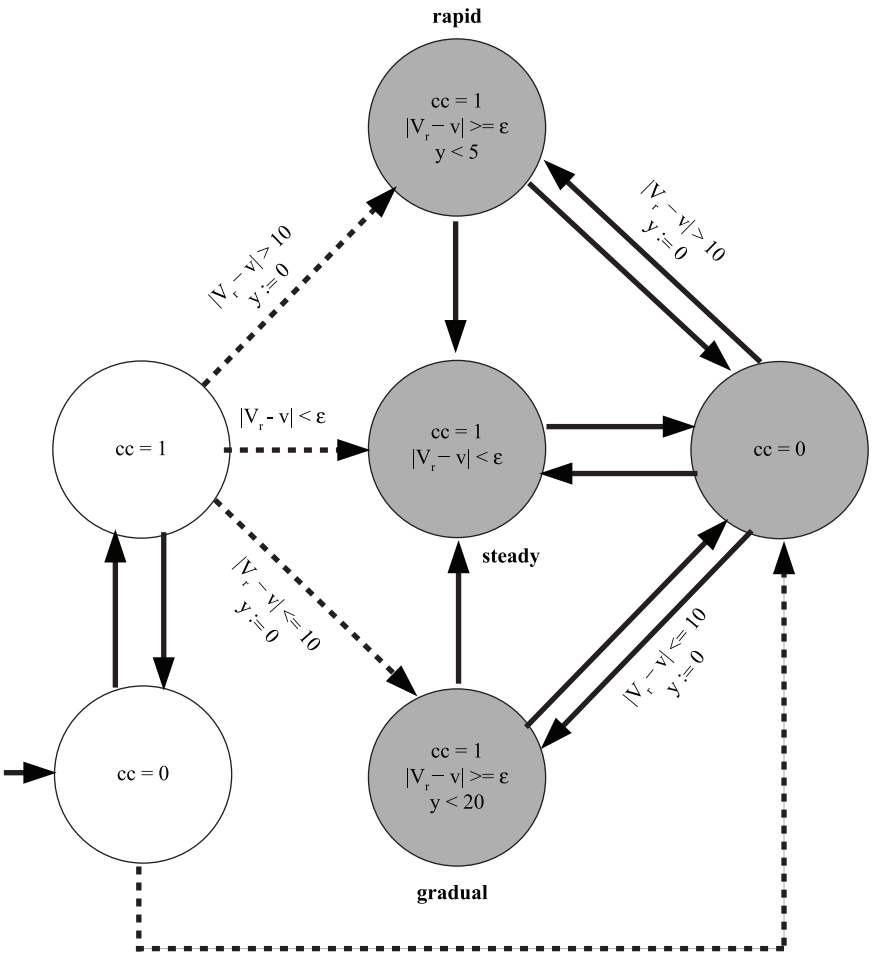

Figure 11: Tolerant cruise control specification $\mathscr{S}'_1$. Here $\dot{y} = 1$.

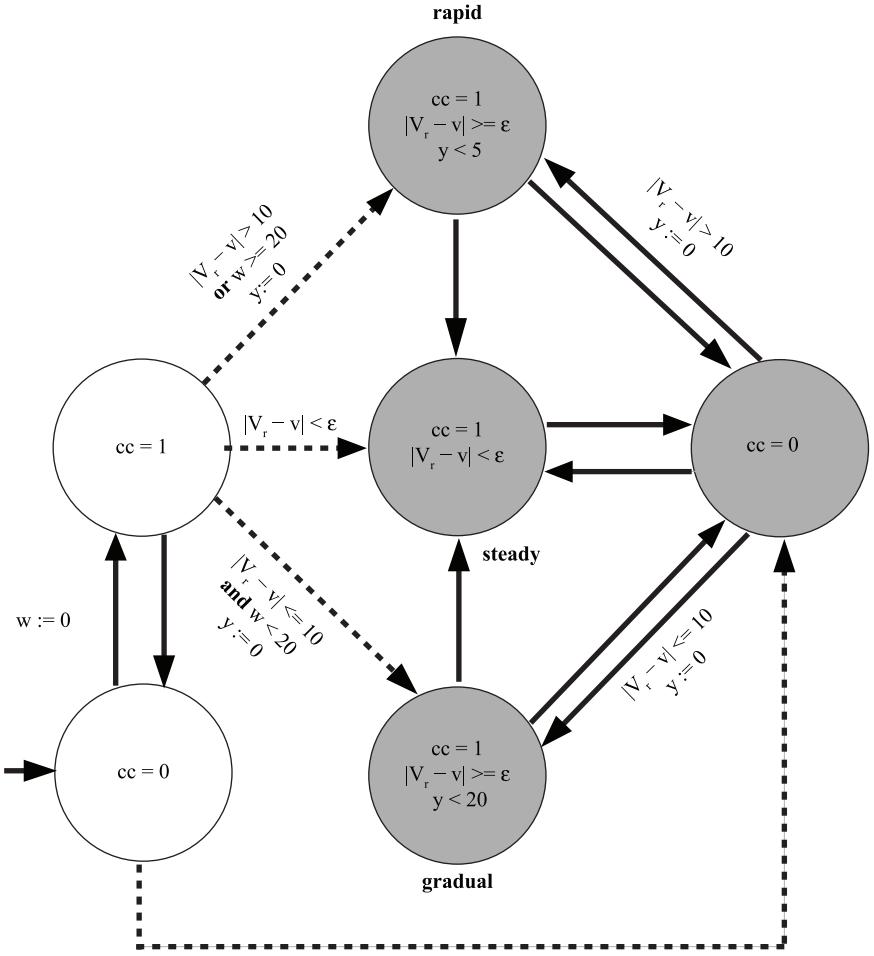

Figure 12: Tolerant cruise control specification $\mathscr{S}_2'$. Here $\dot{y} = \dot{w} = 1$.

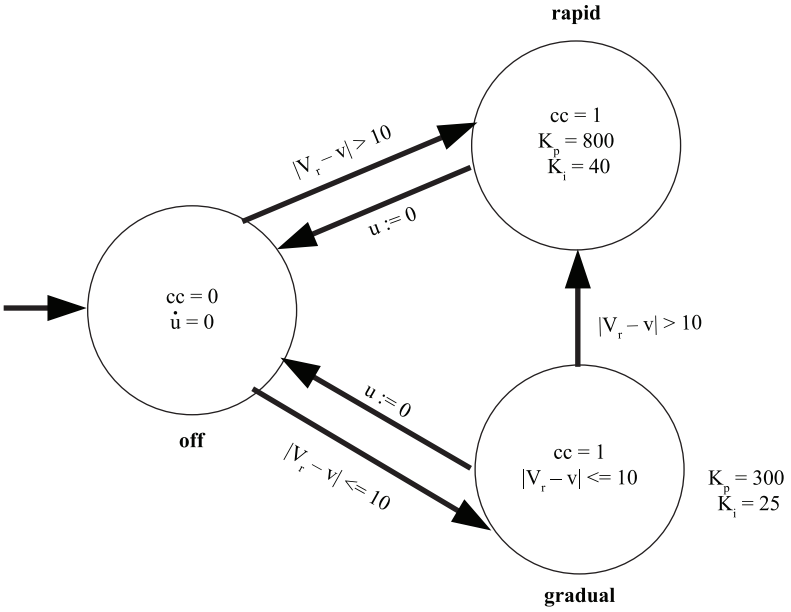

Figure 13: Controller $\mathscr{C}_2$ for tolerant specification $\mathscr{S}_1'$.

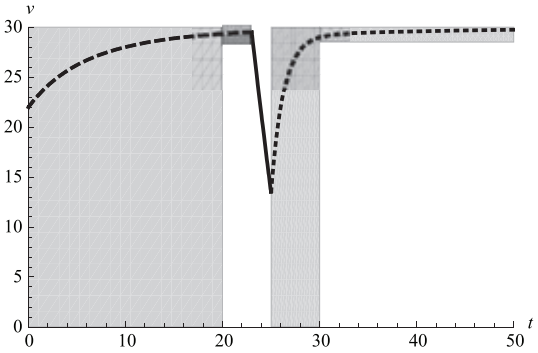

Figure 14: Conflict resolution.

speed. However, stability control intervenes after 23 seconds as the speed has to be reduced while traveling on a curve. Let us say the conflict between the controllers shown in Fig. 10 is resolved in favour of the stability controller, and subsequently cruise control is resumed after the conflict, as shown in Fig. 14. When cruise control is resumed, $|V_r - v| > 10$ and hence the advice changes to accelerate rapidly. Since $\mathscr{C}_2$ is a controller that satisfies the tolerant specification $\mathscr{S}'_2$, we have a guarantee that the controlled system conforms to the tolerant specification during the periods when $\mathscr{C}_2$ is in control (shown shaded in the figure).

5 Controller Verification

The verification problem for conflict-tolerant specifications, is the following: given a plant $\mathscr{P}$, a conflict-tolerant hybrid automaton specification $\mathscr{S}'$, and a controller $\mathscr{C}$, does $\mathscr{C}$ satisfy $\mathscr{S}'$ with respect to $\mathscr{P}$ (see Section 4). We now show how this can be solved algorithmically when the plant $\mathscr{P}$, controller $\mathscr{C}$, and the tolerant specification $\mathscr{S}'$ are given as initialized rectangular automata. We say that a conflict-tolerant hybrid automaton $\mathscr{S}' = (Acc, Adv, E)$ is a *conflict-tolerant initialized rectangular automaton* when the acceptor *Acc* and the advisor *Adv* are initialized rectangular automata, and the advice relation E is rectangular in that if $(\mathbf{p},\mathbf{g},reset,\mathbf{q}) \in E$, then g and $reset(\mathbf{w})$ are rectangular sets for all $\mathbf{w}$.

We recall the basic definitions of rectangular automata from the literature.[13] A *rectangular* set $R \subseteq \mathbb{R}^n$ is of the form $R_1 \times \cdots \times R_n$ where each R_i is a bounded or unbounded interval, i.e. R is a product of n intervals of the real line. In a rectangular hybrid automaton, the sets $init_q$, tcp_q, F_q are rectangular. Also, the set of states from which a jump is enabled (guard set) is rectangular and during the jump, if a variable is reset, it is set to a value within a fixed constant interval. In addition, in every mode, the derivative of each variable always lies between two fixed bounds, for example $\dot{x} \in [1,2]$. These bounds may vary from one mode to another. A rectangular automaton is said to be *initialized* if for every variable v, after a jump from mode $\mathbf{q}$ to mode $\mathbf{q}'$, either the flow condition of v remains the same or v has been reset.

Rectangular hybrid automata are an interesting subclass of hybrid automata because they can be used for conservatively approximating sets of arbitrary hybrid trajectories. Thus, rectangular automata could be used to create abstractions of complex hybrid systems which can then be verified.[3] Let s,t be two states of a hybrid automaton $\mathscr{H}$. In *the reachability problem* for hybrid automata, we are interested in checking whether there is a trajectory of $\mathscr{H}$ that starts at s and ends at t. It has been shown that the reachability problem for initialized rectangular automata is decidable.[13]

This result is obtained by first translating an initialized rectangular automaton $\mathscr{R}$ to an initialized multirate automaton $\mathscr{M}_{\mathscr{R}}$ in which each variable evolves according to a constant, rational slope, which may be different in different control states. Then the multirate automaton $\mathscr{M}_{\mathscr{R}}$ is translated into a timed automaton $\mathscr{T}_{\mathscr{R}}$ for which the reachability problem is known to be decidable.[2]

We now sketch the key ideas behind this two-step translation. Let $\mathscr{R}$ be an initialized rectangular automaton of n variables. Consider a variable x_i with $\dot{x}_i = [l,u]$ in $\mathscr{R}$. In the corresponding initialized multirate automaton $\mathscr{M}_{\mathscr{R}}$, each variable x_i is replaced by two variables y_{2i-1} and y_{2i} such that $\dot{y}_{2i-1} = l$ and $\dot{y}_{2i} = u$. Let $h_{\mathscr{M}} : \mathbf{Q} \times \mathbb{R}^{2n} \to \mathbf{Q} \times \mathbb{R}^n$ be a function which maps the state space of $\mathscr{M}_{\mathscr{R}}$ to that of $\mathscr{R}$ defined by

$$h_{\mathscr{M}}((\mathbf{q},\mathbf{y})) = \{(\mathbf{q},\mathbf{x}) \mid \mathbf{y}_{2i-1} \le \mathbf{x}_i \le \mathbf{y}_{2i}\}.$$

The variable y_{2i-1} tracks the least possible value of x_i and the variable y_{2i} tracks the greatest possible value of x_i. The jump relation of $\mathscr{M}_{\mathscr{R}}$ is constructed such that this continues to hold even after a jump. The initialized multirate automaton $\mathscr{M}_{\mathscr{R}}$ is further translated into a timed automaton $\mathscr{T}_{\mathscr{R}}$ by rescaling the state space. Let $h_{\mathscr{I}} : \mathbf{Q} \times \mathbb{R}^{2n} \to \mathbf{Q} \times \mathbb{R}^{2n}$ be a function which maps the state space of $\mathscr{T}_{\mathscr{R}}$ to that of $\mathscr{M}_{\mathscr{R}}$ defined by

$$h_{\mathscr{I}}((\mathbf{q},(\mathbf{y}_1,\ldots,\mathbf{y}_{2n}))) = (\mathbf{q},(l_1 \cdot \mathbf{y}_1,\ldots,l_{2n} \cdot \mathbf{y}_{2n}))$$

where $l_i = \dot{y}_i$ if $\dot{y}_i \neq 0$ and $l_i = 1$ otherwise.

We say that a set of configurations A of the automaton $\mathscr{R}$ is "representable" as a set of regions (see[2]) if there exists a set of regions B of the corresponding timed automaton $\mathscr{T}_{\mathscr{R}}$ such that $\bigcup h_{\mathscr{M}}(h_{\mathscr{I}}(B)) = A$. Let $Reach_{\mathscr{H}}(I)$ denote the set

of states that can be reached from the set of states I using a trajectory of the automaton $\mathscr{H}$.

Lemma 1. *Let A be a set of configurations of an initialized rectangular automaton $\mathscr{R}$ which is representable as a set of regions B. Then $Reach_{\mathscr{R}}(A)$ is representable as the set of regions Reach $\mathscr{T}_{\mathscr{R}}(B)$.*

Let $\mathscr{S}' = (Acc, Adv, E)$ be a conflict-tolerant initialized rectangular automaton. Recall that the advice relation E induces a map m_E which maps a configuration of the acceptor Acc to a configuration of the advisor Adv. Corresponding to m_E, we can obtain an equivalent map m'_E which maps a configuration $(\mathbf{p},\mathbf{y})$ of the timed automaton corresponding to the acceptor, to a configuration $(\mathbf{q},\mathbf{y}')$ of the timed automaton corresponding to the advisor, such that $(\mathbf{q},\mathbf{y}') = m'_E((\mathbf{p},\mathbf{y}))$ iff $h_{\mathscr{M}}(h_{\mathscr{I}}(\mathbf{q},\mathbf{y}')) = m_E(h_{\mathscr{M}}(h_{\mathscr{I}}(\mathbf{p},\mathbf{y})))$ (we lift m_E to work on sets of configurations in the natural way).

Lemma 2. *Let $\mathscr{S}' = (Acc, Adv, E)$ be a conflict-tolerant initialized rectangular automaton. Let A be a set of configurations of an initialized rectangular automaton $\mathscr{R}$ representable as a set of regions B of $\mathscr{T}_{\mathscr{R}}$. Then the set of configurations $m_E(A)$ is representable as the set of regions $m'_E(B)$.*

Theorem 1. *Given a plant $\mathscr{P}$, a controller $\mathscr{C}$ for $\mathscr{P}$ and a conflict-tolerant initialized rectangular automaton $\mathscr{S}' = (Acc, Adv, E)$ over (X, U, Y) such that $\mathscr{P}$ and $\mathscr{C}$ are initialized rectangular automata, it is decidable to check whether $\mathscr{C}$ satisfies $\mathscr{S}'$ with respect to $\mathscr{P}$.*

Proof. Given a deterministic rectangular hybrid automaton $\mathscr{H}$, we note that it will be stuck in a mode $\mathbf{q}$ if both continuous evolution and discrete jump are not possible from $\mathbf{q}$. We can complete $\mathscr{H}$ by adding a trap mode $\mathbf{t}$ and then adding jump transitions $\mathbf{q} \to \mathbf{t}$ from every other mode $\mathbf{q}$ such that if the automaton gets stuck in a mode $\mathbf{q}$, it can transition to $\mathbf{t}$. We complete the advisor automaton Adv to obtain Adv'.

Let $\mathscr{H}_1 = \mathscr{P} \| \mathscr{C}' \| Acc$ and $\mathscr{H}_2 = \mathscr{P} \| \mathscr{C}' \| Adv'$ where $\mathscr{C}'$ is obtained from $\mathscr{C}$ by renaming every variable $u \in U$ to its primed version u'. In order to check if the controller $\mathscr{C}$ does not satisfy the tolerant specification $\mathscr{S}'$ with respect to $\mathscr{P}$, it is necessary and sufficient to check if there exists a "trajectory" of the form:

$$i \rightsquigarrow_{\mathscr{H}_1} ((\mathbf{p},\mathbf{q},\mathbf{a}_1),\mathbf{v}) \to ((\mathbf{p},\mathbf{q},\mathbf{a}_2),\mathbf{w}) \rightsquigarrow_{\mathscr{H}_2} ((\mathbf{p}',\mathbf{q}',\mathbf{t}),\mathbf{w}'),$$

i.e. there exists a configuration $((\mathbf{p},\mathbf{q},\mathbf{a}_1), \mathbf{v})$ of $\mathscr{H}_1$ reachable from an initial configuration of $\mathscr{H}_1$ and in $\mathscr{H}_2$, we can reach a trap configuration $((\mathbf{p}',\mathbf{q}',\mathbf{t}), \mathbf{w}')$ from $((\mathbf{p},\mathbf{q},\mathbf{a}_2), \mathbf{w})$ such that $\mathbf{v} \triangleright X = \mathbf{w} \triangleright X$, $\mathbf{v} \triangleright U' = \mathbf{w} \triangleright U$ and $m_E((\mathbf{a}_1, \mathbf{v} \triangleright Y)) = (\mathbf{a}_2, \mathbf{w} \triangleright Y)$. Thus, even after the plant follows the advice of the controller $\mathscr{C}$ from the configuration $((\mathbf{p},\mathbf{q},\mathbf{a}_2), \mathbf{w})$, the resulting plant behaviour violates the specification as the advisor automaton reaches a trap mode (recall that m_E maps a configuration of the acceptor to sets of configurations of the advisor).

In order to check whether a trap mode is reachable even after the plant follows the controller's advice, we first translate the initialized rectangular automata $\mathscr{H}_1$ and $\mathscr{H}_2$ to the corresponding timed automata $\mathscr{I}_{\mathscr{H}_1}$ and $\mathscr{I}_{\mathscr{H}_2}$. Given

$$I \rightsquigarrow_{\mathscr{H}_1} \text{Reach}_{\mathscr{H}_1}(I) \to_{m_E} J \rightsquigarrow_{\mathscr{H}_2} \text{Reach}_{\mathscr{H}_2}(J),$$

where J is the set of configurations of the advisor automaton reachable from the set of initial configurations I of the acceptor automaton, we are interested in checking whether $\text{Reach}_{\mathscr{H}_2}(J)$ contains a configuration of the form $((\mathbf{p}',\mathbf{q}',\mathbf{t}),\mathbf{w}')$ for some $\mathbf{p}',\mathbf{q}',\mathbf{w}'$. The reachability check can be carried out in $\mathscr{I}_{\mathscr{H}_1}$ and $\mathscr{I}_{\mathscr{H}_2}$ as to whether $I_{\mathscr{I}} \rightsquigarrow_{\mathscr{I}_{\mathscr{H}_1}} Reach_{\mathscr{I}_{\mathscr{H}_1}}(I_{\mathscr{I}}) \to_{m'_E} I_{\mathscr{I}} \rightsquigarrow_{\mathscr{I}_{\mathscr{H}_2}} Reach_{\mathscr{I}_{\mathscr{H}_2}}$. By Lemmas 1 and 2, it is sufficient to check if $Reach_{\mathscr{I}_{\mathscr{H}_2}}$ contains a configuration of the form $((\_, \_, \mathbf{t}), \_)$. This check can be carried out in time linear in the size of the region automata for $\mathscr{I}_{\mathscr{H}_1}$ and $\mathscr{I}_{\mathscr{H}_2}$. □

6 Conclusion

In this paper we have studied the problem of specifying and verifying controllers in a hybrid setting. We have introduced the notion of conflict-tolerant specifications in this setting and provided a novel hybrid automata based mechanism for respresenting such specifications. We have also given a decision procedure for the problem of verifying whether a given controller satisfies a given conflict-tolerant specification when the plant, controller and the specification are modeled as initialized rectangular automata.

If valid conflict-tolerant controllers can be constructed to satisfy tolerant specifi-cations, then supervisory controller design is considerably simplified, thus easing the system builder's task of integrating independently designed controllers procured from different vendors. We note that each controller need only be specified, developed and verified *once* regardless of which other controllers it is integrated with.

In earlier work in a discrete and timed setting, we had proposed a specification mechanism using automata with "advised" and "not-advised" transitions. This mechanism (as also the definition of an advice function) required specifications to be "consistent," in that whenever a behaviour

τ was advised after a behaviour σ (i.e. $\tau \in f(\sigma)$), the advice for $\sigma \cdot \tau$ should be precisely the extensions of τ in the advice after σ (namely $f(\sigma)$). This restriction precludes specifications like the one in Figure 2(b) which are natural requirements to specify when building controllers that can be suspended and then resumed later. Such specifications can now be specified using the hybrid automaton based mechanism in this paper.

Given controllers that are verified to conform to their respective tolerant specifications, and a priority ordering on these controllers, one could ask for a supervisory controller that "maximizes" the advice of the controllers in that it ignores the advice of a controller only if the controller's advice conflicts with that of a higher priority controller. In a restricted setting of switched control (where a controller can only switch the discrete "mode" of the plant) we have shown how to build such a supervisory controller. The reader is referred to[11] for further details.

Received 20 August 2013.

References

1. Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theor. Comput. Sci.*, 138(1):3–34, 1995.
2. Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
3. Rajeev Alur, Thomas A. Henzinger, Gerardo Lafferriere and George J. Pappas. Discrete abstractions of hybrid systems. *Proc. of the IEEE*, 88(7):971–984, Jul 2000.
4. E. Asarin, O. Bournez, T. Dang, O. Maler and A. Pnueli. Effective synthesis of switching controllers for linear systems. *Proc. of the IEEE*, 88(7):1011–1025, Jul 2000.
5. Y.L. Chen, S. Lafortune and F. Lin. Modular supervisory control with priorities for discrete event systems. In *Conf. on Decision and Control*, pages 409–415. IEEE, 1995.
6. Deepak D'Souza and Madhu Gopinathan. Conflict-tolerant features. In *Computer Aided Verification*, pages 227–239, 2008.
7. Deepak D'Souza, Madhu Gopinathan, S. Ramesh and Prahladavaradan Sampath. Conflict-tolerant real-time features. In *Quantitative Evaluation of Systems*, pages 274–283, 2008.
8. Deepak D'Souza, Madhu Gopinathan, S. Ramesh and Prahladavaradan Sampath. Supervisory control for real-time systems based on conflict-tolerant controllers. In *Conference on Automation Science and Engineering*, 2009.
9. Amy P. Felty and Kedar S. Namjoshi. Feature specification and automated conflict detection. *ACM Trans. Softw. Eng. Methodol.*, 12(1):3–27, 2003.
10. Kathi Fisler and Shriram Krishnamurthi. Decomposing verification by features. *IFIP Working Conference on Verified Software: Theories, Tools, Experiments*, 2006.
11. Madhu Gopinathan. *Conflict Tolerant Features*. PhD thesis, Department of Computer Science and Automation, Indian Institute of Science, Bangalore, Oct 2009.
12. Jonathan D. Hay and Joanne M. Atlee. Composing features and resolving interactions. In *SIGSOFT Found. of Softw. Engg.*, pages 110–119, 2000.
13. Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *J. Comput. Syst. Sci.*, 57(1):94–124, 1998.
14. Dirk O. Keck and Paul J. Kühn. The feature and service interaction problem in telecommunications systems. a survey. *IEEE Trans. Software Eng.*, 24(10):779–796, 1998.
15. MSNBC. The top 10 safety features for the future –http://www.msnbc.msn. com/id/23300261, 2008.
16. Rajesh Rajamani. *Vehicle Dynamics and Control*. Springer, 2006.
17. Peter J.G. Ramadge and W. Murray Wonham. The control of discrete event systems. In *Proc. of the IEEE*, volume 77, pages 81–98, 1989.
18. C.J. Tomlin, J. Lygeros and S. Shankar Sastry. A game theoretic approach to controller design for hybrid systems. *Proc. of the IEEE*, 88(7):949–970, Jul 2000.
19. K.C. Wong, J.G. Thistle, H.H. Hoang, and R.P. Malhamé. Supervisory control of distributed systems: Conflict resolution. In *Conf. on Decision and Control*, pages 416–421. IEEE, 1995.

Deepak D'Souza received his Ph.D. from Chennai Mathematical Institute in 2000. Since 2003 he has been at the Department of Computer Science and Automation of the Indian Institute of Science, Bangalore, where he is currently an Associate Professor. Among his areas of interest are specification and analysis of real-time and hybrid systems, program analysis, and program verification.

Madhu Gopinathan completed his Ph.D. at the Indian Institute of Science, Bangalore in the area of formal verification after working in the industry for several years. He is currently Chief Technology Officer at vMobo Inc. working on data mining and machine learning. His current interests include creating systems that can learn and adapt to their environment.

S. Ramesh earned his B.E. degree in Electronics and Communication Engineering from Indian Institute of Science Bangalore and his Ph.D. degree in Computer Science & Engineering from Indian Institute of Technology Bombay. He has been with General Motors Global R&D for the last seven years. Prior to that, he was on the faculty of the department of Computer Science & Engineering at IIT Bombay, for more than fifteen years. His areas of interests are Rigorous Software Engineering, Embedded Systems and Real-Time Systems. This work was done while the author was at GM R&D Bangalore.

Prahladavaradan Sampath received his BSc degree in computer science from Brunel University, and both MSc and Ph.D. degrees in computer science from Imperial College, University of London. He is currently a manager at MathWorks India, where he leads a team developing formal methods based tools for verification and validation of model-based designs. His interests include program analysis, formal specification techniques, and developing tools for rigorous development of correct software. This work was done while the author was at GM R&D Bangalore.