Conflict-Tolerant Specifications for Hybrid Systems

Deepak D'Souza Madhu Gopinathan S. Ramesh
Prahladavaradan Sampath

IISc-CSA-TR-2010-6
http://archive.csa.iisc.ernet.in/TR/2010/6/

Computer Science and Automation
Indian Institute of Science, India

October 2010

Conflict-Tolerant Specifications for Hybrid Systems

Deepak D'Souza* Madhu Gopinathan† S. Ramesh‡
Prahladavaradan Sampath§

Abstract

We propose a framework for developing 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 first motivate the need for conflict-tolerant specifications by illustrating conflict between two controllers in a real world system. We then propose a way of defining conflict-tolerant specifications for general hybrid systems. 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. With tolerant specifications and verified controllers, it is easy to construct a simple priority based supervisory controller for resolving conflict among controllers. We also show that in the restricted setting of switched control, it is possible to construct a supervisory controller which guarantees the "maximal" use of every controller by ensuring that the advice of each controller is always followed except at time instants when each of the controller's advice conflicts with that of a higher priority controller.

1 Introduction

The problem of engineering large embedded systems is growing exponentially with the increasing sophistication of software. This has inspired a

*deepakd@csa.iisc.ernet.in
†madhug@csa.iisc.ernet.in
‡ramesh.s@gm.com
§p.sampath@gm.com

number of approaches for organizing software to improve the reliability of software systems. Many of these approaches propose a *feature oriented* development paradigm in which feature specifications are derived from domain requirements and features are implemented to satisfy such specifications. A software product line [23] is a set of features from which multiple software products can be engineered by suitably integrating a subset of the available features. In the automotive industry, advanced features such as adaptive cruise control, collision avoidance, electronic stability control etc. [12] are developed as part of a software product line, and a subset of these features is integrated into automotive products based on market needs and regulatory requirements.

We view embedded systems developed using this paradigm as consisting of a plant along with multiple *features* or controllers, where each controller advises the plant on how to conform to its feature specification. In such a setting, controllers intermittently control the plant and they do not always agree on how the plant should behave. When multiple controllers control the plant, the plant may reach a point of "conflict" at which the controllers offer conflicting advice. This situation is an instance of what is referred to in the literature as the *feature interaction* problem [16].

Automotive manufacturers like General Motors integrate various controllers (some built by third party vendors) into a final product. Detecting and resolving conflicts between controllers poses a significant challenge during the integration phase. Conflicts between two controllers can, of course, be resolved by respecifying or redesigning one of the controllers. However there is no guarantee that the redesigned controller will not conflict with some other existing controller. Moreover, redesign for handling specific conflicts reduces the scope for reusing the controller in multiple contexts.

An alternative to redesign is to prioritize controllers during the integration phase and then resolve conflicts at run time by *suspending* the controller with lower priority so that the plant can continue with the advice of the higher priority controller. However the issue now is how and when to *resume* the suspended controller so as to *maximize* its use.

Our Approach. In this paper we provide a solution to 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 [7, 8]. The main idea is to design controllers that are "tolerant" or "resilient" in the sense that they continue to advise the plant even when their advice has been disregarded in the past. This lets us build simple automated priority-based supervisors, which ignore the advice of a controller if it conflicts with that of a higher priority controller. A controller for the plant is again a hybrid automaton that reads the variables of the plant and

provides inputs to the plant. Behaviours of the plant are modeled as piecewise continuous functions or "signals."

The starting point of this framework is the notion of a *conflict-tolerant specification*. A classical safety specification can be thought of as a prefix-closed set of signals containing all the plant behaviours which are considered *safe*. If a controller is overridden due to a conflict, then the classical specification for that controller is typically violated and the specification does not say anything about the desired set of behaviours when that controller is resumed. A conflict-tolerant specification on the other hand can be viewed as an *advice function* f that specifies for every plant behaviour σ, a prefix-closed set of all future behaviours that are considered safe. 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 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 4cm. A classical specification for this feature is shown in Fig. 1(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 4cm or fall below 2cm. The classical specification does not specify the desired behaviour if the water level falls below 2cm or rises above 4cm. In contrast, the tolerant specification in Fig. 1(b) specifies that ideally the water level should be between 2 and 4cm 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. 1(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. 1(c) specifies that ideally the water level should be between 2 and 4cm, and if this is violated, then the water level must be kept below 2cm if it is currently below 2cm, and kept above 4cm if it is currently above 4cm.

We note that both the tolerant specifications of Fig. 1(b) and (c) induce the same classical specification shown in Fig. 1(a), in that the behaviours of the plant that are *always* according to their advice coincide with the safety

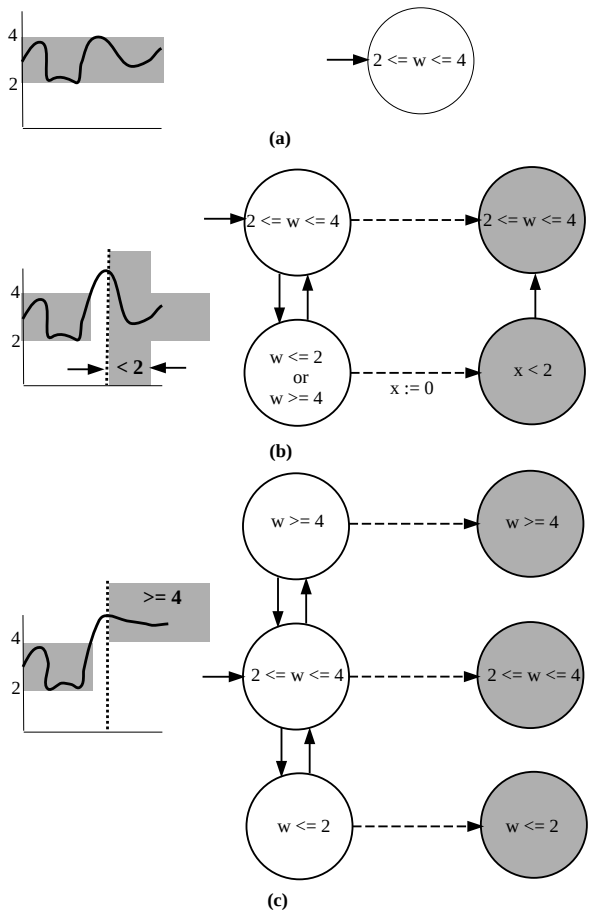

Figure 1: A classical specification (a), and conflict-tolerant specifications (b) and (c).

language of Fig. 1(a). However, as tolerant specifications they are quite different.

When multiple controllers are controlling a plant, it is likely that a controller is suspended by the supervisory controller during periods of conflict and then resumed later. Figure 2 illustrates the partitioning of a plant behaviour into intervals where a controller's advice is followed (shaded) and not followed (not shaded). For example, let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two controllers which control a water tank. Let the specification of $\mathcal{C}_2$ require that always the water level should rise at a rate between 1 and 2. Suppose that the controller $\mathcal{C}_2$ is suspended due to a conflict with $\mathcal{C}_1$ (at time instant marked S in Fig. 2) and then resumed later (at time instant marked R). For verifying whether a controller satisfies a conflict-tolerant specification, we need to check whether

the plant exhibits a behaviour that is not allowed by the specification after the controller is resumed. A controller satisfies a conflict-tolerant specification if in each interval where the controller's advice is followed, the plant behaviour is contained within the set of behaviours allowed by the specification. For the plant behaviour $\sigma \cdot \tau$ in Fig. 2, the controller $\mathcal{C}_2$ satisfies its tolerant specification since water level rises at a rate between 1 and 2 after resuming $\mathcal{C}_2$. In this paper we show how we can algorithmically solve the verification problem, when the plant, the controller, and the acceptor and advisor components of the tolerant specification are given as initialized rectangular hybrid automata.

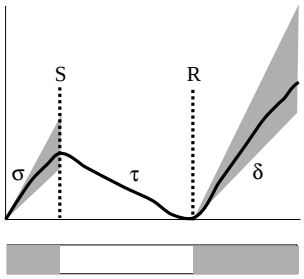

Figure 2: Partitioning of a plant behaviour into intervals in which the advice of a controller is followed (shaded) and not followed (not shaded) when the controller is suspended.

If the individual controllers are verified to conform to their respective tolerant specifications, then we can build a simple supervisory controller based on a given priority ordering of the respective controllers. We note that each controller need only be specified, developed and verified *once* regardless of which other controllers with which it is integrated.

In earlier work [7, 8, 9] we have proposed a conflict-tolerant framework in the discrete and timed settings. Apart from the fact that we deal with much more general behaviours in the form of signals, the work in this paper departs from our earlier work in several ways. The contributions of this paper are

- In earlier work, tolerant specifications did not have acceptor and advisor components. The notion of acceptor and advisor components have enabled us to create a more general specification mechanism, not limited to only "consistent" specifications as done in [7, 8].

- 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 itself is also a valid controller for the plant.

- We give a decision procedure for verifying whether a controller satisfies a tolerant specification with respect to a plant when the plant, controller and the tolerant specification are modeled using initialized rectangular automata.

Related Work. 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 [16]. The problem of conflict detection at the specification stage is addressed in [11], where conflict between two feature specifications in temporal logic is detected automatically.

We now focus on previous work in the untimed setting which address conflict resolution. Our approach of viewing features as discrete event controllers [22] follows that of [25, 6]. 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 [6], 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 [14] 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 [24], controllers that satisfy safety specifications are synthesized 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 this case, the supervisor can resume a suspended controller only when the plant state is in the safe set of that controller. Using this method, supervisor design becomes more complex as the plant has to be driven to a safe state before resuming the controller. In contrast, our solution simplifies supervisor design.

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

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 $k \in 0, \cdots, n-1$, 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 [5]. We denote by $\mathcal{W}$ the set of all signals over the set of variables W. A *signal language* over W is simply a subset of $\mathcal{W}$. 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 $\mathcal{W}$. 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

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

For signals $\sigma_1, \sigma_2 \in \mathcal{W}$, we say that σ_1 is a *prefix* of σ_2, denoted $\sigma_1 \preceq \sigma_2$, if there exists a signal $\sigma \in \mathcal{W}$ 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 $L \subseteq \mathcal{W}$ and a signal σ, we denote by $ext_\sigma(L)$, the *set of extensions* of σ that are in L, i.e.

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

Our definition of a hybrid automaton is adapted from the definitions of hybrid automata in the literature [24, 17, 3, 5, 1, 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.

A *hybrid automaton* is a tuple

$$\mathcal{H} = \langle Q, V, C, F, init, tcp, \rightarrow \rangle$$

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 $\mathcal{H}$.

- V is a finite set of state variables. The set $\mathbf{V}$ represents the states of $\mathcal{H}$. A *configuration* of $\mathcal{H}$ 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 $\mathcal{H}$, i.e. $\mathcal{H}$ constrains the initial value, continuous flow and resets of variables in C. Let $\overline{C}$ denote the set of uncontrolled variables of $\mathcal{H}$, i.e. $\overline{C} = V \setminus C$.

- 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 $\vec{\mathbf{D}}$ 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.

- $\mathit{init} \subseteq \mathbf{Q} \times \mathbf{C}$ specifies a set of initial configurations given by $\{(\mathbf{q}, \mathbf{v}) \in \mathbf{Q} \times \mathbf{V} \mid (\mathbf{q}, \mathbf{v} \triangleright C) \in \mathit{init}\}$. Thus, *init* does not constrain the initial values of the uncontrolled variables.

- $\mathit{tcp} : \mathbf{Q} \to 2^{\mathbf{V}}$ assigns to each mode a **time can progress** condition. If $\mathbf{v} \in \mathit{tcp}(\mathbf{q})$, then it is possible for the automaton $\mathcal{H}$ 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 $e = (\mathbf{q}, \mathbf{g}, \mathit{reset}, \mathbf{q}') \in \to$, $\mathbf{q}$ 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 $\mathit{reset} : \mathbf{V} \to 2^{\mathbf{C}}$ 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 $\mathit{tcp}_{\mathbf{q}}$ we mean the time can progress condition $\mathit{tcp}(\mathbf{q})$. 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 $\mathit{tcp}_{\mathbf{q}}$ 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 $\mathcal{H}$. Let $\tau : [a, b) \to \mathbf{V}$ be a function which is continuous in $[a, b)$ and differentiable in (a, b). We say that $\mathcal{H}$ *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 $c \in C, \dot{\tau}_c(t) \in F_{\mathbf{q}}(\tau(t))(c)$, i.e. the signal for the controlled variables obeys the flow condition of mode $\mathbf{q}$.

We say that $\mathcal{H}$ *jumps* from $(\mathbf{q}, \mathbf{v})$ to $(\mathbf{q}, \mathbf{v}')$, written $(\mathbf{q}, \mathbf{v}) \to (\mathbf{q}', \mathbf{v}')$ iff there exists $(\mathbf{q}, \mathbf{g}, \mathit{reset}, \mathbf{q}') \in \to$ such that $\mathbf{v} \in \mathbf{g}$, $\mathbf{v}' \triangleright C \in \mathit{reset}(\mathbf{v})$, and $\mathbf{v}' \triangleright \overline{C} = \mathbf{v} \triangleright \overline{C}$.

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

- Either $\nu(0) = \mathbf{q}$ and $\sigma(0) = \mathbf{v}$, or $\nu(0) = \mathbf{q}'$ and $\sigma(0) = \mathbf{v}'$ and $(\mathbf{q}, \mathbf{v}) \to (\mathbf{q}', \mathbf{v}')$. 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 $n - 1$, ν 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 ν in the interval I_k. Then $\mathcal{H}$ 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 $(\mathbf{q_{k-1}}, \sigma(t_k^-)) \to (\mathbf{q_k}, \sigma(t_k))$, i.e. $\mathcal{H}$ jumps,
  - or $\mathbf{q_{k-1}} = \mathbf{q_k}$ and $(\sigma \trianglerighteq C)(t_k^-) = (\sigma \trianglerighteq C)(t_k)$, i.e. $\mathcal{H}$ does not jump and the controlled variables do not change.

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

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

The *signal language* of $\mathcal{H}$ starting from a given configuration $(\mathbf{q}, \mathbf{v})$, denoted $L_{(\mathbf{q},\mathbf{v})}(\mathcal{H})$ is defined as

$$L_{(\mathbf{q},\mathbf{v})}(\mathcal{H}) = \{\sigma \mid \exists \nu : \ (\nu, \sigma) \in \mathit{Traj}_{(\mathbf{q},\mathbf{v})}(\mathcal{H})\}.$$

The *signal language* of $\mathcal{H}$, denoted $L(\mathcal{H})$, is defined as

$$L(\mathcal{H}) = \{\sigma \mid \exists \nu : \ (\nu, \sigma) \in \mathit{Traj}(\mathcal{H})\}.$$

We say a hybrid automaton $\mathcal{H}$ is *deterministic* if for any signal $\sigma \in L(\mathcal{H})$, there is exactly one trajectory of the form $(\nu, \sigma) \in \mathit{Traj}(\mathcal{H})$.

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

$$\mathit{config}_{\mathcal{H}}(\tau) = \{(\mathbf{q}, \mathbf{v}) \mid \ \exists (\nu, \sigma) \in \mathit{Traj}(\mathcal{H}) \text{ s.t. } (\nu(r^-) = \mathbf{q} \text{ and } \sigma(r^-) = \mathbf{v}\}.$$

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

$$L_\sigma(\mathcal{H}) = \bigcup_{(\mathbf{q},\mathbf{v}) \in \mathit{config}_{\mathcal{H}}(\sigma)} L_{(\mathbf{q},\mathbf{v})}(\mathcal{H}).$$

Finally, before we close this section, we define the synchronous product of two hybrid automata. Let $\mathcal{H}_1 = (Q_1, V, C_1, F_1, \mathit{init}_1, \mathit{tcp}_1, \to_1)$ and $\mathcal{H}_2 = (Q_2, V, C_2, F_2, \mathit{init}_2, \mathit{tcp}_2, \to_2)$ be two hybrid automata over the same set of variables V. The automata $\mathcal{H}_1$ and $\mathcal{H}_2$ 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 $\mathcal{H}_1$ and $\mathcal{H}_2$, denoted $\mathcal{H}_1 \| \mathcal{H}_2$ is defined to be the hybrid automaton $\mathcal{H} = (Q, V, C, F, \mathit{init}, \mathit{tcp}, \to)$ where

- $Q = Q_1 \times Q_2$,
- $C = C_1 \cup C_2$,
- Let $\mathbf{D} = C \to 2^{\mathbb{R}}$ be a set of maps specifying a set of derivatives for variables in C. Then $F : \mathbf{Q} \to (\mathbf{V} \to \mathbf{D})$ 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 $\mathbf{d} \triangleright C_2 \in F_2(\mathbf{q}_2)(\mathbf{v})$,
- $\mathit{init} \subseteq \mathbf{Q} \times \mathbf{C}$ where $((\mathbf{q}_1, \mathbf{q}_2), \mathbf{c}) \in \mathit{init}$ iff $(\mathbf{q}_1, \mathbf{c} \triangleright C_1) \in \mathit{init}_1$ and $(\mathbf{q}_2, \mathbf{c} \triangleright C_2) \in \mathit{init}_2$,
- $\mathit{tcp} : \mathbf{Q} \to 2^{\mathbf{V}}$ where $\mathit{tcp}((\mathbf{q}_1, \mathbf{q}_2)) = \mathit{tcp}_1(\mathbf{q}_1) \cap \mathit{tcp}_2(\mathbf{q}_2)$,

- $((\mathbf{q}_1, \mathbf{q}_2), \mathbf{g}, \mathit{reset}, (\mathbf{q}'_1, \mathbf{q}'_2)) \in \rightarrow$ if one of the conditions below is satisfied:
  - $\mathbf{q}_1 = \mathbf{q}'_1, \exists(\mathbf{q}_2, \mathbf{g}_2, \mathit{reset}_2, \mathbf{q}'_2) \in \rightarrow_2$ s.t. $\mathbf{g} = \mathbf{g}_2$ and $\mathbf{c} \in \mathit{reset}(\mathbf{v})$ iff $\mathbf{c} \triangleright C_2 \in \mathit{reset}_2(\mathbf{v})$ and $\mathbf{c} \triangleright C_1 = \mathbf{v} \triangleright C_1$.
  - $(\mathbf{q}_1, \mathbf{g}_1, \mathit{reset}_1, \mathbf{q}'_1) \in \rightarrow_1, \mathbf{g} = \mathbf{g}_1$ and $\mathbf{c} \in \mathit{reset}(\mathbf{v})$ iff $\mathbf{c} \triangleright C_1 \in \mathit{reset}_1(\mathbf{v})$, $\mathbf{c} \triangleright C_2 = \mathbf{v} \triangleright C_2, \mathit{and} \mathbf{q}_2 = \mathbf{q}'_2$.
  - $(\mathbf{q}_1, \mathbf{g}_1, \mathit{reset}_1, \mathbf{q}'_1) \in \rightarrow_1, (\mathbf{q}_2, \mathbf{g}_2, \mathit{reset}_2, \mathbf{q}'_2) \in \rightarrow_2, \mathbf{g} = \mathbf{g}_1 \cap \mathbf{g}_2$ and $\mathbf{c} \in \mathit{reset}(\mathbf{v})$ iff $\mathbf{c} \triangleright C_1 \in \mathit{reset}_1(\mathbf{v})$ and $\mathbf{c} \triangleright C_2 \in \mathit{reset}_2(\mathbf{v})$.

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

3 Conflict between controllers

In this section we introduce a running example and illustrate conflict between two controllers. 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 1 (Plant)**.** *Let $V = X \cup U$. A plant over (X, U, Y) is a deterministic hybrid automaton $\mathcal{P}$ over V, which controls the variables in X. We assume that the plant $\mathcal{P}$ is **non-blocking** in that if $\sigma \in L(\mathcal{P})$, then $L_\sigma(\mathcal{P}) \neq \{\varepsilon\}$.*

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 from [13] in which it is assumed that the rotational inertia of the wheels is negligible and the friction retarding the motion of the car is proportional to the car's speed. The equation of 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 hybrid automaton model of the plant 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 cruise control 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.

The flow conditions of the plant are not shown in the figure and are as follows.

$$\begin{aligned} \dot{x} &= v \\ \dot{v} &= -\frac{b}{m}v + \frac{1}{m}u. \end{aligned}$$

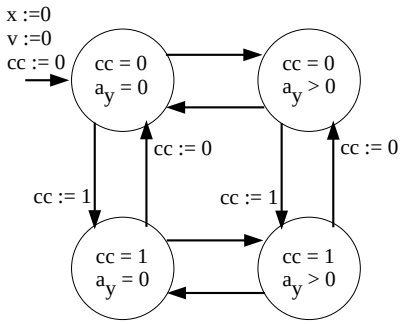

Figure 3: Plant $\mathcal{P}$.

Definition 2 (Controller)**.** *A* ***controller*** *over* (X, U, Y) *is a deterministic hybrid automaton* $\mathcal{C} = (Q, V, U, F, init, tcp, \rightarrow)$ *over* V *which controls the variables in* U*. The controller* $\mathcal{C}$ *is valid with respect to a plant* $\mathcal{P}$ *over* (X, U, Y) *if* $\mathcal{C}$ *is* ***non-blocking*** *with respect to* $\mathcal{P}$*, i.e. if* $\sigma \in L(\mathcal{P}\|\mathcal{C})$*, then* $L_\sigma(\mathcal{P}\|\mathcal{C}) \neq \{\varepsilon\}$*.*

Let $W = X \cup Y$. For a classical safety specification, we use a deterministic hybrid automaton $\mathcal{H}$ over W which gives a safety language $L(\mathcal{H})$. Let $\mathcal{P}$ be a plant over (X, U, Y), $\mathcal{C}$ be a controller for $\mathcal{P}$ and $\mathcal{S}$ be a safety specification over (X, U, Y). We say that the controller $\mathcal{C}$ for $\mathcal{P}$ *satisfies* $\mathcal{S}$ if $L(\mathcal{P}\|\mathcal{C}) \trianglerighteq X \subseteq L(\mathcal{S}) \trianglerighteq X$.

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 whereas 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 $|V_r - v| < \epsilon$. threshold.

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.

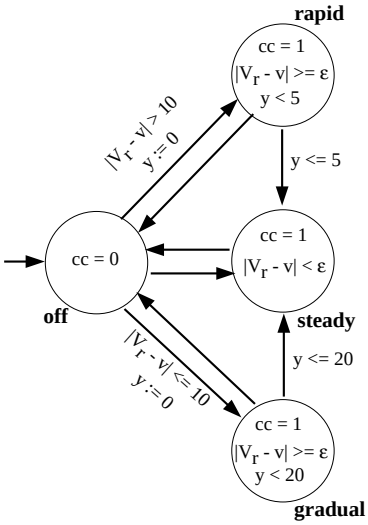

Figure 4: Cruise Control Specification $\mathcal{S}_{cc}$. Here y is a clock variable ($\dot{y} = 1$).

The controllers are implemented using proportional, integral control [13]. The input u is of the form [21]

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

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 $\cdot$ s/m, the reference speed set by the driver $V_r = 30$ m/s and the steady state error threshold $\epsilon = 5\%$ 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 $\mathcal{P}$ and satisfies the cruise control specification $\mathcal{S}_{cc}$.

Now consider another feature called *electronic 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

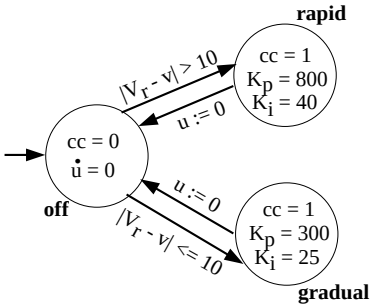

Figure 5: Cruise control implementation. In gradual and rapid modes, $\dot{u} = -K_p\dot{v} + K_i(V_r - \dot{x})$

.

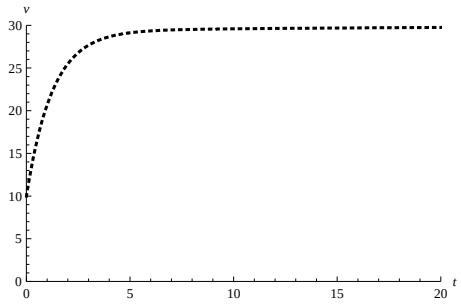

Figure 6: Cruise Control Rapid Mode ($V_r = 30$ m/s, $K_p = 800, K_i = 40$).

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 3 (Conflict)**.** *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be valid controllers for a plant $\mathcal{P}$. The controllers $\mathcal{C}_1$ and $\mathcal{C}_2$ are in* conflict *with respect to $\mathcal{P}$, if there exists a behaviour τ in $L(\mathcal{P}\|\mathcal{C}_1\|\mathcal{C}_2)$ such that $L_\tau(\mathcal{P}\|\mathcal{C}_1\|\mathcal{C}_2) = \{\varepsilon\}$, i.e. after the plant behaviour τ, the controllers $\mathcal{C}_1$ and $\mathcal{C}_2$ 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).

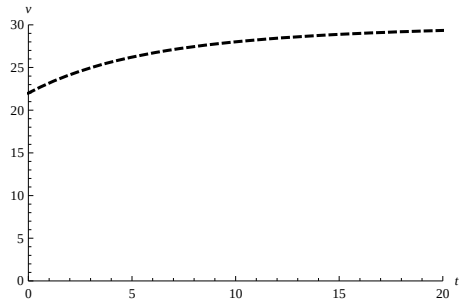

Figure 7: Cruise Control Gradual Mode ($V_r = 30$ m/s, $K_p = 300, K_i = 25$).

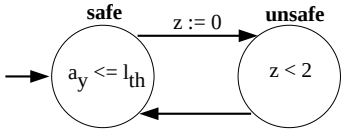

Figure 8: Electronic Stability Control Specification. z is a clock variable ($\dot{z} = 1$).

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.

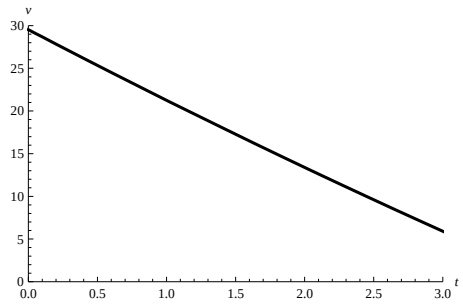

Figure 9: Electronic Stability Control Implementation.

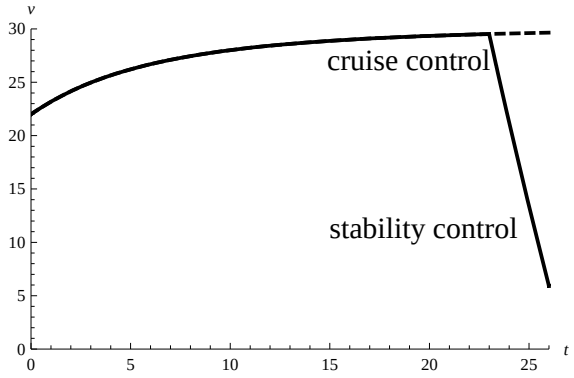

Figure 10: Conflict between Cruise Control and Stability Control.

4 Tolerant specifications

In the previous section, we saw a classical safety specification over a set of variables W being represented as a hybrid automaton whose language is a prefix-closed subset of $\mathcal{W}$. We now introduce our notion of a conflict-tolerant specification.

A *conflict-tolerant* (safety) specification over a set of variables W is an advice function as defined below:

Definition 4 (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 $\mathcal{P}$ be a plant over a partitioned set of variables (X, U, Y). Let $\mathcal{C}$ be a controller for $\mathcal{P}$. Let σ be a behaviour of the plant not necessarily generated under the control of the controller $\mathcal{C}$. Then we can define the behaviour of the plant $\mathcal{P}$ controlled by $\mathcal{C}$, *after* the plant behaviour σ, denoted $L^c_\sigma(\mathcal{P}\|\mathcal{C})$, as follows. Suppose the plant $\mathcal{P}$ has reached a configuration $(\mathbf{p}, (\mathbf{x}, \mathbf{u}))$ after σ. Let the configuration reached by the controller $\mathcal{C}$ after observing σ be $(\mathbf{q}, (\mathbf{x}, \mathbf{u}'))$. Then $L^c_\sigma(\mathcal{P}\|\mathcal{C})$ is defined to be the set of behaviours of $\mathcal{P}\|\mathcal{C}$ 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 $\mathcal{C}$.

Definition 5 ($\mathcal{C}$ satisfies f)**.** *Let $\mathcal{P}$ be a plant over (X, U, Y) and let f be a conflict-tolerant advice function over (X, U, Y). A controller $\mathcal{C}$ for $\mathcal{P}$ satisfies f if for each $\sigma \in L(\mathcal{P}) \trianglerighteq X$, $L^c_\sigma(\mathcal{P}\|\mathcal{C}) \trianglerighteq X \subseteq f(\sigma) \trianglerighteq X$. Thus after **any** plant behaviour σ over X, if the plant follows the advice of $\mathcal{C}$, then the resulting behaviour over X conforms to the safety language over X prescribed by f after observing σ.*

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 6 (Conflict Tolerant Specification)**.** *A conflict-tolerant specification given by hybrid automata over the partitioned set of variables* (X, U, Y) *is a tuple* $\mathcal{S}' = (Acc, Adv, E)$ *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* $(\mathbf{q}, \mathbf{w}') \in m((\mathbf{p}, \mathbf{w}))$ *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* $m_E((\mathbf{p}, \mathbf{w})) = (\mathbf{q}, \mathbf{w}')$*, then* $\mathbf{w}' \triangleright X = \mathbf{w} \triangleright X$*.*

The conflict-tolerant specification $\mathcal{S}'$ above defines an advice function over the set of variables X as follows. We define the *unconstrained* signal language of $\mathcal{S}'$, denoted $L_{(\mathbf{p},\mathbf{w})}(\mathcal{S}')$ to be simply $L(Acc)$. We define the *constrained* signal language of $\mathcal{S}'$ starting from a configuration $(\mathbf{p}, \mathbf{w})$, denoted $L^c_{(\mathbf{p},\mathbf{w})}(\mathcal{S}')$, to be $L_{(\mathbf{q},\mathbf{w}')}(Adv)$ where $(\mathbf{q}, \mathbf{w}') = m_E((\mathbf{p}, \mathbf{w}))$.

Let σ be a signal in $L(\mathcal{S}') \trianglerighteq X$. Then there is a unique configuration $(\mathbf{p}, \mathbf{w})$ reached by σ in the acceptor automaton. By $L_\sigma(\mathcal{S}')$, we mean $L_{(\mathbf{p},\mathbf{w})}(\mathcal{S}')$ and by $L^c_\sigma(\mathcal{S}')$, we mean $L^c_{(\mathbf{p},\mathbf{w})}(\mathcal{S}')$.

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

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

Given a plant $\mathcal{P}$, a controller $\mathcal{C}$, and a conflict-tolerant hybrid specification $\mathcal{S}'$ over (X, U, Y), we say $\mathcal{C}$ satisfies $\mathcal{S}'$ with respect to $\mathcal{P}$ if $\mathcal{C}$ satisfies the advice function $f_{\mathcal{S}'}$ with respect to $\mathcal{P}$.

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 $|V_r - v| < \epsilon$ 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.

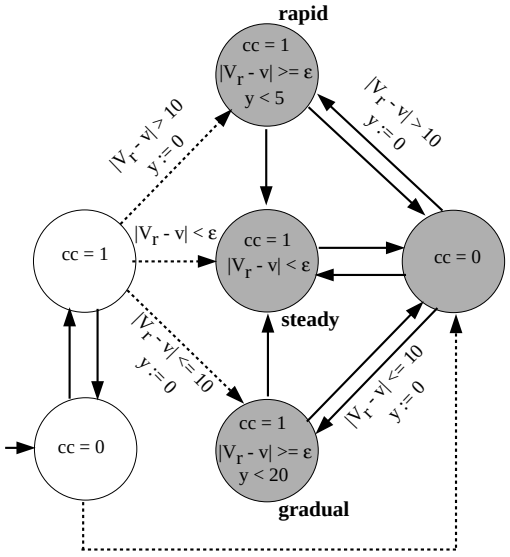

Figure 11: Tolerant Cruise Control Specification $\mathcal{S}'_1$. Here $\dot{y} = 1$.

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,(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.

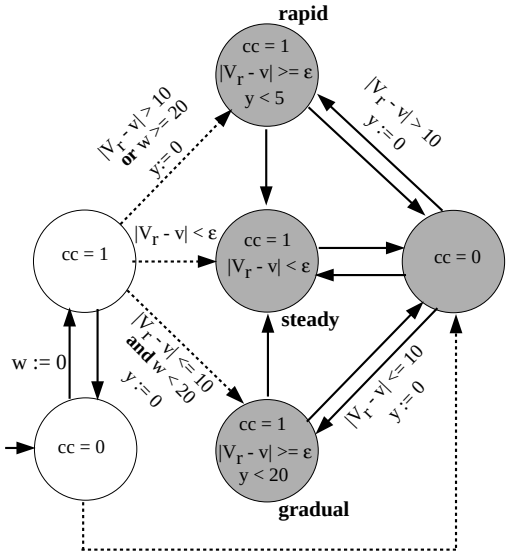

Figure 12: Tolerant Cruise Control Specification $\mathcal{S}_2'$. Here $\dot{y} = \dot{w} = 1$.

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

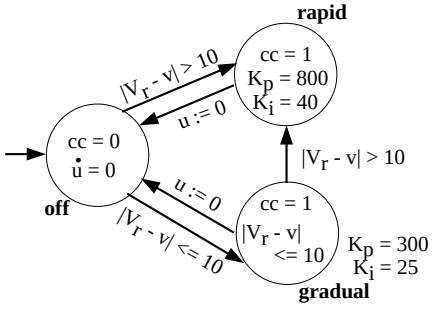

Figure 13: Controller $\mathcal{C}_2$ for tolerant spec $\mathcal{S}_1'$.

Consider now the example plant under the control of the controller $\mathcal{C}_2$ 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 speed. However, stability control intervenes after 23 seconds as the speed has to be reduced while travelling 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 $\mathcal{C}_2$ is a controller that satisfies the tolerant specification $\mathcal{S}'_2$, we have a guarantee that the controlled system conforms to the tolerant specification during the periods when $\mathcal{C}_2$ is in control (shown shaded in the figure).

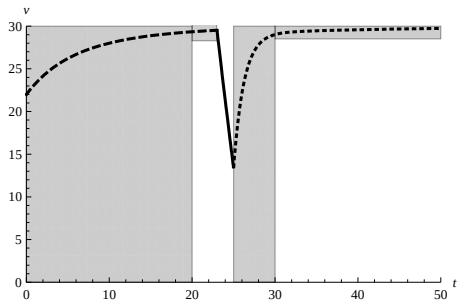

Figure 14: Conflict Resolution.

5 Verification

The verification problem for conflict-tolerant specifications, is the following: given a plant $\mathcal{P}$, a conflict-tolerant specification $\mathcal{S}'$, and a controller $\mathcal{C}$, does $\mathcal{C}$ satisfy $\mathcal{S}'$ with respect to $\mathcal{P}$, in the sense of Definition 5. We now show how this can be solved when the plant $\mathcal{P}$, controller $\mathcal{C}$, and the tolerant specification $\mathcal{S}'$ are given as initialized rectangular automata. We say that the specification $\mathcal{S}'$ is an 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 $(\mathbf{p}, \mathbf{g}, \mathit{reset}, \mathbf{q}) \in E$, then $\mathbf{g}$ and $\mathit{reset}(\mathbf{w})$ are rectangular sets for all $\mathbf{w}$.

We recall the basic definitions of rectangular automata from the literature [15]. 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 $\mathit{init}_q, \mathit{tcp}_q, F_q$ 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 is 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 [4]. Let s, t be two states of a hybrid automaton $\mathcal{H}$. In the *reachability problem* for hybrid automata, we are interested in checking whether there is a trajectory of $\mathcal{H}$ that starts at s and ends at t. It has been shown that the reachability problem for initialized rectangular automata is decidable [19, 15, 20].

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

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

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

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 $\mathcal{M}_{\mathcal{R}}$ is constructed such that this continues to hold even after a jump. The initialized multirate automaton $\mathcal{M}_{\mathcal{R}}$ is further translated into a timed automaton $\mathcal{T}_{\mathcal{R}}$ by rescaling the state space. Let $h_{\mathcal{T}} : \mathbf{Q} \times \mathbb{R}^{2n} \to \mathbf{Q} \times \mathbb{R}^{2n}$ be a function which maps the state space of $\mathcal{T}_{\mathcal{R}}$ to that of $\mathcal{M}_{\mathcal{R}}$ defined by

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

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 $\mathcal{R}$ is "representable" as a set of regions if there exists a set of regions B of $\mathcal{T}_{\mathcal{R}}$ such that $h_{\mathcal{M}}(h_{\mathcal{T}}(B)) = A$. Let $Reach_{\mathcal{H}}(I)$ denote the set of states that can be reached from the set of states I using a trajectory of the automaton $\mathcal{H}$.

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

Let $\mathcal{S}' = (Acc, Adv, E)$ be a conflict-tolerant specification given by hybrid automata. Recall that the advice relation E induces a map m_E which maps a configuration of Acc to a configuration of Adv (Definition 6). Given a map m_E of a tolerant specification, we can obtain an equivalent map m'_E such that $m'_E((\mathbf{p}, \mathbf{y})) = (\mathbf{q}, \mathbf{y}')$ iff $m_E(h_{\mathcal{M}}(h_{\mathcal{T}}(\mathbf{p}, \mathbf{y}))) = h_{\mathcal{M}}(h_{\mathcal{T}}(\mathbf{q}, \mathbf{y}'))$.

Lemma 2. *Let A be a set of configurations of an initialized rectangular automaton $\mathcal{R}$ representable as a set of regions B of $\mathcal{T}_{\mathcal{R}}$. Then the set of configurations $m_E(A)$ is representable as the set of regions $m'_E(B)$.*

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

Proof. Given a deterministic rectangular hybrid automaton $\mathcal{H}$, 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 $\mathcal{H}$ 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 $\mathcal{H}_1 = \mathcal{P} \| \mathcal{C}' \| Acc$ and $\mathcal{H}_2 = \mathcal{P} \| \mathcal{C} \| Adv'$ where $\mathcal{C}'$ is obtained from $\mathcal{C}$ by renaming every variable $u \in U$ to its primed version u'. In order to check if $\mathcal{C}$ does not satisfy $\mathcal{S}'$ with respect to $\mathcal{P}$, it is necessary and sufficient to check the following: $i \rightsquigarrow_{\mathcal{H}_1} ((\mathbf{p}, \mathbf{q}, \mathbf{a}_1), \mathbf{v}) \to ((\mathbf{p}, \mathbf{q}, \mathbf{a}_2), \mathbf{w}) \rightsquigarrow_{\mathcal{H}_2} ((\mathbf{p}', \mathbf{q}', \mathbf{t}), \mathbf{w}')$, i.e. there exists a configuration $((\mathbf{p}, \mathbf{q}, \mathbf{a}_1), \mathbf{v})$ of $\mathcal{H}_1$ reachable from an initial configuration of $\mathcal{H}_1$ and in $\mathcal{H}_2$, we can reach a 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 $\mathcal{C}$ 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.

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 $\mathcal{H}_1$ and $\mathcal{H}_2$ to the corresponding timed automata $\mathcal{T}_{\mathcal{H}_1}$ and $\mathcal{T}_{\mathcal{H}_2}$. Given $I \rightsquigarrow_{\mathcal{H}_1} Reach_{\mathcal{H}_1}(I) \to_{m_E} J \rightsquigarrow_{\mathcal{H}_2} Reach_{\mathcal{H}_2}(J)$, we are interested in checking whether $Reach_{\mathcal{H}_2}(J)$ 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 $\mathcal{T}_{\mathcal{H}_1}$ and $\mathcal{T}_{\mathcal{H}_2}$ as

to whether $I_{\mathcal{T}} \rightsquigarrow_{\mathcal{T}_{\mathcal{H}_1}} Reach_{\mathcal{T}_{\mathcal{H}_1}}(I_{\mathcal{T}}) \rightarrow_{m'_E} J_{\mathcal{T}} \rightsquigarrow_{\mathcal{T}_{\mathcal{H}_2}} Reach_{\mathcal{T}_{\mathcal{H}_2}}$. By Lemmas 1 and 2, it is sufficient to check if $Reach_{\mathcal{T}_{\mathcal{H}_2}}$ 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 $\mathcal{T}_{\mathcal{H}_1}$ and $\mathcal{T}_{\mathcal{H}_2}$. □

We can also check whether a controller is valid with respect to a plant in a similar fashion.

6 Composition of Switching Controllers

We use X, U and Y to denote three disjoint sets of variables to model the variables controlled by the plant, controller and the specification respectively. The variables in U are the discrete input variables using which a controller controls the plant. The set of variables Y is for use of the specification, for e.g. clock variables. For the rest of the paper, we fix a triple (X, U, Y) and we set $W = X \cup Y$. For a classical safety specification, we use a deterministic hybrid automaton $\mathcal{H}$ over W which gives a safety language $L(\mathcal{H})$.

For simplicity, we assume that there is only one discrete variable u in U. Let $V = X \cup \{u\}$. Let $\mathcal{H}$ be a hybrid automaton over V. Let $\sigma \in L(\mathcal{H}) \trianglerighteq X$. We define *u-set after* σ of $\mathcal{H}$, i.e. the value of u in the extension of σ, denoted $\mathcal{H}(\sigma)$, as

$$\mathcal{H}(\sigma) = \{\tau(0) \triangleright u \mid \tau \in L_\sigma(\mathcal{H})\}.$$

Definition 7 (Plant)**.** *A plant over (X, U, Y) is a deterministic hybrid automaton $\mathcal{P}$ over V, which controls the variables in X. We assume that the plant $\mathcal{P}$ is* ***non-blocking*** *in that if $\sigma \in L(\mathcal{P})$, then $L_\sigma(\mathcal{P}) \neq \{\varepsilon\}$.*

As a running example, we consider a water tank equipped with a pump. The hybrid automaton for the water tank (the plant) is shown in Figure 15. The controlled variables of $\mathcal{P}$, i.e. variables in X, are shown in bold and the uncontrolled variables, i.e. variables in U, are shown in italics. The set $X = \{\mathbf{w}, \mathbf{s}\}$, where $\mathbf{w}$ is a continuous variable, which denotes the water level and $\mathbf{s}$ is a discrete variable, which denotes a sensor with value either 0 or 1. The set $U = \{p\}$, where p is a discrete variable with value 0 when the pump is turned off and with value 1 when the pump is turned on.

Initially, the water level is 2 cm and the sensor $\mathbf{s}$ is 0. The water level rises at the rate of 1 cm/sec when the pump is on and falls at the rate of 1 cm/sec when the pump is off. The sensor $\mathbf{s}$ can become 1 only when $\mathbf{w} \geq 3$ and can return to 0 again only when $\mathbf{w} \geq 5$. Note that the water level $\mathbf{w}$ will always be between 0 and 6 cm.

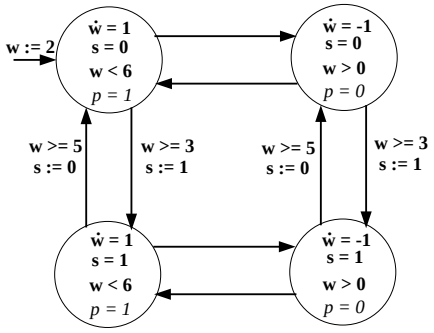

Figure 15: Water Tank (plant) $\mathcal{P}$.

Definition 8 (Switching Controller)**.** *A* ***switching controller*** *over* (X, U, Y) *is a hybrid automaton* $\mathcal{C} = (Q, V, U, F, init, tcp, \rightarrow)$ *which controls the variable* $u \in U$*. The controller* $\mathcal{C}$ *is valid with respect to a plant* $\mathcal{P}$ *over* (X, U, Y) *if* $\mathcal{C}$ *is* ***strongly non-blocking*** *with respect to* $\mathcal{P}$*, i.e. if* $\sigma \in L(\mathcal{P}\|\mathcal{C})$*, then* $\mathcal{C}(\sigma) \subseteq \mathcal{P}(\sigma)$ *and* $\mathcal{C}(\sigma) \neq \emptyset$*.*

Let $\mathcal{P}$ be a plant over (X, U, Y) and let $\mathcal{C}$ be a controller for $\mathcal{P}$. Let $\sigma : [0, r) \rightarrow \mathbf{V} \in L(\mathcal{P})$. We say that σ is according to the advice of $\mathcal{C}$ at time t $(0 \leq t < r)$, if $\sigma = \sigma_1 \cdot \sigma_2$ and $\sigma_2(0) \in \mathcal{C}(\sigma_1)$. We say σ *is according to* $\mathcal{C}$ if σ is according to the advice of $\mathcal{C}$ at all times t in $[0, r)$.

Let $\mathcal{S}$ be a safety specification over (X, U, Y). We say a switching controller $\mathcal{C}$ for the plant $\mathcal{P}$ *satisfies* $\mathcal{S}$ if $L(\mathcal{P}\|\mathcal{C}) \trianglerighteq X \subseteq L(\mathcal{S}) \trianglerighteq X$.

As an example, we consider a system with respect to $(\{w, s\}, \{p\}, \{y\})$ where a feature requires that the water level w must always be between 2 cm and 4 cm. Figure 16(a) shows a specification $\mathcal{S}_1$ and Figure 16(b) shows a controller $\mathcal{C}_1$ for this requirement. Note that $\mathcal{C}_1$ controls only the pump p. It switches the pump off when the water level reaches 4 cm and switches the pump on when the water level falls to 2 cm. Initially, the pump is switched on.

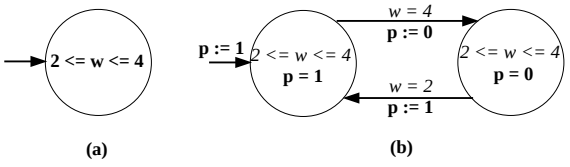

Figure 16: (a) Specification $\mathcal{S}_1$ and (b) controller $\mathcal{C}_1$.

It is easy to see that the controller $\mathcal{C}_1$ is valid for $\mathcal{P}$ and satisfies $\mathcal{S}_1$.

Now consider another feature which requires that if the sensor **s** is set to 1, then either the sensor is reset or within 2 seconds, the water level must be kept greater than or equal to 5 cm. Figure 17 shows a specification $\mathcal{S}_2$ for this requirement. This specification introduces a clock variable **y** (i.e. $\dot{\mathbf{y}} = 1$) which is used to indicate that the water level must be greater than or equal to 5 cm within 2 seconds of setting **s** to 1.

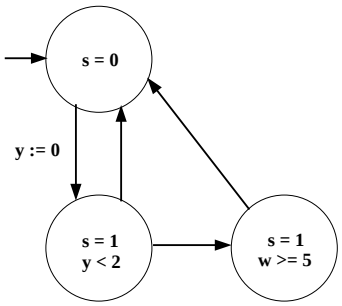

Figure 17: Specification $\mathcal{S}_2$.

Figure 18 shows a controller $\mathcal{C}_2$ for satisfying specification $\mathcal{S}_2$. If the sensor **s** is set to 1, then the controller $\mathcal{C}_2$ controls the pump as follows:

- If $\mathbf{w} \leq 5$, then the pump must be switched on.
- If $5 < \mathbf{w} < 6$, then the pump can be either on or off.
- If $\mathbf{w} = 6$, then the pump must be switched off.

Note that the sensor **s** can be set to 1 only when the water level is greater than or equal to 3 cm (see Figure 15). Therefore, when the sensor **s** is set to 1 and the water level is less than 5 cm, the controller $\mathcal{C}_2$ can satisfy the specification by keeping the pump on (mode q_3).

We now illustrate the notion of conflict between controllers.

Definition 9 (Conflict)**.** *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be valid controllers for a plant $\mathcal{P}$. The controllers $\mathcal{C}_1$ and $\mathcal{C}_2$ are in* conflict *with respect to $\mathcal{P}$, if there exists a behaviour τ in $L(\mathcal{P}\|\mathcal{C}_1\|\mathcal{C}_2)$ such that $L_\tau(\mathcal{P}\|\mathcal{C}_1\|\mathcal{C}_2) \neq \{\varepsilon\}$. In other words, there exists a behaviour $\sigma \in L(\mathcal{P}\|\mathcal{C}_1\|\mathcal{C}_2) \rhd X$ such that the controllers do not agree on a u-set after σ, i.e. $\mathcal{C}_1(\sigma) \cap \mathcal{C}_2(\sigma) = \emptyset$.*

Consider the example plant behaviour shown in Figure 19. The sensor **s** is set to 1 at time 1.5. This requires $\mathcal{C}_2$ to keep the pump on until the

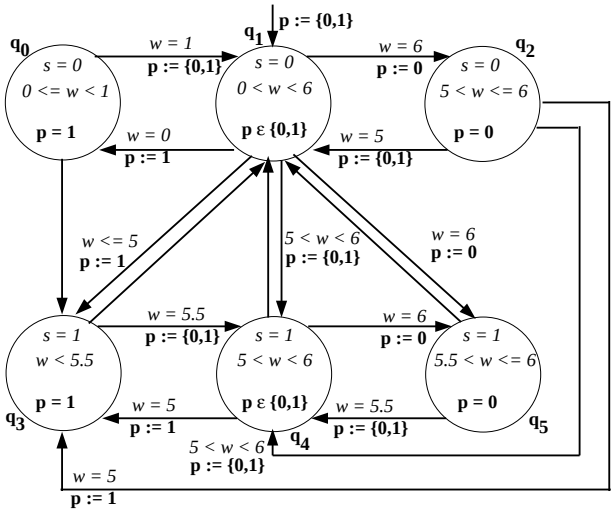

Figure 18: Controller $\mathcal{C}_2$.

water level reaches 5.5 cm. However, $\mathcal{C}_1$ requires the pump to be switched off at time 2 as the water level has reached 4 cm. The controlled system is blocked at time 2 as $\mathcal{C}_1$ requires the pump to be switched off and $\mathcal{C}_2$ requires the pump to be kept on. If $\mathcal{P}$ were to follow the advice of $\mathcal{C}_1$, then water level will start falling at time 2. However, if $\mathcal{P}$ were to follow the advice of $\mathcal{C}_2$, then water level will keep rising after 2 seconds (shown in gray).

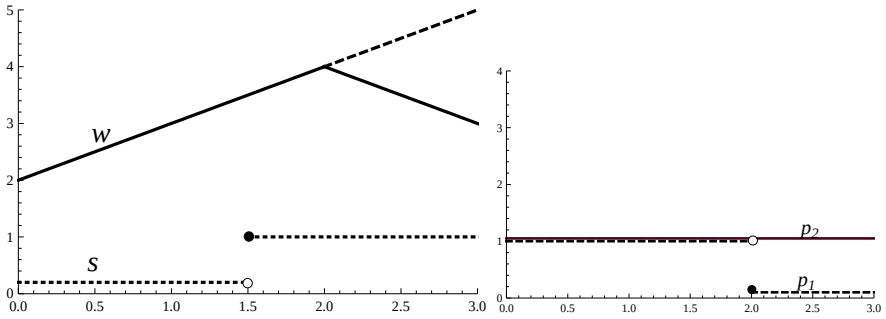

Figure 19: Conflicting advice from $\mathcal{C}_1$ (p_1 shown dashed) and $\mathcal{C}_2$ (p_2).

In this section we introduce our notion of conflict-tolerance in a hybrid systems setting. A *conflict-tolerant* (safety) specification over a set of variables W is an advice function as defined below:

Definition 10. *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.*

Definition 11. *A conflict-tolerant hybrid specification over* (X, U, Y) *is a tuple* $\mathcal{S}' = (Acc, Adv, E)$ *where*

- *Acc is a hybrid automaton over* W *called the* ***acceptor****. Let* $Acc = (P, W, W, F_1, init_1, tcp_1, \to_1)$*. 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* $Adv = (Q, W, W, F_2, init_2, tcp_2, \to_2)$*.*
- $E \subseteq \mathbf{P} \times 2^{\mathbf{W}} \times (\mathbf{W} \to 2^{\mathbf{W}}) \times \mathbf{Q}$ *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,* $reset : \mathbf{W} \to 2^{\mathbf{W}}$ *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* $m_E : (\mathbf{P} \times \mathbf{W}) \to 2^{(\mathbf{Q} \times \mathbf{W})}$ *be the map induced by the advice relation* E *such that* $(\mathbf{q}, \mathbf{w}') \in m((\mathbf{p}, \mathbf{w}))$ *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* $m_E((\mathbf{p}, \mathbf{w})) = (\mathbf{q}, \mathbf{w}')$*, then* $\mathbf{w}' \triangleright X = \mathbf{w} \triangleright X$*.*

Let $\mathcal{S}'$ be a conflict-tolerant specification over (X, U, Y) as above. The *unconstrained* signal language of $\mathcal{S}'$ starting from a configuration $(\mathbf{p}, \mathbf{w}) \in \mathbf{P} \times \mathbf{W}$, denoted $L_{(\mathbf{p},\mathbf{w})}(\mathcal{S}')$, is defined to be $L_{(\mathbf{p},\mathbf{w})}(Acc)$. The unconstrained signal language of $\mathcal{S}'$, denoted $L(\mathcal{S}')$, is defined to be $\bigcup_{(\mathbf{p},\mathbf{w})} L_{(\mathbf{p},\mathbf{w})}(\mathcal{S}')$ where $(\mathbf{p}, \mathbf{w}) \in init_1$. The *constrained* signal language of $\mathcal{S}'$ starting from a configuration $(\mathbf{p}, \mathbf{w})$, denoted $L^c_{(\mathbf{p},\mathbf{w})}(\mathcal{S}')$, is defined to be $L_{(\mathbf{q},\mathbf{w}')}(Adv)$ where $(\mathbf{q}, \mathbf{w}') = m_E((\mathbf{p}, \mathbf{w}))$.

Let σ be a signal in $L(\mathcal{S}') \trianglerighteq X$. Then there is a unique configuration $(\mathbf{p}, \mathbf{w})$ reached by σ in the acceptor automaton. By $L_\sigma(\mathcal{S}')$, we mean $L_{(\mathbf{p},\mathbf{w})}(\mathcal{S}')$ and by $L^c_\sigma(\mathcal{S}')$, we mean $L^c_{(\mathbf{p},\mathbf{w})}(\mathcal{S}')$.

A conflict-tolerant specification $\mathcal{S}'$ over X induces a function $f_{\mathcal{S}'}$ given by for all $\sigma \in L(\mathcal{S}')$, $f_{\mathcal{S}'}(\sigma) = L^c_\sigma(\mathcal{S}')$, and $\mathcal{X}$ otherwise.

Returning to our example, we now consider tolerant specifications. Figure 20 shows a tolerant specification for feature 1. The automaton on the left hand side (with modes p_1, p_2) is the acceptor and the automaton on the

right hand side (with modes q_1, q_2) is the advisor. The edges in the advice relation are shown dashed. Note that the classical specification shown in Figure 16(a) does not specify the required behaviour if the water level falls below 2 cm or rises above 4 cm. In contrast, the tolerant specification $\mathcal{S}_1'$ specifies that if the water level is not between 2 and 4 cm, then it must be kept within 2 and 4 cm within 2 seconds.

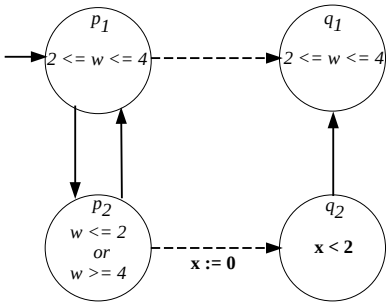

Figure 20: Tolerant specification $\mathcal{S}_1'$.

Figure 21 shows a tolerant specification for feature 2. Note that the classical specification shown in Figure 17 does not specify the required behaviour when 2 seconds have elapsed after the sensor **s** is set to 1 and the water level is still below 5 cm. In contrast, the tolerant specification $\mathcal{S}_2'$ continues to advise that the water level must be kept within greater than or equal to 5 cm within 2 seconds (mode p_3).

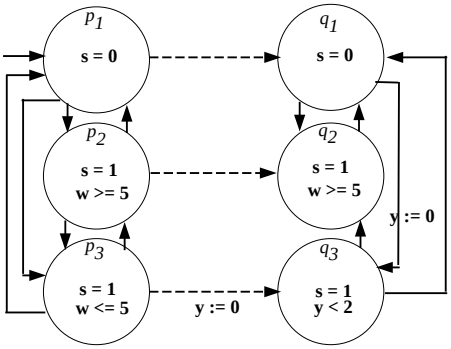

Figure 21: Tolerant specification $\mathcal{S}_2'$.

We now define the notion of a tolerant switching controller which is a switching controller with certain properties as given below. Let $\mathcal{P}$ be a plant

over (X, U, Y). Let $\mathcal{C} = (Q, V, U, F, init, tcp, \rightarrow)$ be a switching controller for $\mathcal{P}$. We say that $\mathcal{C}$ is *complete* with respect to $L(\mathcal{P})$ if for all $\sigma \in L(\mathcal{P})$, there exists a trajectory $\tau \in Traj(\mathcal{C})$ such that $\tau \trianglerighteq X = \sigma \trianglerighteq X$. We say that $\mathcal{C}$ is *mode deterministic* with respect to $L(\mathcal{P})$ if for all $\sigma \in L(\mathcal{P})$ and for all trajectories $\tau, \tau' \in Traj(\mathcal{C})$ such that $\tau \trianglerighteq X = \sigma \trianglerighteq X$ and $\tau' \trianglerighteq X = \sigma \trianglerighteq X$, the mode signals are equal, i.e. $\tau \trianglerighteq Q = \tau' \trianglerighteq Q$.

Let the switching controller $\mathcal{C}$ be complete and mode deterministic with respect to $L(\mathcal{P})$. Let $\sigma \in L(\mathcal{P})$ and let $config_{\mathcal{C}}(\sigma \trianglerighteq X)$, i.e. the possible configurations of $\mathcal{C}$ after σ be $\{(\mathbf{q}, \mathbf{v}_1), (\mathbf{q}, \mathbf{v}_2), \cdots, (\mathbf{q}, \mathbf{v}_n)\}$, where $n \geq 1$. We say that $\mathcal{C}$ is *u-switch permissive* iff whenever $v_i \in tcp(\mathbf{q})$, $\mathcal{C}(\sigma \trianglerighteq X) = tcp(\mathbf{q}) \triangleright \{u\}$, i.e. in every mode $\mathbf{q}$, $\mathcal{C}$ allows switching of a u-signal to any of the values permitted by the time can progress condition of that mode.

Figure 22 shows a u-switch permissive controller which controls the discrete variable $\mathbf{p}$. Note that $\dot{\mathbf{p}} = 0$ as $\mathbf{p}$ is a discrete variable. If the controller is u-switch permissive, then the value of $\mathbf{p}$ can be switched between 0 and 1, i.e. the pump can be switched on or off when $0 < w < 6$.

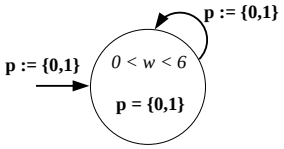

Figure 22: A controller that is u-switch permissive.

Let $\sigma \in L(\mathcal{P})$. Let $config_{\mathcal{P}}(\sigma)$, i.e. the configuration of $\mathcal{P}$ after σ be $\{(\mathbf{b}, \mathbf{v})\}$. Let $config_{\mathcal{C}}(\sigma \trianglerighteq X)$, i.e. the possible configurations of $\mathcal{C}$ after σ be $\{(\mathbf{q}, \mathbf{v}_1), (\mathbf{q}, \mathbf{v}_2), \cdots, (\mathbf{q}, \mathbf{v}_n)\}$, where $n \geq 1$. Note that for $i = 1..n$, $\mathbf{v} \triangleright X = \mathbf{v}_i \triangleright X$. The *controlled behaviour after* σ from a configuration $(\mathbf{b}, \mathbf{q}, \mathbf{v}_i)$ is defined to be $L_{(\mathbf{b},\mathbf{q},\mathbf{v}_i)}(\mathcal{P} \| \mathcal{C})$, i.e. the state of the variables controlled by $\mathcal{P}$ remain the same, but u is assigned the value $\mathbf{v}_i \triangleright u$. The controlled behaviour after σ, denoted $L_\sigma(\mathcal{P} \| \mathcal{C})$, is defined as $\bigcup_{i=1..n} L_{(\mathbf{b},\mathbf{q},\mathbf{v}_i)}(\mathcal{P} \| \mathcal{C})$.

Definition 12. *Let $\mathcal{P}$ be a plant over (X, U, Y). A conflict-tolerant switching controller $\mathcal{C}'$ for $\mathcal{P}$ is a switching controller for $\mathcal{P}$ that is complete, mode deterministic and u-switch permissive with respect to $L(\mathcal{P})$. The switching controller $\mathcal{C}'$ is valid with respect to $\mathcal{P}$ if $\mathcal{C}'$ is* ***strongly non-blocking*** *with respect to $\mathcal{P}$, i.e. if $\sigma \in L(\mathcal{P})$, then $\mathcal{C}'(\sigma) \subseteq \mathcal{P}(\sigma)$ and $\mathcal{C}'(\sigma) \neq \emptyset$.*

Thus a conflict-tolerant controller $\mathcal{C}'$ must observe and advise how to extend each behaviour σ of the plant. Note that such a behaviour σ may not be according to the advice of $\mathcal{C}'$, in that $\sigma \notin L(\mathcal{P} \| \mathcal{C}')$. In contrast, a classical

controller $\mathcal{C}$ (see Definition 2) assumes that its advice is always followed by the plant, and hence it needs to advise extensions of only controlled behaviours (i.e. those in $L(\mathcal{P}\|\mathcal{C})$).

Definition 13 ($\mathcal{C}'$ satisfies f)**.** *Let $\mathcal{P}$ be a plant over (X, U, Y) and let f be a conflict-tolerant advice function over (X, U, Y). A conflict-tolerant switching controller $\mathcal{C}'$ for $\mathcal{P}$ satisfies f if for each $\sigma \in L(\mathcal{P}) \trianglerighteq X$, $L_\sigma(\mathcal{P}\|\mathcal{C}') \trianglerighteq X \subseteq f(\sigma) \trianglerighteq X$. Thus after **any** plant behaviour σ over X, if the plant follows the advice of $\mathcal{C}'$, then the resulting behaviour over X conforms to the safety language over X prescribed by f after observing σ.*

Figure 23 shows a tolerant controller $\mathcal{C}_1'$ that satisfies the tolerant specification given by $\mathcal{S}_1'$ of Figure 20. The controller $\mathcal{C}_1'$ advises that the pump must be kept on in mode q_0 and it must be kept off in mode q_2. As the water tank has a capacity of only 6 cm, this controller can satisfy the specification $\mathcal{S}_1'$ if the water level drops below 2 cm or rises above 4 cm. In mode q_1, the controller $\mathcal{C}_1'$ advises that the pump can be either on or off. Note that the transitions into mode q_1 from other modes must set $\mathbf{p}$ to $\{0, 1\}$ and the self loop must be present for $\mathcal{C}_1'$ to be u-switch permissive.

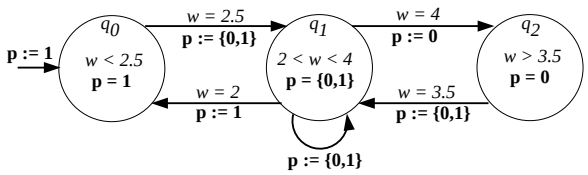

Figure 23: Tolerant controller $\mathcal{C}_1'$.

Figure 24 shows a tolerant controller $\mathcal{C}_2'$ that satisfies the tolerant specification given by $\mathcal{S}_2'$ of Figure 21. Note that this is the same as the classical controller in Figure 18 except the transition from mode q_2 to q_5 and the self loops in mode q_1 and q_4.

Our aim now is to show that a natural priority based supervisory controller can be built for a given set of conflict-tolerant controllers. The supervisory controller will guarantee that each of the individual controllers are used maximally.

Let $\mathcal{C}_1'$ and $\mathcal{C}_2'$ be valid conflict-tolerant controllers for a plant $\mathcal{P}$ over (X, U, Y). Let P be a priority ordering between $\mathcal{C}_1'$ and $\mathcal{C}_2'$, and say P assigns a higher priority to $\mathcal{C}_1'$. We denote this by $\mathcal{C}_1' >_P \mathcal{C}_2'$. Figure 25 shows that both $\mathcal{C}_1'$ and $\mathcal{C}_2'$ observe the base system and offer their advice which is used by the supervisory controller $\mathcal{C}$ to control the base system.

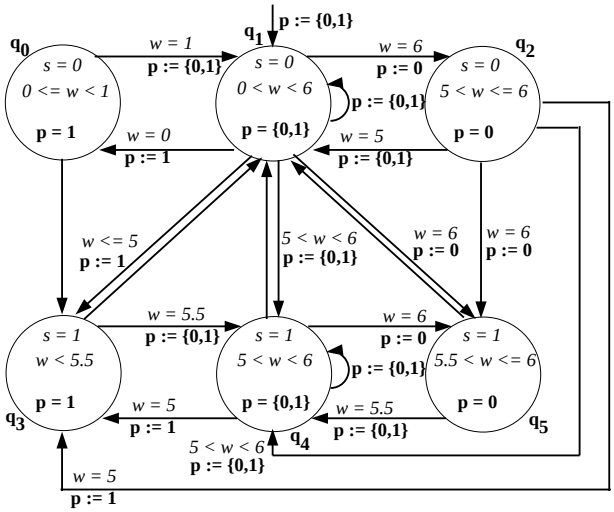

Figure 24: Tolerant controller $\mathcal{C}_2'$.

We rename the variable u in $\mathcal{C}_1'$ to u_1 and get $\mathcal{C}_1''$. Similarly, we rename the variable u in $\mathcal{C}_2'$ to u_2 and get $\mathcal{C}_2''$. We extend the set U such that $U' = \{u, u_1, u_2\}$. Let $\mathcal{C}_{12}$ be the hybrid automaton $\mathcal{C}_1''\|\mathcal{C}_2''$ over $V' = X \cup U'$. For a predicate ϕ over the set of variables U' (for example, $u = u_1 \wedge u = u_2$), we write $[\![\phi]\!]$ to denote the set of valuations $\mathbf{v}$ that satisfy ϕ.

Definition 14 (Prioritized Composition)**.** *The P-prioritized composition of the controllers $\mathcal{C}_1'$ and $\mathcal{C}_2'$ with respect to the plant $\mathcal{P}$, denoted $\|_{P,\mathcal{P}}(\mathcal{C}_1', \mathcal{C}_2')$, is defined to be the hybrid automaton $\mathcal{C} = (Q, V', U', F, init, tcp, \rightarrow)$ obtained from $\mathcal{C}_{12}$ after making changes according to the following rules:*

1. *Let $I = init_{12} \cap [\![u = u_1 \wedge u = u_2]\!]$. Then*

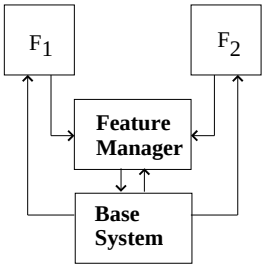

Figure 25: Supervisory Controller $\mathcal{C}$ maximally utilizes the advice of $\mathcal{C}_1'$ and $\mathcal{C}_2'$.

$$init := \begin{cases} I & \text{if } I(\mathbf{q}) \neq \emptyset \\ init_{12} \cap [\![u = u_1]\!] & \text{otherwise.} \end{cases}$$

2. *For all* $\mathbf{q} \in \mathbf{Q}$, *let* $T(\mathbf{q}) = tcp_{12}(\mathbf{q}) \cap [\![u = u_1 \wedge u = u_2]\!]$. *Then*

$$tcp(\mathbf{q}) := \begin{cases} T(\mathbf{q}) & \text{if } T(\mathbf{q}) \neq \emptyset \\ tcp_{12}(\mathbf{q}) \cap [\![u = u_1]\!] & \text{otherwise.} \end{cases}$$

   If $T(\mathbf{q}) = \emptyset$, *then we call* $\mathbf{q}$ *a conflict mode. Otherwise,* $\mathbf{q}$ *is a non-conflict mode.*

3. *For each* $e = (\mathbf{q}, \mathbf{g}, reset, \mathbf{q}') \in \rightarrow_{12}$, *we add a transition* $e' = (\mathbf{q}, \mathbf{g}', reset', \mathbf{q}')$ *to* $\rightarrow$ *where* $\mathbf{g}'$ *and* $reset'$ *are obtained as follows. When* $\mathbf{q}'$ *is a non-conflict mode, the guard and reset conditions are given in Table 1. By* $tcp_{u_2}(\mathbf{q}')$, *we mean the valuations over* u_2 *which satisfy the time can progress condition of* $\mathbf{q}'$. *When* $\mathbf{q}'$ *is a conflict mode, then* $\mathbf{g}' = \mathbf{g}$ *and* $reset' = \lambda v \cdot (reset(v) \cap [\![u = u_1]\!])$.

Transition Type	Guard	Reset
Joint $\mathcal{C}_1', \mathcal{C}_2'$	$\mathbf{g}$	$\lambda v \cdot (reset(v) \cap tcp_{U'}(\mathbf{q}') \cap [\![u = u_1]\!])$
Only $\mathcal{C}_1'$	$\mathbf{g} \cap [\![\mathbf{u}_2 \in tcp_{u_2}(\mathbf{q}')]\!]$	$\lambda v \cdot (reset(v) \cap tcp_{U'}(\mathbf{q}') \cap [\![u = u_1]\!])$
Only $\mathcal{C}_2'$	$\mathbf{g} \cap [\![\mathbf{u}_1 \in tcp_{u_1}(\mathbf{q}')]\!]$	$\lambda v \cdot (reset(v) \cap tcp_{U'}(\mathbf{q}') \cap [\![u = u_1]\!])$

Table 1: Changes to guard and reset conditions of an edge e when the target of e is a non-conflict mode.

Lemma 3. *Let* $\mathcal{P}$ *be a plant over* (X, U, Y). *For each* $\sigma \in L(\mathcal{P} \| \mathcal{C}) \trianglerighteq X$, *the* u*-set of the switching controller* $\mathcal{C}$, $\mathcal{C}(\sigma)$ *is a subset of* $\mathcal{P}(\sigma)$ *and* $\mathcal{C}(\sigma) \neq \emptyset$. *Furthermore,*

$$\mathcal{C}(\sigma) = \begin{cases} \mathcal{C}_1'(\sigma) \cap \mathcal{C}_2'(\sigma) \\ \qquad \text{if } \mathcal{C}_1'(\sigma) \cap \mathcal{C}_2'(\sigma) \neq \emptyset \\ \mathcal{C}_1'(\sigma) \quad \text{otherwise.} \end{cases}$$

Proof. Let $\sigma \in L(\mathcal{P} \| \mathcal{C}) \trianglerighteq X$. Let $t_1, \ldots, t_n$, where $n \geq 0$ be the sequence of time points at which the switching controller $\mathcal{C}$ jumps during a run on σ. Note that both $\mathcal{C}_1'$ and $\mathcal{C}_2'$ have a run on σ. Furthermore, the trajectories of $\mathcal{C}_1'$ and $\mathcal{C}_2'$ agree with that of $\mathcal{C}$ on the mode signal. Also, the union of the time points at which $\mathcal{C}_1'$ and $\mathcal{C}_2'$ jump is exactly $\{t_1, \ldots, t_n\}$.

Next, we note that the *reset* of all edges in $\mathcal{C}$ matches the u-set given by the *tcp* condition in the target mode. This can be verified from Table 1.

Hence the u-set advised by $\mathcal{C}$ in each mode is determined fully by the *tcp* condition of that mode. We can now verify that in any interval $[t_i, t_{i+1})$, the u-sets S, S_1, S_2 corresponding to the runs of $\mathcal{C}$, $\mathcal{C}'_1$ and $\mathcal{C}'_2$ respectively satisfy the property that $S = S_1 \cap S_2$ whenever $S_1 \cap S_2$ is nonempty (in the non-conflict modes) and $S = S_1$ otherwise (in the conflict modes). This is clear from the way the *tcp* conditions of $\mathcal{C}$ are defined. The lemma now follows. □

Let $\mathcal{S}'_1, \ldots, \mathcal{S}'_n$ be conflict-tolerant specifications. Suppose that each $\mathcal{C}'_j$ individually satisfies the specification $\mathcal{S}'_j$ w.r.t. the plant $\mathcal{P}$. By the nature of our composition construction, it is not difficult to see that:

Theorem 2. *The supervisory controller $\mathcal{C} = \|_{P,\mathcal{P}}(\mathcal{C}'_1, \ldots, \mathcal{C}'_n)$ is a valid switching controller for $\mathcal{P}$. Furthermore, $\mathcal{C}$ satisfies each of the specifications $\mathcal{S}'_1, \ldots, \mathcal{S}'_n$ in the following "maximal" sense. For every $\tau : [0, r) \rightarrow \mathbf{V} \in L(\mathcal{P}\|\mathcal{C})$:*

1. *$\tau \trianglerighteq X \in L^c_\varepsilon(\mathcal{S}'_1) \trianglerighteq X$, i.e. the controller $\mathcal{C}$ always satisfies the specification $\mathcal{S}'_1$.*

2. *For all prefixes $\sigma : [0, s) \rightarrow \mathbf{V}$ of τ, if $C(\sigma \trianglerighteq X) \not\subseteq \mathcal{C}'_j(\sigma \trianglerighteq X)$, then there is a controller $\mathcal{C}'_k$ such that $\mathcal{C}'_k >_P \mathcal{C}'_j$ and $\mathcal{C}'_k(\sigma \trianglerighteq X) \cap \mathcal{C}'_j(\sigma \trianglerighteq X) = \emptyset$, i.e. if the plant $\mathcal{P}$ is switched to a mode that is not advised by $\mathcal{C}'_j$, then $\mathcal{C}'_j$ is in conflict with the advice of the higher priority controller $\mathcal{C}'_k$. Furthermore, the portions of τ according to and not according to $\mathcal{C}'_j$s advice are "finitely varying" in that we can cover τ by a finite sequence of adjacent non-empty intervals such that in every interval, either τ is according to the advice of $\mathcal{C}'_j$ or τ is never according to the advice of $\mathcal{C}'_j$ (see Figure 2).*

□

Consider the plant behaviour which we used to illustrate conflict in Section 3. The switching controller $\mathcal{C}_1$ required the pump to be switched off after 2 seconds as the water level has reached 4 cm. However, the switching controller $\mathcal{C}_2$ required the pump to be kept on so that it can maintain the water level at or above 5 cm as long as the sensor **s** is set to 1. This conflict can be resolved by using tolerant controllers and composing them by prioritizing one controller over the other.

Suppose that we compose the tolerant switching controllers $\mathcal{C}'_1$ (Figure 23) and $\mathcal{C}'_2$ (Figure 24) with the priority order $\mathcal{C}'_2 >_P \mathcal{C}'_1$. Then the conflict is resolved in favour of the switching controller $\mathcal{C}'_2$ as shown in Figure 26. The water level continues to rise till 3.5 seconds following the advice of $\mathcal{C}'_2$. When

the water level reaches 5.5 cm, the advice of $\mathcal{C}_2'$ changes such that the pump can be kept on or off. From then on, the water level starts falling as the pump can be switched off following the advice of both $\mathcal{C}_2'$ and $\mathcal{C}_1'$.

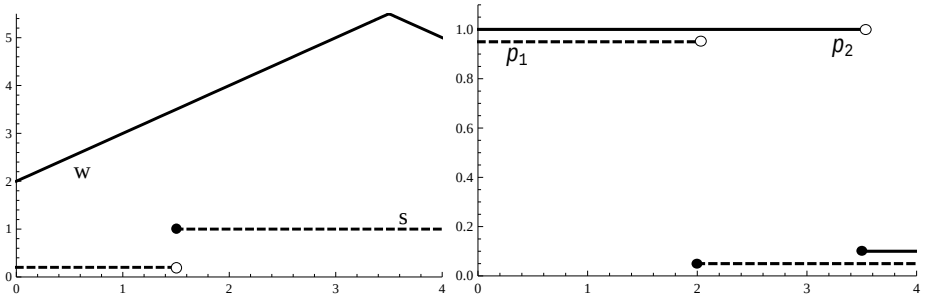

Figure 26: Conflict resolved in favour of $\mathcal{C}_2'$.

7 Conclusion

In this paper we have introduced conflict-tolerant specifications in the hybrid setting. We have also given a decision procedure for verifying whether a controller satisfies a conflict-tolerant specification when the plant, controller and the specification are modeled using initialized rectangular automata. Given controllers that are verified to conform to their respective tolerant specifications, it is easy to build a priority based supervisory controller which ignores the input of a controller if it conflicts with that of a higher priority controller. In the restricted setting of switched control, we have shown how to construct a supervisory controller which guarantees the "maximal" use of every controller by ensuring that the advice of each controller is always followed *except* at time instants when *each* of the controller's advice conflicts with that of a higher priority controller [10].

In earlier work we had required specifications to be "consistent," in that for all advised signals $\sigma \cdot \tau$, the advice after $\sigma \cdot \tau$ is exactly the same as all the extensions of τ in the advice after σ. However, this restriction precludes specifications like the one in Figure 1(b) which are natural requirements to specify when building controllers that can be suspended and then resumed later. The specifications we consider are however "weakly consistent" in the sense that the advice after $\sigma \cdot \tau$ always *contains* the extensions of τ in the advice after σ.

If valid conflict-tolerant controllers can be constructed to satisfy such specifications, 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 with which it is integrated.

References

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

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[3] R. Alur, T. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *Real-Time Systems Symposium, 1993., Proceedings.*, pages 2–11, Dec 1993.

[4] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proc. of the IEEE*, 88(7):971–984, Jul 2000.

[5] 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.

[6] 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.

[7] D. D'Souza and M. Gopinathan. Conflict-tolerant features. In *Computer Aided Verification*, pages 227–239, 2008.

[8] D. D'Souza, M. Gopinathan, S. Ramesh, and P. Sampath. Conflict-tolerant real-time features. In *Quantitative Evaluation of Systems*, pages 274–283, 2008.

[9] D. D'Souza, M. Gopinathan, S. Ramesh, and P. Sampath. Supervisory control for real-time systems based on conflict-tolerant controllers. In *Conference on Automation Science and Engineering*, 2009.

[10] D. D'Souza, M. Gopinathan, R. S., and P. Sampath. Conflict-tolerant specifications for hybrid systems. Technical Report IISc-CSA-TR-2010-6, Indian Institute of Science, 2010. `http://www.csa.iisc.ernet.in/TR/2010/6/`.

[11] A. P. Felty and K. S. Namjoshi. Feature specification and automated conflict detection. *ACM Trans. Softw. Eng. Methodol.*, 12(1):3–27, 2003.

[12] The top 10 safety features for the future – `http://www.msnbc.msn.com/id/23300261`.

[13] G. F. Franklin, J. D. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems*. Prentice-Hall, 2006.

[14] J. D. Hay and J. M. Atlee. Composing features and resolving interactions. In *SIGSOFT Found. of Softw. Engg.*, pages 110–119, 2000.

[15] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *J. Comput. Syst. Sci.*, 57(1):94–124, 1998.

[16] D. O. Keck and P. J. Kühn. The feature and service interaction problem in telecommunications systems. a survey. *IEEE Trans. Software Eng.*, 24(10):779–796, 1998.

[17] Lecture notes on hybrid systems – john lygeros.

[18] N. A. Lynch, R. Segala, and F. W. Vaandrager. Hybrid i/o automata. *Inf. Comput.*, 185(1):105–157, 2003.

[19] A. Puri and P. Varaiya. Decidability of hybrid systems with rectangular differential inclusion. In *CAV*, pages 95–104, 1994.

[20] A. Puri and P. Varaiya. Decidable hybrid systems. In *In Hybrid Systems III, LNCS 1066*, pages 413–423. Springer, 1996.

[21] R. Rajamani. *Vehicle Dynamics and Control*. Springer, 2006.

[22] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. In *Proc. of the IEEE*, volume 77, pages 81–98, 1989.

[23] Software Engineering Institute. Software product lines. `http://www.sei.cmu.edu/productlines`.

[24] C. 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.

[25] 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.