

Conflict-Tolerant Switching Controllers

Deepak D'Souza
Madhu Gopinathan
Indian Institute of Science
Bangalore, India
{deepakd,gmadhu}@csa.iisc.ernet.in

S. Ramesh
Prahладavaradan Sampath
GM India Science Lab
Bangalore, India
{ramesh.s.p.sampath}@gm.com

ABSTRACT

This paper addresses the problem of detecting and resolving conflicts between switching controllers of hybrid systems. We consider systems composed of a base system with multiple switching controllers, each of which independently control the base system dynamics so as to conform to their individual specifications. We propose a methodology for developing such systems in a modular manner based on the notion of **conflict-tolerant** specifications that specify the future behaviour even when the specification has been violated in the past. Conflict-tolerant switching controllers must be designed to continue offering advice even when their advice has been overridden in the past. We show that such switching controllers can be composed using a simple priority based scheme which guarantees the **maximal** use of each switching controller. We also give an algorithm for verifying conflict-tolerant switching controllers when the specification, the base system and the controller are given as rectangular hybrid automata.

1. INTRODUCTION

The problem of engineering large software intensive systems is growing exponentially with the increasing sophistication of software. This has inspired a number of approaches for organizing software to improve the reliability of software systems. Many of these approaches propose a *feature oriented* development paradigm, where feature specifications are derived from domain requirements and features are implemented to satisfy such specifications. By suitably composing features, multiple software products can be engineered [19]. Historically, this approach has been followed in the telecommunications industry. In the automotive industry, advanced safety features such as electronic stability control, collision avoidance etc. [12] are developed as part of a software product line, and a subset of these features is integrated into different automotive products based on the needs of customers.

We view systems developed using this paradigm as consisting of a base system along with multiple *features* where each

feature advises the base system on how to conform to the feature specification. One of the problems faced in integrating various features in such systems is that the system may reach a point of “conflict” between two (or more) features, where the features do not agree on a common action for the base system to perform. This situation is an instance of what is referred to in the literature as the *feature interaction* problem [15, 13, 10].

Consider a development model where individual features are specified by original equipment manufacturers (OEMs) and implemented by third party vendors. The OEM must *verify* that feature implementations conform to their specifications. In addition, the OEM must *integrate* various features into a final product. Detecting and resolving conflicts between features during feature integration poses a significant challenge for OEMs. Conflict between two features can, of course, be resolved by respecifying or redesigning one of the features. However this is often not possible in practice and there is no guarantee that the redesigned feature does not conflict with some other feature. Moreover, redesign of a feature for handling specific conflicts reduces the scope for reusing the feature in multiple contexts.

An alternative to redesign is to *suspend* the feature with lower priority so that the base system can continue with the advice of the higher priority feature. However the issue now is how and when to *resume* the suspended feature so as to maximize its use.

Our Approach. In this paper we propose a formal framework for developing feature based systems in a way that overcomes some of the problems outlined above. We work in the setting of real-time features so as to model more closely the timed-dependent features that arise in the automotive domain. The framework is based on the novel notion of *conflict-tolerance*, which requires features to be *resilient* or *tolerant* with regard to violations of their advice. Thus, unlike the classical notion of a feature, a conflict-tolerant feature can observe that its advice has been over-ridden, takes into account the over-riding event, and proceeds to offer advice for *subsequent* behaviour of the system.

The starting point of this framework is the notion of a conflict-tolerant *specification* of a feature. A classical safety specification can be viewed as a prefix-closed language of finite words containing all the system behaviours which are considered *safe*. This can be pictured as a safety *cone* in the

tree representing all possible behaviours, as shown in Fig. 1 (a). A conflict-tolerant specification on the other hand can be viewed as an *advice function* that specifies for *each* behaviour w of the base system, a safety cone comprising all future behaviours that are considered safe, *after* the system has exhibited behaviour w (Fig. 1 (b)).



Figure 1: The conflict-tolerant specification on the right advises on how to extend w even though its advice has been overridden (dashed line) in the past when generating w .

To illustrate how a conflict-tolerant specification can capture a specifier’s intent more richly than a classical specification, consider a feature that is required to release (denoted by the event “**rel**”) a single unit of lubrication every 1 second. A classical specification for this feature may be given by the timed transition system shown in Fig. 2(a). The state invariant “ $x < 1$ ” is to be interpreted as a “time-can-progress” condition: thus as long as the value of the clock x is less than 1, the specification recommends letting time elapse. However when $x = 1$, time elapse is no more recommended and instead the action **rel** is recommended “urgently”. Fig. 2(b) and (c) show annotated timed transition systems denoting conflict-tolerant specifications that induce the same classical specification shown in (a). These transition systems differ from the classical one in two ways. The dashed transitions are to be read as “not-advised”, and they enable the specification to keep track of events that occur in violation of its advice at a given state. Secondly, in a state the time-can-progress condition can be violated to let time elapse against the advice of the specification. In this case time elapses but control remains in the same state. Thus specification (b), advises **rel** urgently at all times after 1 second from the previous release, until its advice is taken. When $x < 1$, its advice is only to let time elapse; If the event **rel** is performed against its advice, it uses the dashed transition to keep track of this and resets its clock x . If its advice is not followed when $x = 1$, it continues to advise that **rel** be done urgently, i.e. time elapse is no more recommended

The specification (c) keeps track of whether the previous release was “on time” or not and changes its advice accordingly. Thus it advises **rel** 0.5 seconds after the last release if the previous release was “late” (lightly shaded state) and advises **rel** 2 seconds after the last release if it was “early” (darkly shaded state).

A conflict-tolerant feature implementation can be viewed as a timed transition system with transitions annotated as *advised* and *not-advised*, similar to the conflict-tolerant specifications described above. A feature implementation is now said to satisfy a conflict-tolerant specification (with respect to a given base system), if after every possible behaviour w of the base system, the behaviours of the base system that are according to the advice of the feature implementation are

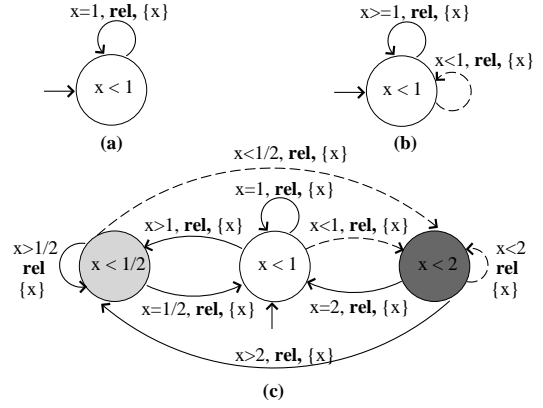


Figure 2: A classical spec (a), and conflict-tolerant specs (b) and (c) that induce the same classical spec (a).

contained in the safety cone specified by the specification for w . We address the natural feasibility and verification problems in this framework and give decision procedures to solve these problems in the setting of Alur-Dill timed transition systems.

An important aspect of our framework is the fact that conflict-tolerant features admit a simple and effective composition scheme based on a prioritization of the features being composed. The composition scheme can also be viewed as a conflict resolution technique. The composition scheme ensures that the resulting system always satisfies the specification of the highest priority feature. Additionally, it follows the advice of all other features F , *except* at points where each action in the advice of F conflicts with the advice of a higher priority feature. It is in this sense that each feature is *maximally* utilized.

In summary, the contributions of this paper are:

- Formulation of the novel notion of conflict-tolerance in the context of timed systems.
- Algorithms for (i) checking whether it is feasible to construct a conflict-tolerant feature for a base system and a conflict-tolerant specification, and (ii) for verifying whether a conflict-tolerant feature is valid for a base system and whether it satisfies a conflict-tolerant specification.
- Formulation of a *prioritization* scheme for composition of conflict tolerant features in a way that *maximizes* the use of each feature.

Related Work. In [11] 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 state machines and a conflict (in the untimed setting) is detected by checking whether a state in which the features advise conflicting system actions, is reached [15]. 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 discrete event controllers [18] follows that of [21, 5]. 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 [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 recent work [7] we have studied the notion of conflict-tolerance in an untimed setting, and exhibited a similar framework. The contribution of the present paper is the formulation of conflict-tolerance in the real-time setting, and the solutions to the feasibility and verification problems. The techniques we use are essentially along the lines of [2, 8].

In [6], an algorithm for checking compatibility between two components, i.e. whether they satisfy each other's assumptions with respect to timing constraints, is provided. Our work on the other hand is more a methodology for conflict-resolution and modular system design.

The rest of the paper is structured as follows: After preliminary definitions, in Section 3 we elaborate in a hybrid setting the view of features as controllers and illustrate conflict between features. We then introduce the notion of conflict-tolerance in Section 4. Finally in Section 5, we describe our composition scheme and provide a precise formulation of the claim that the controllers are maximally utilized.

2. PRELIMINARIES

Let X, Y be sets. Let R be a relation between X and Y , i.e. $R \subseteq X \times Y$. We define the domain of R , denoted $domain(R)$, to be $\{x \in X \mid \exists y \in Y \text{ and } (x, y) \in R\}$, and the range of R , denoted $range(R)$, to be $\{y \in Y \mid \exists x \in X \text{ and } (x, y) \in R\}$. Let Z be a subset of $domain(R)$. We define the restriction of R to the set Z , denoted $R \triangleright Z$, as

$$R \triangleright Z = \{(x, y) \in R \mid x \in Z\}.$$

Let $X \rightarrow Y$ denote the set of functions from X to Y . Given a function $f : X \rightarrow (Y \rightarrow Z)$ and a set W such that $W \subseteq Y$, we denote by $f \triangleright W$, the function $g : X \rightarrow (W \rightarrow Z)$ such that for all $x \in domain(g)$, $g(x) = f(x) \triangleright W$. Given a set $L \subseteq X \rightarrow (Y \rightarrow Z)$, by $L \triangleright W$ we mean $\{f \triangleright W \mid f \in L\}$. By $f \triangleright w$ we mean $f \triangleright \{w\}$.

Let W be a finite set of variables. We will assume in the sequel that for each variable w , there is a set $K_w \subseteq \mathbb{R}$, which denotes the set of values that w can take. We say that w is a

discrete variable if K_w is finite, otherwise it is a continuous variable. A *valuation* for a variable $w \in W$ is a function $\mathbf{w} : W \rightarrow \mathbb{R}$ such that $\mathbf{w}(w) \in K_w$. For a set of variables W , we denote by \mathbf{W} the set of all valuations for the variables in W .

Let \mathbb{R}^n , where $n \geq 1$, denote the set of all n -tuples of real numbers, i.e. $\{(x_1, \dots, x_n) \mid x_i \in \mathbb{R}\}$. For a set of variables $W = \{w_1, \dots, w_n\}$, we can represent a valuation \mathbf{w} as a vector $\vec{\mathbf{w}} \in \mathbb{R}^n$, defined as $\vec{\mathbf{w}} = (\mathbf{w}(w_1), \dots, \mathbf{w}(w_n))$ with the implicit ordering $w_1 < w_2 < \dots < w_n$. For a set of valuations \mathbf{W} , $\vec{\mathbf{W}}$ denotes the set of vectors associated with the valuations in \mathbf{W} .

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be a real-valued function. The function f has a *left limit* l at a point $p \in \mathbb{R}$, denoted $\lim_{x \rightarrow p^-} f(x) = l$, if for any real number $\epsilon > 0$, there exists a real number $\delta > 0$ such that for all $x \in \mathbb{R}$, if $p - \delta < x < p$, then we have $|f(x) - l| < \epsilon$. By $f(p^-)$, we mean the left limit of f at p . The function f has a *right limit* l at a point $p \in \mathbb{R}$, denoted $\lim_{x \rightarrow p^+} f(x) = l$, if for any real number $\epsilon > 0$, there exists a real number $\delta > 0$ such that for all $x \in \mathbb{R}$, if $p < x < p + \delta$, then we have $|f(x) - l| < \epsilon$. The function f has a *limit* l at a point $p \in \mathbb{R}$, denoted $\lim_{x \rightarrow p} f(x) = l$, if $\lim_{x \rightarrow p^-} f(x) = l$ and $\lim_{x \rightarrow p^+} f(x) = l$.

We say f is *left continuous* at a point $p \in \mathbb{R}$ if $\lim_{x \rightarrow p^-} f(x) = f(p)$ and is *right continuous* at p if $\lim_{x \rightarrow p^+} f(x) = f(p)$. We say f is *continuous* at p iff it is left continuous and right continuous at p , i.e. $\lim_{x \rightarrow p} f(x) = f(p)$.

We say f is *differentiable* at a point $p \in \mathbb{R}$ if the function h has a limit at p , where h is defined on $\mathbb{R} \setminus \{p\}$ as

$$h(x) = \frac{f(x) - f(p)}{x - p}.$$

The limit of h at p , i.e. $\lim_{x \rightarrow p} h(x)$, if it exists, is called the derivative of f at p and is denoted $\dot{f}(p)$.

Let $[a, b)$ be an interval that is a subset of \mathbb{R} . We say f is continuous in $[a, b)$ if f is continuous at every point in (a, b) and f is right continuous at a , i.e. $\lim_{x \rightarrow a^+} f(x) = f(a)$. We say f is differentiable in (a, b) if f is differentiable at every point in (a, b) .

A *vector-valued* function is of the form $f : \mathbb{R} \rightarrow \mathbb{R}^n$, i.e. for any point $x \in \mathbb{R}$, $f(x)$ is a vector in \mathbb{R}^n . Such a function f can be described in terms of n component functions $f_i : \mathbb{R} \rightarrow \mathbb{R}$ such that $f(x) = (f_1(x), \dots, f_n(x))$. The limit of f at a point $p \in \mathbb{R}$ is obtained by taking the limits of its component functions in the following sense:

$$\lim_{x \rightarrow p} f(x) = (\lim_{x \rightarrow p} f_1(x), \dots, \lim_{x \rightarrow p} f_n(x)).$$

The one sided limits can be obtained in a similar manner. The function f is continuous at p if $\lim_{x \rightarrow p} f(x) = f(p)$. If the component functions of f are differentiable, then the derivative of f can be obtained by differentiating the component functions as follows:

$$\dot{f}(p) = (\dot{f}_1(p), \dots, \dot{f}_n(p)).$$

Let $W = \{w_1, \dots, w_n\}$ be a set of variables. Let \mathbf{W} be

the set of all valuations for the variables in W . Given $\sigma : \mathbb{R}_{\geq 0} \rightarrow \mathbf{W}$, let $\sigma' : \mathbb{R}_{\geq 0} \rightarrow \mathbf{W}$ be a function such that for all $t \in \mathbb{R}_{\geq 0}$, $\sigma'(t) = \mathbf{w}$, where $\mathbf{w} = \sigma(t)$. We say σ is continuous (right continuous) at a point $p \in \mathbb{R}$ if σ' is continuous (right continuous) at p . We say σ is differentiable at p if σ' is differentiable at p . We extend these definitions to intervals $[a, b] \subseteq \mathbb{R}$ in the expected manner.

2.1 Signal

We will model the behaviour of a hybrid system using the notion of a signal as defined below.

DEFINITION 1 (SIGNAL). A signal over a set of variables W is a function $\sigma : I \rightarrow \mathbf{W}$ where

- the domain I is an interval $[0, r)$ for some $r \in \mathbb{R}_{\geq 0}$ and
- σ has only finitely many points of discontinuity. Thus, there is a strictly increasing sequence of time points $t_0 = 0 < t_1 < t_2 < \dots < t_n = r$ such that for every $k \in \{0, \dots, n-1\}$, σ is continuous in the interval $I_k = [t_k, t_{k+1})$.

The definition above is adapted from the definition of temporal behaviour in [4]. Note that given an interval $I_k = [t_k, t_{k+1})$ where $k \in \{0, \dots, n-1\}$, σ is right continuous at the points of discontinuity t_k .

We denote by \mathcal{W} the set of all signals over W . A signal language L over W is a subset of \mathcal{W} . Let $\varepsilon : [0, 0) \rightarrow \mathbf{W}$ denote the empty signal, i.e. the signal whose domain is an empty set. Let $\sigma_1 : [0, t_1) \rightarrow \mathbf{W}$ and $\sigma_2 : [0, t_2) \rightarrow \mathbf{W}$ 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 $\sigma : [0, t_1 + t_2) \rightarrow \mathbf{W}$ 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 $\sigma'_1 \in \mathcal{W}$ such that $\sigma_1 \cdot \sigma'_1 = \sigma_2$. 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.

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

2.2 Hybrid Automata

Our definition of a hybrid automaton is adapted from the definitions of hybrid automata in the literature [20, 16, 3, 4, 1, 17]. 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 2 (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. The finite set \mathbf{Q} is the set of modes of \mathcal{H} .
- V is a finite set of state variables (discrete or continuous). 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 \bar{C} denote the set of uncontrolled variables of \mathcal{H} , i.e. $\bar{C} = V \setminus C$.
- Let $\mathbf{D} = C \rightarrow 2^{\mathbb{R}}$ be the set of maps specifying a set of derivatives for variables in C . Then, $F : \mathbf{Q} \rightarrow (\mathbf{V} \rightarrow \mathbf{D})$ assigns to each mode a **flow condition** which constrains the time derivative of the continuous flow of the controlled variables. Thus, $\forall \mathbf{q} \in \mathbf{Q}, \forall \mathbf{v} \in \mathbf{V}$, if $\mathbf{d} = F(\mathbf{q})(\mathbf{v})$, then the time derivative of the signal of $c \in C$ must be in $\mathbf{d}(c)$. Alternatively, \mathbf{D} can be looked at as defining a set-valued vector field $\bar{\mathbf{D}}$ which prescribes the laws of continuous flow for the controlled variables. We assume that such a vector field is Lipschitz continuous (see Section ??). We also assume that for a discrete variable u , F is such that $\forall \mathbf{q} \in \mathbf{Q}, \forall \mathbf{v} \in \mathbf{V}, ((F(\mathbf{q})(\mathbf{v}))u) = \{0\}$, i.e. the signals of u are constant in every mode.
- $init \subseteq \mathbf{Q} \times \mathbf{C}$ specifies a set of initial configurations given by $\{(\mathbf{q}, \mathbf{v}) \mid (\mathbf{q}, \mathbf{v} \triangleright C) \in init\}$. Thus, $init$ does not constrain the initial values of the uncontrolled variables.
- $tcp : \mathbf{Q} \rightarrow 2^{\mathbf{V}}$ assigns to each mode a **time can progress condition**, i.e. if $\mathbf{v} \in tcp(\mathbf{q})$, then it is possible for the automaton \mathcal{H} to evolve continuously from the state \mathbf{v} .
- $\rightarrow \subseteq \mathbf{Q} \times 2^{\mathbf{V}} \times (\mathbf{V} \rightarrow 2^{\mathbf{C}}) \times \mathbf{Q}$ is a **jump relation**. Then for a jump $e = (\mathbf{q}, \mathbf{g}, reset, \mathbf{q}') \in \rightarrow$, \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 $reset : \mathbf{V} \rightarrow 2^{\mathbf{C}}$ 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 $[a, b)$ be an interval that is a subset of $\mathbb{R}_{\geq 0}$. Let $\tau : [a, b) \rightarrow \mathbf{V}$ be a function which is continuous in $[a, b)$ and differentiable in (a, b) . Let $(\mathbf{q}, \mathbf{v}), (\mathbf{q}, \mathbf{v}')$ be configurations of \mathcal{H} . We say that τ is a *witness* of \mathcal{H} evolving continuously from (\mathbf{q}, \mathbf{v}) to $(\mathbf{q}, \mathbf{v}')$ iff the following conditions are satisfied:

- $\tau(a) = \mathbf{v}$,

- $\tau(b^-) = \mathbf{v}'$,
- for all $t \in [a, b)$, $\tau(t) \in tcp_{\mathbf{q}}$ and
- τ can be decomposed into the functions $\tau \triangleright C : [a, b) \rightarrow \mathbf{C}$ and $\tau \triangleright \bar{C} : [a, b) \rightarrow \bar{\mathbf{C}}$ such that for all $t \in (a, b)$, $\tau(t) \triangleright C \in F_{\mathbf{q}}(\tau(t))$.

Let (\mathbf{q}, \mathbf{v}) be a configuration of \mathcal{H} . We say that *time can progress* from (\mathbf{q}, \mathbf{v}) to reach $(\mathbf{q}, \mathbf{v}')$ iff there exists a function $\tau : [a, b) \rightarrow \mathbf{V}$ which is a witness for a valid continuous evolution from (\mathbf{q}, \mathbf{v}) to $(\mathbf{q}, \mathbf{v}')$. By $(\mathbf{q}, \mathbf{v}) \xrightarrow{\tau} (\mathbf{q}, \mathbf{v}')$, we mean that τ is a witness of \mathcal{H} evolving continuously from (\mathbf{q}, \mathbf{v}) to $(\mathbf{q}, \mathbf{v}')$.

DEFINITION 3 (TRAJECTORY OF \mathcal{H}). Let $I = [0, r) \subseteq \mathbb{R}_{\geq 0}$ and $W = Q \cup V$. A **trajectory** of \mathcal{H} starting from a configuration (\mathbf{q}, \mathbf{v}) is a signal $\tau : I \rightarrow \mathbf{W}$ such that the component signals $\tau \triangleright Q : I \rightarrow \mathbf{Q}$ and $\tau \triangleright V : I \rightarrow \mathbf{V}$ satisfy the following conditions:

- Either $\tau \triangleright Q(0) = \mathbf{q}$, $\tau \triangleright V(0) = \mathbf{v}$ or $\tau \triangleright Q(0) = \mathbf{q}'$, $\tau \triangleright V(0) = \mathbf{v}'$ and there exists $(\mathbf{q}, \mathbf{g}, \text{reset}, \mathbf{q}') \in \rightarrow$ such that $\mathbf{v} \in \mathbf{g}$ and $\mathbf{v}' \triangleright C \in \text{reset}(\mathbf{v})$. Thus, τ 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 $n - 1$, in the interval $I_k = [t_k, t_{k+1})$, $\tau \triangleright Q$ is a constant function and $\tau \triangleright V$ is continuous. Let \mathbf{q}_k denote the value of $\tau \triangleright Q$ in the interval I_k . Let σ denote the restriction of τ to the interval $[t_k, t_{k+1})$, i.e. $\sigma = \tau \triangleright [t_k, t_{k+1})$. Then $(\mathbf{q}_k, \tau \triangleright V(t_k)) \xrightarrow{\sigma} (\mathbf{q}_k, \tau \triangleright V(t_{k+1}))$ (valid continuous evolution).
- $\tau \triangleright V$ can be further decomposed into the signals $\tau \triangleright C : I \rightarrow \mathbf{C}$ and $\tau \triangleright \bar{C} : I \rightarrow \bar{\mathbf{C}}$ such that for every k from 1 to $n - 1$
 - either there exists $(\mathbf{q}_{k-1}, \mathbf{g}, \text{reset}, \mathbf{q}_k) \in \rightarrow$ such that $\tau \triangleright V(t_k^-) \in \mathbf{g}$ and $\tau \triangleright C(t_k) \in \text{reset}(\tau \triangleright V(t_k^-))$ (i.e. \mathcal{H} jumps);
 - or $\mathbf{q}_{k-1} = \mathbf{q}_k$ and $\tau \triangleright C(t_k^-) = \tau \triangleright C(t_k)$ (\mathcal{H} does not jump).

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

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

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

$$L_{(\mathbf{q}, \mathbf{v})}(\mathcal{H}) = \{\tau \triangleright V \mid \tau \in \text{Traj}_{(\mathbf{q}, \mathbf{v})}(\mathcal{H})\}.$$

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

$$L(\mathcal{H}) = \{\tau \triangleright V \mid \tau \in \text{Traj}(\mathcal{H})\}.$$

We say a hybrid automaton \mathcal{H} is *deterministic* if for any signal $\sigma \in L(\mathcal{H})$, there is exactly one trajectory $\tau \in \text{Traj}(\mathcal{H})$ such that $\tau \triangleright V = \sigma$.

Let $\sigma : [0, r) \rightarrow \mathbf{X}$ be a signal over X where $X \subseteq V$. We define the *configurations* of \mathcal{H} after σ , denoted $\text{config}_{\mathcal{H}}(\sigma)$, to be

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

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

DEFINITION 4 (PRODUCT). Let $\mathcal{H}_1 = (Q_1, V, C_1, F_1, \text{init}_1, tcp_1, \rightarrow_1)$ and $\mathcal{H}_2 = (Q_2, V, C_2, F_2, \text{init}_2, tcp_2, \rightarrow_2)$ be two hybrid automata over the same set of variables V . Then the *synchronized product* of \mathcal{H}_1 and \mathcal{H}_2 , denoted $\mathcal{H}_1 \parallel \mathcal{H}_2$, is given by the hybrid automaton $\mathcal{H} = (Q, V, C, F, \text{init}, tcp, \rightarrow)$ 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 $\forall \mathbf{q}_1, \mathbf{q}_2 \in \mathbf{Q}, \forall v \in \mathbf{V}, \forall \mathbf{d} \in \mathbf{D}$, $\mathbf{d} \in F(\mathbf{q}_1, \mathbf{q}_2)(v)$ iff $\mathbf{d} \triangleright C_1 \in F_1(\mathbf{q}_1)(v)$ and $\mathbf{d} \triangleright C_2 \in F_2(\mathbf{q}_2)(v)$,
- $\text{init} \subseteq \mathbf{Q} \times \mathbf{C}$ where $((\mathbf{q}_1, \mathbf{q}_2), \mathbf{c}) \in \text{init}$ iff $(\mathbf{q}_1, \mathbf{c} \triangleright C_1) \in \text{init}_1$ and $(\mathbf{q}_2, \mathbf{c} \triangleright C_2) \in \text{init}_2$,
- $tcp : \mathbf{Q} \rightarrow 2^{\mathbf{V}}$ where $tcp((\mathbf{q}_1, \mathbf{q}_2)) = tcp_1(\mathbf{q}_1) \cap tcp_2(\mathbf{q}_2)$,
- $((\mathbf{q}_1, \mathbf{q}_2), \mathbf{g}, \text{reset}, (\mathbf{q}'_1, \mathbf{q}'_2)) \in \rightarrow$ if one of the conditions below is satisfied:
 - $\mathbf{q}_1 = \mathbf{q}'_1, (\mathbf{q}_2, \mathbf{g}_2, \text{reset}_2, \mathbf{q}'_2) \in \rightarrow_2, \mathbf{g} = \mathbf{g}_2$ and $\mathbf{c} \in \text{reset}(\mathbf{v})$ iff $\mathbf{c} \triangleright C_2 \in \text{reset}_2(\mathbf{v})$ and $\mathbf{c} \triangleright C_1 = \mathbf{v} \triangleright C_1$.
 - $(\mathbf{q}_1, \mathbf{g}_1, \text{reset}_1, \mathbf{q}'_1) \in \rightarrow_1, \mathbf{g} = \mathbf{g}_1$ and $\mathbf{c} \in \text{reset}(\mathbf{v})$ iff $\mathbf{c} \triangleright C_1 \in \text{reset}_1(\mathbf{v})$ and $\mathbf{c} \triangleright C_2 = \mathbf{v} \triangleright C_2, \mathbf{q}_2 = \mathbf{q}'_2$.
 - $(\mathbf{q}_1, \mathbf{g}_1, \text{reset}_1, \mathbf{q}'_1) \in \rightarrow_1, (\mathbf{q}_2, \mathbf{g}_2, \text{reset}_2, \mathbf{q}'_2) \in \rightarrow_2, \mathbf{g} = \mathbf{g}_1 \cap \mathbf{g}_2$ and $\mathbf{c} \in \text{reset}(\mathbf{v})$ iff $\mathbf{c} \triangleright C_1 \in \text{reset}_1(\mathbf{v})$ and $\mathbf{c} \triangleright C_2 \in \text{reset}_2(\mathbf{v})$.

3. CONVENTIONAL SETTING FOR SWITCHING CONTROL

Let X, U, Y be three disjoint sets of variables such that U is a set of discrete variables. The variables in X are controlled by the base system and the variables in U are the discrete input variables using which a controller controls the base system. Let $W = X \cup Y$. A *safety specification* over (X, U, Y) is a prefix-closed signal language over W which can also be viewed as an “advice function” as defined below.

DEFINITION 5 (ADVICE FUNCTION). An *advice function* over (X, U, Y) is a function $f : \mathcal{W} \rightarrow 2^{\mathcal{W}}$ which satisfies the following conditions:

- $f(\varepsilon)$ is a prefix-closed set of signals. Recall that ε is the empty signal (see Section ??).

- f is consistent in the sense that for all $\sigma \in f(\varepsilon)$ and all $\tau \in f(\sigma)$, we have $f(\sigma \cdot \tau) = \text{ext}_\tau(f(\sigma))$.

A safety specification L over (X, U, Y) induces an advice function f_L given by $f_L(\sigma) = \text{ext}_\sigma(L)$. Conversely, an advice function f induces a safety specification L_f given by $L_f = f(\varepsilon)$.

A deterministic hybrid automaton \mathcal{H} over W (which controls the variables in W) can be used to give a safety specification $L(\mathcal{H})$ over (X, U, Y) . The induced advice function $f_{\mathcal{H}}$ is given by $f_{\mathcal{H}}(\sigma) = L_\sigma(\mathcal{H})$ if $\sigma \in L(\mathcal{H})$, and \emptyset otherwise.

For simplicity, we assume that there is only one discrete variable u in U . Let $V = X \cup \{u\}$. Recall that X is the set of variables controlled by the base system. Let \mathcal{H} be a hybrid automaton over V . Let $\sigma \in L(\mathcal{H}) \triangleright X$. We define u -set after σ of \mathcal{H} , denoted $\mathcal{H}(\sigma)$, as

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

We now define the notion of a base system.

DEFINITION 6 (BASE SYSTEM). A base system (or plant) over (X, U, Y) is a deterministic hybrid automaton \mathcal{B} over V , which controls the variables in X . We assume that the base system \mathcal{B} is **non-blocking** in that if $\sigma \in L(\mathcal{B})$, then $\mathcal{B}(\sigma) \neq \emptyset$.

As a running example, we consider a water tank equipped with a pump. The hybrid automaton for the water tank (the base system) is shown in Figure 3. The controlled variables of \mathcal{B} , 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.

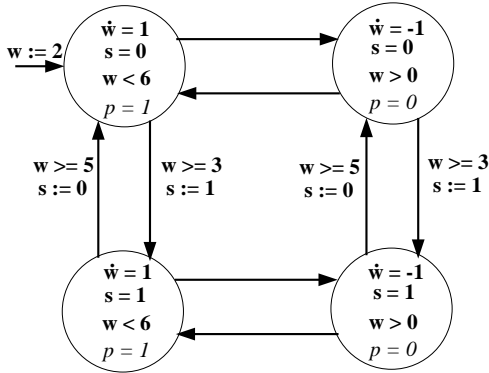


Figure 3: Water Tank (Base System) \mathcal{B} .

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 be assigned a value 1 only when $\mathbf{w} \geq 3$ and can be assigned 0 again only when $\mathbf{w} \geq 5$. Note that the water level \mathbf{w} will always be between 0 and 6 cm.

Let \mathcal{B} be a base system over (X, U, Y) . We define a switching controller for \mathcal{B} as follows.

DEFINITION 7 (SWITCHING CONTROLLER). A **switching controller** for \mathcal{B} is a hybrid automaton $\mathcal{C} = (Q, V, \{u\}, F, \text{init}, \text{tcp}, \rightarrow)$ over V , which controls the variable u . The controller \mathcal{C} is valid with respect to \mathcal{B} if \mathcal{C} is **strongly non-blocking** with respect to \mathcal{B} , i.e. if $\sigma \in L(\mathcal{B} \parallel \mathcal{C})$, then $\mathcal{C}(\sigma) \subseteq \mathcal{B}(\sigma)$ and $\mathcal{C}(\sigma) \neq \emptyset$.

Let \mathcal{B} be a base system over (X, U, Y) and let \mathcal{C} be a controller for \mathcal{B} . Let $\sigma : [0, r) \rightarrow \mathbf{V} \in L(\mathcal{B})$. We say that σ is according to the advice of \mathcal{C} at time t ($0 \leq t < r$), if $\sigma = \sigma_1 \cdot \sigma_2$, where σ_1, σ_2 are signals of the form $\sigma_1 : [0, t) \rightarrow \mathbf{V}, \sigma_2 : [0, r - t) \rightarrow \mathbf{V}$ respectively, 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 such that $0 \leq t < r$.

Let \mathcal{S} be a safety specification over (X, U, Y) . We say a switching controller \mathcal{C} for the base system \mathcal{B} satisfies \mathcal{S} if $L(\mathcal{B} \parallel \mathcal{C}) \triangleright X \subseteq L(\mathcal{S}) \triangleright X$.

As an example, consider a feature which requires that the water level must always be between 2 cm and 4 cm. Figure 4(a) shows a specification \mathcal{S}_1 and Figure 4(b) shows a controller \mathcal{C}_1 for this requirement. Note that \mathcal{C}_1 controls only the pump. 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.

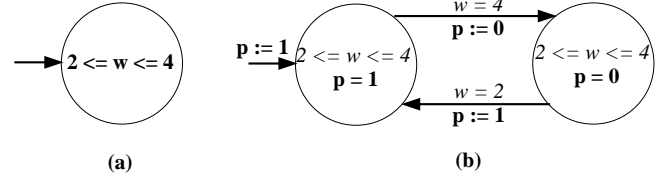


Figure 4: (a) Specification \mathcal{S}_1 and (b) controller \mathcal{C}_1 .

Figure 5 shows a behaviour of \mathcal{B} that is according to the advice of \mathcal{C}_1 . The signal $\sigma \triangleright w$ is shown with a solid line and the signal $\sigma \triangleright p$ is shown dashed.

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

Now consider another feature which requires that if the sensor \mathbf{s} is set to 1, then within 2 seconds, the water level must be kept greater than or equal to 5 cm. Figure 6 shows a specification \mathcal{S}_2 for this requirement. This specification introduces a clock variable \mathbf{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 \mathbf{s} to 1.

Figure 7 shows a controller \mathcal{C}_2 for satisfying specification \mathcal{S}_2 . If the sensor \mathbf{s} is set to 1, then the controller \mathcal{C}_2 controls the

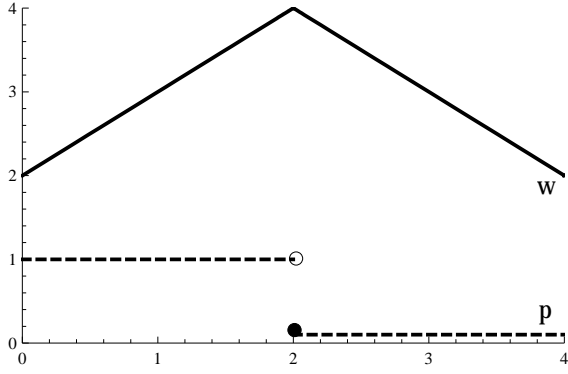


Figure 5: Signal σ is a behaviour of \mathcal{B} that is according to the advice of \mathcal{C}_1 .

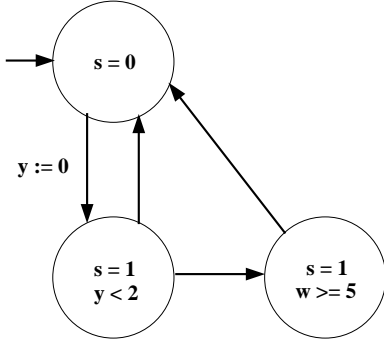


Figure 6: Specification \mathcal{S}_2 .

pump as follows:

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

Figure 8 illustrates whether water level will rise or fall given a mode of controller \mathcal{C}_2 . For example, if the controller \mathcal{C}_2 is in mode q_1 and the water level is between 0 and 6, then the water level can either rise or fall depending on whether the pump is on or off respectively. The water level w must rise in mode q_3 , and must fall in mode q_5 .

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 3). 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 8 (CONFLICT). Let \mathcal{C}_1 and \mathcal{C}_2 be valid controllers for a base system \mathcal{B} . The controllers \mathcal{C}_1 and \mathcal{C}_2 are in conflict with respect to \mathcal{B} , if there exists a behaviour

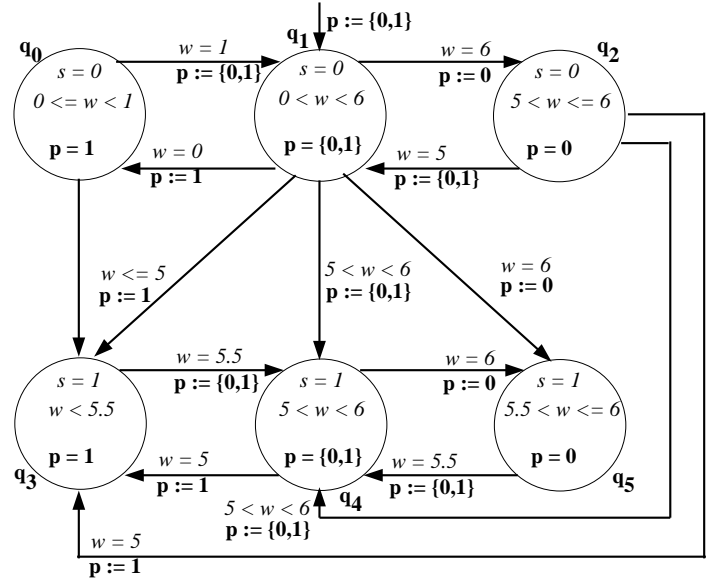


Figure 7: Controller \mathcal{C}_2 .

τ in $L(\mathcal{B}||\mathcal{C}_1||\mathcal{C}_2)$ such that $L_\tau(\mathcal{B}||\mathcal{C}_1||\mathcal{C}_2) \neq \{\varepsilon\}$. In other words, there exists a behaviour $\sigma \in L(\mathcal{B}||\mathcal{C}_1||\mathcal{C}_2) \supseteq X$ 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 base system behaviour shown in Figure 9. The sensor s is set to 1 at time 1.5. This requires \mathcal{C}_2 to keep the pump on until the 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{B} were to follow the advice of \mathcal{C}_1 , then water level will start falling at time 2. However, if \mathcal{B} were to follow the advice of \mathcal{C}_2 , then water level will keep rising after 2 seconds (shown dashed).

4. CONFLICT-TOLERANT SWITCHING CONTROLLERS

In this section we introduce our notion of conflict-tolerance in a hybrid systems setting. Analogous to the notion of a specification as an advice function given in Section 3, a *conflict-tolerant* safety specification over a set of variables W is a *conflict-tolerant* advice function as defined below:

DEFINITION 9. A *conflict-tolerant* advice function over a set of variables W is a function $f : \mathcal{W} \rightarrow 2^{\mathcal{W}}$ which satisfies the following conditions:

- for **every** signal $\sigma \in \mathcal{W}$, $f(\sigma)$ is a prefix-closed set of signals.
- f is consistent in the sense that for **all** $\sigma \in \mathcal{W}$ with $\tau \in f(\sigma)$, we have $f(\sigma \cdot \tau) = \text{ext}_\tau(f(\sigma))$.

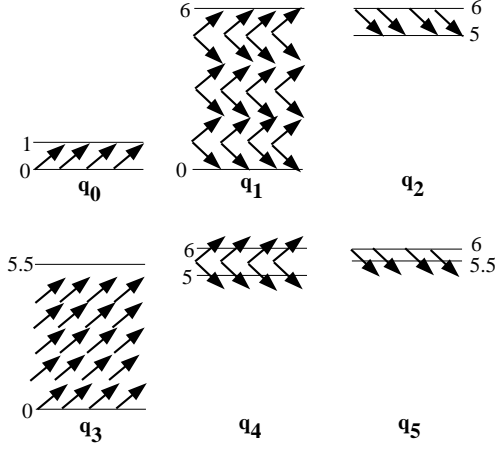


Figure 8: Advice in various modes of controller C_2 .

Let X, U, Y be three disjoint sets of variables. Let $W = X \cup Y$.

DEFINITION 10. A *conflict-tolerant specification over (X, U, Y) using hybrid automata* 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 $\sigma \in L(Acc) \triangleright X$, there is a unique trajectory τ of Acc such that $\tau \triangleright X = \sigma$.
- 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 $m : (\mathbf{P} \times \mathbf{W}) \rightarrow 2^{(\mathbf{Q} \times \mathbf{W})}$ be the map induced by the advice relation E such that $(\mathbf{q}, \mathbf{w}') \in m((\mathbf{p}, \mathbf{w}))$ iff $e = (\mathbf{p}, \mathbf{g}, reset, \mathbf{q}) \in E, \mathbf{w} \in \mathbf{g}$ and $\mathbf{w}' = reset(\mathbf{w})$. For all reachable configurations (\mathbf{p}, \mathbf{w}) of Acc , the map m must be such that $|m((\mathbf{p}, \mathbf{w}))| = 1$ and if $m((\mathbf{p}, \mathbf{w})) = (\mathbf{q}, \mathbf{w}')$, 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*

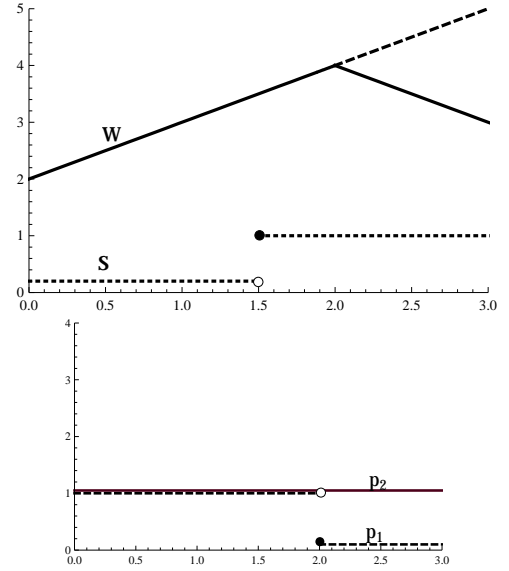


Figure 9: Conflicting advice from C_1 (p_1 shown dashed) and C_2 (p_2).

signal language of \mathcal{S}' starting from a configuration (\mathbf{p}, \mathbf{w}) , denoted $L_{(\mathbf{p}, \mathbf{w})}^c(\mathcal{S}')$, is defined to be $L_{(\mathbf{q}, \mathbf{w}')}(\mathcal{S}')$ where $(\mathbf{q}, \mathbf{w}') = m((\mathbf{p}, \mathbf{w}))$. Recall that m is the map induced by the advice relation E .

Let σ be a signal in $L(\mathcal{S}') \triangleright X$. 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 $L_\sigma^c(\mathcal{S}')$, we mean $L_{(\mathbf{p}, \mathbf{w})}^c(\mathcal{S}')$.

Let $L \subseteq \mathcal{X}$ be a prefix-closed signal language over X . We say that the tolerant specification \mathcal{S}' is *complete* with respect to L if $L \subseteq L(\mathcal{S}') \triangleright X$. If \mathcal{S}' is complete with respect to L , then for every $\sigma \in L$, there is a unique trajectory τ of the acceptor automaton Acc such that $\tau \triangleright X = \sigma$. A conflict-tolerant specification \mathcal{S}' that is complete with respect to a signal language L over X induces a function $f_{\mathcal{S}'}$ given by for all $\sigma \in L$, $f_{\mathcal{S}'}(\sigma) = L_\sigma^c(\mathcal{S}')$, and \emptyset otherwise. We say that the tolerant specification \mathcal{S}' is *advice consistent* with respect to L iff the induced advice function $f_{\mathcal{S}'}$ is consistent. From now on, we consider only tolerant specifications that are advice consistent.

Figure 10 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 4 (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 11 shows a tolerant specification for feature 2. Note that the classical specification shown in Figure 6 does not specify the required behaviour when 2 seconds have elapsed

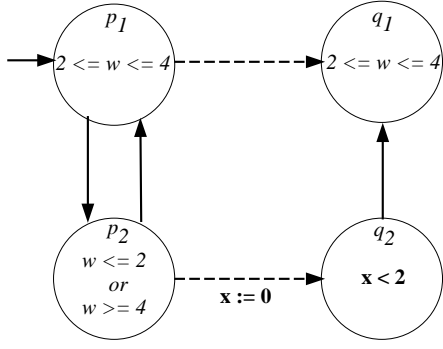


Figure 10: Tolerant specification S'_1 .

after the sensor s is set to 1 and the water level is still below 5 cm. In contrast, the tolerant specification 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).

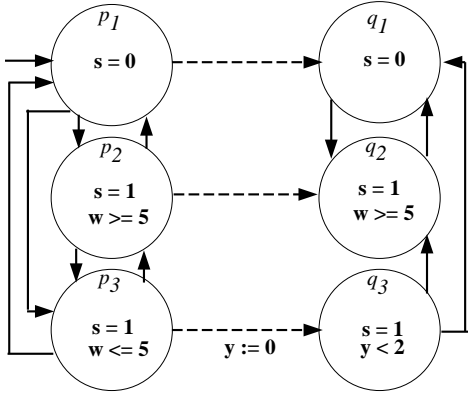


Figure 11: Tolerant specification S'_2 .

We now define the notion of a tolerant switching controller which is a switching controller with certain properties as given below. Let \mathcal{B} be a base system over (X, U, Y) . Let $\mathcal{C} = (Q, V, U, F, \text{init}, \text{tcp}, \rightarrow)$ be a switching controller for \mathcal{B} . We say that \mathcal{C} is *complete* with respect to $L(\mathcal{B})$ if for all $\sigma \in L(\mathcal{B})$, there exists a trajectory $\tau \in \text{Traj}(\mathcal{C})$ such that $\tau \triangleright X = \sigma \triangleright X$. We say that \mathcal{C} is *mode deterministic* with respect to $L(\mathcal{B})$ if for all $\sigma \in L(\mathcal{B})$ and for all trajectories $\tau, \tau' \in \text{Traj}(\mathcal{C})$ such that $\tau \triangleright X = \sigma \triangleright X$ and $\tau' \triangleright X = \sigma \triangleright X$, the mode signals are equal, i.e. $\tau \triangleright Q = \tau' \triangleright Q$.

Let the switching controller \mathcal{C} be complete and mode deterministic with respect to $L(\mathcal{B})$. Let $\sigma \in L(\mathcal{B})$ and let $\text{config}_{\mathcal{C}}(\sigma \triangleright X)$, i.e. the possible configurations of \mathcal{C} after σ be $\{(\mathbf{q}, \mathbf{v}_1), (\mathbf{q}, \mathbf{v}_2), \dots, (\mathbf{q}, \mathbf{v}_n)\}$, where $n \geq 1$. We say that \mathcal{C} is *u-switch permissive* iff whenever $v_i \in \text{tcp}(\mathbf{q})$, $\mathcal{C}(\sigma \triangleright X) = \text{tcp}(\mathbf{q}) \triangleright \{u\}$, 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 12 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 $0 < w < 6$.

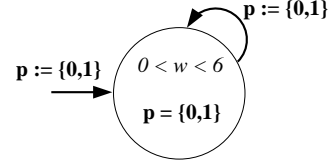


Figure 12: A controller that is u -switch permissive.

Let $\sigma \in L(\mathcal{B})$. Let $\text{config}_{\mathcal{B}}(\sigma)$, i.e. the configuration of \mathcal{B} after σ be $\{(\mathbf{b}, \mathbf{v})\}$. Let $\text{config}_{\mathcal{C}}(\sigma \triangleright X)$, i.e. the possible configurations of \mathcal{C} after σ be $\{(\mathbf{q}, \mathbf{v}_1), (\mathbf{q}, \mathbf{v}_2), \dots, (\mathbf{q}, \mathbf{v}_n)\}$, 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 $L_{(\mathbf{b}, \mathbf{q}, \mathbf{v}_i)}(\mathcal{B} \parallel \mathcal{C})$, i.e. the state of the variables controlled by \mathcal{B} remain the same, but u is assigned the value $\mathbf{v}_i \triangleright u$. The controlled behaviour after σ , denoted $L_{\sigma}(\mathcal{B} \parallel \mathcal{C})$, is defined as $\bigcup_{i=1..n} L_{(\mathbf{b}, \mathbf{q}, \mathbf{v}_i)}(\mathcal{B} \parallel \mathcal{C})$.

DEFINITION 11. Let \mathcal{B} be a base system over (X, U, Y) . A *conflict-tolerant switching controller \mathcal{C}'* for \mathcal{B} is a switching controller for \mathcal{B} that is complete, mode deterministic and u -switch permissive with respect to $L(\mathcal{B})$. The switching controller \mathcal{C}' is valid with respect to \mathcal{B} if \mathcal{C}' is **strongly non-blocking** with respect to \mathcal{B} , i.e. if $\sigma \in L(\mathcal{B})$, then $\mathcal{C}'(\sigma) \subseteq \mathcal{B}(\sigma)$ 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 base system. Note that such a behaviour σ may not be according to the advice of \mathcal{C}' , in that $\sigma \notin L(\mathcal{B} \parallel \mathcal{C}')$. In contrast, a classical controller \mathcal{C} (see Definition 7) assumes that its advice is always followed by the base system, and hence it needs to advise extensions of only controlled behaviours (i.e. those in $L(\mathcal{B} \parallel \mathcal{C})$).

DEFINITION 12 (\mathcal{C}' SATISFIES S'). Let \mathcal{B} be a base system over (X, U, Y) and let S' be a conflict-tolerant specification over (X, U, Y) . A conflict-tolerant switching controller \mathcal{C}' for \mathcal{B} satisfies S' if for each $\sigma \in L(\mathcal{B}) \triangleright X$, $L_{\sigma}(\mathcal{B} \parallel \mathcal{C}') \triangleright X \subseteq L_{\sigma}^c(S') \triangleright X$. Thus after **any** base system behaviour σ over X , if the base system follows the advice of \mathcal{C}' , then the resulting behaviour over X conforms to the safety language over X prescribed by S' after observing σ .

Figure 13 shows a tolerant controller \mathcal{C}'_1 that satisfies the tolerant specification S'_1 of Figure 10. 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 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.

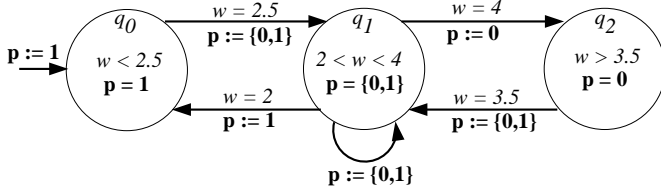


Figure 13: Tolerant controller C'_1 .

Figure 14 shows a tolerant controller C'_2 that satisfies the tolerant specification S'_2 of Figure 11. Note that this is the same as the classical controller in Figure 7 except the transition from mode q_2 to q_5 and the self loops in mode q_1 and q_4 .

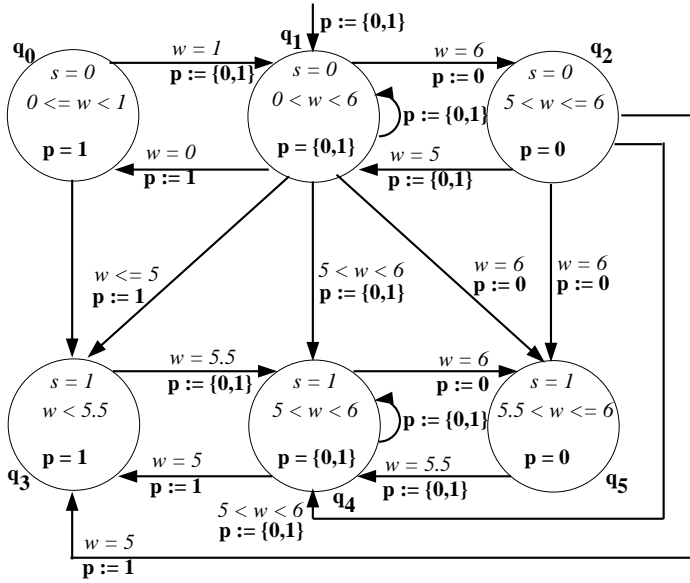


Figure 14: Tolerant controller C'_2 .

5. PRIORITIZED COMPOSITION

We now show how to compose conflict-tolerant switching controllers to obtain a classical switching controller. The prioritized composition that we define below guarantees that the advice of each tolerant switching controller is used whenever possible.

For the rest of this section, let us fix \mathcal{B} to be a base system over (X, U, Y) . Let C'_1 and C'_2 be valid conflict-tolerant controllers for \mathcal{B} . Let P be a priority ordering between C'_1 and C'_2 , and say P assigns a higher priority to C'_1 . We denote this by $C'_1 >_P C'_2$. Our objective is to obtain a switching controller \mathcal{C} by composing C'_1 and C'_2 using the priority order P . Figure 15 shows that both C'_1 and C'_2 observe the base system and offer their advice which is used by the switching controller \mathcal{C} to control the base system.

We rename the variable u in C'_1 to u_1 and get C''_1 . Similarly, we rename the variable u in C'_2 to u_2 and get C''_2 . We extend

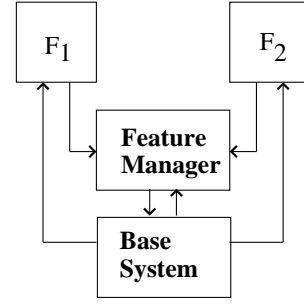


Figure 15: Switching Controller \mathcal{C} maximally utilizes the advice of C'_1 and C'_2 .

the set U such that $U' = \{u, u_1, u_2\}$. Let \mathcal{C}_{12} be a hybrid automaton over V' (where $V' = X \cup U'$) such that $\mathcal{C}_{12} = (Q, V', U', F, \text{init}_{12}, \text{tcp}_{12}, \rightarrow_{12})$ is the product $C'_1 \parallel C'_2$.

For a predicate ϕ over the set of variables U' (for example, $u = u_1 \wedge u = u_2$), we write $\llbracket \phi \rrbracket$ to denote the set of valuations \mathbf{v} that satisfy ϕ .

DEFINITION 13 (PRIORITIZED COMPOSITION). *The P -prioritized composition of the controllers C'_1 and C'_2 with respect to the base system \mathcal{B} , denoted $\llbracket_{P, \mathcal{B}}(C'_1, C'_2) \rrbracket$, is defined to be the hybrid automaton $\mathcal{C} = (Q, V', U', F, \text{init}, \text{tcp}, \rightarrow)$ obtained from \mathcal{C}_{12} after making changes according to the following rules:*

1. Let $I = \text{init}_{12} \cap \llbracket u = u_1 \wedge u = u_2 \rrbracket$. Then

$$\text{init} := \begin{cases} I & \text{if } I(\mathbf{q}) \neq \emptyset \\ \text{init}_{12} \cap \llbracket u = u_1 \rrbracket & \text{otherwise.} \end{cases}$$

2. For all $\mathbf{q} \in Q$, let $T(\mathbf{q}) = \text{tcp}_{12}(\mathbf{q}) \cap \llbracket u = u_1 \wedge u = u_2 \rrbracket$. Then

$$\text{tcp}(\mathbf{q}) := \begin{cases} T(\mathbf{q}) & \text{if } T(\mathbf{q}) \neq \emptyset \\ \text{tcp}_{12}(\mathbf{q}) \cap \llbracket u = u_1 \rrbracket & \text{otherwise.} \end{cases}$$

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

3. For each $e = (\mathbf{q}, \mathbf{g}, \text{reset}, \mathbf{q}') \in \rightarrow_{12}$, we add a transition $e' = (\mathbf{q}, \mathbf{g}', \text{reset}', \mathbf{q}')$ 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 $\text{tcp}_{u_2}(\mathbf{q}')$, 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 $\text{reset}' = \lambda v \cdot (\text{reset}(v) \cap \llbracket u = u_1 \rrbracket)$.

LEMMA 1. *Let \mathcal{B} be a base system over (X, U, Y) . For each $\sigma \in L(\mathcal{B} \parallel \mathcal{C}) \triangleright X$, the u -set of the switching controller \mathcal{C} , $\mathcal{C}(\sigma)$ is a subset of $\mathcal{B}(\sigma)$ and $\mathcal{C}(\sigma) \neq \emptyset$. Furthermore,*

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

Transition Type	Guard	Reset
Joint C'_1, C'_2	\mathbf{g}	$\lambda v \cdot (\text{reset}(v) \cap \text{tcp}_{U'}(\mathbf{q}') \cap \llbracket u = u_1 \rrbracket)$
Only C'_1	$\mathbf{g} \cap \llbracket \mathbf{u}_2 \in \text{tcp}_{u_2}(\mathbf{q}') \rrbracket$	$\lambda v \cdot (\text{reset}(v) \cap \text{tcp}_{U'}(\mathbf{q}') \cap \llbracket u = u_1 \rrbracket)$
Only C'_2	$\mathbf{g} \cap \llbracket \mathbf{u}_1 \in \text{tcp}_{u_1}(\mathbf{q}') \rrbracket$	$\lambda v \cdot (\text{reset}(v) \cap \text{tcp}_{U'}(\mathbf{q}') \cap \llbracket u = u_1 \rrbracket)$

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

PROOF. Let $\sigma \in L(\mathcal{B} \parallel \mathcal{C}) \triangleright X$. Let t_1, \dots, 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 C'_1 and C'_2 have a run on σ . Furthermore, the trajectories of C'_1 and C'_2 agree with that of \mathcal{C} on the mode signal. Also, the union of the time points at which C'_1 and C'_2 jump is exactly $\{t_1, \dots, t_n\}$.

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}, C'_1 and 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. \square

THEOREM 1. *Let S'_1 and S'_2 be conflict-tolerant specifications. Let \mathcal{B} be a base system. Let C'_1, C'_2 be valid conflict-tolerant controllers for \mathcal{B} that satisfy S'_1, S'_2 respectively. Let P be a priority order such that $C'_1 >_P C'_2$ (without loss of generality). Then the switching controller $\mathcal{C} = \parallel_{P, \mathcal{B}}(C'_1, C'_2)$ is a valid switching controller for \mathcal{B} . Furthermore, \mathcal{C} satisfies the specifications S'_1 and S'_2 , in the following “maximal” sense. For every $\tau : [0, \tau) \rightarrow \mathbf{V} \in L(\mathcal{B} \parallel \mathcal{C})$:*

1. $\tau \triangleright X \in L_\varepsilon^c(S'_1) \triangleright X$, i.e. the switching controller \mathcal{C} always satisfies the specification S'_1 .
2. For all prefixes $\sigma : [0, s) \rightarrow \mathbf{V}$ of τ , if $\mathcal{C}(\sigma \triangleright X) \not\subseteq C'_2(\sigma \triangleright X)$, then $C'_1(\sigma \triangleright X) \cap C'_2(\sigma \triangleright X) = \emptyset$, i.e. if the base system \mathcal{B} is switched to a mode that is not advised by C'_2 , then C'_2 is in conflict with the advice of the higher priority controller C'_1 . We can cover τ by a sequence of adjacent non-empty intervals such that in every interval, either τ is according to the advice of C'_2 or τ is not according to the advice of C'_2 .

PROOF. Note that C'_1 and C'_2 are both valid switching controllers for \mathcal{B} . Therefore, by Lemma 1, it is clear that \mathcal{C} is also a valid switching controller. It is always the case that $\mathcal{C}(\sigma) \subseteq C'_1(\sigma)$. If $\mathcal{C}(\sigma) \not\subseteq C'_2(\sigma)$, then by Lemma 1, C'_1 and C'_2 are in conflict. \square

Figure 16 illustrates this result. We can partition a base system behaviour τ into intervals in which the advice of C'_2 is not followed (shaded) and where it is followed (not shaded).

Consider the base system behaviour which we used to illustrate conflict in Section 3. The switching controller C_1

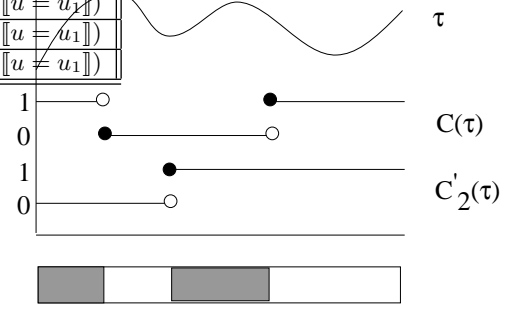


Figure 16: Partitioning of a base system behaviour τ into intervals in which the advice of C'_2 is not followed (shaded) and where it is followed (not shaded).

required the pump to be switched off after 2 seconds as the water level has reached 4 cm. However, the switching controller 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 C'_1 (Figure 13) and C'_2 (Figure 14) with the priority order $C'_2 >_P C'_1$. Then the conflict is resolved in favour of the switching controller C'_2 as shown in Figure 17. The water level continues to rise till 3.5 seconds following the advice of C'_2 . When the water level reaches 5.5 cm, the advice of 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 C'_2 and C'_1 .

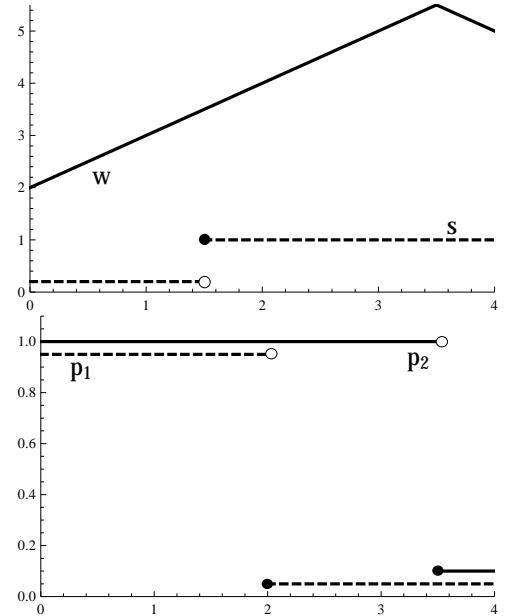


Figure 17: Conflict resolved in favour of C'_2 .

6. VERIFICATION

7. CONCLUSION

8. 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] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli. Effective synthesis of switching controllers for linear systems. *Proceedings 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] L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Timed interfaces. In *EMSOFT*, pages 108–122, 2002.
- [7] D. D’Souza and M. Gopinathan. Conflict-tolerant features. In A. Gupta and S. Malik, editors, *Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 227–239. Springer, 2008.
- [8] D. D’Souza and P. Madhusudan. Timed control synthesis for external specifications. In *STACS*, pages 571–582, 2002.
- [9] A. P. Felty and K. S. Namjoshi. Feature specification and automated conflict detection. *ACM Trans. Softw. Eng. Methodol.*, 12(1):3–27, 2003.
- [10] S. Ferber, J. Haag, and J. Savolainen. Feature interaction and dependencies: Modeling features for reengineering a legacy product line. In *SPLC*, pages 235–256, 2002.
- [11] K. Fisler and S. Krishnamurthi. Decomposing verification by features. *IFIP Working Conference on Verified Software: Theories, Tools, Experiments*, 2006.
- [12] Safety features for the future – <http://www.forbesautos.com/advice/toptens/future-safety-features/lander.html>.
- [13] R. J. Hall. Feature interactions in electronic mail. In *FIW*, pages 67–82, 2000.
- [14] J. D. Hay and J. M. Atlee. Composing features and resolving interactions. In *SIGSOFT Found. of Softw. Engg.*, pages 110–119, 2000.
- [15] 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.
- [16] Lecture notes on hybrid systems – john lygeros.
- [17] N. A. Lynch, R. Segala, and F. W. Vaandrager. Hybrid i/o automata. *Inf. Comput.*, 185(1):105–157, 2003.
- [18] 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.
- [19] C. M. Software Engineering Institute. Software product lines. <http://www.sei.cmu.edu/productlines>.
- [20] C. Tomlin, J. Lygeros, and S. Shankar Sastry. A game theoretic approach to controller design for hybrid systems. *Proceedings of the IEEE*, 88(7):949–970, Jul 2000.
- [21] 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.