Conflict-Tolerant Real-Time Features

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

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

Abstract

This paper addresses the problem of detecting and resolving conflicts due to timing constraints imposed by features in real-time systems. We consider systems composed of a base system with multiple ***features*** *or* ***controllers****, each of which independently advise the system on how to react to input events 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*** *features that are designed to continue offering advice even when their advice has been overridden in the past. We give a simple priority based scheme for composing such features. This guarantees the* ***maximal*** *use of each feature. We provide a formal framework for specifying such features, and a compositional technique for verifying systems developed in this framework.*

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 [20]. 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)).

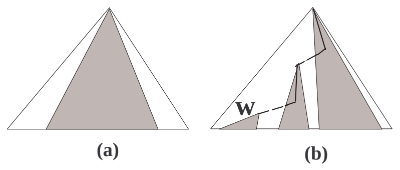

Figure 1. The conflict-tolerant specification on the right advises on how to extend w even though its advice has been overridden (dashed line) in the past when generating w.

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).

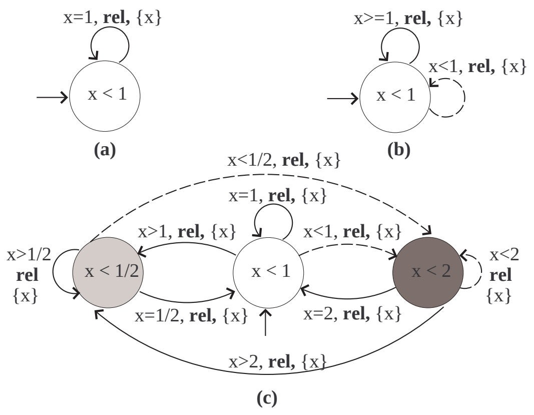

Figure 2. A classical spec (a), and conflict-tolerant specs (b) and (c) that induce the same classical spec (a).

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 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 [16] follows that of [22, 3]. 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 [3], 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 [6] 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 [1, 7].

In [5], 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 timed setting the view of features as controllers and illustrate conflict between features due to timing constraints. We then introduce the notion of conflict-tolerance in Section 4 and address the feasibility and verification problems in Section 5. Finally in Section 6, we describe our composition scheme and provide a precise formulation of the claim that the controllers are maximally utilized.

2. Preliminaries

Let Σ be a finite alphabet of events and let Σ^* denote the set of finite words over Σ. A *language* over Σ is a subset of Σ^*. We write $w \cdot x$ to denote the concatenation of two words w and x.

We use timed words to model behaviours with timing information. Timed words extend classical words with information about the time at which events occur. Let $\mathbb{R}_{\geq 0}, \mathbb{R}_{>0}, \mathbb{Q}_{\geq 0}$ denote the set of nonnegative reals, positive reals and nonnegative rationals respectively. We use a delay based representation for timed words. A *free timed word* over an alphabet Σ is a string in $(\Sigma \cup \mathbb{R}_{>0})^*$ in which two events from Σ never occur contiguously. Thus a free timed word may begin with an event or a delay and end with an event or a delay. The time stamp of an event in a timed word can be obtained by summing the preceding delays. For example, the time stamp of the event b in the free timed word $2 \cdot a \cdot 1 \cdot 1 \cdot b \cdot 1$ is 4. The concatenation of two free timed words may not be a free timed word. For example, $1a \cdot b2$ is not a free timed word.

A free timed word can be represented in a *canonical* way by adding 0 at the beginning and the end, and then collapsing contiguous delays. A canonical timed word is a string of the form $d_0 a_1 d_1 a_2 d_2 \cdots a_n d_n$, where $n \geq 0$, $d_0, d_n \in \mathbb{R}_{\geq 0}$, $d_i \in \mathbb{R}_{>0}$ for $0 < i < n$, and $a_j \in \Sigma$ for $1 \leq j \leq n$. Note that every free timed word has a unique canonical representation. Let ϵ denote the empty timed word whose canonical representation is 0. Let σ and τ be canonical timed words such that $\sigma \cdot \tau$ is a free timed word. We define the concatenation of σ and τ, denoted $\sigma \circ \tau$ (or simply $\sigma\tau$ when clear from the context), to be the canonical representation of the free timed word $\sigma \cdot \tau$. We say a canonical timed word σ is a *prefix* of the canonical timed word τ, written $\sigma \preceq \tau$ if there exists a canonical timed word σ' such that $\sigma \circ \sigma' = \tau$. We denote the set of all canonical timed words over Σ by $T\Sigma^*$. Henceforth, by timed words we will mean canonical timed words.

A *timed language* over Σ is a set of timed words over Σ. A timed language L is called *prefix-closed* if whenever $\tau \in L$ and $\sigma \preceq \tau$, we have $\sigma \in L$. For a timed language $L \subseteq T\Sigma^*$ and a timed word σ, we denote by $ext_\sigma(L)$, the *set of extensions* of σ that are in L, i.e.

$$ext_\sigma(L) = \{\tau \in T\Sigma^* \mid \sigma \circ \tau \in L\}.$$

This induces in a natural way an *immediate* extension function $iext_\sigma(L)$ which we define as follows. Let "δ" denote a symbolic event that stands for "time elapse". Then for any $\sigma \in T\Sigma^*$, the "immediate" extensions in $\Sigma \cup \{\delta\}$ is given by:

$$iext_\sigma(L) = \begin{cases} \{c \in \Sigma \mid \sigma \circ c \in L\} \cup \{\delta\} \\ \qquad \text{if } \exists d > 0 \text{ such that } \sigma \circ d \in L \\ \{c \in \Sigma \mid \sigma \circ c \in L\} \text{ otherwise.} \end{cases}$$

For modeling specifications and feature implementations, we use Alur-Dill timed transition system [1] equipped with the notion of deadlines proposed in [19]. Deadlines are constraints on transitions that are useful in modelling urgency. They can equivalently be specified as "time-can-progress" conditions on states, and we will follow this model.

Let C be a finite set of clocks. A *valuation* for the clocks in C is a map $v : C \to \mathbb{R}_{\geq 0}$. We denote by $\vec{0}$ the valuation which maps all the clocks in C to the value 0. For a valuation v, by $v + t$ (for $t \in \mathbb{R}_{\geq 0}$) we denote the valuation which maps each $x \in C$ to $v(x) + t$; and by $v[0/X]$ (for a subset of clocks X of C) the valuation which maps x to 0 if $x \in X$ and $v(x)$ otherwise. A clock constraint over C is given by the following syntax:

$$\phi := x \sim c \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi$$

where x is a clock in C, $\sim \in \{<, \leq, =, >, \geq\}$ and c is a constant in $\mathbb{Q}_{\geq 0}$. We write $v \vDash \phi$ to denote that the valuation v satisfies the constraint ϕ, with the expected meaning. We denote the set of clock constraints over C by $\Phi(C)$.

A *timed transition system* (TTS) over the alphabet Σ, is a structure of the form $\mathcal{T} = \langle Q, C, s, \to, tcp\rangle$, where Q is a finite set of states, C is a finite set of clocks, $s \in Q$ is the initial state, $\to \subseteq Q \times \Sigma \times \Phi(C) \times 2^C \times Q$ is a finite set of transitions, and $tcp : Q \to \Phi(C)$ specifies the condition under which *time can progress* from a given state. For a transition $e = (q, a, g, X, q') \in \to$, the guard g specifies when e may be taken and X specifies the set of clocks to be reset when e is taken.

A *configuration* of $\mathcal{T}$ is a pair (q, v) where $q \in Q$ and v is a valuation for the clocks in C. From a given configuration (q, v) of $\mathcal{T}$, there are two kinds of transitions:

discrete $(q, v) \xrightarrow{a} (q', v')$ if there is a transition $(q, a, g, X, q') \in \to$ such that $v \vDash g$ and $v' = v[0/X]$.

delay Let $d, d' \in \mathbb{R}_{\geq 0}$ be nonnegative time delays. Then $(q, v) \xrightarrow{d} (q, v + d)$ if for all $0 < d' < d$, $(v + d') \vDash tcp(q)$.

A run of $\mathcal{T}$ on a timed word $\tau = d_0 a_1 d_1 \cdots a_n d_n$ starting from a configuration (q, v) is a finite sequence of configurations $(q_0, v_0)(q_1, v_1)(q_2, v_2) \ldots (q_{2n+1}, v_{2n+1})$ where $q_0 = q, v_0 = v$ and for each i such that $0 \leq i \leq n$, $(q_{2i}, v_{2i}) \xrightarrow{d_i} (q_{2i+1}, v_{2i+1})$ and $(q_{2i-1}, v_{2i-1}) \xrightarrow{a_i} (q_{2i}, v_{2i})$. The timed language of $\mathcal{T}$, denoted $L(\mathcal{T})$, is the set of all timed words on which $\mathcal{T}$ has a run starting from the initial configuration $(s, \vec{0})$. Similarly, we denote by $L_{(q,v)}(\mathcal{T})$ the language of timed words on which $\mathcal{T}$ has a run starting from the configuration (q, v).

We say a TTS $\mathcal{T}$ is *deterministic* if there is at most one run of $\mathcal{T}$ on any timed word $\tau \in T\Sigma^*$. A sufficient condition for $\mathcal{T}$ to be deterministic is that for every pair of distinct transitions of the form (q, a, g_1, X_1, q') and (q, a, g_2, X_2, q''), the constraint $g_1 \wedge g_2$ is not satisfiable. A TTS $\mathcal{T}$ is *complete* if there is at least one run of $\mathcal{T}$ on any timed word $\tau \in T\Sigma^*$. A sufficient condition for $\mathcal{T}$ to be complete is that for every $q \in Q$ and every $a \in \Sigma$, $\bigvee_{(q,a,g_i,X_i,q_i) \in \to} g_i$ is valid and $tcp(q)$ is valid. For a deterministic and complete TTS $\mathcal{T}$, there is a unique run of $\mathcal{T}$ on a given timed word σ. Let (q, v) be the unique configuration reached by $\mathcal{T}$ on σ. Then we define $L_\sigma(\mathcal{T}) = L_{(q,v)}(\mathcal{T})$.

Let $\mathcal{T}_1 = \langle Q_1, C_1, s_1, \to_1, tcp_1\rangle$ and $\mathcal{T}_2 = \langle Q_2, C_2, s_2, \to_2, tcp_2\rangle$ be two timed transition systems over the same alphabet Σ. We assume that the clocks C_1 and C_2 are disjoint. Then the *synchronized product* of $\mathcal{T}_1$ and $\mathcal{T}_2$, denoted $\mathcal{T}_1 \| \mathcal{T}_2$, is given by the TTS $\langle Q_1 \times Q_2, C_1 \cup C_2, (s_1, s_2), \to, tcp\rangle$ where $\to$ and tcp are defined as: (i) $((p, q), g, a, X, (p', q')) \in \to$ if $(p, g_1, a, X_1, p') \in \to_1$ and $(q, g_2, a, X_2, q') \in \to_2$, $g = g_1 \wedge g_2$, and $X = X_1 \cup X_2$; and (ii) $tcp((p, q)) = tcp_1(p) \wedge tcp_2(q)$. We note that the synchronized product as defined above generates the intersection of the timed languages of $\mathcal{T}_1$ and T_2.

3. Features as Controllers

In this section we elaborate in a timed setting, the view of features as modular discrete event controllers [16], as proposed in [3, 22].

A *safety specification* over an alphabet Σ is a prefix-closed timed language over Σ, denoting the set of timed behaviours which are considered "safe" with respect to a certain aspect of safety. A safety specification can also be viewed as an "advice function" as defined below. This view will be useful when we introduce the notion of conflict tolerance in Section 4.

Definition 1 (Advice Function)**.** *An advice function over Σ is a function $f : T\Sigma^* \to 2^{T\Sigma^*}$ which satisfies the following conditions:*

- *$f(\epsilon)$ is a prefix-closed timed language.*
- *f is consistent in the sense that for all $\sigma \in f(\epsilon)$ and all $\tau \in f(\sigma)$, we have $f(\sigma\tau) = ext_\tau(f(\sigma))$.*

Note that for all $\sigma \in f(\epsilon)$, $f(\sigma)$ is a prefix-closed timed language. A safety specification L over Σ induces an advice function f_L given by $f_L(\sigma) = ext_\sigma(L)$. Conversely, an advice function f induces a safety language L_f given by $L_f = f(\epsilon)$. An advice function f induces in a natural way an *immediate* advice function $f^i : T\Sigma^* \to 2^{\Sigma \cup \{\delta\}}$ given by

$$f^i(\sigma) = iext_\sigma(f(\epsilon)).$$

We say a timed word τ is *according to* an immediate advice function f^i if for each prefix σ of τ, we have $iext_\sigma(\{\tau\}) \subseteq f^i(\sigma)$. Similarly, we say τ is according to an advice function f if τ is according to the induced immediate advice function f^i. A deterministic timed transition system $\mathcal{T}$ over Σ induces an advice function $f_\mathcal{T}$ given by $f_\mathcal{T}(\tau) = L_\tau(\mathcal{T})$ if $\tau \in L(\mathcal{T})$, and $\emptyset$ otherwise. We say a safety specification L over Σ is (timed) *regular* if it is given by a deterministic timed transition system $\mathcal{T}$ over Σ.

We now define the notion of a base system. Let Σ be an alphabet which is partitioned into "environment events" Σ_e and "system events" Σ_s.

Definition 2 (Base System). *A base system (or plant) over Σ is a deterministic timed transition system $\mathcal{B}$ over Σ, which we assume to be* ***non-blocking*** *in that whenever $\sigma \in L(\mathcal{B})$, then $iext_\sigma(L(\mathcal{B})) \neq \emptyset$. Further, for all states q, we assume that $tcp(q) = true$, i.e. $\mathcal{B}$ does not impose any constraints on progress of time in any of its states.*

Let $\mathcal{B}$ be a base system over Σ which is partitioned into environment events Σ_e and system events Σ_s.

Definition 3 (Controller). *A controller (or feature implementation) for $\mathcal{B}$ is a deterministic timed transition system $\mathcal{C}$ over Σ. The controller $\mathcal{C}$ is* ***valid*** *$\mathcal{B}$ if*

- *$\mathcal{C}$ is* ***non-restricting****: If $\sigma \in L(\mathcal{B}\|\mathcal{C})$ and $\sigma e \in L(\mathcal{B})$ for some environment event $e \in \Sigma_e$, then $\sigma e \in L(\mathcal{C})$. Thus the controller must not restrict any environment event e enabled in the base system after any controlled behaviour σ.*

- *$\mathcal{C}$ is* ***non-blocking****: If $\sigma \in L(\mathcal{B}\|\mathcal{C})$, then $ext_\sigma(L(\mathcal{B}\|\mathcal{C})) \neq \emptyset$. Thus the controller must not block the base system after any controlled behavior σ.*

We carry over the notions of advice function $f_\mathcal{C}$ and corresponding immediate advice function $f^i_\mathcal{C}$ for a controller $\mathcal{C}$. Let $\mathcal{B}$ be a base system and L a safety specification over Σ. We say a controller $\mathcal{C}$ for $\mathcal{B}$ *satisfies* L if $L(\mathcal{B}\|\mathcal{C}) \subseteq L$.

As a running example, we consider an electronic control unit (ECU) that controls the motors for locking and unlocking the doors of a car. The transition system for the motor (the base system) is shown in Fig. 3. We denote the environment events in italics and system events in bold. The set Σ_e comprises $lock\_req$, $unlock\_req$ and $crash$, while Σ_s is $\{\mathbf{lock}, \mathbf{unlock}\}$. We use the convention that self-loops on states are labeled with all events in Σ, excluding those on which there is an outgoing transition from the state, as well as those events a for which the self-loop has a label $\sim a$. Thus, the state labeled U has an edge to itself on all events except $\mathbf{lock}$ and $\mathbf{unlock}$. The initial state is shown by an incoming arrow. To avoid clutter, we do not show guards or tcp when they are true, and resets when they are empty.

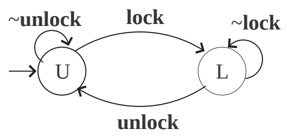

Figure 3. Motor Base System $\mathcal{B}$.

The environment events $lock\_req$ or $unlock\_req$ occur respectively, when the lock or unlock button on a remote key is pressed. The environment event $crash$ occurs when a car crash is sensed. The system event $\mathbf{lock}$ occurs when the door is locked and the system event $\mathbf{unlock}$ occurs when the door is unlocked.

The base system is typically run with several controllers including a "vanilla" controller which locks or unlocks the doors in response to passenger requests. Consider a feature called *crash safety* [4] which requires that if the doors are locked and a crash occurs, then the doors must be unlocked within 15 seconds. Fig. 4 shows a possible specification $\mathcal{S}_{CS}$ for this requirement. If the doors are locked and a crash occurs, then the clock z is reset and we are in the state labeled C. The delay advised in this state is at most 15 seconds as specified by the time can progress condition.

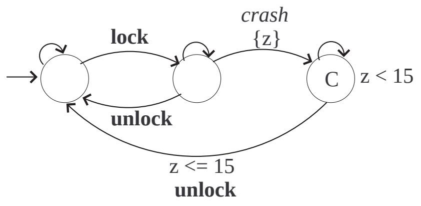

Figure 4. Crash Safety Spec $\mathcal{S}_{CS}$.

Fig. 5 shows a possible controller implementation $\mathcal{C}_{CS}$ for the specification $\mathcal{S}_{CS}$. This controller advises the base system to unlock the doors within 10 seconds after a car crash. In general, controllers may constrain the base system further than the specification based on other implementation considerations.

Now consider another feature called *overheat protection* [4]. This feature is required as it is found that excessive locking and unlocking of the doors within a short time (say, when children play with a remote key) could cause the motor to burn due to overheating. To prevent such a scenario, it

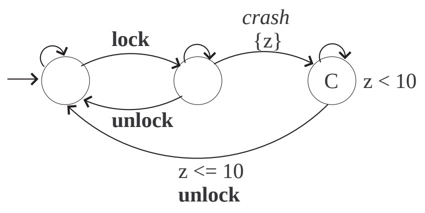

Figure 5. Crash Safety Controller $\mathcal{C}_{CS}$.

is recommended that $\mathbf{unlock}$ must be disabled for at least 180 seconds if there are 3 successive $\mathbf{lock}$ operations during a 30 second interval. Note that the designer of this feature may not be aware of the details of the crash safety feature. Figure 6 shows a possible specification $\mathcal{S}_{OHP}$ for this feature. The state labeled XD or YD is reached after a third successive lock during a 30 second interval. Then $\mathbf{unlock}$ is advised only after 180 seconds. We take the controller $\mathcal{C}_{OHP}$ for this feature to be the same as the specification $\mathcal{S}_{OHP}$. It is easy to see that each of the controllers $\mathcal{C}_{CS}$ and $\mathcal{C}_{OHP}$ are valid with respect to the base system $\mathcal{B}$ and satisfy their respective specifications.

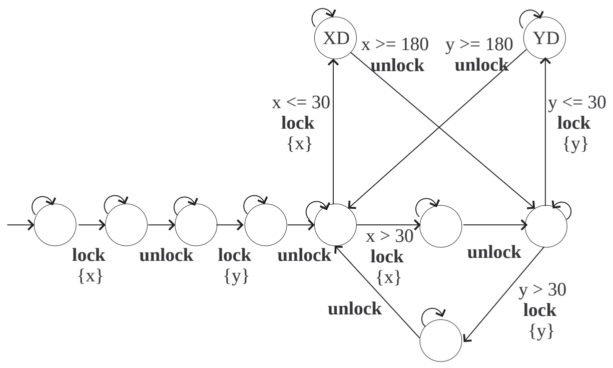

Figure 6. Overheat Protection Spec $\mathcal{S}_{OHP}$.

We now illustrate the notion of conflict between controllers.

Definition 4 (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 τ in $L(\mathcal{B}\|\mathcal{C}_1\|\mathcal{C}_2)$ such that $ext_\tau(L(\mathcal{B}\|\mathcal{C}_1\|\mathcal{C}_2))$ is empty. In other words, there exists a behaviour $\tau \in L(\mathcal{B})$ which is according to both $\mathcal{C}_1$ and $\mathcal{C}_2$, but $f^i_{\mathcal{B}}(\tau) \cap f^i_{\mathcal{C}_1}(\tau) \cap f^i_{\mathcal{C}_2}(\tau) = \emptyset$.*

Thus $\mathcal{C}_1$ and $\mathcal{C}_2$ are in conflict with respect to $\mathcal{B}$ if $\mathcal{C}_1\|\mathcal{C}_2$ is blocking with respect to $\mathcal{B}$. Consider the behaviour from the initial state of $\mathcal{B}$

$$0\ l_r\ 1\ \mathbf{l_s}\ 1\ u_r\ 1\ \mathbf{u_s}\ 1\ l_r\ 1\ \mathbf{l_s}\ 1\ u_r\ 1\ \mathbf{u_s}\ 1\ l_r\ 1\ \mathbf{l_s}\ 5\ c\ 10$$

where l_r, u_r, c denote the environment events $lock\_req, unlock\_req, crash$ and $\mathbf{l_s}, \mathbf{u_s}$ the system events $\mathbf{lock}, \mathbf{unlock}$ respectively. Here, the third lock operation has occured at 9 seconds, a crash 5 seconds later, and then 10 seconds have elapsed. The base system $\mathcal{B}$ allows $\mathbf{unlock}$ or delay. The controlled system is blocked as the advice of $\mathcal{C}_{CS}$ is $\{\mathbf{unlock}\}$ and the advice of $\mathcal{C}_{OHP}$ (in the state XD as a third lock operation has occurred in a 30 second interval) is $\{\delta\}$, i.e. $\mathcal{C}_{CS}$ does not advise any more delay before $\mathbf{unlock}$ whereas $\mathcal{C}_{OHP}$ advises further delay.

4. Conflict-Tolerant Controllers

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

Definition 5 (Conflict-Tolerant Advice Function)**.** *A conflict-tolerant advice function over an alphabet Σ is a function $f : T\Sigma^* \to 2^{T\Sigma^*}$ which satisfies the following conditions:*

- *for **every** timed word $\sigma \in T\Sigma^*$, $f(\sigma)$ is a prefix-closed language.*
- *f is consistent in the sense that for **all** $\sigma \in T\Sigma^*$ with $\tau \in f(\sigma)$, we have $f(\sigma\tau) = ext_\tau(f(\sigma))$.*

A *conflict-tolerant* timed transition system over Σ is a tuple $\mathcal{T}' = (\mathcal{T}, N, tcp)$ where $\mathcal{T}$ is a deterministic TTS over Σ whose timed-can-progress condition for each state is $true$, $N \subseteq \rightarrow$ is a subset of transitions designated as *not-advised*, and tcp is a time-can-progress condition for states in $\mathcal{T}$. Let (q, v) be a configuration of the TTS $\mathcal{T}$. Then the *unconstrained* language generated by $\mathcal{T}'$ starting from (q, v), denoted $L_{(q,v)}(\mathcal{T}')$, is defined to be $L_{(q,v)}(\mathcal{T})$. The *constrained* language generated by $\mathcal{T}'$ starting from (q, v) is denoted $L^c_{(q,v)}(\mathcal{T}')$, and defined to be $L_{(q,v)}(\widetilde{\mathcal{T}})$, where $\widetilde{\mathcal{T}}$ is the transition system obtained from $\mathcal{T}$ by deleting all not-advised transitions (i.e. transitions in N) and adding the time-can-progress conditions given by tcp. Let σ be a timed word in $L(\mathcal{T})$. Then there is a unique configuration (q, v) reached by $\mathcal{T}$ on σ. Then, by $L^c_\sigma(\mathcal{T}')$ we mean $L^c_{(q,v)}(\mathcal{T}')$. We say $\mathcal{T}'$ is *complete* wrt a timed language $L \subseteq T\Sigma^*$ if $L \subseteq L_\epsilon(\mathcal{T}')$.

A complete conflict-tolerant TTS $\mathcal{T}'$ induces a natural conflict-tolerant advice function $f_{\mathcal{T}'}$ given by $f_{\mathcal{T}'}(\sigma) = L^c_\sigma(\mathcal{T}')$. We say a conflict-tolerant advice function is *timed regular* if it is given by a conflict-tolerant TTS over Σ.

We define the synchronized product of a timed transition system $\mathcal{T}_1$ and a conflict-tolerant TTS $\mathcal{T}'_2 = (\mathcal{T}_2, N_2, tcp_2)$ to be the conflict-tolerant TTS $(\mathcal{T}_1\|\mathcal{T}_2, N'_2, tcp'_2)$, where the unadvised transition set N_2' and tcp_2' are inherited from N_2 and tcp_2 respectively.

Let $\mathcal{B}$ be a base system over the partitioned alphabet Σ.

Definition 6 (Conflict-Tolerant Controller)**.** *A conflict-tolerant controller $\mathcal{C}'$ for $\mathcal{B}$ is a conflict-tolerant TTS over Σ that is complete with respect to $L(\mathcal{B})$. The controller $\mathcal{C}'$ is* valid *wrt $\mathcal{B}$ if*

- *$\mathcal{C}'$ is **non-restricting**: If $\sigma e \in L(\mathcal{B})$ for some environment event $e \in \Sigma_e$, then $\sigma e \in L^c_\sigma(\mathcal{C}')$ (or equivalently $e \in f^i_{\mathcal{C}'}(\sigma)$). Thus the controller must not restrict any environment event e enabled in the base system after **any** base system behaviour σ.*
- *$\mathcal{C}'$ is **non-blocking**: If $\sigma \in L(\mathcal{B})$, then $L^c_\sigma(\mathcal{B}\|\mathcal{C}') \neq \emptyset$ (equivalently $f^i_{\mathcal{C}'}(\sigma) \cap f^i_{\mathcal{B}}(\sigma) \neq \emptyset$). Thus the controller must not block the base system after **any** base system behaviour σ.*

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 $\sigma \notin L^c_\epsilon(\mathcal{C}')$. In contrast, a classical controller $\mathcal{C}$ (see Definition 3) assumes that its advice is always followed by the base system, and advises extensions of only controlled behaviours (i.e. those in $L_\epsilon(\mathcal{B}\|\mathcal{C})$).

Definition 7 ($\mathcal{C}'$ satisfies f)**.** *Let f be a conflict-tolerant specification over Σ. A conflict-tolerant controller $\mathcal{C}'$ for $\mathcal{B}$ satisfies f if for each $\sigma \in L(\mathcal{B})$, $L^c_\sigma(\mathcal{B}\|\mathcal{C}') \subseteq f(\sigma)$. Thus after **any** base system behaviour σ, if the base system follows the advice of $\mathcal{C}'$, then the resulting behaviour conforms to the safety language $f(\sigma)$.*

We now illustrate these definitions with our running example. Fig. 7 shows a possible conflict-tolerant specification $\mathcal{S}'_{CS}$ for the crash safety feature. Note that the classical specification for crash safety shown in Fig. 4 required the doors to be unlocked within 15 seconds after a car crash and did not specify what needs to be done if $z > 15$ (this could be the case if a feature with priority greater than crash safety requires more delay before the doors can be unlocked). In contrast, $\mathcal{S}'_{CS}$ specifies that the doors must be unlocked as soon as possible when $z \geq 15$, i.e. δ is not advised from the state C when $z \geq 15$. A conflict-tolerant controller that satisfies this specification must meet the obligation that delay is not advised if 15 or more seconds have elapsed after the car crashed and the doors have not yet been opened.

Fig. 8 shows a possible conflict-tolerant specification $\mathcal{S}'_{OHP}$ for the overheat protection feature. The not-advised transitions are shown using dashed transitions. In this specification, even if the doors are unlocked before 180 seconds in one of the states XD or YD, the specification still looks for a lock operation that may correspond to a "bad" lock (i.e. a third consecutive lock in a 30 second interval), and advises a 180 second break before the next unlock. We note that the induced classical specification is the same as $\mathcal{S}_{CS}$ in Fig. 6.

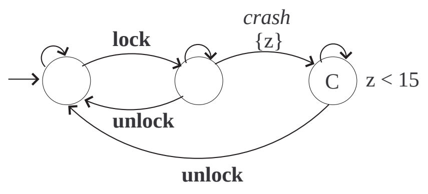

Figure 7. Conflict-Tolerant Crash Safety Specification $\mathcal{S}'_{CS}$.

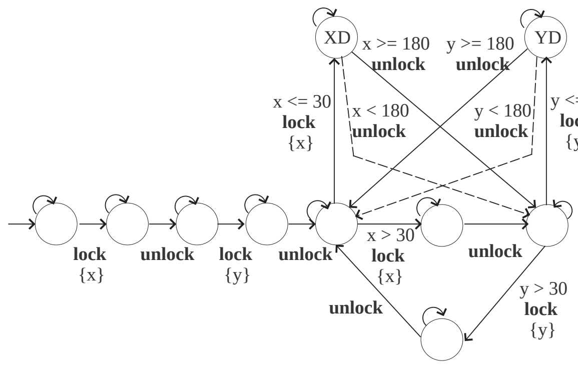

Figure 8. Conflict-Tolerant Overheat Protection Specification $\mathcal{S}'_{OHP}$.

As example valid CT controllers for $\mathcal{B}$ which satisfy $\mathcal{S}'_{CS}$ and $\mathcal{S}'_{OHP}$, we can take the specifications $\mathcal{S}'_{CS}$ and $\mathcal{S}'_{OHP}$ themselves, respectively. In general however, a conflict-tolerant controller may be quite different from the specification. This is because the specification captures only one aspect of safety, while there may be other functional, performance, and resource related requirements on the controller. Thus, it makes sense to consider the natural feasibility and verification problems in this setting, which we do in the next section.

5. Verification

In this section we address the natural problems of feasibility and verification in the conflict-tolerant framework.

Let Σ be an alphabet, partitioned into Σ_e and Σ_s. For a TTS $\mathcal{T}$ over Σ, a state $q \in Q$, and an event $a \in \Sigma$, we denote by $cond_{\mathcal{T}}(q, a)$, the condition under which a is enabled in q, i.e. $cond_{\mathcal{T}}(q, a) = \bigvee_{(q,g,a,X,q') \in \rightarrow} g$, where $\rightarrow$

is the transition relation of $\mathcal{T}$. For a conflict-tolerant TTS $\mathcal{T}'$, $cond_{\mathcal{T}'}(q,a)$ gives the condition under which a is advised from q, i.e. $cond_{\mathcal{T}'}(q,a) = \bigvee_{(q,g,a,X,q') \in \rightarrow \setminus N} g$.

Theorem 1 (Feasibility)**.** *Given a base system $\mathcal{B}$ over Σ, and a regular conflict-tolerant specification $\mathcal{S}'$ over Σ, we can check if $\mathcal{S}'$ is* feasible *with respect to $\mathcal{B}$ in that there exists a valid conflict-tolerant controller for $\mathcal{B}$ that satisfies $\mathcal{S}'$.*

Proof. We claim that there exists a valid controller for $\mathcal{B}$ satisfying $\mathcal{S}'$ iff in the synchronized product $\mathcal{B} \| \mathcal{S}'$, there does not exist a configuration $((b,q),v)$ which is reachable from the initial configuration $((s_{\mathcal{B}}, s_{\mathcal{S}'}), \vec{0})$, and satisfies one of the following conditions:

1. ($((b,q),v)$ is "restricting") there is an environment event $e \in \Sigma_e$ allowed by $\mathcal{B}$, but not advised by $\mathcal{S}'$, i.e. there exists $e \in \Sigma_e$ such that $v \vDash cond_{\mathcal{B}}(b,e)$ and $v \nvDash cond_{\mathcal{S}'}(q,e)$.

2. ($((b,q),v)$ is "blocking") there is no discrete or delay step that is both enabled in $\mathcal{B}$ and advised by $\mathcal{S}'$, i.e. $v \nvDash tcp(q)$, and for each $a \in \Sigma$, $v \nvDash (cond_{\mathcal{B}}(b,a) \wedge cond_{\mathcal{S}'}(q,a))$. Recall the assumption that $\mathcal{B}$ is non-blocking.

If such a configuration $((b,q),v)$ exists, then clearly a controller cannot be valid for $\mathcal{B}$ and satisfy $\mathcal{S}'$ at the same time. Conversely, if no such $((b,q),v)$ exists, then $\mathcal{S}'$ itself is a valid controller for $\mathcal{B}$ that satisfies $\mathcal{S}'$.

The condition above can easily be checked in linear time using the region automaton [1] for $\mathcal{B} \| \mathcal{S}'$. The size of the region automaton is exponential in the number of clocks and the maximal constants in the guards. □

Theorem 2 (Verification)**.** *Given a base system $\mathcal{B}$ over Σ, a regular conflict-tolerant specification $\mathcal{S}'$, and a conflict-tolerant controller $\mathcal{C}'$, we can check whether $\mathcal{C}'$ is a valid conflict-tolerant controller for $\mathcal{B}$ satisfying $\mathcal{S}'$.*

Proof. It is easy to see that a necessary and sufficient condition for $\mathcal{C}'$ to be a valid controller for $\mathcal{B}$ and satisfying $\mathcal{S}'$, is to check that in the synchronized product $\mathcal{B} \| \mathcal{C}' \| \mathcal{S}'$, there does *not* exist a configuration $((b,p,q),v)$ which is reachable from the initial configuration $((s_{\mathcal{B}}, s_{\mathcal{C}'}, s_{\mathcal{S}'}), \vec{0})$, and satisfies one of the following conditions:

1. ($\mathcal{C}'$ is restricting) there is an environment event $e \in \Sigma_e$ allowed by $\mathcal{B}$, but not advised by $\mathcal{C}'$, i.e. there exists $e \in \Sigma_e$ such that $v \vDash cond_{\mathcal{B}}(b,e)$ and $v \nvDash cond_{\mathcal{C}'}(p,e)$.

2. ($\mathcal{C}'$ is blocking) there is no discrete or delay step that is enabled by $\mathcal{B}$ and advised by $\mathcal{C}'$, i.e. $v \nvDash tcp(p)$ and $v \nvDash (\bigvee_{a \in \Sigma} cond_{\mathcal{B}}(b,a) \wedge cond_{\mathcal{C}'}(p,a))$.

3. ($\mathcal{C}'$ does not satisfy $\mathcal{S}'$) there is a discrete or delay step that is both enabled by $\mathcal{B}$ and advised by $\mathcal{C}'$, but not advised by $\mathcal{S}'$. That is, either for some $a \in \Sigma$, $v \vDash cond_{\mathcal{B}}(b,a) \wedge cond_{\mathcal{C}'}(q,a)$ but $v \nvDash cond_{\mathcal{S}'}(q,a)$, or $v \vDash tcp_{\mathcal{C}'}(p)$ and $v \nvDash tcp_{\mathcal{S}'}(q)$.

Once again this condition can be checked in linear time using the region automaton for the synchronised product of $\mathcal{B}$, $\mathcal{C}'$ and $\mathcal{S}'$. □

6. Composition

We now give a way of composing conflict-tolerant controllers based on a prioritization of the controllers. The composition guarantees that the advice of each controller is used in the "best possible" way.

Let $\mathcal{B} = \langle B, C_0, s_0, \rightarrow_{\mathcal{B}} \rangle$, be a base system over Σ, and let $\mathcal{C}_1' = \langle Q_1, C_1, s_1, \rightarrow_1, N_1, tcp_1 \rangle$ and $\mathcal{C}_2' = \langle Q_2, C_2, s_2, \rightarrow_2, N_2, tcp_2 \rangle$ be valid conflict-tolerant controllers for $\mathcal{B}$. We assume that the set of clocks C_0, C_1 and C_2 are disjoint. 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 $\mathcal{C}_1' >_P \mathcal{C}_2'$.

For $a \in \Sigma$, and $p \in Q$, $q_1 \in Q_1$, and $q_2 \in Q_2$, let $g_{11}^a(p,q_1,q_2)$ stand for the disjunction of constraints of the form $g \wedge g_1 \wedge g_2$ such that there exist transitions $t = (p,a,g,X,p')$, $t_1 = (q_1,a,g_1,X_1,q_1')$ and $t_2 = (q_2,a,g_1,X_1,q_2')$ in $\mathcal{B}$, $\mathcal{C}_1'$ and $\mathcal{C}_2'$ respectively, with $t_1 \notin N_1$ and $t_2 \notin N_2$. Let $g_{11}(p,q_1,q_2)$ stand for the constraint $\bigvee_{a \in \Sigma} g_{11}^a(p,q_1,q_2)$. Also, let $tcp_{11}(p,q_1,q_2)$ denote the constraint $tcp_1(q_1) \wedge tcp_2(q_2)$. Then:

Definition 8 (Prioritized Composition of Controllers)**.** *The* P-prioritized composition *of the controllers $\mathcal{C}_1'$ and $\mathcal{C}_2'$, with respect to the base system $\mathcal{B}$, is denoted $\|_{P,\mathcal{B}}(\mathcal{C}_1', \mathcal{C}_2')$, and defined to be the timed transition system $\mathcal{C} = \langle B \times Q_1 \times Q_2, C_0 \cup C_1 \cup C_2, (s_0,s_1,s_2), \Rightarrow, tcp \rangle$, where $\Rightarrow$ and tcp are defined as follows. Let $a \in \Sigma$, and $t = (p,a,g,X,p')$, $t_1 = (q_1,a,g_1,X_1,q_1')$ and $t_2 = (q_2,a,g_1,X_1,q_2')$ be transitions in $\mathcal{B}$, $\mathcal{C}_1'$ and $\mathcal{C}_2'$ respectively. Then we have transitions in $\mathcal{C}$ according to the following rules:*

1. *If $t_1 \notin N_1$ and $t_2 \notin N_2$ (that is they are both advised transtions) then add the transition*

   $$((p,q_1,q_2), a, g \wedge g_1 \wedge g_2, X \cup X_1 \cup X_2, (p',q_1',q_2')).$$

2. *If $t_1 \notin N_1$ and $t_2 \in N_2$ (that is t_1 is advised but t_2 is not) then add the transition*

   $$((p,q_1,q_2), a, g'', X \cup X_1 \cup X_2, (p',q_1',q_2')$$

   where $g'' = g \wedge g_1 \wedge g_2 \wedge \neg g_{11}(p,q_1,q_2) \wedge \neg tcp_{11}(p,q_1,q_2)$.

The time-can-progress condition tcp is given by $tcp(p, q_1, q_2) = tcp_{11}(p, q_1, q_2) \vee tcp_{10}(p, q_1, q_2)$, where $tcp_{10}(p, q_1, q_2) = tcp_1(q_1) \wedge \neg tcp_2(q_2) \wedge \neg g_{11}(p, q_1, q_2)$.

Lemma 1. *The controller $\mathcal{C} = \|_{P,\mathcal{B}}(\mathcal{C}'_1, \mathcal{C}'_2)$ defined above is a* valid *classical controller for $\mathcal{B}$. The immediate advice function $f^i_{\mathcal{C}}$ it induces is given as follows. For each $\tau \in L(\mathcal{B})$:*

$$f^i_{\mathcal{C}}(\tau) = \begin{cases} f^i_{\mathcal{B}}(\tau) \cap f^i_{\mathcal{C}'_1}(\tau) \cap f^i_{\mathcal{C}'_2}(\tau) \\ \qquad \text{if } f^i_{\mathcal{B}}(\tau) \cap f^i_{\mathcal{C}'_1}(\tau) \cap f^i_{\mathcal{C}'_2}(\tau) \neq \emptyset \\ f^i_{\mathcal{B}}(\tau) \cap f^i_{\mathcal{C}'_1}(\tau) \quad \text{otherwise.} \end{cases} \qquad \Box$$

We can now generalize this prioritized composition to any number of controllers. Let $\mathcal{C}'_1, \ldots, \mathcal{C}'_n$ be valid conflict-tolerant controllers for the base system $\mathcal{B}$, with each $\mathcal{C}'_i = \langle Q_i, C_i, s_i, \rightarrow_i, N_i, tcp_i \rangle$. Let P be a priority ordering that induces a total ordering $>_P$ on the controllers, say $\mathcal{C}'_1 >_P \cdots >_P \mathcal{C}'_n$. For $a \in \Sigma$, and $p \in Q$, and $q_i \in Q_i$ we extend the earlier definitions of g^a_{11}, g_{11}, and tcp_{11} in the expected way to $g^a_{1^n}(p, q_1, \ldots, q_n)$, $g_{1^n}(p, q_1, \ldots, q_n)$, and $tcp_{1^n}(p, q_1, \ldots, q_n)$. Then the transitions in $\mathcal{C} = \|_{P,\mathcal{B}}(\mathcal{C}'_1, \ldots, \mathcal{C}'_n)$ are defined inductively (on decreasing values of n-length bit strings u) as follows: Let $a \in \Sigma$, and $t = (p, a, g, X, p')$, $t_i = (q_i, a, g_i, X_i, q'_i)$ be transitions in $\mathcal{B}$, and each $\mathcal{C}'_i$ respectively. By the *rank* of the transitions $t_1, \ldots, t_n$ we mean the n-length bitstring u such that $u(i) = 1$ iff $t_i \notin N_i$. Then we have transitions in $\mathcal{C}$ according to the following rules:

1. If the rank of $t_1, \ldots, t_n$ is 1^n (that is they are all advised transtions) then add the transition

   $$((p, q_1, \ldots, q_2), a, g'', X \cup X_1 \cup \cdots \cup X_n, (p', q'_1, \ldots, q'_n))$$

   where $g'' = g \wedge g_1 \wedge \cdots \wedge g_n$.

2. If rank of $t_1, \ldots, t_n$ is u with $1^n > u \geq 10^{n-1}$, then add

   $$((p, q_1, \ldots, q_n), a, g'', X \cup X_1 \cup \cdots \cup X_n, (p', q'_1, \ldots, q'_n))$$

   where $g'' = g \wedge g_1 \wedge \cdots \wedge g_n \wedge \neg \bigvee_{u' > u} g_{u'}(p, q_1, \ldots, q_n) \wedge \neg \bigvee_{u' > u} tcp_u(p, q_1, \ldots, q_n)$.

The time-can-progress condition tcp is given by $tcp(p, q_1, \ldots, q_n) = \bigvee_u tcp_u(p, q_1, \ldots, q_n)$ (for $u \geq 10^{n-1}$), where tcp_{1^n} is defined as above, and inductively $tcp_u(p, q_1, \ldots, q_n) = \bigwedge_{i:u(i)=1} tcp_i(q_i) \wedge \bigwedge_{i:u(i)=0} \neg tcp_i(q_i) \wedge \neg \bigvee_{u' > u} g_{u'}(p, q_1, \ldots, q_n)$.

Before giving the "correctness" condition satisfied by our composition scheme, we state a useful observation which says that the set of points along a timed word which are according to a regular advice function, is finitely varying. It will be convenient to talk of "intervals" of prefixes in a given timed word σ. Thus, for example, for prefixes τ and τ' of σ such that $\tau \preceq \tau'$, the interval $(\tau, \tau']$ denotes the set of prefixes $\{\alpha \preceq \sigma \mid \tau \prec \alpha \preceq \tau'\}$. We say that intervals I and J of σ are *adjacent* if $I \cap J = \emptyset$ and $I \cup J$ forms an interval of σ.

Lemma 2. *Let $\mathcal{C}'$ be a conflict-tolerant controller over Σ, and let $\sigma \in T\Sigma^*$. Then σ can be covered by a sequence of adjacent nonempty intervals $I_0, I_1, \ldots, I_k$ ($k \geq 0$) such that σ is alternately according to $f^i_{\mathcal{C}'}$ in each I_i. More precisely, either in each even interval (of the form I_{2i}) σ is according to $f^i_{\mathcal{C}'}$, and in each odd interval (of the form I_{2i+1}) σ is* never *according to $f^i_{\mathcal{C}'}$; or in each odd interval σ is according to $f^i_{\mathcal{C}'}$, and in each even interval σ is* never *according to $f^i_{\mathcal{C}'}$.*

Proof. It is sufficient to observe that in the unique run of $\mathcal{C}'$ on σ, the prefixes corresponding to action points are either according to the advice of $\mathcal{C}'$ or not, and for the periods of time elapse, the run remains in a single state throught this period. Each such period can be partitioned into a finite number of periods which lie alternately inside the tcp condition for the state or outside. $\Box$

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. $\mathcal{B}$. By the nature of our composition construction it is not difficult to see that:

Theorem 3. *The timed transition system $\mathcal{C} = \|_{P,\mathcal{B}}(\mathcal{C}'_1, \ldots, \mathcal{C}'_n)$ is a valid controller for $\mathcal{B}$. Furthermore, $\mathcal{C}$ satisfies each of the specifications $\mathcal{S}'_1, \ldots, \mathcal{S}'_n$ in the following "maximal" sense. For every $\sigma \in L(\mathcal{B} \| \mathcal{C})$:*

1. *$\sigma \in L^c_\epsilon(\mathcal{S}'_1)$.*

2. *Let $I_0, I_1, \ldots I_k$ be the sequence of adjacent intervals induced by $f_{\mathcal{C}'_j}$ on σ as given by Lemma 2. Then throughout the even intervals (WLOG) σ is according to the advice of $\mathcal{C}'_j$, and throughout the odd intervals σ is conflict with the higher priority controllers, in that for each prefix τ of σ in these intervals, for each $c \in f^i_{\mathcal{C}'_j}(\tau)$, there is a controller $\mathcal{C}'_k$ such that $\mathcal{C}'_k >_P \mathcal{C}'_j$ and $c \notin f^i_{\mathcal{B}}(\tau) \cap f^i_{\mathcal{C}'_k}(\tau)$.* $\Box$

Consider the base system behaviour which we used to illustrate conflict in Section 3. The controller for crash safety advises that the doors must be unlocked without any delay whereas the controller for overheat protection advises further delay before unlocking the doors. With the priority order $\mathcal{C}'_{OHP} < \mathcal{C}'_{CS}$, the conflict is resolved such that one possible extension (starting from *crash*) is $c_e\ 15\ \mathbf{u_s}\ 3\ l_e\ 1\ \mathbf{l_s}$, i.e. the doors are unlocked ignoring the advice of $\mathcal{C}'_{OHP}$. Later, the doors are locked again (perhaps, this was a minor crash). Note that $\mathcal{C}'_{OHP}$ is in the state labeled YD as this is a third successive lock during a 30 second interval.

The advice of $\mathcal{C}'_{OHP}$ can now be followed if there is no further conflict. We emphasise that the same controllers can be composed with the priority $\mathcal{C}'_{CS} < \mathcal{C}'_{OHP}$ to obtain a system in which conflicts are resolved in favour of $\mathcal{C}'_{OHP}$, while maximally utilizing the advice of $\mathcal{C}'_{CS}$.

7. Implementation

For verifying conflict-tolerant controllers, we use the infinite state, bounded model checker [2, 21] from the SAL [17] tool suite. We first translate the timed transition systems to SAL modules. An explicit environment module $\mathcal{E}$ is synchronously composed with the other modules. This module (i) nondeterministically chooses a discrete or a delay event that is enabled by all the other modules, and (ii) alternates between a discrete and a delay transition [8]. The synchronous composition ensures that time progresses by the same amount in all the modules.

We translate the given transitions systems for base system $\mathcal{B}$, conflict-tolerant specification $\mathcal{S}'$, and conflict-tolerant controller $\mathcal{C}'$ to SAL modules. We then generate a formula corresponding to each of the following conditions (see Theorem 2 in Section 5).

1. ($\mathcal{C}'$ is non-restricting) Let $((b,p),v)$ be the configuration reached after generating $\tau \in L(\mathcal{B}\|\mathcal{C}')$. Then $\forall e \in \Sigma_e \; v \vDash cond_{\mathcal{B}}(b,e) \Rightarrow v \vDash cond_{\mathcal{C}'}(p,e)$ must hold.

2. ($\mathcal{C}'$ is non-blocking) Let $((b,p),v)$ be the configuration reached after generating $\tau \in L(\mathcal{B}\|\mathcal{C}')$. Then $v \vDash tcp((b,p)) \vee (\bigvee_{a\in\Sigma} cond_{\mathcal{B}\|\mathcal{C}'}(b,p))$ must hold.

3. ($\mathcal{C}'$ satisfies $\mathcal{S}'$) For every transition $((b,p,q),v) \rightarrow ((b',p',q'),v')$ of the composed system $\mathcal{B}\|\mathcal{C}'\|\mathcal{S}'$, the auxiliary boolean variable $s\_advised$ is set to true iff $q \rightarrow q'$ is advised by $\mathcal{S}'$ and the auxiliary boolean variable $c\_advised$ is set to true iff $p \rightarrow p'$ is advised by $\mathcal{C}'$. Then $Always(c\_advised \Rightarrow s\_advised)$ must hold.

The safety properties generated above are either proved using k-induction [18] or a counterexample is obtained.

8. Conclusion

We note that conflict-tolerant specifications are somewhat stronger than classical specifications, and may not always admit a conflict-tolerant controller even when the induced classical specifications admits a classical controller. Nonetheless, whenever the conflict-tolerant controllers can be constructed, our framework provides a flexible way of composing the controllers to obtain systems with guarantees on the usage of each controller. Our framework is also amenable to more flexible priority schemes like according priority dynamically based on history of events.

References

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

[2] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.

[3] 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.

[4] W. Damm and M. Cohen. Advanced validation techniques meet complexity challenge in embedded software development. In *Embedded Systems Journal*, 2001.

[5] L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Timed interfaces. In *EMSOFT*, pages 108–122, 2002.

[6] D. D'Souza and M. Gopinathan. Conflict-tolerant features. In *Computer Aided Verification (to appear)*, 2008.

[7] D. D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In *STACS*, pages 571–582, 2002.

[8] B. Dutertre and M. Sorea. Timed systems in sal. Technical Report SRI-SDL-04-03, SRI, October 2004.

[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] 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.

[17] SAL – http://sal.csl.sri.com.

[18] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a sat-solver. In *FMCAD*, pages 108–125, 2000.

[19] J. Sifakis and S. Yovine. Compositional specification of timed systems (an extended abstract). In *Symp. on Theoretical Aspects of Comp. Science*, pages 347–359, 1996.

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

[21] M. Sorea. Bounded model checking for timed automata. *Electr. Notes Theor. Comput. Sci.*, 68(5), 2002.

[22] 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.