2010 17th International Symposium on Temporal Representation and Reasoning

Conflict-Tolerant Real-time Specifications in Metric Temporal Logic

Sumesh Divakaran, Deepak D'Souza and Raj Mohan M
Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560012, India.
{sumeshd,deepakd,raj}@csa.iisc.ernet.in

***Abstract*—A framework based on the notion of "conflict-tolerance" was proposed in [1], [2] as a compositional methodology for developing and reasoning about systems that comprise multiple independent controllers. A central notion in this framework is that of a "conflict-tolerant" specification for a controller. In this work we propose a way of defining conflict-tolerant real-time specifications in Metric Interval Temporal Logic (MITL). We call our logic** CT-MITL **for Conflict-Tolerant MITL. We then give a clock optimal "delay-then-extend" construction for building a timed transition system for monitoring past-MITL formulas. We show how this monitoring transition system can be used to solve the associated verification and synthesis problems for** CT-MITL**.**

***Keywords*-conflict-tolerance; delay-then-extend construction; verification; synthesis;**

I. Introduction

A framework based on the notion of "conflict-tolerance" was proposed in [1], [2] as a way of developing and reasoning about systems that are composed of a base system along with multiple independent controllers that each implement a certain feature for the system. Such systems appear commonly in software intensive domains, examples of which include a telecom switch which provides different features to subscribers, like call forwarding and call screening; or an automobile with several time-dependent features like cruise control and stability control. Typically the controller for each feature is developed independently, and the controllers are all integrated together using a hand-coded supervisory controller. Unfortunately in certain configurations of the system – as the reader may well imagine for the example features mentioned above – the individual controllers may profer conflicting advices on how the system should proceed next. These conflicts are typically resolved by suspending the lower-priority controller and then waiting for a "reset" state of the system before restarting the controller. As a result the system loses out on the suspended feature's utility during this period.

The framework in [1], [2] proposes a way of designing each controller so that, based on a priority ordering among the features, it is easy to compose them in a way in which each controller is utilised "maximally." Thus each controller's advice is taken at all times except when *each* of its advised actions is in conflict with a higher priority controller. The key idea in this framework is to specify a "conflict-tolerant" behaviour for each feature, and then to build a controller for each feature that meets its conflict-tolerant specification. Unlike a classical safety specification, which can be viewed as a prefix-closed language of behaviours, a conflict-tolerant specification is an *advice function* which specifies a safety language for *each* possible finite behaviour of the system. This is depicted in Fig. 1: If one considers the set of all possible behaviours of the system as a tree growing downwards, then part (a) shows the shaded "cone" denoting a classical safety language, and part (b) depicts what a tolerant specification may look like, with safety cones prescribed for each possible behaviour of the system. A controller for such a specification must itself be "tolerant" in that it not only advises the system on what actions to take next (like a classical controller), but also keeps track of possible deviations from its advice, and goes on to advise next events so as to control the *subsequent* behaviour of the system. A conflict-tolerant controller satisfies a tolerant specification given as an advice function f if after *every* system behaviour σ, the subsequent controlled behaviour of the system stays within the safety language $f(\sigma)$.

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

An important component missing in the framework of [1], [2] is the ability to specify conflict-tolerant specifications in a popular specification language like temporal logic. In this paper our aim is to fill this gap in the domain of real-time systems, by proposing a way of specifying conflict-tolerant specifications in Metric Interval Temporal Logic (MITL) [3], [4].

The logic we propose, called CT-MITL for "Conflict-Tolerant MITL," is a syntactic fragment of MITL. A CT-MITL specification is a conjunction of formulas of the form $\Box(\varphi \Rightarrow \psi)$, where φ is a past-MITL formula, and ψ is a disjunction of formulas of the form c, where c is a system action or the symbolic event δ which stands for "time

1530-1311/10 $26.00 © 2010 IEEE
DOI 10.1109/TIME.2010.23

elapse." A CT-MITL formula defines an "immediate" advice function in a natural way: at the end of any behaviour σ, we check whether the past formula φ is true, and if so, advise a set of next actions that satisfy ψ.

The associated verification problem for CT-MITL is to check, given a base system $\mathcal{B}$ and a conflict-tolerant controller C (both modelled as Alur-Dill timed transition systems), and a conflict-tolerant specification in the form of a CT-MITL formula θ, whether C satisfies the *advice function* induced by θ, with respect to the given base system $\mathcal{B}$ (as described above). We note that an advice function is in general a richer object than a classical safety specification, and thus the verification problem for CT-MITL is more general than the classical verification problem for MITL. In general, a controller C may satisfy θ as a classical MITL specification, but *may not* satisfy it as a conflict-tolerant specification.

Nevertheless, we show that the verification problem, as well as the associated feasibility and synthesis problems, for CT-MITL can be solved algorithmically, using essentially the same technique as for classical MITL [4], [5], [6]. The main step is to build for a given past-MITL formula φ a deterministic transition system that "monitors" the truth of φ along every timed word it reads. The construction we give is inspired by the elegant *compositional* construction of a deterministic timed automaton for a past-MITL formula given in [6]. Our construction is slightly different, as we use a "delay-then-extend" idea to handle the main inductive case of $\diamondsuit_I \varphi$, which is provably clock-optimal and potentially uses one clock less than earlier constructions in [4], [6]. This step can also be implemented very simply in modelling and simulation tools like SIMULINK [7], using simple off-the-shelf "delay" and "pulse" blocks (see [8]).

A more detailed technical report is available in [8]. Our logic and techniques specialize easily to the discrete (untimed) setting as reported in [9].

II. Preliminaries

For an alphabet of symbols Σ we denote the set of words over Σ by Σ^*, and by ϵ the empty word. Let $\mathbb{R}_{\geq 0}$ and $\mathbb{Q}_{\geq 0}$ denote the set of non-negative reals and rationals respectively. We will use the standard notation to describe intervals of reals. So for example the interval $(1, 2]$ denotes the set $\{t \in \mathbb{R}_{\geq 0} \mid 1 < t \leq 2\}$. An interval is *non-singular* if the set it denotes is not a singleton set. Whenever convenient we use '$\langle$' (resp. '$\rangle$') to denote a left-open or left-closed (resp. right-open or right-closed) interval bracket.

A *timed word* σ over Σ is a string in $(\Sigma \cup \mathbb{R}_{\geq 0})^*$ 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 $1 \leq j \leq n$. We use the notation $dur(\sigma)$ to denote the duration of σ, which in this case is $\sum_{i=0}^{n} d_i$. We use the symbol ε to denote the empty timed word whose representation is simply "0."

Let $\sigma = d_0 a_1 d_1 \cdots a_m d_m$, and $\tau = e_0 b_1 e_1 \cdots b_n e_n$, be timed words such that it is not the case that: $n, m > 0$ and $d_m = e_0 = 0$. Then we define the *concatenation* of σ and τ, denoted $\sigma \cdot \tau$ (or simply $\sigma\tau$ when clear from the context), to be the timed word $d_0 a_1 d_1 \cdots a_m (d_m + e_0) b_1 e_1 \cdots b_n e_n$. We say a timed word σ is a *prefix* of a timed word τ, written $\sigma \preceq \tau$, if there exists a timed word σ' such that $\sigma \cdot \sigma' = \tau$. We denote the set of all timed words over Σ by $T\Sigma^*$. We note that our timed words are essentially Alur-Dill timed words except that we use a delay-based representation and allow timed words to end with delays.

It will often be convenient to view a timed word σ over Σ as a map from $[0, dur(\sigma)]$ to $\Sigma \cup \{\delta\}$, which tells us whether the event at time t in $[0, dur(\sigma)]$ is an action point $a \in \Sigma$, or the symbolic event "δ" which denotes a non-action point or "time elapse." Thus if $\sigma = d_0 a_1 d_1 \cdots a_n d_n$, for each $t \in [0, dur(\sigma)]$ we can define

$$\sigma(t) = \begin{cases} a_k & \text{if } n > 0 \text{ and } \exists k \in \{1, \ldots, n\} : \ t = \Sigma_{i=0}^{k-1} d_i, \\ \delta & \text{otherwise.} \end{cases}$$

By $last(\sigma)$ we will mean the value $\sigma(dur(\sigma))$ in $\Sigma \cup \{\delta\}$. For $t \in [0, dur(\sigma)]$ we use the notation $\sigma[0, t)$ to denote the prefix of σ of duration t which does not end in an action point. Thus $\sigma[0, t) = \tau$ where $\tau \preceq \sigma$, $dur(\tau) = t$, and $last(\tau) = \delta$.

A *timed language* over Σ is simply a set of timed words over Σ. A timed language L is called *prefix-closed* if whenever $\sigma \in L$ and $\tau \preceq \sigma$, we have $\tau \in L$. We denote by $L_{\preceq}$ the prefix-closure of 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. Thus $ext_\sigma(L) = \{\tau \in T\Sigma^* \mid \sigma \cdot \tau \in L\}$.

We use a variant of Alur-Dill timed transition systems [10] which have "time can progress" conditions [11] as state invariants, to model the systems we consider. To begin with 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 clocks in C to 0. For a valuation v and $t \in \mathbb{R}_{\geq 0}$, by $v + t$ we mean the valuation that 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 each x in X to 0, and each x in $C - X$ to $v(x)$. A *clock constraint* g over C is a boolean combination of atomic constraints of the form $x \sim c$, where x is a clock in C, $\sim \in \{<, \leq, =, >, \geq\}$ and c is a rational constant. We write $v \models g$ to say that the valuation v satisfies the clock constraint g, with the expected meaning. By $\Phi(C)$ we denote the set of clock constraints over C.

A *timed transition system* (TTS) over an alphabet Σ is a structure of the form $\mathcal{T} = (Q, C, s, \to, tcp)$, where Q is a finite set of states, C is a finite set of clocks, $s \in Q$ is the initial state, $\to \subseteq Q \times \Sigma \cup \{\epsilon\} \times \Phi(C) \times 2^C \times Q$ is a finite set of transitions, and $tcp : Q \to \Phi(C)$ specifies the condition under which *time can progress* in a given state. 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**) $(q, v) \xrightarrow{c} (q', v')$ where

$c \in \Sigma \cup \{\epsilon\}$, if there exists $(q, c, g, X, q') \in \rightarrow$, such that $v \models g$, and $v' = v[0/X]$; and (**Delay**) $(q, v) \xrightarrow{d} (q, v + d)$ where $d \in \mathbb{R}_{\geq 0}$, if for all $0 \leq d' < d$, we have $(v + d') \models tcp(q)$.

A *run* of $\mathcal{T}$ on a timed word σ staring from a configuration (q, v) is a sequence of time points $0 = t_0 \leq t_1 < t_2 < \cdots < t_n \leq t_{n+1} = dur(\sigma)$, with $n \geq 0$, along with a sequence of configurations of $\mathcal{T}$, $(q_0, v_0), (q_1, v_1), \ldots, (q_{2n+1}, v_{2n+1})$, satisfying $(q_0, v_0) = (q, v)$; for each $i \in \{1, \ldots, n+1\}$, $(q_{2i-2}, v_{2i-2}) \xrightarrow{t_i - t_{i-1}} (q_{2i-1}, v_{2i-1})$; and for each $i \in \{1, \ldots, n\}$, $(q_{2i-1}, v_{2i-1}) \xrightarrow{c_i} (q_{2i}, v_{2i})$, where $c_i = \sigma(t_i)$ if $\sigma(t_i) \in \Sigma$, and ϵ otherwise. A *semirun* of $\mathcal{T}$ on σ starting from a configuration (q, v) is defined to be the run of $\mathcal{T}$ on σ if $t_n < t_{n+1}$ and is defined to be the sequence of time points $0 = t_0 \leq t_1 < t_2 < \cdots < t_n = dur(\sigma)$, along with the sequence of configurations of $\mathcal{T}$, $(q_0, v_0), (q_1, v_1), \ldots, (q_{2n-1}, v_{2n-1})$, satisfying the expected conditions, otherwise.

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 semirun of $\mathcal{T}$ on any timed word $\sigma \in T\Sigma^*$. For a deterministic TTS $\mathcal{T}$, and a timed word σ on which $\mathcal{T}$ has a semirun , we note that the semirun is unique. Let (q, v) be the unique configuration reached by the semirun of $\mathcal{T}$ on σ and let us denote it by $config_{\mathcal{T}}(\sigma)$. Then we define $L_\sigma(\mathcal{T}) = L_{config_{\mathcal{T}}(\sigma)}(\mathcal{T})$.

It will be convenient sometimes to use constraints over both the states and clocks in a TTS. A *configuration constraint* g over a set of states Q and a set of clocks C is a boolean combination of atomic constraints of the form q (for $q \in Q$) and $x \sim c$ (as for clock constraints). Once again, the notion of when a configuration (q, v) satisfies a configuration constraint g is defined in the expected way. A TTS with configuration constraints is a TTS which uses configuration constraints instead of clock constraints, and the notions of runs, etc, are defined in a similar manner as for TTS's.

Let $\mathcal{T}_1 = (Q_1, C_1, s_1, \rightarrow_1, tcp_1)$ and $\mathcal{T}_2 = (Q_2, C_2, s_2, \rightarrow_2, tcp_2)$ be two timed transition systems, possibly with configuration constraints, over the same alphabet Σ. We assume that the set of clocks C_1 and C_2, and states Q_1 and Q_2 are disjoint. Then the *synchronized product* of $\mathcal{T}_1$ and $\mathcal{T}_2$, denoted by $\mathcal{T}_1 \parallel \mathcal{T}_2$, is given by the TTS $(Q_1 \times Q_2, C_1 \cup C_2, (s_1, s_2), \rightarrow, tcp)$ where $\rightarrow$ and tcp are defined as follows.

- $((p, q), g, c, X, (p', q')) \in \rightarrow$ if either there exist transitions $(p, g_1, c, X_1, p') \in \rightarrow_1$ and $(q, g_2, c, X_2, q') \in \rightarrow_2$, and $g = g_1 \wedge g_2$, and $X = X_1 \cup X_2$; or $c = \epsilon$ and there exists a transition $(p, g, \epsilon, X, p') \in \rightarrow_1$, and $q = q'$ (or vice-versa).
- $tcp((p, q)) = tcp_1(p) \wedge tcp_2(q)$.

We note that the synchronized products as defined above generates the intersection of the timed languages $L(\mathcal{T}_1)$ and $L(\mathcal{T}_2)$.

Finally, we define the notion of an "open" TTS which we will make use of in our inductive construction of monitoring TTS's. An *open* TTS (over Σ) with respect to the pairwise-disjoint set of states and clocks $Q_1, C_1, \ldots, Q_n, C_n$, is a structure $\mathcal{U} = (Q, C, s, \rightarrow, tcp)$, similar to a TTS over Σ, that uses "open" configuration constraints over $Q_1, C_1, \ldots, Q_n, C_n$. An *open* configuration constraint is a boolean combination of configuration constraints over Q, C, and atomic constraints of the form $\mathcal{T}_i.q$ (where $q \in Q_i$), and $\mathcal{T}_i.x \sim c$ (where $x \in C_i$). Given TTS's $\mathcal{T}_i = (Q_i, C_i, s_i, \rightarrow_i, tcp_i)$, we can define the composition of $\mathcal{U}$ and $\mathcal{T}_1, \ldots, \mathcal{T}_n$, denoted $\mathcal{U} \parallel \mathcal{T}_1 \parallel \cdots \parallel \mathcal{T}_n$, to be the TTS with configuration constraints obtained by taking the parallel composition of $\mathcal{U}, \mathcal{T}_1, \ldots, \mathcal{T}_n$ as defined above, but where the constraints $\mathcal{T}_i.q$ and $\mathcal{T}_i.y \sim c$ are interpreted as the configuration constraints $\bigvee_{u \in Q \times Q_1 \times \cdots \times Q_n,\ u(i)=q} u$ and $y \sim c$ respectively. Wherever convenient we denote constraints $\mathcal{T}_i.q$ and $\mathcal{T}_i.y \sim c$ by q and $y \sim c$ respectively.

III. Conflict-Tolerant Control

In this section we recall some of the key notions in the real-time conflict-tolerant framework from [2]. We begin with the notion of a base system. Some of the example systems we use here are adapted from [2], [12].

A base system is modelled as a TTS over an alphabet comprising "system" or "controllable" events Σ_s, and "environment" or uncontrollable events Σ_e. We call such an alphabet a *partitioned* alphabet, and use the convention that $\Sigma = \Sigma_s \cup \Sigma_e$. Let us fix a partitioned alphabet (Σ_s, Σ_e) for the rest of this section. A *base system* (or *plant*) over (Σ_s, Σ_e) is a deterministic TTS $\mathcal{B}$ over Σ, which we assume to be *non-blocking* in that whenever $\sigma \in L(\mathcal{B})$, then $ext_\sigma(L(\mathcal{B})) \neq \{\varepsilon\}$.

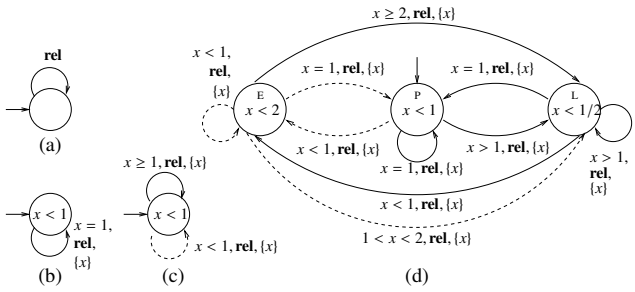


Figure 2. (a) Base system, (b) classical controller, and (c), (d) conflict-tolerant controllers.

Fig. 2 (a) shows a simple base system model which has only one system event "**rel**" (say for "releasing" a unit of lubricant), and no environment events.

In the classical framework, a real-time safety specification for a controller is given by a prefix-closed timed language. A controller for a given base system satisfies this specification if all behaviours of the controlled base system are contained

in the specified safety language. For example, the TTS shown in Fig. 2(b) is a classical controller for the example base system that ensures a **rel** event after every 1 time unit.

A *conflict-tolerant specification* on the other hand is a collection of safety languages, one for *each* possible behaviour of the base system. This is formalised as an "advice function" below, which advises a timed language of future extensions for each possible behaviour.

Definition 1. *A (real-time)* advice function *over an alphabet* Σ *is a mapping* $f : T\Sigma^* \to 2^{T\Sigma^*}$ *which satisfies the following conditions:*

- *for every timed word* σ *over* Σ, $f(\sigma)$ *is a prefix-closed language.*
- f *is consistent, i.e. for all timed words* σ, τ *over* Σ, *if* $\tau \in f(\sigma)$ *then* $f(\sigma\tau) = ext_\tau(f(\sigma))$.

An alternate way of describing an advice function is an *immediate* advice function which is a mapping $g : T\Sigma^* \to 2^{\Sigma \cup \{\delta\}}$. Given a timed word σ over Σ, and a timepoint $t \in [0, dur(\sigma)]$, we say σ is *according to* the immediate advice g at time t if $\sigma(t) \in g(\sigma[0,t))$. The immediate advice function g above induces an advice function $f_g : T\Sigma^* \to 2^{T\Sigma^*}$ given by:

$$f_g(\sigma) = \{\, \tau \in T\Sigma^* \mid \sigma\tau \text{ is according to } g \text{ in } \langle dur(\sigma), dur(\sigma\tau)]\}$$

where '⟨' = '[' if $\sigma(t) = \delta$ and '(' otherwise. It is easy to verify that the two conditions in Def. 1 are satisfied by f_g, and hence that it is a valid advice function.

A conflict-tolerant controller is similar to a classical controller which synchronizes with the base system and controls the choice of possible next system events available to the base system. The main difference however is that a conflict-tolerant controller also keeps track of the system events that are *against* its advice, and goes on to control the *subsequent* behaviour of the system. A real-time conflict-tolerant controller is modelled as an annotated TTS described below.

A *conflict-tolerant* timed transition system (CTTS) over Σ is a tuple $\mathcal{T}' = (\mathcal{T}, N, tcp')$ where $\mathcal{T} = (Q, C, s, \to, tcp)$ is a deterministic TTS over Σ, $N \subseteq \to$ is a subset of transitions designated as *not-advised*, and tcp' is the advised time-can-progress condition for states of $\mathcal{T}$. 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 replacing the time-can-progress condition tcp by tcp'. Let σ be a timed word in $L(\mathcal{T})$. Let (q, v) be the unique configuration reached by $\mathcal{T}$ on σ. Then, the constrained extensions of σ, denoted $L^c_\sigma(\mathcal{T}')$, is $L^c_{(q,v)}(\mathcal{T}')$. Thus a CTTS can be viewed as working in two "modes:" *tracking* (corresponding to the unconstrained behaviour) and *advising* (corresponding to constrained behaviour). Finally, we say $\mathcal{T}'$ is *complete* w.r.t. a timed language $L \subseteq T\Sigma^*$ if $L \subseteq L(\mathcal{T})$.

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

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

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

Fig. 2(c) and (d) show two conflict-tolerant controllers for the example base system in Fig. 2(a). In the figure we use the convention that the "not-advised" transitions are shown with dotted arrows, and the advised tcp' conditions are shown in the state (the tracking tcp conditions are all "true"). The first tolerant controller's behaviour is to advise a **rel** event if 1 or more time units have elapsed after the last **rel** event, no matter when it took place. On the other hand, the second tolerant controller in part (d) also advises a **rel** event after 1 time unit if the last two **rel** events were *exactly* 1 time unit apart or "punctual;" but if the last two **rel** events were *less* than 1 time unit apart ("early") it advises a **rel** event after 2 time units, and if the last two **rel** events were *more* than 1 time unit apart ("late") it advises a **rel** event after 0.5 time units. The states of the controller are marked "P", "E", and "L" for "Punctual", "Early", and "Late" respectively, and keep track of this property for the last two **rel** events.

We note that as *classical* controllers, both the controllers have the same effect as the classical controller in part (b): behaviours that are controlled according to their advice will always be (prefixes of) behaviours of the form $(1 \cdot \mathbf{rel})^*$. However as *tolerant* controllers their behaviours are quite different. We will return to these examples in the next section when we illustrate our logic for specifying tolerant specifications.

IV. Conflict-Tolerant Metric Interval Temporal Logic

In this section we present our logic for specifying real-time conflict-tolerant specifications, called CT-MITL. Our logic is based on the well-known timed temporal logic

Metric Interval Temporal Logic (MITL) [4] which is a popular logic for specifying real-time properties.

We begin by recalling the syntax and semantics of MITL. The syntax of an MITL formula over an alphabet Σ is given by: $\theta ::= \top \mid \bot \mid a \mid \neg\theta \mid \theta \vee \theta \mid \theta U_I \theta \mid \theta S_I \theta$ where $a \in \Sigma$ and I is a non-singular interval with end points in $\mathbb{Q}_{\geq 0} \cup \{\infty\}$.

We interpret formulas of MITL in a "continuous" manner over timed words. Let σ be a timed word, t a timepoint in $[0, dur(\sigma)]$, and θ an MITL formula. Then we define the satisfaction relation $\sigma, t \models \theta$, as follows. We always have $\sigma, t \models \top$ and $\sigma, t \not\models \bot$. In addition we have:

$$\begin{array}{lll} \sigma, t \models a & \text{iff} & \sigma(t) = a \\ \sigma, t \models \theta_1 U_I \theta_2 & \text{iff} & \exists t'' > t : t'' - t \in I \text{ and } \sigma, t'' \models \theta_2 \text{ and} \\ & & \forall t' : t < t' < t'', \sigma, t' \models \theta_1 \\ \sigma, t \models \theta_1 S_I \theta_2 & \text{iff} & \exists t'' < t : t - t'' \in I \text{ and } \sigma, t'' \models \theta_2 \text{ and} \\ & & \forall t' : t'' < t' < t, \sigma, t' \models \theta_1, \end{array}$$

with the boolean operators interpreted in the standard way. We note that we have used versions of the "Until" and "Since" operators that are "strict" in both the arguments, as is often done when the underlying time domain is dense.

We will make use of the following derived operators: $\delta = \neg \bigvee_{a \in \Sigma} a$, $\phi S \psi = \phi S_{[0,\infty)} \psi$, $init = \neg(\top S \top)$ (*init* thus characterizes the timepoint 0), $\Diamond_I \theta \equiv \top U_I \theta$, $\Box_I \theta \equiv \neg \Diamond_I \neg\theta$, $\Diamond\!\!\!-_I \theta \equiv \top S_I \theta$, and $\Box\!\!\!-_I \theta \equiv \neg \Diamond\!\!\!-_I \neg\theta$.

For an MITL formula θ and a timed word σ, we say $\sigma \models \theta$ iff $\sigma, 0 \models \theta$. We set $L(\theta) = \{\sigma \in T\Sigma^* \mid \sigma \models \theta\}$.

The "past" fragment of MITL is obtained by disallowing the U operator. We will make use of the "strict past" fragment, denoted MITL$_P$, which are past MITL formulas which are not a boolean combination of formulas, one of which is of the form a for some $a \in \Sigma$. Thus an MITL$_P$ formula cannot refer to the current time point. For an MITL$_P$ formula φ, we will say $\sigma \models \varphi$ to mean $\sigma, dur(\sigma) \models \varphi$.

We can now introduce the logic CT-MITL. A *Conflict-Tolerant MITL* (CT-MITL) formula over an alphabet Σ is an MITL formula θ over Σ of the form

$$\Box \bigwedge_{i \in \{1,\ldots,n\}} (\varphi_i \Rightarrow \psi_i)$$

where $n \geq 0$, and for each $i \in \{1, \ldots, n\}$, φ_i is an MITL$_P$ formula and ψ_i is of the form $\bigvee_{c \in X_i} c$ for some $X_i \subseteq \Sigma \cup \{\delta\}$.

The CT-MITL formula θ given above induces the immediate advice function $g_\theta : T\Sigma^* \to 2^{\Sigma \cup \{\delta\}}$ given by:

$$g_\theta(\sigma) = \bigcap_{i \in \{1,\ldots,k\},\ \sigma \models \varphi_i} X_i.$$

We use the convention that the intersection of an empty collection of sets is the universal set, in this case $\Sigma \cup \{\delta\}$. Thus, if σ is such that none of the φ_i's is true at the end of σ, then the immediate advice given by g_θ on σ is the set $\Sigma \cup \{\delta\}$. Finally, we will write f_θ to denote the corresponding advice function f_{g_θ}.

We now give a few examples to illustrate the logic. Returning to our example base system of Fig. 2(a), we give a couple of conflict-tolerant specifications in CT-MITL. For convenience we use notation of the form "≥ 1" to denote the interval $[1, \infty)$.

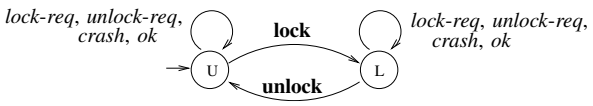


Figure 3. Door motor base system.

Example 1. *The* CT-MITL *formula θ_1 below specifies an immediate advice function "advise a* **rel** *event if one unit or more of time has elapsed since the last* **rel** *or init event, and time elapse otherwise."*

$$\Box(((\neg \mathbf{rel})\, S_{\geq 1}\, (\mathbf{rel} \vee \mathit{init})) \Rightarrow \mathbf{rel}) \wedge ((\Diamond\!\!\!-_{<1}\, (\mathbf{rel} \vee \mathit{init})) \Rightarrow \delta)).$$

The tolerant controller of Fig. 2(c) can be seen to satisfy this conflict-tolerant specification w.r.t. the base system of Fig. 2(a). □

Example 2. *The* CT-MITL *formula θ_2 below advises a* **rel** *event after 1 time unit (respectively 2 time units and 0.5 time units) depending on whether the last two* **rel** *events were 1 time unit apart ("Punctual"), less than 1 time unit apart ("Early"), or more than 1 time unit apart ("Late").*

$$\begin{array}{ll} \Box(& (((\neg \mathbf{rel}) S_{\geq 1} P) \Rightarrow \mathbf{rel}) \wedge (((\neg \mathbf{rel}) S_{<1} P) \Rightarrow \delta) \wedge \\ & (((\neg \mathbf{rel}) S_{\geq 2} E) \Rightarrow \mathbf{rel}) \wedge (((\neg \mathbf{rel}) S_{<2} E) \Rightarrow \delta) \wedge \\ & (((\neg \mathbf{rel}) S_{\geq 0.5} L) \Rightarrow \mathbf{rel}) \wedge (((\neg \mathbf{rel}) S_{<0.5} L) \Rightarrow \delta)), \end{array}$$

where $P = init \vee (\mathbf{rel} \wedge ((\neg \mathbf{rel}) S_{=1} \mathbf{rel}))$, $L = \mathbf{rel} \wedge ((\neg \mathbf{rel}) S_{>1} \mathbf{rel}))$ and $E = \mathbf{rel} \wedge (\Diamond\!\!\!-_{<1} \mathbf{rel}))$. The tolerant controller of Fig. 2(d) can be seen to satisfy the tolerant specification given by this formula. □

It is interesting to note that viewed as classical MITL specifications, both θ_1 and θ_2 describe *exactly* the same timed language: $L(\theta_1) = L(\theta_2) = (1 \cdot \mathbf{rel})^*_{\leq}$. However, as illustrated by the examples, the tolerant specifications induced by θ_1 and θ_2 are very different. The controllers in Fig. 2 each satisfy their respective specifications, but not the other's. Further, for any CT-MITL formula θ, we always have $L(\theta) = f_\theta(\varepsilon)$. Thus the timed language denoted by θ corresponds to the initial safety cone of the tolerant spec denoted by θ.

As a final example we look at a couple of features for a car door motor system, adapted from [2]. The base system here (shown in Fig. 3) models a car door motor, which has environment events $\Sigma_e = \{\mathit{lock\text{-}req}, \mathit{unlock\text{-}req}, \mathit{crash}, \mathit{ok}\}$ and system events $\Sigma_s = \{\mathbf{lock}, \mathbf{unlock}\}$. The "Overheat Protection" feature aims to protect the motor from overheating due to the excessive locking and unlocking of doors within a short period of time. It recommends that if two **lock** events occur within 30 sec the system events **lock** and **unlock** be disabled for 180 sec.

Example 3. *A possible tolerant specification for this feature is:*

$$\Box((\diamondsuit_{\leq 180}(\textbf{lock} \wedge \diamondsuit_{\leq 30}\textbf{lock})) \Rightarrow \delta \vee \mathit{env}).$$

Here env stands for the formula $\bigvee_{e\in\Sigma_e} e$. *See [8] for a possible tolerant controller that satisfies this specification.* □

Example 4. *The "Crash Safety" feature for the door motor requires that if a crash occurs, then unless an ok event occurs the door must be unlocked within* 15 *seconds, and kept unlocked till an ok event occurs. A possible tolerant specification for this feature is:*

$$\Box(\quad ((\neg(\mathit{ok} \vee \textbf{unlock})\, S_{<15}\, \mathit{crash}) \Rightarrow (\delta \vee \textbf{unlock} \vee \mathit{env})) \wedge ((\neg(\mathit{ok} \vee \textbf{unlock})\, S_{\geq 15} \mathit{crash}) \Rightarrow (\textbf{unlock} \vee \mathit{env}))).$$

See [8] for a tolerant controller satisfying the crash safety specification above. □

We can now define the natural verification and synthesis problems for the logic CT-MITL.

Definition 2. *(*Verification Problem for CT-MITL*) Given a base system* $\mathcal{B}$ *over* Σ*, a conflict-tolerant controller* C *for* $\mathcal{B}$*, and a* CT-MITL *formula* θ*, check whether* C *is a valid conflict-tolerant controller for* $\mathcal{B}$ *which satisfies the advice function* f_θ.

Definition 3. *(*Synthesis Problem for CT-MITL*) Given a base system* $\mathcal{B}$ *over* Σ*, and a* CT-MITL *formula* θ*, check whether there exists a valid conflict-tolerant controller for* $\mathcal{B}$ *which satisfies the advice function* f_θ*. If so, construct one.*

V. Monitoring Timed Transition System for MITL$_P$

The main step in solving the verification and synthesis problems for CT-MITL is the construction of a "monitoring" TTS for a given CT-MITL specification. Without loss of generality we assume that the specification uses only the modalities untimed S and $\diamondsuit$ as the modality S_I can be expressed in terms of modalities $\diamondsuit$ and S (see [13]).

A *monitoring guard* g over a set of states Q and clocks C is a boolean combination of configuration constraints and constraints of the form "a" for $a \in \Sigma$. A monitoring guard g is evaluated over a triple (c, q, v), where $c \in \Sigma \cup \{\delta\}$, $q \in Q$, and v is a valuation over C, as follows: $(c, q, v) \models g$ iff g is a configuration constraint and $(q, v) \models g$; and $(c, q, v) \models a$ iff $c = a$.

Let φ be an MITL$_P$ formula. A *monitoring* TTS (MTTS) for φ is a pair $(\mathcal{T}_\varphi, g_\varphi)$ where $\mathcal{T}_\varphi$ is a TTS with configuration constraints and g_φ is monitoring guard wrt $\mathcal{T}_\varphi$, such that $\forall \sigma \in T\Sigma^*, \sigma \models \varphi \iff (\mathit{last}(\sigma), \mathit{config}_{\mathcal{T}}(\sigma)) \models g_\varphi$.

Lemma 1. *Given an* MITL$_P$ *formula* φ *we can effectively construct an MTTS* $(\mathcal{T}_\varphi, g_\varphi)$ *for* φ.

Proof. We prove this lemma by induction on the structure of φ. For the base case $a \in \Sigma$, the monitoring TTS is given by $(\mathcal{T}_a, a)$ where $\mathcal{T}_a$ is single state transition system .

The monitoring TTS for the formula $\neg\psi$ is derived from the MTTS $(\mathcal{T}_\psi, g_\psi)$ for ψ, and is given by $(\mathcal{T}_\psi, \neg g_\psi)$.

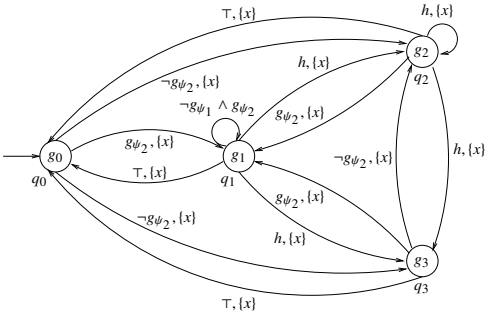
For the case $\psi_1 \vee \psi_2$ we assume that we have the monitoring TTS $(\mathcal{T}_{\psi_1}, g_{\psi_1})$ for ψ_1 and $(\mathcal{T}_{\psi_2}, g_{\psi_2})$ for ψ_2. Then the monitoring TTS for $\psi_1 \vee \psi_2$ is given by $(\mathcal{T}_{\psi_1} \parallel \mathcal{T}_{\psi_2}, g)$ where $g = \mathcal{T}_{\psi_1}.g_{\psi_1} \vee \mathcal{T}_{\psi_2}.g_{\psi_2}$.

Before we treat the two remaining cases, for a monitoring guard g we introduce the notation for the *left-closure* of g, denoted $[\![g$:

$$\begin{aligned}
[\![u &= u, u \in Q \cup \Sigma,\\
[\![(x \bowtie c) &= x \geq c, \bowtie \in \{>, \geq\},\\
[\![(x \bowtie c) &= x \bowtie c, \bowtie \in \{<, \leq\},\\
[\![(g_1 \vee g_2) &= [\![g_1 \vee [\![g_2,\\
[\![(g_1 \wedge g_2) &= [\![g_1 \wedge [\![g_2.
\end{aligned}$$

The notation $g]\!]$ is defined symmetrically, and we set $[\![g]\!] = ([\![g) \vee (g]\!])$.

Let us now construct a monitoring OpenTTS for the formula $\psi_1 S \psi_2$. Let $(\mathcal{T}_{\psi_1}, g_{\psi_1})$ and $(\mathcal{T}_{\psi_2}, g_{\psi_2})$ be MTTS for ψ_1 and ψ_2 respectively. Let $\mathcal{U}$ be the OpenTTS shown in Fig. V. Then the TTS $\mathcal{T}_{\psi_1 S \psi_2} = \mathcal{U} \parallel \mathcal{T}_{\psi_1} \parallel \mathcal{T}_{\psi_2}$ is an MTTS for $\psi_1 S \psi_2$ with $g_{\psi_1 S \psi_2} = (q_1 \vee q_2) \wedge (x > 0)$.



g_0	$(x > 0) \Rightarrow \neg g_{\psi_1}$
g_1	$(x > 0) \Rightarrow g_{\psi_1}$
g_2	$((x = 0) \Rightarrow ([\![g_{\psi_2}) \wedge ((x > 0) \Rightarrow g_{\psi_1})$
g_3	$((x \geq 0) \Rightarrow (\neg[\![g_{\psi_2}) \wedge ((x > 0) \Rightarrow g_{\psi_1})$
h	$\neg g_{\psi_1} \wedge \neg g_{\psi_2}$

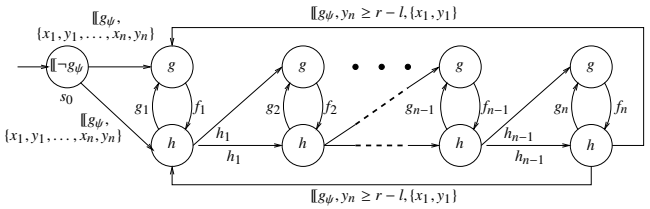
Figure 4. OpenTTS $\mathcal{U}$ for $\psi_1 S \psi_2$.

We now give our construction, called *delay-then-extend* or DTE for short, for the case $\diamondsuit_{(l,r)}\psi$, $0 < l < r - l$. Our construction uses an optimal number of clocks and in general it uses one clock less than the constructions in [4], [5].

Let $(\mathcal{T}_\psi, g_\psi)$ for be the monitoring automaton for ψ. Let $\mathbb{N}_n = \{1, \ldots, n\}$ and let $\mathcal{U}_1, \mathcal{U}_2$ and $\mathcal{U}_3$ be the OpenTTS shown in Fig. 5, Fig. 6 and Fig. 7 respectively.

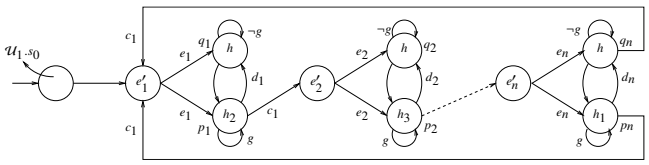
Let $\mathcal{T} = \mathcal{U}_3 \| \mathcal{U}_2 \| \mathcal{U}_1 \| \mathcal{T}_\psi$. For a timed word σ let ρ be the unique semirun of $\mathcal{T}$ on σ and $v_t(x)$ be the valuation of clock x of $\mathcal{T}$ at time t in the semirun ρ.

Lemma 2. *Let* $g_1 = \neg s_0 \wedge (\bigvee_{i\in\mathbb{N}_n}(q_i \wedge (x_i > l) \vee p_i \wedge (x_i > l) \wedge (y_i < l)))$ *and* $g_2 = \neg s_0 \wedge (\bigvee_{i\in\mathbb{N}_n}((x_i > l) \wedge (y_i \geq l) \wedge (z < r - l)))$. *Then for any timed word* σ, $(\mathit{config}_{\mathcal{T}_\varphi}(\sigma), t) \models g_1 \vee g_2$. $(\mathit{config}_{\mathcal{T}_\varphi}(\sigma), t) \models g_1 \vee g_2 \iff \sigma, t \models \varphi$.



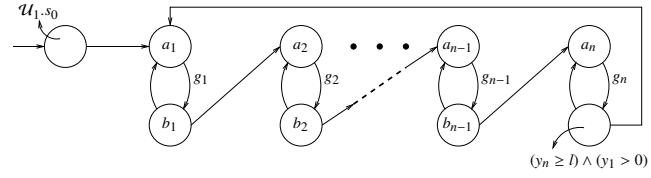
g	$[\![g_\psi]\!]$ for $i \in \mathbb{N}_n$
h	$[\![\neg g_\psi]\!]$ for $i \in \mathbb{N}_n$
f_i	$(y_i > 0), \{y_i\}$ for $i \in \mathbb{N}_n$
g_i	$0 < y_i < r - l$ for $i \in \mathbb{N}_n$
h_i	$y_i \geq r - l, \{x_{i+1}, y_{i+1}\}$ for $i \in \{1, \ldots, n-1\}$

Figure 5. OpenTTS $\mathcal{U}_1$.



g	g_ψ
h	$[\![g_\psi]\!]$
h_i	$[\![\neg g_\psi]\!] \wedge (x_i > 0)$ for $i \in \mathbb{N}_n$
c_i	$y_i \geq r - l$ for $i \in \mathbb{N}_n$
d_i	$y_i < r - l$ for $i \in \mathbb{N}_n$
e_i	$x_i = l$ for $i \in \mathbb{N}_n$
e'_i	$\neg s_0 \wedge (x_i \leq l)$ for $i \in \mathbb{N}_n$

Figure 6. OpenTTS $\mathcal{U}_2$.



a_i	$(\neg s_0) \wedge (y_i \leq l)$ for $i \in \mathbb{N}_n$
b_i	$(y_i \geq l) \wedge (y_{i+1} > 0)$ for $i \in \{1, \ldots, n-1\}$
g_i	$y_i = l, \{z\}$ for $i \in \mathbb{N}_n$

Figure 7. OpenTTS $\mathcal{U}_3$.

Proof: First let us prove that $(config_{\mathcal{T}_\varphi}(\sigma), t) \models g_1 \vee g_2 \Rightarrow \sigma, t \models \varphi$. Since $(config_{\mathcal{T}_\varphi}(\sigma), t) \models g$ there exists a clock x_i such that $v_t(x_i) \leq v_t(x_j)$ for all $i \in \mathbb{N}_n$ and that $\sigma, t - v_t(x_i) \models [\![g_\psi$. Now if $(config_{\mathcal{T}_\varphi}(\sigma), t) \models g_1$ (resp. $(config_{\mathcal{T}_\varphi}(\sigma), t) \models g_2$) then there does not exist a subinterval of $(t - v_t(x_i), t)$ (resp. $(t - v_t(x_i), t - v_t(y_i) + l)$) of size atleast $r - l$ such that g_ψ is nowhere true in the subinterval. Therefore $\sigma, t \models \varphi$. See [8] for details.

Now let us assume that $\sigma, t \models \varphi$ and prove the converse. Since $\sigma, t \models \varphi$ there exists a non negative point $t' \in (t-r, t-l)$ such that $\sigma, t' \models g_\psi$. Let x_i be the clock in X that is last reset on or before t' (our construction guarantees that such a clock exists in X). Then:

Case (a): the clock y_i is not reset in the interval (t', t). Then either g_ψ is everywhere true in the interval (t', t) in which case $(config_{\mathcal{T}_\varphi}(\sigma), t) \models \neg s_0 \wedge q_i \wedge (x_i > l)$; or g_ψ is nowhere true in the interval (t', t) in which case $(config_{\mathcal{T}_\varphi}(\sigma), t) \models \neg s_0 \wedge (x_i > l) \wedge (y_i \geq l) \wedge (z < r - l)$.

Case (b): Otherwise let t'' be the point where y_i last reset before t. Then $(config_{\mathcal{T}_\varphi}(\sigma), t'') \models g_\psi]\!]$ and there does not exist a subinterval of $(t - v_t(x_i), t - v_t(y_i) + l))$ of size atleast $r - l$ such that g_ψ is nowhere true in the subinterval. So $(config_{\mathcal{T}_\varphi}(\sigma), t) \models g_1$ if $t - t'' < l$ and $(config_{\mathcal{T}_\varphi}(\sigma), t) \models g_1 \vee g_2$ otherwise. See [8] for details. ■

We observe that as $\mathcal{T}_\varphi$ needs to reset at most a pair of clocks in an interval of size $r - l$, the no. of clocks required by $\mathcal{T}_\varphi$ is at most $2 * \lceil l/(r-l) \rceil + 1$. We also observe that the construction can be extended to any given MITL$_P$ formula which is of the form $\diamondsuit_{\langle l,r \rangle}\psi$ albeit with increased complexity.

Theorem 1. *Our construction for $\diamondsuit_{\langle l,r \rangle}\psi$ uses the optimal number of clocks.*

Proof: Let us give a brief sketch of the proof. Let σ be a timed word over a such that $|\sigma| = 2n + 2$ and for each $i \in \{0, \ldots, n\}$, the interval $[i*(r-l), (i+1)*(r-l))$ has exactly two a's. Furthermore, the distance between any a's is not an integer and the distance between the a's at the even position and the next odd position is greater than $(r - l)$. Now we argue that in order to monitor σ for $\diamondsuit_{\langle l,r \rangle}a$ we need atleast $2 * \lceil l/(r-l) \rceil + 1$ clocks and hence the theorem. See [8] for a detailed proof. ■

We close this section with a sample construction for a monitoring TTS for the formula $\varphi = \diamondsuit_{\leq 180}(\mathbf{lock} \wedge \diamondsuit_{\leq 30}\mathbf{lock})$ which is the premise of the specification for the feature overheat protection given in section IV. Let $\mathcal{T}_1$ be the TTS shown in Fig. 8(a) and let $g_1 = \mathcal{T}_1.(p_1 \wedge (x \leq 30))$. Then $(\mathcal{T}_1, g_1)$ is an MTTS for $\diamondsuit_{\leq 30}\mathbf{lock}$. Let $\mathcal{T}_2$ be the OpenTTS shown in Fig. 8(b). Then $(\mathcal{T}_2 || \mathcal{T}_1, g_2)$ is an MTTS for φ with $g_2 = \mathcal{T}_2.(q_1 \wedge (y \leq 180))$.

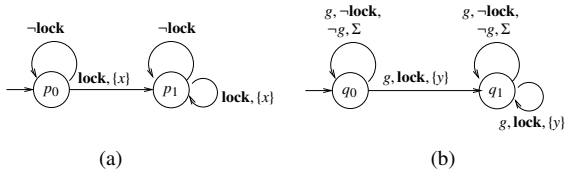


Figure 8. (a): TTS $\mathcal{T}_1$ and (b): OpenTTS $\mathcal{T}_2$.

VI. Verification and Synthesis

First let us discuss the verification problem for CT-MITL. We solve the problem by first constructing the monitoring automaton for the specification and then taking the product of the base system and the controller with it. Finally we construct the region graph of the product automaton. Once we have the region graph we can run a reachability analysis

similar to one given in [2] on it to see whether it adheres to our requirements. A detailed proof can be found in [8].

We now do a feasibility study on the conflict-tolerant controller for a given base system $\mathcal{B}$ and a CT-MITL specification $\theta = \Box(\bigwedge_{i=1}^{i=n}(\varphi_i \Rightarrow \psi_i))$ where each $\psi_i \subseteq \bigvee_{a\in\Sigma} a$ over an alphabet Σ. This we do by first synthesizing a "*maximal canonical conflict-tolerant controller*" C_θ which respects the specification θ. Now we do a reachability analysis on the region graph once again similar to the one given in [2] of the synchronized product $\mathcal{B}\|C_\theta$ and check whether there exists a "restricting" or "non blocking" configuration in the graph. See [8] for a detailed proof.

References

[1] D. D'Souza and M. Gopinathan, "Conflict-tolerant features," in *CAV*, ser. Lecture Notes in Computer Science, A. Gupta and S. Malik, Eds., vol. 5123. Springer, 2008, pp. 227–239.

[2] D. D'Souza, M. Gopinathan, S. Ramesh, and P. Sampath, "Conflict-tolerant real-time features," in *QEST*. IEEE Computer Society, 2008, pp. 274–283.

[3] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Systems*, vol. 2, no. 4, pp. 255–299, 1990.

[4] R. Alur, T. Feder, and T. A. Henzinger, "The benefits of relaxing punctuality," *J. ACM*, vol. 43, no. 1, pp. 116–146, 1996.

[5] O. Maler, D. Nickovic, and A. Pnueli, "Real time temporal logic: Past, present, future," in *FORMATS*, ser. Lecture Notes in Computer Science, P. Pettersson and W. Yi, Eds., vol. 3829. Springer, 2005, pp. 2–16.

[6] ——, "From mitl to timed automata," in *FORMATS*, ser. Lecture Notes in Computer Science, E. Asarin and P. Bouyer, Eds., vol. 4202. Springer, 2006, pp. 274–289.

[7] R. H. Bishop, *Modern control systems analysis and design using MATLAB and SIMULINK*. Reading, MA, USA: Addison-Wesley, 1997.

[8] S. Divakaran, D. D'Souza, and M. R. Mohan, "Conflict-Tolerant Real-Time Specifications in Metric Temporal Logic," Department of CSA, Indian Institute of Science, Technical Report IISc-CSA-TR-2009-9, 2009, uRL: `http://archive.csa.iisc.ernet.in/TR/2009/9/`.

[9] ——, "Conflict-Tolerant Specifications in Temporal Logic," in *ISEC '10: Proceedings of the 3rd India software engineering conference*. New York, NY, USA: ACM, 2010, pp. 103–110.

[10] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

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

[12] M. Gopinathan, "Conflict Tolerant Features," Ph.D. dissertation, Department of CSA, Indian Institute of Science, 2009.

[13] D. D'Souza and N. Tabareau, "On timed automata with input-determined guards," in *FORMATS/FTRTFT*, ser. Lecture Notes in Computer Science, Y. Lakhnech and S. Yovine, Eds., vol. 3253. Springer, 2004, pp. 68–83.