Conflict-Tolerant Specifications in Temporal Logic

Sumesh Divakaran Deepak D'Souza
Raj Mohan M.
Department of Computer Science and Automation,
Indian Institute of Science,
Bangalore 560012, India.

Abstract

A framework based on the notion of conflict-tolerance was proposed in [1] as a methodology for developing and reasoning about systems that are composed of multiple independent features. In [1] the authors use annotated transition systems to specify conflict-tolerant features. In this paper we propose a way of specifying conflict-tolerant features in Temporal Logic, which is a specification language widely used in practice. We call our logic Conflict-Tolerant LTL or CT-LTL. We provide an algorithm for verifying whether a given feature implementation meets a specification given in our logic. The paper concludes by providing a constructive procedure for synthesising a finite-state feature implementation from a given CT-LTL specification.

1 Introduction

A framework based on the notion of "conflict-tolerance" was proposed in [1] 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 an automobile with several features like cruise control and stability control; or a telecom switch which provides different features to subscribers, like call forwarding and call screening. Typically the controller for each feature is developed independently, and the controllers are all integrated together using a supervisory controller or feature manager. Unfortunately in certain configurations of the system – as the reader may well imagine for the example features mentioned above – the individual controllers may prefer conflicting advises on how the system should proceed next. These conflicts are typically resolved by suspending the lower-priority controller(s) and then waiting for a "reset" state of the system before restarting the controller. As a result the suspended feature's utility is lost out on during this period.

The framework in [1] proposes a way of designing each controller so that, given 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.

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

The key idea in this framework is to specify a "conflict-tolerant" behaviour for each feature, and to build controllers for each feature that meet 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 w, the subsequent controlled behaviour of the system stays within the safety language $f(w)$.

In this paper our aim is to propose a way of specifying conflict-tolerant specifications in Linear-Time Temporal Logic (LTL) [2]. LTL is a popular specification language in both academia and industry, and is considered to be a language that is easy to use by the specifier. Hence we believe that a mechanism for specifying conflict-tolerant specifications in LTL would be a useful addition to the conflict-tolerant framework.

The logic we propose, called CT-LTL for "Conflict-Tolerant LTL," is a syntactic fragment of LTL. A CT-LTL specification is a conjunction of formulas of the form $\Box(\varphi \implies \psi)$, where φ is a past LTL formula, and ψ is a disjunction of formulas of the form $\bigcirc a$ ("next a"), where a is system action. A CT-LTL formula defines an advice function in a natural way: for any behaviour w, 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-LTL is to check, given a base system $\mathcal{B}$ and a conflict-tolerant controller $\mathcal{C}$ (both modelled as a finite-state transition systems), and a conflict-tolerant specification in the form of a CT-LTL formula θ, whether $\mathcal{C}$ 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-LTL is more general different than the classical verification problem for LTL. Thus, a controller $\mathcal{C}$ may satisfy θ as a classical LTL specification, but *not* as a conflict-tolerant specification.

Nonetheless, we show that the verification problem, as well as the associated feasibility and synthesis problems, for CT-LTL can be solved algorithmically, using essentially the same technique as for classical LTL. The main step is to build for a given past LTL formula φ a deterministic transition system that "monitors" the truth of φ along every word it reads. This is similar to the "formula automaton" for classical LTL [3, 4].

The rest of the paper is organized as follows. In Sections 2 and 3 we introduce conflict-tolerant systems and associated notions along with some examples. In Section 4 we introduce LTL and our logic CT-LTL for specifying conflict-tolerant specifications. In Section 5 we give a procedure for generating the monitoring automaton for a past-LTL formula. Sections 6 and 7 address the verification and synthesis problems respectively.

2 Preliminaries

Let Σ be a finite alphabet of events and let Σ^* be the set of all finite words over Σ. A language over Σ is a subset of Σ^*. We denote the empty word by ϵ and the concatenation of the words u and v by $u \cdot v$ (or simply uv). We say u is a prefix of v, and write $u \preceq v$, if there exists w in Σ^* such that $uw = v$. We say a language L is *prefix-closed* if whenever $v \in L$ and $u \preceq v$, we have $u \in L$.

For any language L and a word u over Σ the set of all extensions of u in L, denoted $ext_u(L)$, is $\{v \in \Sigma^* \mid uv \in L\}$ and the set of all *immediate* extensions of u in L, denoted $iext_u(L)$, is $\{a \in \Sigma \mid av \in \ ext_u(L)\}$.

A *transition system* $\mathcal{T}$ over Σ is a tuple $(Q, s, \rightarrow)$ where Q is a finite set of states, $s \in Q$ is the start state and $\rightarrow \subseteq Q \times \Sigma \times Q$ is a Σ-labelled transition relation. A *run* of the transition system $\mathcal{T}$ on a word $w = a_1 \dots a_n$ starting from a state q_0 is a sequence $q_0, \dots, q_n$ of states in Q such that for all $i \in \{1, \dots, n\}$, $(q_{i-1}, a_i, q_i) \in \rightarrow$. Let $L_q(\mathcal{T})$ denote the set of all words on which $\mathcal{T}$ has a run starting from the state q. Then we define the *language* generated by $\mathcal{T}$, denoted $L(\mathcal{T})$, to be $L_s(\mathcal{T})$. We say the transition system $\mathcal{T}$ is *deterministic* if for every $p \in Q$ and $a \in \Sigma$ there exists at most one $q \in Q$ such that $(p, a, q) \in \rightarrow$. For a deterministic transition system $\mathcal{T}$ and a word w on which $\mathcal{T}$ has a run let q be the unique state reached by $\mathcal{T}$ on w. Then we define $L_w(\mathcal{T}) = L_q(\mathcal{T})$.

Next we define the standard "synchronized product" of two transition systems.

Definition 1. *Let* $T_1 = (Q_1, s_1, \rightarrow_1)$ *and* $T_2 = (Q_2, s_2, \rightarrow_2)$ *be two transition systems over* Σ*. Then the* synchronized product *of* T_1 *and* T_2*, denoted by* $T_1 \parallel T_2$*, is defined to be the transition system* $(Q_1 \times Q_2, (s_1, s_2), \rightarrow)$ *over* Σ*, where* $((p_1, p_2), a, (q_1, q_2)) \in \rightarrow$ *iff* $(p_1, a, q_1) \in \rightarrow_1$ *and* $(p_2, a, q_2) \in \rightarrow_2$*.*

3 Conflict-Tolerant Controllers

In this section we recall some of the key notions in the conflict-tolerant framework from [1]. To begin with, following a line of work suggested earlier in the literature [5, 6], feature implementations are viewed as *controllers* of a base system.

A base system is modelled as a finite-state transition system in which "system" events are performed in response to "environment" events. In this regard we define a *partitioned* alphabet to be one of the form (Σ_s, Σ_e) where Σ_s is a finite set of controllable or *system* events, and Σ_e is a finite set of uncontrollable or *environment* events. We will use the convention that $\Sigma_s \cup \Sigma_e = \Sigma$.

Definition 2. *A* base system *or* plant *over a partitioned alphabet* (Σ_s, Σ_e) *is a deterministic finite state transition system* $\mathcal{B}$ *over* Σ *satisfying the following conditions:*

- $L(\mathcal{B})$ *is alternating, i.e.* $L(\mathcal{B}) \subseteq (\Sigma_e \cdot \Sigma_s)^* \cup ((\Sigma_e \cdot \Sigma_s)^* \cdot \Sigma_e)$
- $\mathcal{B}$ *is non-blocking, i.e.* $w \in L(\mathcal{B}) \implies \exists a \in \Sigma$ *s.t.* $wa \in L(\mathcal{B})$*.*

As a running example we consider the base system shown in Fig. 2(a). The base system models a system that can perform the system events **noRel**, **rel** and **relDouble** for releasing zero, one, or two units of oxygen respectively, in response to the environment event *timer*.

In the classical framework, a (safety) specification for a feature is given by a prefix-closed language. A controller implementing the feature meets this specification with respect to a given base system if all behaviours of the controlled base system lie within the specified language. For example, the transition system shown in Fig. 2(b) is a classical controller for the given base system that ensures that every *timer* event is followed by a **rel** event.

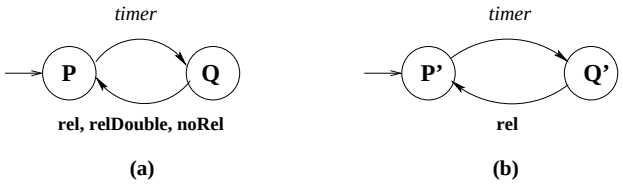

Figure 2: (a) Example base system and (b) a classical controller.

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 safety language of future extensions for each possible behaviour.

Definition 3. *An* advice function *over an alphabet* Σ *is a map* $f : \Sigma^* \to 2^{\Sigma^*}$ *which satisfies the following conditions:*

- *for every word* $w \in \Sigma^*$, $f(w)$ *is a prefix-closed language.*
- f *is* consistent *in the sense that for all* $u \in \Sigma^*$ *if* $v \in f(u)$ *then* $f(uv) = ext_v(f(u))$.

Another way of describing an advice function is as an *immediate* advice function:

Definition 4. *An* immediate *advice function over an alphabet* Σ *is a map* $g : \Sigma^* \to 2^{\Sigma}$.

An immediate advice function g over Σ induces an advice function f_g over Σ in the following way. We say a word w in Σ^* is *according to* the immediate advice function g at a non-empty prefix ua of w, if $a \in g(u)$. We say w is *always* according to g if w is according to g at all non-empty prefixes of w. The advice function f_g induced by g can now be defined for each $u \in \Sigma^*$ as

$$f_g(u) = \{v \in \Sigma^* \mid \forall wa \preceq v,\ uv \text{ is according to } g \text{ at } uwa\}.$$

One can verify that the two conditions for an advice function in Def. 3 are satisfied by f_g and hence that it is a valid advice function.

A conflict-tolerant controller (or feature implementation) 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 key difference 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 conflict-tolerant controller is modelled as an annotated transition system described below.

A *conflict-tolerant transition system* (CTTS for short) over an alphabet Σ is a tuple $\mathcal{T}' = (\mathcal{T}, N)$ where $\mathcal{T}$ is a deterministic transition system over Σ and $N \subseteq \rightarrow$ is a subset of transitions designated as *not-advised*.

A CTTS $\mathcal{T}'$ (as above) generates two type of languages: an "unconstrained" language, and a "constrained" language. Let $\mathcal{T}''$ be the transition system obtained from $\mathcal{T}'$ by deleting all the not-advised transitions (i.e. transitions in N) from $\mathcal{T}'$. Then starting from a configuration q in $\mathcal{T}'$ the *unconstrained* language generated by $\mathcal{T}'$, denoted $L_q(\mathcal{T}')$, is defined to be $L_q(\mathcal{T})$ and the *constrained* language generated by $\mathcal{T}'$, denoted $L^c_q(\mathcal{T}')$, is defined to be $L_q(\mathcal{T}'')$. For any word $w \in L(\mathcal{T})$ we use $L^c_w(\mathcal{T}')$ to denote the language $L^c_q(\mathcal{T}')$ where q is the unique configuration reached by $\mathcal{T}$ on w. The CTTS $\mathcal{T}'$ is said to be *complete* with respect to a language L if $L \subseteq L(\mathcal{T})$.

We can now define a "conflict-tolerant" controller.

Definition 5. *A* conflict-tolerant controller $\mathcal{C}$ *for a base system* $\mathcal{B}$ *over a partitioned alphabet* (Σ_s, Σ_e) *is a conflict-tolerant transition system over* Σ *that is complete with respect to* $L(\mathcal{B})$.

The controller $\mathcal{C}$ *is said to be* valid *with respect to* $\mathcal{B}$ *if the following conditions hold:*

- $\mathcal{C}$ *is* non-restricting*: If* $w \cdot e \in L(\mathcal{B})$ *for some environment event* $e \in \Sigma_e$*, then* $e \in L^c_w(\mathcal{C})$*. Thus the controller must not restrict any environment event* e *enabled in the base system after any system behaviour* w*.*
- $\mathcal{C}$ *is* non-blocking*: If* $w \in L(\mathcal{B})$*, then* $L^c_w(\mathcal{C}) \cap L_w(\mathcal{B}) \neq \{\epsilon\}$*. Thus the controller must not block the system after any system behaviour* w*.*

Let $\mathcal{C}$ be a conflict-tolerant controller for a base system $\mathcal{B}$ as above. Then we can view the product transition system $\mathcal{B} \parallel \mathcal{C}$ as a CTTS over Σ where the non-advised transitions are inherited from $\mathcal{C}$. Thus, a joint transition $((p, q), a, (p', q'))$ is not-advised iff the transition (q, a, q') is not-advised in $\mathcal{C}$.

Let f be an advice function over Σ. A conflict-tolerant controller $\mathcal{C}$ for $\mathcal{B}$ *satisfies* the conflict-tolerant specification f if for each $w \in L(\mathcal{B})$, $L^c_w(\mathcal{B} \parallel \mathcal{C}) \subseteq f(w)$. In other words, after any system behaviour w, if the base system follows the advice of $\mathcal{C}$, the resulting behaviours must all conform to the safety language $f(w)$.

Figure 3 shows two valid conflict-tolerant controllers for the example base system of Fig 2(a). The not-advised transitions are shown with dotted arrows. The first controller's advice is always to wait for a *timer* event and advise a **rel** event in response, no matter what behaviour has ensued in the past. The second controller on the other hand tries to maintain a "unit average" with respect to the last system action. We note that while both these controllers have rather different behaviours as conflict tolerant controllers, as classical controllers they are the same as the controller of Fig. 2(b).

We will return to these examples after seeing how to specify advice functions in temporal logic.

4 Conflict-Tolerant Specifications in CT-LTL

Linear-time Temporal Logic (LTL) [2] is a formalism for specifying systems whose behaviours are viewed as a linear sequence of events.

The syntax of an LTL formula over an alphabet Σ is given by:

$$\varphi ::= \top \mid \bot \mid a \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi U \varphi \mid \ominus\varphi \mid \varphi S \varphi$$

where $a \in \Sigma$.

Let w be a word over Σ and let $|w|$ denote the length of w. Let $i \in \{0, \ldots, |w|\}$ be a position in w corresponding to a prefix of w. Then the semantics of LTL is defined inductively via the relation $w, i \models \varphi$ ("w satisfies φ at position i") as follows.

For all $i \in \{0, \ldots, |w|\}$, $(w,i) \models \top$ and $(w,i) \not\models \bot$.

$$
\begin{array}{rcl}
(w,i) \models a & \iff & i > 0 \text{ and } w(i) = a. \\
(w,i) \models \neg\varphi & \iff & (w,i) \not\models \varphi. \\
(w,i) \models \varphi \vee \psi & \iff & (w,i) \models \varphi \text{ or } (w,i) \models \psi. \\
(w,i) \models \varphi \wedge \psi & \iff & (w,i) \models \varphi \text{ and } (w,i) \models \psi. \\
(w,i) \models \bigcirc\varphi & \iff & i < |w| \text{ and } (w,i+1) \models \varphi. \\
(w,i) \models \ominus\varphi & \iff & i > 0 \text{ and } (w,i-1) \models \varphi. \\
(w,i) \models \varphi U \psi & \iff & \exists j:\ i \leq j < |w|,\ (w,j) \models \psi \text{ and} \\
 & & \forall k:\ i \leq k < j \implies (w,k) \models \varphi. \\
(w,i) \models \varphi S \psi & \iff & \exists j, 0 \leq j \leq i, (w,j) \models \psi \text{ and} \\
 & & \forall k:\ j < k \leq i \implies (w,k) \models \varphi.
\end{array}
$$

We will make use of the derived operators $\Diamond\varphi = \top U \varphi$, $\diamondminus\varphi = \top S \varphi$, $\Box\varphi = \neg\Diamond\neg\varphi$, and $\boxminus\varphi = \neg\diamondminus\neg\varphi$. We also make use of the "weaker" version of $\bigcirc$ defined as $\odot\,\varphi = \bigcirc\varphi \vee \neg\bigcirc\top$. Thus $\odot\,\varphi$ says that either we are at the end of the word or there is a next position in the word and φ is satisfied there. Finally, we will make use of the abbreviation *init* which is defined to be $\neg\ominus\top$, which is true precisely at position 0 in any given word.

For an LTL formula φ and a word $w \in \Sigma^*$, we say $w \models \varphi$ iff $w, 0 \models \varphi$. We set $L(\varphi) = \{w \in \Sigma^* \mid w \models \varphi\}$. As an example, the LTL formula $\Box(\ominus a \implies b)$, specifies all words over $\{a, b\}$ in which every a is immediately followed by a b.

We will make use of the past fragment of LTL, obtained by disallowing the operators U and $\bigcirc$, and denote it by LTL_p. Thus the syntax of LTL_p formulas over the alphabet Σ is given by:

$$\varphi ::= \top \mid \bot \mid a \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \ominus\varphi \mid \varphi S \varphi$$

where $a \in \Sigma$. For an LTL_p formula φ and a word w in Σ^*, we write $w \models \varphi$ to denote the fact that $(w, |w|) \models \varphi$. Correspondingly, we denote the language associated with φ by $L(\varphi) = \{w \ \in \Sigma^* \mid w \models \varphi\}$.

Let us now discuss our way of specifying conflict-tolerant specifications in LTL. Our logic, denoted CT-LTL, is syntactically a fragment of LTL, as described below. Each formula in this logic will describe an *immediate* advice function.

Definition 6. *A formula of* CT-LTL *over an alphabet* Σ *is an LTL formula over* Σ *of the form*

$$\Box\Big(\bigwedge_{i \in \{1,\ldots,k\}} (\varphi_i \implies \psi_i)\Big)$$

where $k \geq 0$, φ_i *is an* LTL_p *formula, and each* ψ_i *is of the form* $\bigvee_{a \in X_i} \odot\, a$, *where* $X_i \subseteq \Sigma$.

The formula $\theta = \Box(\bigwedge_{i \in \{1,\ldots,k\}} (\varphi_i \implies \psi_i))$ above defines an immediate advice function g_θ given by

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

We adopt the convention that the intersection of an empty set is the full set of events Σ. We denote by f_θ the advice function f_{g_θ} induced by θ.

Let us now illustrate our logic with a couple of examples, with respect to the example base system $\mathcal{B}$ of Fig. 2(a).

Example 1. *The* CT-LTL *formula*

$$\Box(\mathit{timer} \implies \odot \mathbf{rel})$$

specifies the immediate advice function which advises a ***rel*** *event whenever the last event is a timer event and* Σ *otherwise.*

The conflict-tolerant controller of Fig. 3(a) can be seen to satisfy the conflict-tolerant spec given by the CT-LTL formula above, with respect to the example base system.

Example 2. *The* CT-LTL *formula below specifies an immediate advice that tries to maintain a unit average with respect to the last system event.*

$$\begin{aligned}\Box(\quad & ((\text{timer} \wedge (\mathit{init} \vee \ominus \boldsymbol{rel})) \implies \odot \boldsymbol{rel}) \wedge \\ & ((\text{timer} \wedge \ominus \boldsymbol{noRel}) \implies \odot \boldsymbol{relDouble}) \wedge \\ & ((\text{timer} \wedge \ominus \boldsymbol{relDouble}) \implies \odot \boldsymbol{noRel})).\end{aligned}$$

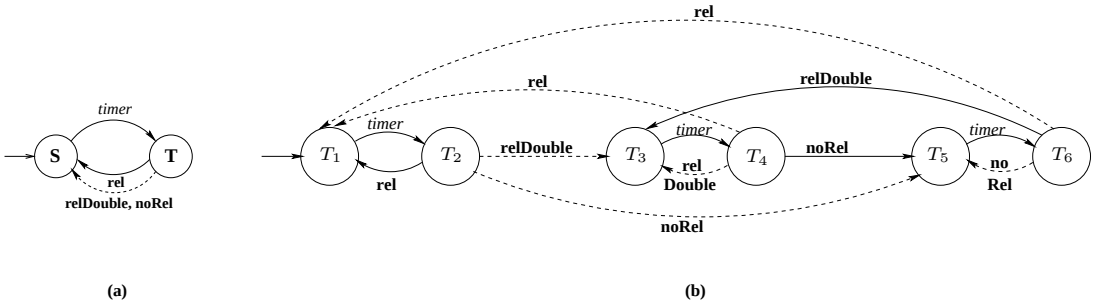

Figure 3: Two conflict-tolerant controllers for the example base system of Fig. 2(a).

The conflict-tolerant controller of Fig. 3(b) can be seen to satisfy the conflict-tolerant spec given by the CT-LTL formula above, with respect to the example base system.

We now consider the verification and synthesis problems induced by the logic CT-LTL.

Definition 7. *(*Verification Problem for CT-LTL*) Given a base system* $\mathcal{B}$ *over* Σ*, a conflict-tolerant controller* $\mathcal{C}$ *for* $\mathcal{B}$*, and a* CT-LTL *formula* θ*, check whether* $\mathcal{C}$ *is a valid conflict-tolerant controller for* $\mathcal{B}$ *which satisfies the advice function* f_θ *with respect to* $\mathcal{B}$*.*

Definition 8. *(*Synthesis Problem for CT-LTL*) Given a base system $\mathcal{B}$ over Σ, and a* CT-LTL *formula θ, check whether there exists a valid conflict-tolerant controller for $\mathcal{B}$ which satisfies the advice function f_θ with respect to $\mathcal{B}$; and if so, construct one.*

We would like to emphasize that these problems for the logic CT-LTL are different from the corresponding problems for classical LTL. Consider the verification problem for a CT-LTL formula θ. Viewed as an LTL formula, θ defines a safety language $L(\theta)$. However as a CT-LTL formula, θ defines an *advice function*, which is in general a richer object than a safety language. In fact, it is easy to see that the safety language $L(\theta)$ corresponds to the initial safety cone of the advice function f_θ induced by θ: in other words, $L(\theta) = f_\theta(\epsilon)$. Thus it is not sufficient to simply check whether $\mathcal{B} \parallel \mathcal{C}$ satisfies θ as an LTL formula to be able to conclude that $\mathcal{C}$ satisfies f_θ with respect to $\mathcal{B}$. Nevertheless, in the subsequent sections we show that we solve these more general problems for CT-LTL using essentially the same techniques as for classical LTL.

5 From an LTL_p formula to a DFA

The main step in solving the verification and synthesis problems for a CT-LTL specification is the construction of the "formula automaton" for the LTL_p sub formulas of the specification. A *formula automaton* $\mathcal{A}_\theta$ of θ is a transition system whose state space is the set of all "consistent" set over θ. And the transition relation of $\mathcal{A}_\theta$ is defined in a "consistent way" the details of which are given in Def. 11.

Let us now define some notions which we require subsequently. Let φ be an LTL_p formula over the alphabet Σ and let $psf(\varphi)$ be the set of *positive sub formulas* of φ, i.e. sub formulas of φ which are not of the form $\neg\psi$. Without loss of generality we assume that ψ does not contain any double negation sub-formulas, i.e formulas of the form $\neg\neg\psi$. Then the *closure* of the formula φ is defined as follows:

Definition 9. *The closure of an* LTL_p *formula φ, denoted $cl(\varphi)$, is defined as*

$$cl(\varphi) = X \cup \{\neg\beta \mid \beta \in X\}$$

where $X = psf(\varphi) \cup \{\ominus(\psi S \mu),\ \psi S \mu \in psf(\varphi)\} \cup \Sigma \cup \{\top\}$.

Now given the set $cl(\varphi)$ we define an *atom* of the formula φ as follows:

Definition 10. *An* atom *of an* LTL_p *formula φ is a subset A of $cl(\varphi)$ satisfying*

the following conditions.

0. $\top \in A$
1. $\forall \neg\psi \in cl(\varphi),\ \psi \in A \iff \neg\psi \notin A$
2. $\forall (\psi \vee \mu) \in cl(\varphi),\ (\psi \vee \mu) \in A \iff \psi \in A$ *or* $\mu \in A$
3. $\forall (\psi \wedge \mu) \in cl(\varphi),\ (\psi \vee \mu) \in A \iff \psi \in A$ *and* $\mu \in A$
4. $\forall (\psi\ S\mu) \in cl(\varphi),\ (\psi\ S\mu) \in A \iff \mu \in A,$ *or,* $\psi \in A$ *and* $\ominus(\psi\ S\mu) \in A$
5. $|\Sigma \cap A| \leq 1$

We denote the set of all atoms over φ by $atoms(\varphi)$.

As simple exercises we construct the sets $cl(\varphi)$ and $atoms(\varphi)$ of the LTL_p formula $\varphi = bSa$ over $\Sigma = \{a, b\}$ in Example 3 and Example 4.

Example 3. ***(Closure)*** *The closure of the formula φ is as given below.*

$$\begin{aligned} psf(\varphi) =& \{a, b, bSa\} \\ X =& \{\top, a, b, bSa, \ominus(bSa)\} \\ cl(\varphi) =& \{\top, \bot, a, b, bSa, \ominus(bSa), \neg a, \neg b, \neg(bSa), \neg\ominus(bSa)\} \end{aligned}$$

Example 4. ***(atom)*** *The atoms of over φ are as given below.*

$$\begin{aligned} A =& \{\top, \neg b, \neg a, \neg(bSa), \neg\ominus(bSa)\} \\ B =& \{\top, b, \neg a, bSa, \ominus(bSa)\} \\ C =& \{\top, b, \neg a, \neg(bSa), \neg\ominus(bSa)\} \\ D =& \{\top, \neg b, a, bSa, \ominus(bSa)\} \\ E =& \{\top, \neg b, a, bSa, \neg\ominus(bSa)\} \\ F =& \{\top, \neg b, \neg a, (\neg b)Sa, \ominus(bSa)\} \\ atoms(\varphi) =& \{A, B, C, D, E, F\} \end{aligned}$$

For any LTL_p formula φ once we have $cl(\varphi)$ and $atoms(\varphi)$ in place we define the *formula automaton* for φ as follows:

Definition 11. *The formula automaton $\mathcal{A}_\varphi = \langle Q, \Sigma, s, F, \rightarrow \rangle$ for a given LTL_p formula φ is given by:*

$$\begin{aligned} Q =& atoms(\varphi) \\ s =& \{\psi \in cl(\varphi) \mid \epsilon \models \psi\} \\ F =& \{A \in atoms(\varphi) \mid \varphi \in A\} \\ \rightarrow \subseteq& Q \times \Sigma \times Q \text{ given by :} \\ (A, a, A') \in \rightarrow & \text{ iff } a \in A' \text{ and } \forall \ominus\psi \in cl(\varphi),\ \psi \in A \Leftrightarrow \ominus\psi \in A' \end{aligned}$$

As a transition system the finite state automaton $\mathcal{A}_\theta$ differs slightly from the definition of transition system given in section 2 as we designate a subset of the set of all states of $\mathcal{A}_\theta$ as the set of final states. The *language* generated by $\mathcal{A}_\theta$, denoted by $L(\mathcal{A}_\theta)$, is the set of all words w on which $\mathcal{A}_\theta$ has a run starting at the initial state and ending in a final state.

Example 5. ***(construction)*** *Consider the* LTL_p *formula* $\varphi = bSa$ *over* $\Sigma = \{a, b\}$. *Then* $cl(\varphi)$ *is as given in Example 3 and* $atoms(\varphi)$ *is as given in Example 4. The formula automaton* $\mathcal{A}_\varphi$ *is shown in Figure 4.*

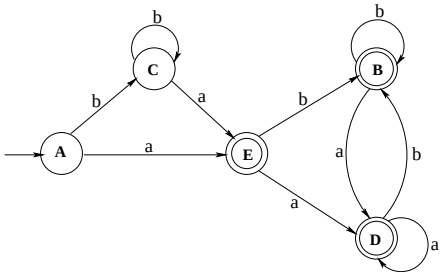

Figure 4: Formula automaton for bSa. The state F is unreachable and is not shown.

Lemma 1. *Let* w *be a word in* Σ^*. *Then there exists a run for* w *in* $\mathcal{A}_\varphi$.

Proof. Let $w = a_1 \cdots a_n$. For each $i \in \{0, \ldots, n\}$ let A_i be the set $\{\psi \in cl(\varphi) \mid (w, i) \models \psi\}$. It is clear from the definition of atoms that each A_i is an atom. Also we can easily verify that each $1 \leq i \leq n$, $(A_{i-1}, a, A_i) \in \rightarrow$. Further $A_0 = s$. Therefore $A_0, A_1, \ldots, A_n$ is a run of w in $\mathcal{A}_\varphi$. □

Lemma 2. *Let* $w = a_1 \cdots a_n$ *be a word in* Σ^* *and* $A_0, \ldots, A_n$ *be a run on* w *in* $\mathcal{A}_\varphi$. *Then for all* $\psi \in cl(\varphi)$ *and for all* $i \in \{0, \ldots, n\}$, $(w, i) \models \psi \iff \psi \in A_i$.

Proof. Proof is by induction on the structure of ψ.
0. $\psi = \top$

For all $i \in \{0, \ldots, n\}, (w, i) \models \top$ by semantics of LTL_p formula and $\top \in A_i$ by definition of an atom.
1. $\psi = a, a \in \Sigma$

$$\begin{aligned} (w, i) \models \psi &\iff i > 0 \text{ and } w(i) = a \\ &\iff a \in A_i \text{ (since } (A_{i-1}, a_i, A_i) \in \rightarrow) \end{aligned}$$

2. $\psi = \neg\mu$

$$\begin{aligned} (w, i) \models \psi &\iff (w, i) \not\models \mu \\ &\iff \mu \notin A_i \text{ (induction hypothesis)} \\ &\iff \neg\mu \in A_i \text{ (definition of atom)} \end{aligned}$$

3. $\psi = \mu \vee \nu$

$$\begin{aligned}(w,i) \models \psi &\iff (w,i) \models \mu \text{ or } (w,i) \models \nu \\ &\iff \mu \in A_i \text{ or } \nu \in A_i \text{ (induction hypothesis)} \\ &\iff \mu \vee \nu \in A_i \text{ (definition of atom)}\end{aligned}$$

4. $\psi = \ominus\mu$

$$\begin{aligned}(w,i) \models \psi &\iff i > 0 \text{ and } (w, i-1) \models \mu \\ &\iff \mu \in A_{i-1} \text{ (induction hypothesis)} \\ &\iff \ominus\mu \in A_i \text{ (since } (A_{i-1}, a_i, A_i) \in \rightarrow)\end{aligned}$$

5. $\psi = \mu S \nu$

Suppose $(w,i) \models \psi$. We need to show that $\psi \in A_i$. As $(w,i) \models \psi$ there exists a $k : 0 \leq k \leq i$ such that $(w,k) \models \nu$ and for all $j, k < j \leq i$, $(w,j) \models \mu$. By a second induction on $(i-k)$, we prove that $\psi \in A_i$.
Base case: $i - k = 0$. Then $k = i$ and $(w,i) \models \nu$. Then by the main induction hypothesis $\nu \in A_i$ and therefore $\psi \in A_i$ by the definition of atom.
Induction step: $i - k = l > 0$. Without loss of generality we assume that $(w,i) \not\models \nu$. Then we have:

$$\Rightarrow (w, i-1) \models \mu S \nu \text{ and } (w,i) \models \mu \tag{1}$$

$$\Rightarrow \mu S \nu \in A_{i-1} \text{(by second. IH) and } \mu \in A_i \text{ (by main IH)} \tag{2}$$

$$\Rightarrow \ominus(\mu S \nu) \in A_i \text{ (as } (A_{i-1}, a_i, A_i) \in \rightarrow) \tag{3}$$

$$\Rightarrow \mu S \nu \in A_i \text{ (from the (2),(3) and the definition of atom)}$$

Conversely suppose $\psi \in A_i$. We need to show that $(w,i) \models \psi$. Since $A_0, \ldots, A_i$ is a run in $\mathcal{A}_\psi$ starting from the initial state there must exist a $k \leq i$ such that $\nu \in A_k$. Choose the greatest such k. Now by a second induction on $(i-k)$ we prove that $(w,i) \models \psi$.
Base case: $i - k = 0$. Then $k = i$ and hence $\nu \in A_i$. Then by main induction hypothesis $(w,i) \models \nu$ and therefore $(w,i) \models \mu S \nu$ as well.
Induction step: $i - k = l > 0$. Once again without loss of generality we assume that $(w,i) \not\models \nu$. Then we have

$$(\mu S \nu) \in A_i \text{ and } \nu \notin A_i \tag{4}$$

$$\Rightarrow i > 0,\ \mu \in A_i \text{ and } \ominus(\mu S \nu) \in A_i \text{ (by def. of atom)} \tag{5}$$

$$\Rightarrow (w,i) \models \mu \text{ (by main IH)} \tag{6}$$

$$\Rightarrow \mu S \nu \in A_{i-1} \text{ (by (5) and } (A_{i-1}, a_i, A_i) \in \rightarrow) \tag{7}$$

$$\Rightarrow (w, i-1) \models \mu S \nu \text{ (by second IH)} \tag{8}$$

$$\Rightarrow (w,i) \models \mu S \nu \text{ (from (6),(8) and def. of modality } S)$$

□

The following theorem proves the language equivalence between the one generated by the formula ψ and the one accepted by $\mathcal{A}_\psi$.

Theorem 1. *Let ψ be an* LTL_p *formula and let $\mathcal{A}_\psi$ be the its formula automaton. Then $L(\mathcal{A}_\psi) = L(\psi)$.*

Proof. ($\Rightarrow$) Let $w \in \mathcal{A}_\psi$. Then there exists a accepting run of $\mathcal{A}_\psi$ on w. So let $A_0, \ldots, A_{|w|}$ be the accepting run of $\mathcal{A}_\psi$ on w. As $A_{|w|}$ is a final state $\psi \in A_{|w|}$ and therefore by lemma 2 we have that $w \models \psi$.

($\Leftarrow$) Suppose $w \models \psi$. Now we need to show that $w \in L(\mathcal{A}_\psi)$. From lemma 1 it follows that there exists a run $A_0, \ldots, A_{|w|}$ of $\mathcal{A}_\psi$ on w. Now by lemma 2 we have that $\psi \in A_{|w|}$. Thus the run $A_0, \ldots, A_{|w|}$ of the formula automaton $\mathcal{A}_\psi$ on w is accepting and therefore $\psi \in L(\mathcal{A}_\psi)$. □

Theorem 2. *Let ψ be an* LTL_p *formula and let $\mathcal{A}_\psi$ be the its formula automaton. Then $\mathcal{A}_\psi$ is both deterministic and complete with respect to Σ.*

Proof. Proof of $\mathcal{A}_\psi$ is complete with respect to Σ follows from lemma 1. Now suppose that $\mathcal{A}_\psi$ is not deterministic. Then there exists two different runs $A_0, \ldots, \mathcal{A}_n$ and $A'_0, \ldots, A'_n$ on some word $w \in \Sigma^*$. Now by lemma 2, $A_i = A'_i$ for every $0 \leq i \leq n$, which is a contradiction. □

6 Verification

In this section we address verification problem for CT-LTL.

Theorem 3. *Given a base system $\mathcal{B}$ over Σ, a conflict-tolerant specification θ in* CT-LTL *and a conflict-tolerant controller $\mathcal{C}$ over Σ for $\mathcal{B}$, one can check if $\mathcal{C}$ is a valid controller for $\mathcal{B}$ satisfying θ.*

Proof. Let θ the specification given by $\Box(\bigwedge(\varphi_i \Rightarrow \psi_i))$. We construct the automaton $\mathcal{A}_{(\bigwedge \varphi_i)}$ for the LTL_p formula $\bigwedge \varphi_i$ and show that $\mathcal{C}$ is a valid controller for $\mathcal{B}$ satisfying θ iff there does not exist a state (p, q, r) reachable from the initial state in the synchronized product $\mathcal{A}' = \mathcal{B} \parallel \mathcal{C} \parallel \mathcal{A}_{(\bigwedge \varphi_i)}$ satisfying one of the following conditions.

- *$\mathcal{C}$ is restricting:* There exists an event $e \in \Sigma_e$ enabled at p in $\mathcal{B}$, but is not advised at q in $\mathcal{C}$.
- *$\mathcal{C}$ is blocking:* There is no event $c \in \Sigma$ which is both enabled at p in $\mathcal{B}$ and advised at q in $\mathcal{C}$.
- *$\mathcal{C}$ does not satisfy θ:* There exists an event $c \in \Sigma$ enabled at p in $\mathcal{B}$ and advised at q in $\mathcal{C}$, but $c \not\models \bigwedge_{\varphi_i \in r} \psi_i$.

If no state (p, q, r) exists in $\mathcal{A}'$ such that the conditions 1 or 2 hold, then clearly $\mathcal{C}$ is a valid CT-controller for $\mathcal{B}$. Conversely if $\mathcal{C}$ is a valid CT-controller for $\mathcal{B}$, then it is easy to see there does not exist a state in $\mathcal{A}'$ where the conditions 1 or 2 hold. If no state (p, q, r) exists in $\mathcal{A}'$ such that the condition 3 holds, then all advises in $\mathcal{C}$ are according to the specification θ and hence $\mathcal{C}$ satisfies θ. Conversely if $\mathcal{C}$ satisfies θ, then no such state exists in $\mathcal{A}'$ where condition 3 holds. Now checking for such a state in $\mathcal{A}'$ which is reachable from start state can easily be done by doing a reachability analysis on $\mathcal{A}'$. □

Let us now illustrate the verification problem with an example.

Example 6. *Consider the base system $\mathcal{B}$ shown in Figure 2(a), the feature specification θ given in Example 1 and a conflict-tolerant controller $\mathcal{C}$ for $\mathcal{B}$ shown in Figure 3(a). Atoms and other details of the construction are given below. The deterministic finite-state transition system $\mathcal{A}_{\bigwedge \varphi_i}$ is shown in Figure 5 and the synchronized product $\mathcal{B} \parallel \mathcal{C} \parallel \mathcal{A}_{\bigwedge \varphi_i}$ is shown in Figure 6. Now we can see that no state (p, q, r) as mentioned in theorem 3 is reachable in $\mathcal{B} \parallel \mathcal{C} \parallel \mathcal{A}_{\bigwedge \varphi_i}$ and hence $\mathcal{C}$ is a valid CTC for $\mathcal{B}$ satisfying θ.*

$$\begin{aligned} \bigwedge \varphi_i &= \{timer\}. \\ psf(\bigwedge \varphi_i) &= \{timer\}. \\ cl(\bigwedge \varphi_i) &= \{\top, \bot, \mathbf{rel}, \mathbf{relDouble}, \mathbf{noRel}, timer, \\ &\qquad \neg\mathbf{rel}, \neg\mathbf{relDouble}, \neg\mathbf{noRel}, \neg timer\}. \end{aligned}$$

The set of atoms is given below (A is the initial state).

$$\begin{aligned} A &= \{\top, \neg\mathbf{rel}, \neg\mathbf{relDouble}, \neg\mathbf{noRel}, \neg timer\}. \\ B &= \{\top, \mathbf{rel}, \neg\mathbf{relDouble}, \neg\mathbf{noRel}, \neg timer\}. \\ C &= \{\top, \neg\mathbf{rel}, \mathbf{relDouble}, \neg\mathbf{noRel}, \neg timer\}. \\ D &= \{\top, \neg\mathbf{rel}, \neg\mathbf{relDouble}, \mathbf{noRel}, \neg timer\}. \\ E &= \{\top, \neg\mathbf{rel}, \neg\mathbf{relDouble}, \neg\mathbf{noRel}, timer\}. \end{aligned}$$

7 Feasibility and Synthesis

Now that we have addressed the verification problem let us now turn our attention to feasibility and synthesis problems in our setting. Informally the controller feasibility problem is that given a base system and a CT-LTL specification does there exist a valid conflict-tolerant controller for the base system which meets the specification. And the synthesis problem is that if there exists a valid conflict-tolerant controller for given a base system and a CT-LTL specification is it possible the synthesise one. We answer the feasibility question affirmatively and also give a procedure for synthesising a conflict-tolerant controller for a given base system from a CT-LTL specification.

Let $\mathcal{B}$ be a base system and let θ be the specification given by $\Box(\bigwedge_{i=1}^{i=n}(\varphi_i \Rightarrow \psi_i))$. Let $\mathcal{A}_i = \langle Q, s, F, \rightarrow \rangle$ be the formula automaton of φ_i (see section 5). Then we construct the conflict tolerant transition system $\mathcal{C}_i = (\mathcal{A}_i, N_i)$ as follows: let A, A' be atoms in $\mathcal{A}_i$ and let $a \in \Sigma$. Then $(A, a, A') \in N_i$ iff $\varphi_i \in A$ and $a \not\models \psi_i$. Let $\mathcal{C} = \mathcal{C}_1 || \ldots || \mathcal{C}_n$. Then we can easily prove that $\mathcal{C}$ is a valid controller for $\mathcal{B}$ iff for every state (p, q) in $\mathcal{B}||\mathcal{C}$ the following conditions hold:

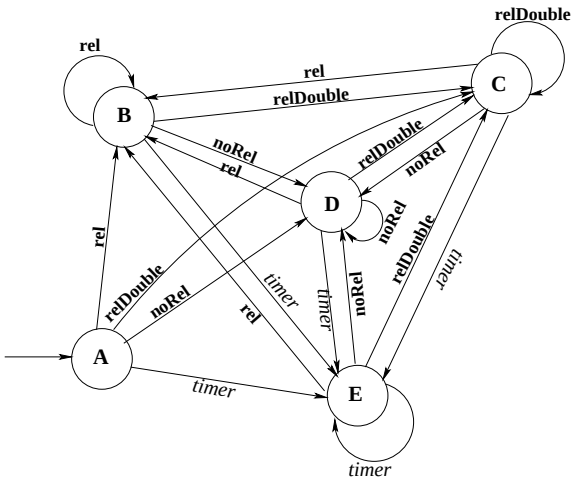

Figure 5: Deterministic finite-state transition system $\mathcal{A}_{\bigwedge \varphi_i}$ for specification given in Example 1.

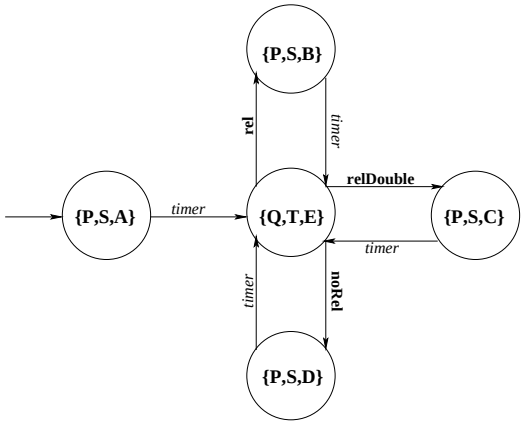

Figure 6: The synchronized product of the base system $\mathcal{B}$ (Figure 2(a)), controller $\mathcal{C}$ of Figure 3(a) and the formula automaton $\mathcal{A}_{\bigwedge \varphi_i}$ of Figure 5.

- $\mathcal{C}$ *is non restricting:* There does not exist an event $e \in \Sigma_e$ such that e is enabled at p in $\mathcal{B}$ but $(e, 0) \not\models \bigwedge_{\varphi_i \in q} \psi_i$.
- $\mathcal{C}$ *is non blocking:* There is an event $c \in \Sigma$ which is enabled at p in $\mathcal{B}$ and $(c, 0) \models \bigwedge_{\varphi_i \in q} \psi_i$.

One can also argue that a valid conflict-tolerant controller for $\mathcal{B}$ meeting the specification θ exists iff $\mathcal{C}$ is a valid controller for $\mathcal{B}$. As checking whether $\mathcal{C}$ is a valid controller for $\mathcal{B}$ is a simple reachability analysis we have that:

Theorem 4. *Given a base system $\mathcal{B}$ and a conflict-tolerant specification θ over Σ the controller feasibility problem for $\mathcal{B}$ meeting the specification θ is decidable.*

Theorem 5. *Given a base system $\mathcal{B}$ and a conflict-tolerant specification θ over Σ one can synthesise a valid conflict-tolerant controller for $\mathcal{B}$ meeting the specification θ provided such a controller is feasible.*

We illustrate the construction of the formula automaton with the following example.

Example 7. *Consider the base system $\mathcal{B}$ shown in Figure 2(a) and the* CT-LTL *specification given in Example 1. Atoms and other details of the construction are given in Example 6. Then the valid CTTS meeting the specification obtained with our construction with state E as the final state is shown in Figure 7.*

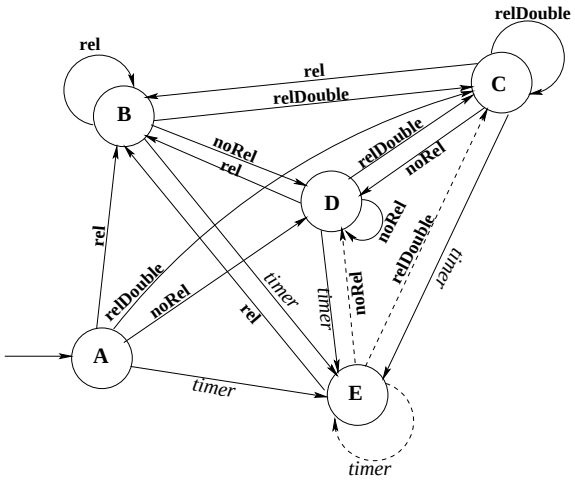

Figure 7: Synthesized conflict-tolerant controller for the specification $\Box(timer \implies \odot \mathbf{rel})$.

8 Conclusion and Future work

In this work we have proposed a way of specifying conflict-tolerant specifications in temporal logic, and given algorithmic solutions to the associated verifying and synthesis problems. Our formalism and associated methodology is therefore a useful addition to the conflict-tolerant framework proposed in [1].

As a future work we would like to look at the integration of verification into standard model checkers like SPIN and SMV. We would also like to explore how to extend our framework to real-time setting.

References

[1] Deepak D'Souza and Madhu Gopinathan. Conflict-tolerant features. In Aarti Gupta and Sharad Malik, editors, *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 227–239. Springer, 2008.

[2] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.

[3] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344. IEEE Computer Society, 1986.

[4] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The glory of the past. In Rohit Parikh, editor, *Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer, 1985.

[5] K. C. Wong, J. G. Thistle, R. P. Malhamé, and H. H. Hoang. Supervisory control of distributed systems: Conflict resolution. *Discrete Event Dynamic Systems*, 10(1-2):131–186, 2000.

[6] Y. L. Chen, S. Lafortune, and F. Lin. Modular Supervisory Control with Priorities for Discrete Event Systems. In *In IEEE Conference on Decision and Control*, pages 409–415, 1995.