Conflict-Tolerant Features*

Deepak D'Souza and Madhu Gopinathan

Indian Institute of Science,
Bangalore, India
{deepakd,gmadhu}@csa.iisc.ernet.in

Abstract. 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 way that guarantees the "maximal" use of each feature. The methodology is 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 composition scheme for such features, which ensures that each feature is maximally utilized. We also provide a formal framework for specifying, verifying, and synthesizing such features. In particular we obtain a compositional technique for verifying systems developed in this framework.

1 Introduction

In this paper we consider systems that are composed of a base system along with multiple "features" or "controllers," each of which are meant to advise the system on how to adhere to their individual feature specifications. Such system models are common in software intensive domains such as telecommunications and automotive electronic control. 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 system to perform [1,2]. Such conflicts can be resolved by redesigning one of the features to satisfy a relaxed specification. However redesigning existing features is often not feasible in practice [3]. Redesign would also defeat the purpose of software product lines [4], which aim to improve software reuse by composing features to derive multiple products from a family of features. Another resolution technique is to suspend the feature with lower priority, and 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.

In this paper we propose a formal framework for developing such systems in a way that overcomes some of these problems. The framework is based on a notion of "conflict-tolerance", which simply requires features to be "resilient" or "tolerant" with regard to violations of their advice due to conflicts with other features.

* This research was partially supported by a grant from India Science Lab, General Motors Corporation.

© Springer-Verlag Berlin Heidelberg 2008

Thus, unlike a classical feature, a conflict-tolerant feature observes that its advice has been overridden, takes into account the violating event, and proceeds to offer advice for *subsequent* behavior of the system. Our methodology includes a way of specifying features, as well as a compositional verification technique for checking whether a feature implementation satisfies its specification.

The starting point in this framework is the notion of a *conflict-tolerant specification* of a feature. A classical specification (which we will assume in this paper to be safety specifications of linear-time behavior) can be viewed as a prefix-closed language of finite words containing all the system behaviors which are considered "safe". This can be pictured as a safety "cone" in the tree representing all possible behaviors, 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* behavior w of the base system, a safety cone comprising all future behaviors that are considered safe, *after* the system has exhibited behavior w (Fig. 1 (b)).

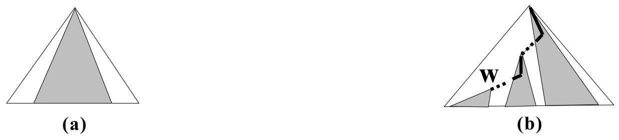

Fig. 1. The conflict-tolerant specification on the right advises on how to extend w even though its advice has been overridden (dashed line) in the past when generating w

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 (by the event "**rel**") a single unit of medication in response to a "*timer*" event from the environment. A classical specification for this feature may be given by the transition system shown in Fig. 2 (a). The transition systems shown in Fig. 2 (b) and (c) denote conflict-tolerant specifications that both induce the same classical specification shown in (a). The dashed transitions are to be read as "not-advised", and their role is to keep track of events that violate the advice at a given state. Thus in specification (b), after an initial *timer* event, the advised action is **rel**; however, if the event **no-rel** occurs against its advice, the specification moves to the lightly shaded copy where on receiving another *timer* event it advises the action **rel-double**. The second specification thus attempts to maintain a unit average in every window of two cycles (it may, for example, be releasing oxygen), while the third specification does not attempt to do this (it may be releasing doses of insulin).

A conflict-tolerant feature implementation can be viewed as a 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 behavior w of the base system, the behaviors of the base system that are according to the advice of the feature implementation, are all contained in the safety cone prescribed by the specification for w.

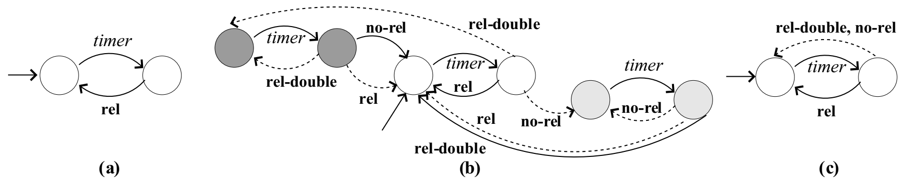

Fig. 2. The specification (a) is a classical specification whereas (b) and (c) are conflict-tolerant specifications. If "release double" (event **rel-double**) occurs against the advice of specification (b), it moves to the darkly shaded copy and advises "no release" (event **no-rel**) in the next cycle; if **no-rel** occurs against its advice, it moves to the lightly shaded copy and advises **rel-double** in the next cycle.

We give decision procedures to solve the natural synthesis and verification problems in this setting. In particular, we give a procedure to check whether a finite-state implementation satisfies a given finite-state conflict-tolerant specification, with respect to a given base system.

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, which 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. Together with our verification procedure, this gives us a compositional way of verifying the composed system, since once the individual features have each been verified to conform to their specifications, the composed system is guaranteed to be correct (in the sense above) "by construction."

Related Work. In [5], 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 is detected by checking whether a state, in which the features advise conflicting system actions, is reached. For a survey, see [1]. The problem of conflict detection is addressed in [6], where features are specified using temporal logic and conflict is detected automatically at the specification stage.

We now focus on previous work addressing conflict resolution. Our approach of viewing features as discrete event controllers [7] follows that of [8,9]. In both these works, the main issue addressed is that of resuming the advice of a controller once it has been overridden due to conflict with a higher priority controller. In [8] (see also [10]), when the lower priority controller (say $\mathcal{C}_2$) is suspended, the behavior of the base system is masked from $\mathcal{C}_2$. The resolution mechanism resumes $\mathcal{C}_2$ when it determines that the base system has reached a state (based on language equivalence) from which it is safe to accept the advice of $\mathcal{C}_2$. The drawback of this scheme is that the base system may never reach a state from which the advice of $\mathcal{C}_2$ can be accepted, and even if such a state was reached eventually, the utility of the controller is lost during the period of suspension.

The work of [9] is closer to ours, in that the 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 behavior, till it sees a specified event that brings it back to an in-spec state. Thus, unlike our controllers, these controllers are designed to work with only certain anticipated conflicts, and moreover do not offer any useful advice in out-of-spec states.

In [11], a rule-based feature model and composition operators for resolving conflicts based on prioritization is presented. Their work is closest to ours in that it implicitly contains the notion of conflict-tolerance with a similar resolution mechanism. However the notion of a conflict-tolerant specification (as against the feature implementation itself) is absent in their work, while it is central in ours. In the absence of a specification one cannot address the important problems of verification and synthesis. With respect to their notion of "weak" (in a sense conflict-tolerant) invariants of features, the feature model is not obliged to offer advice on how to restore the invariant in case of violations.

The rest of the paper is structured as follows: After preliminary definitions, in Sect. 3 we view features as controllers and illustrate conflict between features. We then introduce the notion of conflict-tolerance in Sect. 4 and address the synthesis and verification problems in Sect. 5. Finally in Sect. 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. We denote the empty word by ϵ. A *language* over Σ is a subset of Σ^*. We write $v \cdot w$ (or simply vw) to denote the concatenation of two words v and w. Let L be a language over Σ and let v be a word over Σ. We define the set of *extensions* of v in L to be $ext_v(L) = \{w \in \Sigma^* \mid v \cdot w \in L\}$.

A *transition system* $\mathcal{T}$ over Σ is a tuple $(Q, s, \rightarrow)$, where Q is a set of states, $s \in Q$ is the start state, and $\rightarrow \subseteq Q \times \Sigma \times Q$ is a Σ-labeled transition relation. A *run* of $\mathcal{T}$ on a word $w = a_0 a_1 \cdots a_n$ starting from a state q, is a sequence $q_0, q_1, \ldots, q_{n+1}$ of states in Q such that $q_0 = q$, and for each $i \in \{0, \ldots, n\}$, we have $q_i \xrightarrow{a_i} q_{i+1}$. The language generated by $\mathcal{T}$, denoted $L(\mathcal{T})$, is the set of all words w on which $\mathcal{T}$ has a run starting from s. The language generated by $\mathcal{T}$ starting from a state $q \in Q$, denoted by $L_q(\mathcal{T})$, is the set of all words on which $\mathcal{T}$ has a run starting from q. We say the transition system $\mathcal{T}$ is *complete* (respectively *deterministic*) if for each $q \in Q$ and $a \in \Sigma$, there exists (respectively, exists at most one) q' such that $q \xrightarrow{a} q'$. 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 in $\mathcal{T}$ after generating w. Then, we define $L_w(\mathcal{T}) = L_q(\mathcal{T})$.

Finally, for transition systems $\mathcal{T}_1 = (Q_1, s_1, \rightarrow_1)$ and $\mathcal{T}_2 = (Q_2, s_2, \rightarrow_2)$ over Σ, we define the *synchronized product* of $\mathcal{T}_1$ and $\mathcal{T}_2$, denoted $\mathcal{T}_1 \| \mathcal{T}_2$, to be the transition system $(Q_1 \times Q_2, (s_1, s_2), \rightarrow)$ over Σ, where $(p_1, p_2) \xrightarrow{a} (q_1, q_2)$ iff $p_1 \xrightarrow{a}_1 q_1$ and $p_2 \xrightarrow{a}_2 q_2$.

3 Features as Controllers

In this section we elaborate on the view of features as modular discrete event controllers [7], as proposed in [9,8].

In this paper, we focus on "safety" specifications. A *safety* specification over an alphabet Σ is a prefix-closed language over Σ. 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 Sect. 4.

Definition 1 (Advice Function). *An advice function over Σ is a function $f : \Sigma^* \rightarrow 2^{\Sigma^*}$ such that $f(\epsilon)$ is prefix-closed language, and is* ***consistent*** *in the sense that for all $vw \in f(\epsilon)$ we have $f(vw) = ext_w(f(v))$.*

A safety specification L over Σ induces an advice function f_L given by $f_L(v) = ext_v(L)$. 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 $f^i : \Sigma^* \rightarrow \Sigma$ given by

$$f^i(v) = \{a \in \Sigma \mid \exists w \in \Sigma^* : aw \in f(v)\}.$$

We say a finite word w is *according to* an immediate advice f^i if for each prefix va of w, we have $a \in f^i(v)$. A deterministic transition system $\mathcal{T}$ over Σ induces an advice function $f_{\mathcal{T}}$ given by $f_{\mathcal{T}}(w) = L_w(T)$ for all $w \in L(\mathcal{T})$, and ϵ otherwise. We will say a safety specification L over Σ is *regular* if it is given by a deterministic finite-state transition system $\mathcal{S}$ 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. We model systems over Σ by viewing their executions as a repeated cycle in which an environment event is sampled and a system event is performed in response to it. For simplicity we assume that exactly one environment action is sampled in each cycle.

Definition 2 (Base System). *A base system (or plant) over Σ is a deterministic finite-state transition system $\mathcal{B}$ over Σ, which is*

- ***alternating*** *in that $L(\mathcal{B}) \subseteq (\Sigma_e \cdot \Sigma_s)^* \cup ((\Sigma_e \cdot \Sigma_s)^* \cdot \Sigma_e)$.*
- ***non-blocking*** *in that whenever $w \in L(\mathcal{B})$ there exists $c \in \Sigma$ such that $wc \in L(\mathcal{B})$.*

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

– *$\mathcal{C}$ is **non-restricting**: If $w \in L(\mathcal{B}\|\mathcal{C})$ and $w \cdot e \in L(\mathcal{B})$ for some environment event $e \in \Sigma_e$, then $w \cdot e \in L(\mathcal{C})$. Thus the controller must not restrict any environment event e enabled in the base system after any controlled behavior w.*
– *$\mathcal{C}$ is **non-blocking**: If $w \in L(\mathcal{B}\|\mathcal{C})$, then $wc \in L(\mathcal{B}\|\mathcal{C})$ for some $c \in \Sigma$. Thus the controller must not block the base system after any controlled behavior w.*

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 a lift system (in a building of only two floors – floor 0 and floor 1) and two of its features, adapted from a case study in [12]. Figure 3 shows the base system model. We denote the environment events in italics and system events in bold. In order to avoid clutter, we use a Statechart-like notation. An arrow from a box (dashed rectangle) to another box represents arrows between states with the same label in the two boxes. For example, the arrow labeled **open** represents two arrows: one between states labeled 1, and another between states labeled 0 in the boxes C and O. 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, and those events a for which the self-loop has a label $\sim a$. Thus the state 0 in box C has a transition to itself on the system event $\mathbf{nop_s}$ and on all environment events except *ttfull* and *ttnotfull*. The initial state is shown by an incoming arrow. To keep the figure simple we have not shown the base system as generating alternating environment and system events. However, we consider only the alternating behaviors.

The event fpress_i occurs when a user presses the button on floor i and the event cpress_i occurs when a user presses the car button for floor i inside the lift. The event *ttfull* indicates that the lift is two-thirds full and the event *ttnotfull* indicates that the lift is not two-thirds full. The events nop_e and $\mathbf{nop_s}$ respectively denote that the environment and the system has not performed any action.

The base system is typically run with several controllers including a "vanilla" controller which would keep track of the current direction of travel and require the base system to service all pending requests from floors and from within the lift along that direction before changing direction. However, we will focus on a controller for "executive floor" feature which requires that the requests from the "executive floor" (say, floor 1) must be serviced before other floor requests. Figure 4 shows a possible specification $\mathcal{S}_E$ for this requirement. The transition system simply keeps track of the current floor in its state. When it receives a fpress_1 event, it transitions to the same state in the box P indicating that an executive floor request is pending. It prohibits **open** on floor 0 and **down** on floor 1 so that the executive floor request is serviced before other floors. We take $\mathcal{S}_E$, with the additional constraint that $\mathbf{nop_s}$ is not allowed from the states in box P, as a valid controller $\mathcal{C}_E$ for the base system $\mathcal{B}$.

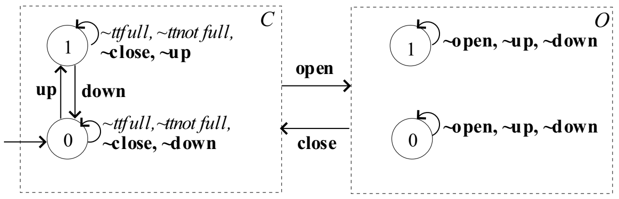
C
1
~ttfull, ~ttnotfull, ~close, ~up
up
down
0
~ttfull, ~ttnotfull, ~close, ~down
open
close
O
1
~open, ~up, ~down
0
~open, ~up, ~down

Fig. 3. Lift base system $\mathcal{B}$. The set Σ_e is $\{fpress_i, cpress_i, ttfull, ttnotfull, nop_e\}$. The events *ttfull* and *ttnotfull* can occur only when the lift door is open (in box O). The set Σ_s is $\{\mathbf{open}, \mathbf{close}, \mathbf{up}, \mathbf{down}, \mathbf{nop_s}\}$. The lift can move up or down only when its door is closed.

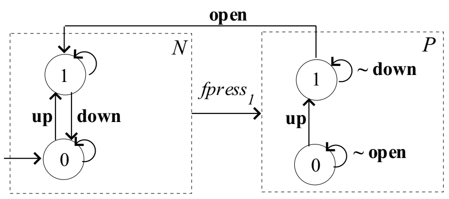

Fig. 4. Executive Floor Specification $\mathcal{S}_E$. Controller $\mathcal{C}_E$ is the same as $\mathcal{S}_E$ except that it does not advise $\mathbf{nop_s}$ from the states in box P.

Consider adding a feature called "two-thirds-full" which requires that the requests from within the lift should be serviced before requests from floors when the lift is two-thirds full. Thus, when the system receives a *ttfull* event, the lift should not service requests from floors as long as there are pending car requests. When the *ttnotfull* event occurs, the lift can go back to its normal functioning. Figure 5 shows a possible specification $\mathcal{S}_T$ for this feature. We take $\mathcal{S}_T$, with the additional constraint that $\mathbf{nop_s}$ is not allowed when the lift is two-thirds full and a car request is pending, as a valid controller $\mathcal{C}_T$ for $\mathcal{B}$. Note that a controller could impose additional constraints such as choosing **open** over *up* at floor 0 when car requests from both floors are pending.

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

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 behavior

$$fpress_1 \cdot \mathbf{up} \cdot nop_e \cdot \mathbf{open} \cdot ttfull \cdot \mathbf{close} \cdot cpress_0 \cdot \mathbf{down} \cdot fpress_1$$

from the initial state of $\mathcal{B}$. The system events allowed by $\mathcal{B}$ are **up**, **open** and $\mathbf{nop_s}$. However, the controlled system is blocked as $\mathcal{C}_E$ does not advise **open** and $\mathbf{nop_s}$ while $\mathcal{C}_T$ does not advise **up** and $\mathbf{nop_s}$.

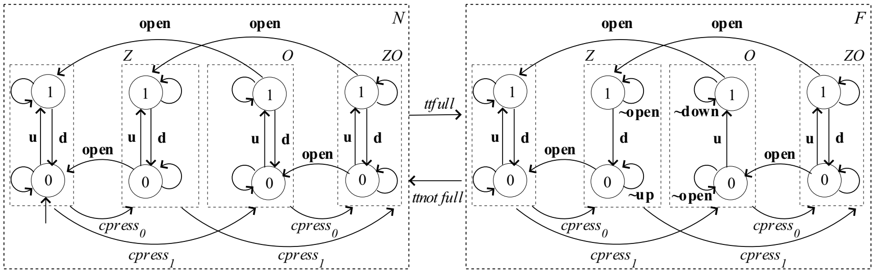

Fig. 5. Two-Thirds Full Specification $\mathcal{S}_T$. In box $Z(O)$, car request for floor 0 (respectively 1) is pending and in box ZO, car requests for both floors are pending. Controller $\mathcal{C}_T$ is the same as $\mathcal{S}_T$ except that it does not advise $\mathbf{nop_s}$ when the lift is two-thirds full (box F) and a car request is pending.

4 Conflict-Tolerant Controllers

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

Definition 5 (Conflict-Tolerant Advice Function). *A conflict-tolerant advice function over an alphabet Σ is a function $f : \Sigma^* \to 2^{\Sigma^*}$ which assigns a prefix-closed language $f(v)$ to every finite word $v \in \Sigma^*$, and is consistent in the sense that for* **all** *$vw \in \Sigma^*$ with $w \in f(v)$, we have $f(vw) = ext_w(f(v))$.*

A *conflict-tolerant* transition system (or CTTS) over Σ is a tuple $\mathcal{T}' = (\mathcal{T}, N)$, where $\mathcal{T} = (Q, s, \to)$ is a deterministic transition system over Σ and $N \subseteq \to$ is a subset of transitions designated as "not-advised." The language generated by $\mathcal{T}'$ starting from a state q in Q, denoted $L_q(\mathcal{T}')$, is defined to be simply $L_q(\mathcal{T})$. The *constrained* language generated by $\mathcal{T}'$, denoted $L^c_q(\mathcal{T}')$, is defined to be $L_q(\widehat{\mathcal{T}})$ where $\widehat{\mathcal{T}}$ is the transition system obtained from $\mathcal{T}$ by deleting all not-advised transitions (i.e. transitions in N). Let $w \in L(\mathcal{T})$, and let q be the unique state reached by $\mathcal{T}$ on w. Then by $L^c_w(\mathcal{T}')$ we mean $L^c_q(\mathcal{T}')$. We say $\mathcal{T}'$ is *complete* with respect to a language $L \subseteq \Sigma^*$ if $L \subseteq L_\epsilon(\mathcal{T}')$.

The CTTS $\mathcal{T}'$ induces a natural conflict-tolerant advice function $f_{\mathcal{T}'}$ given by, for all $w \in \Sigma^*$, $f(w) = L^c_w(\mathcal{T}')$. We say a conflict-tolerant advice function is *regular* if it is given by a finite-state CTTS over Σ.

We define the synchronized product of a transition system $\mathcal{T}_1$ and a CTTS $\mathcal{T}'_2 = (\mathcal{T}_2, N_2)$ to be the CTTS $(\mathcal{T}_1 \| \mathcal{T}_2, N'_2)$ which is complete with respect to $L(\mathcal{T}_1)$ and N'_2 is the set of joint transitions where the $\mathcal{T}_2$ transitions are not-advised (thus $\mathcal{T}_1 \| \mathcal{T}'_2$ inherits the not-advised transitions of $\mathcal{T}'_2$).

In the definitions below let $\mathcal{B}$ be a base system over a partitioned alphabet Σ.

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

– $\mathcal{C}'$ *is* ***non-restricting****: If $w \cdot e \in L(\mathcal{B})$ for some environment event $e \in \Sigma_e$, then $e \in L^c_w(\mathcal{C}')$ (or equivalently $e \in f^i_{\mathcal{C}'}(w)$). Thus the controller must not restrict any environment event e enabled in the base system after* ***any*** *system behavior w.*

– $\mathcal{C}'$ *is* ***non-blocking****: If $w \in L(\mathcal{B})$, then $L^c_w(\mathcal{B}\|\mathcal{C}') \neq \emptyset$ (equivalently $f^i_{\mathcal{C}'}(w) \cap f^i_{\mathcal{B}}(w) \neq \emptyset$). Thus the controller must not block the system after* ***any*** *system behavior w.*

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 $w \in L(\mathcal{B})$, $L^c_w(\mathcal{B}\|\mathcal{C}') \subseteq f(w)$. Thus after any system behavior w, if the base system follows the advice of $\mathcal{C}'$, the resulting behavior conforms to the safety language $f(w)$.*

We now illustrate these definitions with our running example. Figure 6 shows a conflict-tolerant specification $\mathcal{S}'_E$ for the "executive floor" feature. The not-advised transitions are shown using dashed transitions. By ignoring the dashed transitions, we get back the conventional specification in Fig. 4. The dashed transitions in a conflict-tolerant specification indicate the obligation on a controller when the specification is overridden to meet the requirements of a higher priority specification. In $\mathcal{S}'_E$, **open** from state 0 of box P and **down** from state 1 of box P are not advised. However, even if the door opens at floor 0 when a request from floor 1 is pending, $\mathcal{S}'_E$ requires the controller's subsequent advice to be such that the request from floor 1 is serviced. Figure 7 shows a conflict-tolerant specification for the "two-thirds-full" feature.

5 Synthesis and Verification

In this section we address the natural synthesis and verification problems for conflict-tolerant controllers. Let Σ be a partitioned alphabet.

Theorem 1 (Synthesis). *Given a base system $\mathcal{B}$ over Σ, and a regular conflict-tolerant specification $\mathcal{S}'$ over Σ, we can check if there exists a conflict-tolerant controller for $\mathcal{B}$ that satisfies $\mathcal{S}'$, and if so, synthesize a finite-state one.*

Proof. We claim that there exists a controller for $\mathcal{B}$ satisfying $\mathcal{S}'$ iff in the synchronized product $\mathcal{B}\|\mathcal{S}'$ there does not exist a state (b, q) which is reachable from the start state and satisfies one of the conditions

1. (b, q) is "restricting" in the sense that there is an environment event e enabled at b in $\mathcal{B}$, but is not advised at q in $\mathcal{S}'$.
2. (b, q) is "blocking" in that there is no event advised at (b, q) in $\mathcal{B}\|\mathcal{S}'$.

If no such (b, q) exists in $\mathcal{B}\|\mathcal{S}'$, then clearly $\mathcal{S}'$ itself is a valid finite-state controller for $\mathcal{B}$ that satisfies $\mathcal{S}'$. Conversely, if there is a valid controller $\mathcal{C}'$ for $\mathcal{B}$ satisfying $\mathcal{S}'$, then again it is easy to see that no such (b, q) must exist in $\mathcal{B}\|\mathcal{S}'$ (recall that the base system is always non-blocking). These conditions can be checked in time linear in the product of the sizes of $\mathcal{B}$ and $\mathcal{S}'$. □

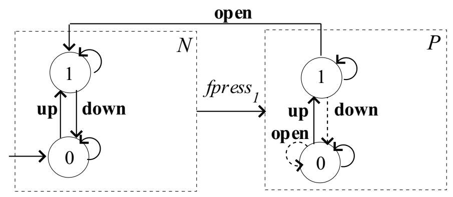

Fig. 6. Tolerant Executive Floor Specification $\mathcal{S}'_E$. Tolerant Controller $\mathcal{C}'_E$ is the same as $\mathcal{S}'_E$ except that $\mathbf{nop_s}$ is not advised from states in box P.

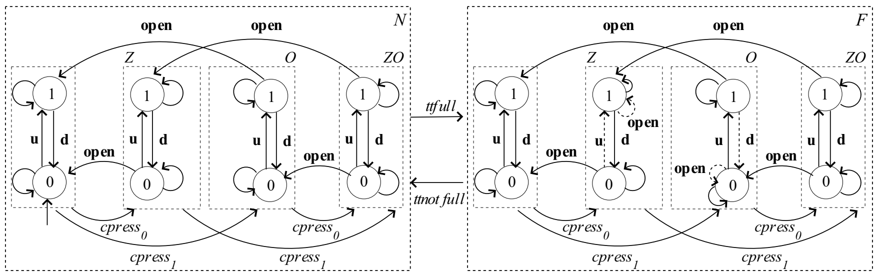

Fig. 7. Tolerant Two-Thirds Full Specification $\mathcal{S}'_T$. Tolerant Controller $\mathcal{C}'_T$ is the same as $\mathcal{S}'_T$ except that it does not advise $\mathbf{nop_s}$ when the lift is two-thirds full and a car request is pending.

Note that even if a state (b, q) as above exists, a classical controller [7] may still exist if it has a strategy to avoid reaching such a state.

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

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 state (b, p, q) which is reachable from the initial state and satisfies one of the following conditions:

1. ($\mathcal{C}'$ is restricting) there exists an event $e \in \Sigma_e$ enabled at b in $\mathcal{B}$, but is not advised at p in $\mathcal{C}'$.
2. ($\mathcal{C}'$ is blocking) there is no event c which is both enabled at b in $\mathcal{B}$ and advised at p in $\mathcal{C}'$.
3. ($\mathcal{C}'$ does not satisfy $\mathcal{S}'$) there is an event c which is both enabled at b in $\mathcal{B}$ and advised at p in $\mathcal{C}'$, but not advised at q in $\mathcal{S}'$.

This check can be carried out in time linear in the product of the sizes 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 a "best possible" way.

Let $\mathcal{B} = (B, r_0, \rightarrow)$ be a base system over an alphabet Σ. Let $\mathcal{C}'_1 = (Q_1, s_1, \rightarrow_1, N_1)$ and $\mathcal{C}'_2 = (Q_2, s_2, \rightarrow_2, N_2)$ be valid conflict-tolerant controllers for $\mathcal{B}$. Let P be a priority ordering between $\mathcal{C}'_1$ and $\mathcal{C}'_2$, and say P assigns a higher priority to $\mathcal{C}'_1$, denoted by $\mathcal{C}'_2 <_P \mathcal{C}'_1$. Then:

Definition 8 (Prioritized Composition). *The* P-prioritized composition *of the controllers $\mathcal{C}'_1$ and $\mathcal{C}'_2$ w.r.t. $\mathcal{B}$, is the conflict-tolerant transition system $\mathcal{C}'$, denoted by $\|_{P,\mathcal{B}}(\mathcal{C}'_1, \mathcal{C}'_2)$, and defined as*

$$\mathcal{C}' = (Q_1 \times Q_2 \times B, (s_1, s_2, r_0), \Rightarrow, N)$$

where $\Rightarrow$ is given by $(p_1, p_2, r) \overset{a}{\Rightarrow} (q_1, q_2, t)$ iff $p_1 \xrightarrow{a}_1 q_1$, $p_2 \xrightarrow{a}_2 q_2$, and $r \xrightarrow{a} t$; and the set of not-advised transitions N is defined as follows. With each transition $u = (p_1, p_2, r) \overset{a}{\Rightarrow} (q_1, q_2, t)$, we associate a bit-string (in this case of length 2) which denotes which of the controllers has advised this transition. Thus the associated bit-string for the transition above is $b_1 b_2$ where b_i is 1 iff $p_i \xrightarrow{a}_i q_i \notin N_i$. Let the "rank" of u be the number represented by this string in binary notation. Then the transition u is advised (i.e. not in N) iff there is no transition of higher rank going out of the state (p_1, p_2, r).

Figure 8 show how the ranks and "advised" status of transitions are calculated (assuming that the base system allows all the events shown). Thus from state (p, p') only the transition on d is advised, while all others are not advised.

Lemma 1. *The CTTS $\mathcal{C}' = \|_{P,\mathcal{B}}(\mathcal{C}'_1, \mathcal{C}'_2)$ defined above is a* valid *(i.e. non-restricting and non-blocking) conflict-tolerant controller for $\mathcal{B}$. In addition, the immediate advice function $f^i_{\mathcal{C}'}$ it induces is given as follows. For each $w \in L(\mathcal{B})$:*

$$f^i_{\mathcal{C}'}(w) = \begin{cases} f^i_{\mathcal{B}}(w) \cap f^i_{\mathcal{C}'_1}(w) \cap f^i_{\mathcal{C}'_2}(w) & \textit{if } f^i_{\mathcal{B}}(w) \cap f^i_{\mathcal{C}'_1}(w) \cap f^i_{\mathcal{C}'_2}(w) \neq \emptyset \\ f^i_{\mathcal{B}}(w) \cap f^i_{\mathcal{C}'_1}(w) & \textit{otherwise.} \end{cases}$$

□

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 a base system $\mathcal{B}$. Let P be a priority that induces a total ordering $<_P$ on the controllers. Then the P-prioritized composition of $\mathcal{C}'_1, \ldots, \mathcal{C}'_n$ (wrt $\mathcal{B}$) is denoted $\|_{P,\mathcal{B}}(\mathcal{C}'_1, \ldots, \mathcal{C}'_n)$ and defined similarly as above. Let $\mathcal{S}'_1, \ldots, \mathcal{S}'_n$ be conflict-tolerant specifications, and suppose each $\mathcal{C}'_j$ individually satisfies the specification $\mathcal{S}'_j$ w.r.t. $\mathcal{B}$.

Theorem 3. *The conflict-tolerant transition system $\mathcal{C}' = \|_{P,\mathcal{B}}(\mathcal{C}'_1, \ldots, \mathcal{C}'_n)$ is a valid conflict-tolerant controller for $\mathcal{B}$. Further, $\mathcal{C}'$ satisfies each of the specifications $\mathcal{S}'_1, \ldots, \mathcal{S}'_n$ in the following "maximal" sense: Each $w \in L(\mathcal{B}\|\mathcal{C}')$ is always*

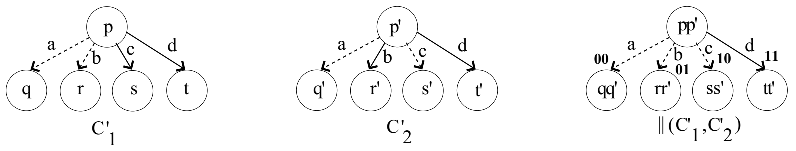

Fig. 8. Computation of ranks in the prioritized composition

according to the immediate advice of $\mathcal{S}'_j$, except *at the points where* $\mathcal{C}'_j$ *is in conflict with the advice of higher-priority controllers, in that for each a in* $f^i_{\mathcal{C}'_j}(w)$, *there is a controller* $\mathcal{C}'_k$ *such that* $\mathcal{C}'_k >_P \mathcal{C}'_j$ *and* $a \notin f^i_{\mathcal{C}'_k}(w) \cap f^i_{\mathcal{B}}(w)$*. In particular, the highest priority specification* $\mathcal{S}'_1$ *is always satisfied.* $\square$

Consider the base system behavior

$$\mathit{fpress}_1 \cdot \mathbf{up} \cdot \mathit{nop}_e \cdot \mathbf{open} \cdot \mathit{ttfull} \cdot \mathbf{close} \cdot \mathit{cpress}_0 \cdot \mathbf{down} \cdot \mathit{fpress}_1$$

which we used to illustrate conflict in Sect. 3. With the priority order $\mathcal{C}'_T > \mathcal{C}'_E$, the conflict is resolved such that one possible extension is

$$\mathbf{open} \cdot \mathit{ttnotfull} \cdot \mathbf{close} \cdot \mathit{nop}_e \cdot \mathbf{up} \cdot \mathit{nop}_e \cdot \mathbf{open}$$

– i.e. the door is opened at floor 0 violating the advice of $\mathcal{C}'_E$. As passengers go out at floor 0, the lift is not two-thirds full and the system immediately follows the advice of $\mathcal{C}'_E$ to service the executive floor. We emphasise that the same controllers can be composed with the priority $\mathcal{C}'_E > \mathcal{C}'_T$, to obtain a system in which conflicts will be resolved in favor of $\mathcal{C}'_E$, while maximally utilizing the advice of $\mathcal{C}'_T$.

7 Discussion

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 admit a classical controller. See [13] for an example. Nonetheless, whenever the conflict-tolerant specifications are realizable, our framework provides a flexible way of composing the controllers to obtain systems with guarantees on the usage of each controller.

We have considered extensions of this framework to include combinations of safety and liveness specifications [13], as well as real-time features [14] with similar results.

Our framework is also amenable to more flexible priority schemes like according priority dynamically based on history of events. Gößler and Sifakis [15] consider transition systems with priorities specified by predicates under which one action is prioritized over another. They provide conditions under which the predicates are consistent in that the prioritized system is non-blocking. Our composition scheme can be thought of as synthesizing consistent priority predicates from possibly inconsistent predicates obtained from individual controllers.

References

1. Keck, D.O., Kühn, P.J.: The feature and service interaction problem in telecommunications systems. a survey. IEEE Trans. Software Eng. 24(10), 779–796 (1998)
2. Hall, R.J.: Feature interactions in electronic mail. In: FIW, pp. 67–82 (2000)
3. Jackson, M., Zave, P.: Distributed feature composition: A virtual architecture for telecommunications services. IEEE Trans. Software Eng. 24(10), 831–847 (1998)
4. Software Engineering Institute: Software product lines, `http://www.sei.cmu.edu/productlines`
5. Fisler, K., Krishnamurthi, S.: Decomposing verification by features. In: IFIP Working Conference on Verified Software: Theories, Tools, Experiments (2006)
6. Felty, A.P., Namjoshi, K.S.: Feature specification and automated conflict detection. ACM Trans. Softw. Eng. Methodol. 12(1), 3–27 (2003)
7. Ramadge, P.J.G., Wonham, W.M.: The control of discrete event systems. Proc. of the IEEE 77, 81–98 (1989)
8. Wong, K.C., Thistle, J.G., Hoang, H.H., Malhamé, R.P.: Supervisory control of distributed systems: Conflict resolution. In: Conf. on Decision and Control, pp. 416–421. IEEE, Los Alamitos (1995)
9. Chen, Y.L., Lafortune, S., Lin, F.: Modular supervisory control with priorities for discrete event systems. In: Conf. on Decision and Control, pp. 409–415. IEEE Computer Society Press, Los Alamitos (1995)
10. Wong, K.C., Thistle, J.G., Hoang, H.H., Malhamé, R.P.: Supervisory control of distributed systems: Conflict resolution. In: Conf. on Decision and Control, pp. 3275–3280. IEEE, Los Alamitos (1998)
11. Hay, J.D., Atlee, J.M.: Composing features and resolving interactions. In: SIGSOFT Found. of Softw. Engg., pp. 110–119 (2000)
12. Plath, M., Ryan, M.: Feature integration using a feature construct. Sci. Comput. Program 41(1), 53–84 (2001)
13. D'Souza, D., Gopinathan, M.: Conflict-tolerant features. Technical Report IISc-CSA-TR-2007-11, Computer Science and Automation, Indian Institute of Science, India (2007), `http://archive.csa.iisc.ernet.in/TR/2007/11/`
14. D'Souza, D., Gopinathan, M., Ramesh, S., Sampath, P.: Conflict-detection and resolution for real-time features (manuscript in preparation)
15. Gößler, G., Sifakis, J.: Priority Systems. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2003. LNCS, vol. 3188, pp. 314–329. Springer, Heidelberg (2004)