

Conflict-Tolerant Features

Deepak D'Souza

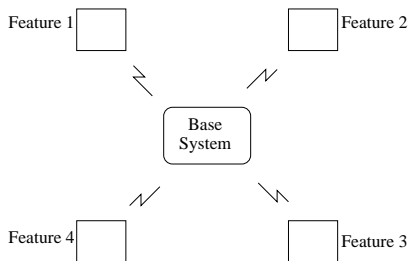
Joint work with Madhu Gopinathan

Computer Science and Automation
Indian Institute of Science, Bangalore.

Bangalore, 12th December 2008

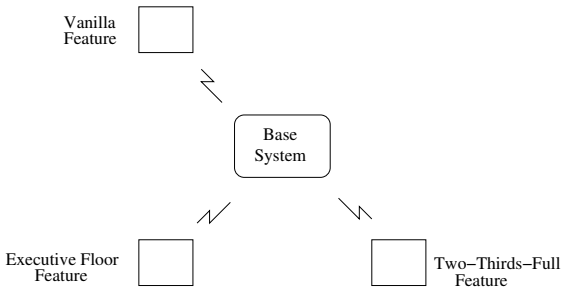
Overview

- Consider systems which are composed of a **base system** + multiple **features** or **controllers**.



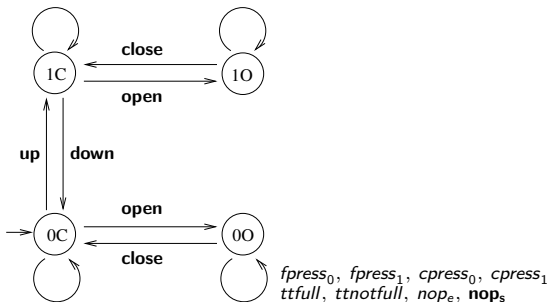
- We propose a methodology for compositionally developing and verifying such systems.
- Idea: Design features to be **conflict-tolerant** or “resilient” to deviations from their advice.

Example System: Lift



Example System: Lift

Lift base system (2 floors):

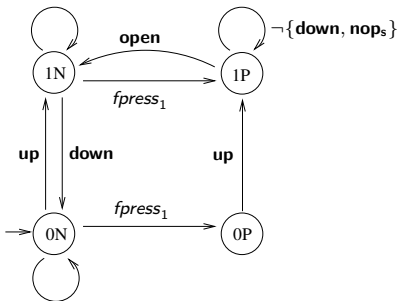


- **System:** **down, up, open, close, \mathbf{nop}_s .**
- **Env:** $fpress_0, fpress_1, cpress_0, cpress_1, ttfull, ttnotfull, nop_e$.
- Consider alternating behaviours only: $(Env \cdot System)^*$.

Executive Floor Feature

Requirement

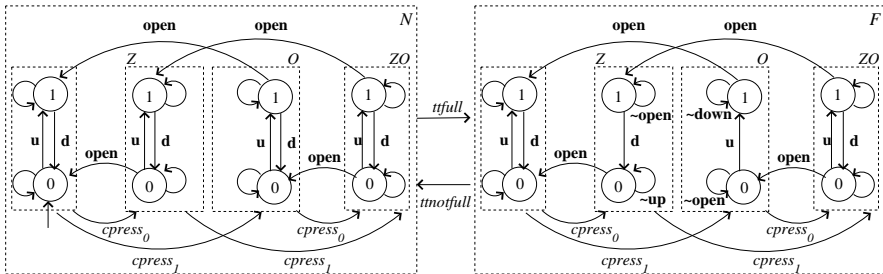
When executive floor (floor 1) call is pending do not service other floor or car calls.



Two-Thirds Full Feature

Requirement

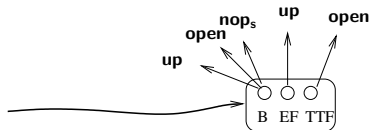
While lift is two-thirds full, service only car calls.



Conflict between features

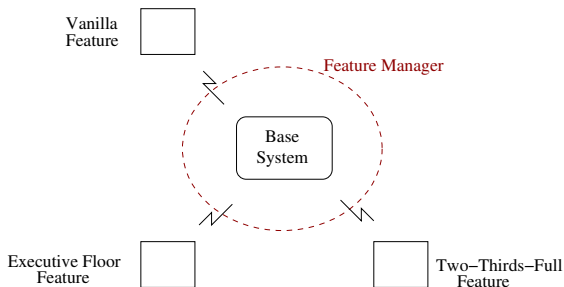
Behaviour after which advices of EF and TTF features diverge:

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



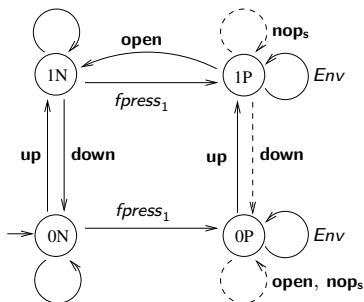
Ad hoc Resolution

Feature Manager resolves conflicts based on priority:



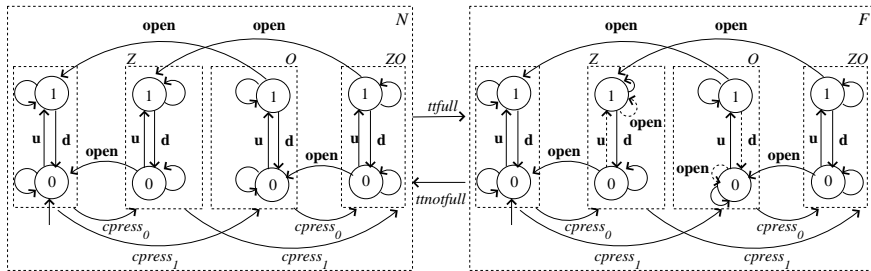
- Suspend lower-priority feature
- Resume lower-priority feature when system is “reset”.
- Our proposal: make features “conflict-tolerant”.

Conflict-Tolerant Executive Floor Feature

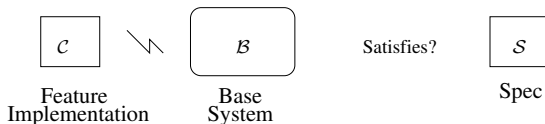


“Tolerant” feature continues to give advice even if overridden.

Conflict-Tolerant TTF Feature



Classical Safety Specs and Verification



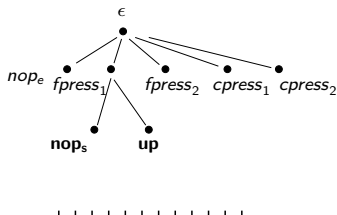
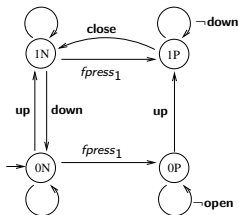
- Useful to have a specification of property we want a feature to satisfy.
- Verification: Does the composition of C and B satisfy S ?

$$C \parallel B \models S$$

- Feasibility: Does there exist a valid controller C for B satisfying S .

Safety Specifications

- A **safety spec** is a prefix-closed language, given by a finite-state transition system \mathcal{S} .
- Prefix-closed language as a pruned **tree**:



When a controller for \mathcal{B} satisfies \mathcal{S}

- A controller is a finite-state transition system.
- A controller \mathcal{C} is **valid** for \mathcal{B} if after every controlled behaviour
 - It allows at least one event which is enabled in the plant.
(**non-blocking**).
 - It allows all environment events enabled in the plant
(**non-restricting**).
- Controller \mathcal{C} for \mathcal{B} satisfies spec \mathcal{S} if controlled language is within $L(\mathcal{S})$:

$$L(\mathcal{C}\|\mathcal{B}) \subseteq L(\mathcal{S}).$$

Verification and Feasibility Problems

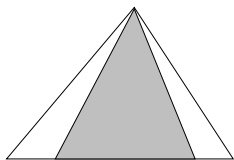
- Verification problem: Given \mathcal{C} , \mathcal{B} , and \mathcal{S} modelled as finite-state transition systems, does $\mathcal{C} \parallel \mathcal{B} \models \mathcal{S}$?
- Feasibility problem: Given \mathcal{B} and \mathcal{S} modelled as finite-state transition systems, does there exist a valid controller \mathcal{C} such that $\mathcal{C} \parallel \mathcal{B} \models \mathcal{S}$?

Decision procedures:

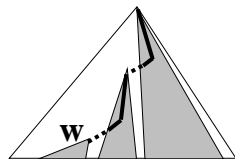
- Simple algorithm to verify that \mathcal{C} for \mathcal{B} satisfies \mathcal{S} : Do DFS on product $\mathcal{C} \parallel \mathcal{B} \parallel \mathcal{S}$ and look for “bad” states.
- Feasibility: Check if “Controller” has a strategy to keep the game within good states no matter what “Environment” does.

Conflict-Tolerant Spec

- Specifies safe behaviours for features after **every** behaviour of the base system.
- Classical spec vs conflict-tolerant spec



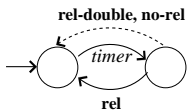
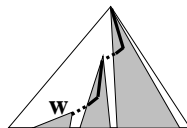
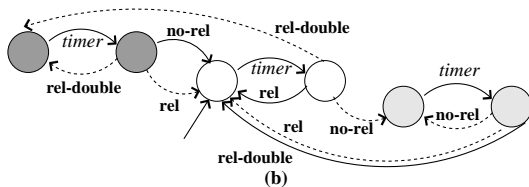
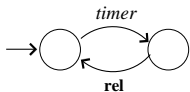
(a)



(b)

- A **conflict-tolerant advice function** $f : \Sigma^* \rightarrow 2^{\Sigma^*}$ assigns a prefix-closed language $f(w)$ to every finite word w , and is **consistent** in the sense that for **all** $wv \in \Sigma^*$ with $v \in f(w)$, we have $f(wv) = \text{ext}_v(f(w))$.
- Can be represented by an “annotated” transition system.

Example Conflict-Tolerant Specs



When a CT controller for \mathcal{B} satisfies a CT \mathcal{S}'

- A CT-controller is an “annotated” finite-state transition system \mathcal{C}' .
- A CT-controller \mathcal{C}' is **valid** for \mathcal{B} if
 - It is **non-blocking**: should allow at least one event which is enabled in the plant (after **any** behaviour of the base system).
 - It is **non-restricting**: should allow all environment events enabled in the plant (after **any** behaviour of the base system).
- CT-Controller \mathcal{C}' for \mathcal{B} satisfies CT spec \mathcal{S}' if after **any** behaviour w of the base system, the “controlled” language of $\mathcal{C}'\|\mathcal{B}$ is within $L_w(\mathcal{S})$:

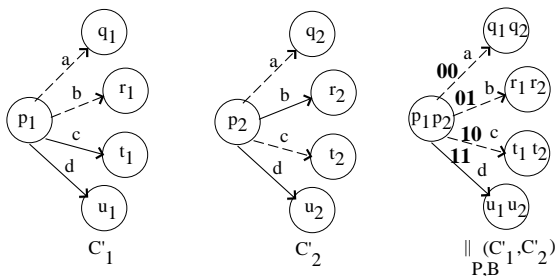
$$L_w^c(\mathcal{C}\|\mathcal{B}) \subseteq L_w^c(\mathcal{S}).$$

- Simple algorithms to solve verification and feasibility problems.

Composing Conflict-Tolerant Features

- Composing CT controllers C'_1 and C'_2 for \mathcal{B} , based on a priority ordering P (say $C'_1 >_P C'_2$)

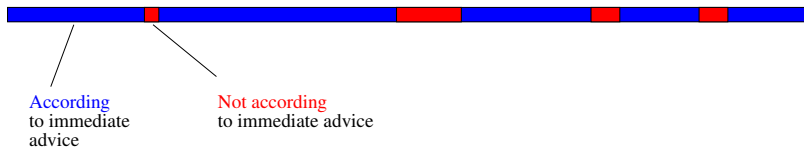
$$\parallel_{\mathcal{B}, P}(C'_1, C'_2)$$



- Generalize to $\parallel_{\mathcal{B}, P}(C'_1, \dots, C'_n)$.
- Composed controller is a valid classical controller for \mathcal{B} .

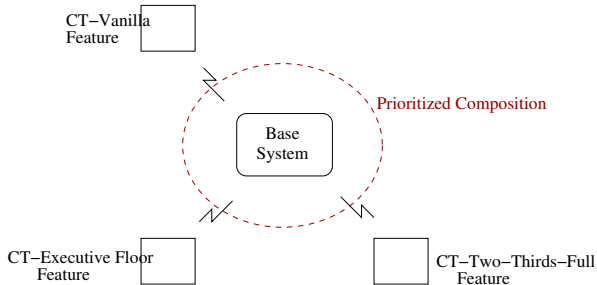
Correctness Guarantee on composed system

Composed system satisfies the spec of *each* feature, *except* when each of its advised actions is incompatible with a higher priority controller.



Thus, composed system satisfies the individual CT-specs **maximally**: system always follows advice of a controller, except when it is unavoidable.

Resolution with guarantees



Summary

- Proposed a methodology for developing systems which are composed of a **base system** + multiple **features** or **controllers**.
- Methodology based on the idea of features that are **conflict-tolerant** or “resilient” to deviations from their advice.
- **Specification**, **verification**, and **synthesis** of such systems.
- **Correct-by-construction** composition.
- **Compositional** verification technique.
- Solution to “feature interaction” problem.

Extending to other Real-Time / hybrid features

- Use deterministic Alur-Dill timed transition systems for base system/Features.
- Model urgency using **deadlines** or **time-can-progress** conditions.
- Same results: verification, feasibility decidable, guarantee on prioritized composition.
- Ongoing work: specifying tolerant controllers for **dynamical systems**.
- Verification? Feasibility? Applicability?

Thanks

Thank You.