Motivation

In a complex, electronic, software-intensive system, a feature is a bundle of system functionality as a user recognizes it.

Features are units that companies use to guide the design and marketing of the functionality offered to their customers.

Features often automate a manual process or activity.

Example: call waiting in telephony.

In the automotive domain, an Advanced Safety Feature makes use of sensors, cameras, and even GPS devices to help the driver be aware of dangers, and when possible, control the dynamics of the vehicle to avoid unsafe outcomes.

Example: collision avoidance in a vehicle.

In the automotive domain, a feature interaction (FI) arises from the activation of two or more features whose output requests to the actuators create contradictory physical forces on the mechanical processes, potentially at distinct times, that cause unsafe outcomes [1,2].

A feature interaction does not arise from the failure of individual components, but from the intended functionality of correctly implemented features.

Feature interactions are becoming more prevalent as systems increase in complexity, and can be a source of significant risk.

Example: when one feature requests to apply the brakes as another feature requests to apply the throttle.

To detect feature interactions in the automotive domain, we use symbolic model checking at design time, which allows us to examine exhaustively all behaviours of the system.

It is advantageous to find all paths that do not satisfy the property before fixing the model because the correction depend on several factors that can only be recognized by looking at all counterexamples.

The set of all counterexamples is often too large to generate and comprehend.

Methodology

Goal: Use our tool to put “abstraction in action” by generating only a reduced number of cases that represent all possible sources of risk (i.e., errors, inconsistencies, contradictions, feature interactions) in a system.

We abstract the counterexamples based on the modelling concepts of control states and transitions.

Our tool AbsAct uses the model checker SMV [3] in an iterative process to generate all sources of risks (counterexamples).

For every abstraction level, we begin by checking the property:

Globally(¬FI).

For each iteration, when a counterexample is generated, we abstract it by representing its equivalence class using a linear temporal logic (LTL) [4] formula according to the desired level of abstraction. Then, we disjunct this LTL expression with the original property and the LTL expressions representing previously generated counterexamples (equivalence classes).

Contributions

Development of the tool AbsAct: Automatic generation of a reduced number of cases that represent all possible sources of risk in a system during model checking iterations.

Definition of a scale of abstraction levels: Representation of equivalence classes as LTL properties, which each create equivalence classes of counterexamples.

Case Study

Detection of sources of risks in automotive features: Collision Avoidance (CA) and Emergency Vehicle Avoidance (EVA), designed in Matlab’s Stateflow and translated into SMV with our tool md2smv [5].

References


More information is available at: http://www.cs.uwaterloo.ca/~ajuarez/research.html