FEATURE INTERACTION DETECTION IN THE AUTOMOTIVE DOMAIN

Alma L. Juarez Dominguez

ASE Doctoral Symposium 2008
September, 2008

Research partially funded by the National Science and Engineering Research Council of Canada, General Motors of Canada and Critical Systems Labs Inc.
The feature interaction problem is not new, but it is appearing in more domains given:

- Increase in system complexity
- Constant demand of new functionality
- Integration of functionality (new and legacy)
- Definition of "feature" varies depending on level of abstraction
A feature is a bundle of system functionality that is recognized by the user, and marketed by a company.

- Call Forwarding, Adaptive Cruise Control
In automotive systems, advanced Safety Features make use of sensors, cameras, and even GPS devices to help the driver to be aware of dangers, and when possible, control the dynamics of the vehicle to avoid unsafe outcomes, *e.g.* Cruise Control, Collision Mitigation.
Main goal

Develop techniques and tools for the detection of feature interactions in the automotive domain.
**OUTLINE**

1. **Characteristics of Automotive Systems**
2. **Proposed Solution**
3. **Evaluation**
4. **Conclusion**
**OUTLINE**

1. **Characteristics of Automotive Systems**
2. **Proposed Solution**
3. **Evaluation**
4. **Conclusion**

**Feature Interaction Detection in the Automotive Domain** September 2008
- Features normally run **simultaneously and independently**, possibly on different hardware, with **no direct communication** between features.

- Indirect communication occurs through the environment.
We are interested in the feature interactions that arise from the activation of two or more features whose output requests to the actuators, potentially at distinct times, create contradictory physical forces that cause unsafe outcomes.
Feature interactions appear in the physical part of the system, creating contradictory physical forces in the environment that cause unsafe behaviours.
Feature interactions appear in the physical part of the system, creating contradictory physical forces in the environment that cause unsafe behaviours.

The set of features is fixed at retail or release time making an off-line solution based on analysis of the designs reasonable.
1 Feature interactions appear in the physical part of the system, creating contradictory physical forces in the environment that cause unsafe behaviours.

2 The set of features is fixed at retail or release time making an off-line solution based on analysis of the designs reasonable.

3 Only one copy of a feature can be active in a vehicle, making it possible to analyze these systems as if the features are invoked statically.
Feature interactions appear in the physical part of the system, creating contradictory physical forces in the environment that cause unsafe behaviours.

The set of features is fixed at retail or release time making an off-line solution based on analysis of the designs reasonable.

Only one copy of a feature can be active in a vehicle, making it possible to analyze these systems as if the features are invoked statically.

The set of actuators and their range of possible values are known.
Summary

The feature interaction problem in automotive systems is bounded, which makes off-line symbolic model checking of feature designs a promising detection technique.
OUTLINE

1. CHARACTERISTICS OF AUTOMOTIVE SYSTEMS
2. PROPOSED SOLUTION
3. EVALUATION
4. CONCLUSION
PROPOSED OFF-LINE DETECTION APPROACH

**Characteristics of Automotive Systems**

**Proposed Solution**

**Evaluation**

**Conclusion**

---

**Feature Interaction Detection in the Automotive Domain**

September 2008
Create a general definition of feature interactions independent of the set of features by considering the elements with multiple feature influence.
Use elements of **multiple feature influence** to construct temporal logic formulas to check the operational behaviour of the features to detect feature interactions.
1. Definition of FIs in the Automotive Domain

- Our definition contains two main parts:
  - **Same Actuator**: for direct actuator conflicts.
  - **Conflicting Actuators**: for actuators that cause feature interactions in the environment.
1. Definition of FIs in the Automotive Domain

Our definition contains two main parts:

- **Same Actuator**: for direct actuator conflicts.
- **Conflicting Actuators**: for actuators that cause feature interactions in the environment.

A feature interaction might be:

- **Immediate**: the conflict is caused by two features making requests in the same step.
- **Temporal**: the conflict is caused by two requests happening within a certain time threshold of each other.

1. Definition of FIs in the Automotive Domain

Example

Conflicting Actuators, Immediate, Degrees

\(\Diamond ((X > \text{value\_threshold}_X) \land (Y > \text{value\_threshold}_Y))\)
2. Translation from Stateflow to SMV

To do analysis at the right level of description:

To do analysis at the right level of description:

- **Stateflow** is extensively used for designing the controllers of embedded components in the automotive and avionics industries.
2. TRANSLATION FROM STATEFLOW TO SMV

- To do analysis at the right level of description:

- **SMV** has powerful features and simple language that allows us to describe the semantics of individual design models in Stateflow, and that of the composition of features.
2. Translation from Stateflow to SMV

- **Nested switch statements** follow the hierarchy of states in the Stateflow model and check whether the current state is in the source state of a transition.

- Within each case statement, there is a series of if-then-else statements, one per transition, checking whether the event and/or condition of the transition are satisfied.

- The order of the if-then-else statements corresponds to the transitions’ priority of execution.
We create a **Stateflow step** through a series of SMV steps. A Stateflow step may consist of a sequence of transitions taken in different components of an AND-state within a feature (**AND-state composition**).
2. Translation from Stateflow to SMV

- At most one transition in the model of a feature can be taken in an SMV step, but...
- We create a Stateflow step through a series of SMV steps.
  - A Stateflow step may consist of a number of transitions taken in different features (feature composition).

![Diagram showing Stateflow and SMV steps]

**Input:** $e_1$

- $i$-th step of execution
  - Active state for A
  - Active state for B
  - Active state for D

- $(i+1)$-th step of execution
  - IDLE
  - (Retain D’s outputs of $i$-th step)

- $(i+2)$-th step of execution

**Input:** $e_3$

...
To perform detection, we **automatically** create combinations of models and properties, based on our proposed definition, to check in SMV.

We use **pruning** to reduce the number of cases to be checked for feature interaction by concentrating only on the feature behaviours that are of interest at a certain point during the detection analysis (e.g., prune cases to only those features that use brakes).
3. Feature Interaction Detection Using SMV

- We **automatically** detect and report **all** feature interactions in every combination of models and properties, and also identify potential:
  - Design errors.
  - Modelling errors.
  - Guidance for environmental restrictions.

- We are investigating **abstractions (time and values)** to use during model checking.

- Based on our findings during the detection process, we intend to propose the **creation of a library of domain-specific abstractions**.
Evaluation of our Approach

- **Correctness:** Does our translator accurately implement the semantics of Stateflow?

- **Completeness:** Does our approach detect all feature interactions?

- **Manageability:** Does our approach present the detected interactions to the designers in a manageable way, *i.e.*, by presenting a summary of the relevant cases?

- **Environmental Modelling:** Can environmental constraints be captured succinctly?
Evaluation of our Approach

- **Traceability**: Can the SMV counterexamples be understood in terms of the STATEFLOW model?

- **Scalability**: Is our approach able to handle the typical size and number of features?

- **Reusability**: Is our approach reusable, with respect to the expert knowledge, feature interaction definition, domain-specific abstractions, and translator?
OUTLINE

1. CHARACTERISTICS OF AUTOMOTIVE SYSTEMS

2. PROPOSED SOLUTION

3. EVALUATION

4. CONCLUSION
A systematic and tractable method for detecting feature interactions:

1. A general, systematic definition of feature interactions in the automotive domain that is independent from the set of features.

2. Identification of Stateflow semantics, and the creation of a translator from Stateflow language into the input language of SMV.

3. Identification of the characteristics of automotive systems that make symbolic model checking a promising technique to detect feature interactions.

4. Use of symbolic model checking and abstractions to detect all feature interactions.

Creation of a set of non-proprietary feature design models in Matlab’s Stateflow.
Work on case study to validate our approach.

- Automatically creating combinations of models and properties to check in SMV.
- Investigating abstractions (time and values) to use during model checking.

Partial results applying our method to our non-proprietary feature set (for same actuator conflicts):

- 74 cases of real feature interactions
- 100 cases of design errors (false positives)
- 8 cases of modelling errors (false positives)
- 8 cases to guide environment restrictions
Questions?

Comments?

Alma Juarez
aljuarez@cs.uwaterloo.ca
## 1. Definition of FIs in the Automotive Domain

<table>
<thead>
<tr>
<th></th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Same Actuator</strong></td>
<td><strong>Boolean</strong> ♦ $(X_1 \neq X_2)$</td>
</tr>
<tr>
<td></td>
<td><strong>Degrees</strong> ♦ $(</td>
</tr>
<tr>
<td><strong>Conflicting Actuators</strong></td>
<td><strong>Boolean</strong> ♦ $(X \neq Y)$</td>
</tr>
<tr>
<td></td>
<td><strong>Degrees</strong> ♦ $(X &gt; \text{value_threshold}_X)$</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## 1. Definition of FIs in the Automotive Domain

<table>
<thead>
<tr>
<th></th>
<th>Temporal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Same Actuator</strong></td>
<td><strong>Boolean</strong></td>
</tr>
<tr>
<td></td>
<td>$\Diamond((X_1 \neq X_{\text{last set}}) \land ((t_{\text{now}} - t_{\text{last } X}) &lt; \text{time threshold}))$</td>
</tr>
<tr>
<td></td>
<td><strong>Degrees</strong></td>
</tr>
<tr>
<td></td>
<td>$\Diamond((</td>
</tr>
<tr>
<td><strong>Conflicting Actuators</strong></td>
<td><strong>Boolean</strong></td>
</tr>
<tr>
<td></td>
<td>$\Diamond((X \neq Y_{\text{last set}}) \land ((t_{\text{now}} - t_{\text{last } Y}) &lt; \text{time threshold}))$</td>
</tr>
<tr>
<td></td>
<td><strong>Degrees</strong></td>
</tr>
<tr>
<td></td>
<td>$\Diamond((X &gt; \text{value threshold}<em>X) \land (Y</em>{\text{last set}} &gt; \text{value threshold}<em>Y) \land ((t</em>{\text{now}} - t_{\text{last } Y}) &lt; \text{time threshold})))$</td>
</tr>
</tbody>
</table>
1. Definition of FIs in the Automotive Domain

Example

Same Actuator, Immediate, Degrees

\[ \diamond (|X_1 - X_2| > \text{value\_threshold}) \]
Example

Conflicting Actuators, Immediate, Degrees

\[ \diamond ((X > value_{threshold_X}) \land (Y > value_{threshold_Y})) \]
1. Definition of FIs in the Automotive Domain

Example

Same Actuator, Temporal, Degrees

\[ \diamond (\left| X_1 - X_{\text{last_set}} \right| > \text{value\_threshold}) \land (t_{\text{now}} - t_{\text{last\_X}} < \text{time\_threshold}) \]
Sequential Execution for AND-states

MODULE F1 (P,Q,X,Y,sys_ready)
{
  sF1 : {A,B};  /*state F1*/
  sA : {A1, A2};  /*state A*/
  sB : {B1,B2};  /*state B*/
  sA1 : {D};  /*state A1*/
  sA2 : {E};  /*state A2*/

  init(sF1) := A;
  init(sA) := A1;
  init(sB) := B1;
  init(sA1) := D;
  init(sA2) := E;

  switch(sF1) {
    A:
      switch(sA) {
        A1:
          if (sys_ready)
            next(Y) := 2;
            next(X) := X;
            next(sA1) := D;
            next(sA) := A2;
            next(sF1) := A;
          else
            next(X) := X;
            next(Y) := Y;
            next(sA1) := sA1;
            next(sA) := sA;
            next(sF1) := sC;
        A2:
          switch(sA2) {
            E:
              if (Q) {
              ...
            }
          }
      }
      else {
        next(X) := X;
        next(Y) := Y;
        next(sA1) := sA1;
        next(sA) := sA;
        next(sF1) := sC; }
    B:
      switch(sB) {
        B1:
          if (P) {
            next(X) := X;
            next(Y) := 1;
            next(sB) := B2;
            next(sF1) := A;
          }
          else {
            next(X) := X;
            next(Y) := Y;
            next(sB) := B1;
            next(sF1) := A; }
        B2:
          ...
      }
    }

  DEFINE readyF1 :=
    sF1=A & sA=A1;
  DEFINE doneF1 :=
    next(sF1)=A & next(sA)=A1;
}

MODULE main() {
  P, Q : boolean;
  X, Y : 0..10;
  init(X):=0; init(Y):=0;
  init(P):={true,false};
  init(Q):={true,false};

  DEFINE sys_ready:=readyF1;
  DEFINE sys_done:=doneF1;

  if (sys_done)
  {next(P) := {true,false};
    next(Q) := {true,false};}
  else
  {next(P) := P;
    next(Q) := Q;}

  F1 (P,Q,X,Y,sys_ready);
}
**Parallel Execution of Integrated Features**

- **i-th step**
- **Input:** \( P \)
- **Current state**
  - A
  - B
  - C
  - D

- **Features**
  - \( F_1 \)
  - \( F_2 \)

- **States**
  - \( A_1 \)
  - \( A_2 \)
  - \( B_1 \)
  - \( B_2 \)

- **Inputs**
  - \( Y=1 \)
  - \( Y=2 \)
  - \( Y=3 \)
  - \( Z=1 \)

- **Values**
  - \( Q \)
  - \( P \)
PARALLEL EXECUTION OF INTEGRATED FEATURES

(i+1)-th step

Current state A
Current state B
Current state C
Current state D

IDLE (Retain D's output)
Parallel Execution of Integrated Features

(i+3)-th step

Input: Q

Current state A  Current state B  Current state C  Current state D
Concurrent Execution of Integrated Features