Translating the Feature Oriented Requirements Modelling Language (FORML) to Alloy

For the purpose of automated analysis

David Dietrich
CS846
Outline

- Problem and Motivation
- FORML
- The Translator
- The Analysis
- Demo
- Limitations
- Future Work
- Conclusions
Problem

• One kind of correctness of a state machine is a lack of conflicts when concurrent transitions are executed.

• A conflict occurs when 2 transitions execute at the same time, but one of their post-conditions does not hold.
  • e.g., Changing the same object with 2 transitions may lead to a non-deterministic post state.

• The goal of this project is to create a translator to Alloy that can use the Alloy Analyzer to locate conflicts when 2 transitions execute concurrently.
The Cost of Requirements Errors

![Bar Graph showing the relative cost to fix requirements errors]

- Requirements: 1
- Coding: 6.5
- Test: 15
- Deployment: 80

Relative Cost to Fix (Requirements = 1)

Image from http://www.stickyminds.com/sitewide.asp?ObjectId=12529&Function=edetail&ObjectType=ARTCOL
FORML

• Created by Pourya Shaker at the University of Waterloo

• Requirements modelling language that provides support for Software Product Lines and feature-oriented modelling

• FORML is designed as a graphical language, but a plain text grammar has also been created

• There are 2 models: the World Model, and the Behaviour Model
FORML – World Model

Diagram of the world model with classes and relationships:

- **PhysicalObject**
  - position: Coord
  - shape: Shape

- **RoadObject**
  - speed: Int
  - acceleration: Int
  - orientation: Int
  - direction: Direction

- **RoadSegment**
  - speedLimit: Int

- **Lane**
  - roadObj
  - roadSeg
  - 1..*

- **Driver**
  - Drives

- **AutoSoftCar**
  - ignition: IgnitionState

- **Contains**
  - AutoSoft

- **«AutoSoft»**
  - BDS

- **«inputs»**
  - IgniteOn()
  - IgniteOff()
  - Steer(value: Int)
  - Accelerate(value: Int)
  - Decelerate(value: Int)

- **enum IgnitionState = {on, off}**
FORML – Behaviour Model

SPL AutoSoft
feature BDS

feature-machine main

on

acceleration

t3: Accelerate(value) / 
a1: AutoSoftCar.acceleration := acceleration()

waitAccelerate

deceleration

t4: Decelrate(value) /
a1: AutoSoftCar.acceleration := deceleration()

waitDecelerate

steering

t5: Steer(value) /
a1: AutoSoftCar.orientation := orientation()

waitSteer

off

t1: IgniteOn() / 
a1: AutoSoftCar.ignition := on

t2: IgniteOff() / 
a1: AutoSoftCar.ignition := off
The Translator

- Takes a FORML model as input and produces an Alloy model
- Written in the Turing eXtender Language (TXL)
- I have only extended the translator, initial parts of it were written by Jan Gorzny (now at U of T)
Translator Architecture

- Input
- FORML Parsing Step
- Alloy Creation Step
- Global Variables
- Output
Translation Overview

FORML Metamodel

Conforms To

Knows

FORML Model

TXL Translator

IN

OUT

Knows

Alloy Metamodel

Knows

Conforms To

Alloy Model
Actions on Transitions

- A transition can have several World Change Actions (WCAs) on it
- The types of WCAs are:
  - Adding an object
  - Removing an object
  - Changing an attribute on an object
- Each object has a predicate created for each of the possible WCAs
How does the analysis take place?

(1) Two transitions are chosen
(2) Each transition has its action(s) executed, resulting in 2 new Future Instances
(3) The resulting futures are compared to find locations where the post conditions overlap
WCA Example

<table>
<thead>
<tr>
<th>AutosoftCar</th>
</tr>
</thead>
<tbody>
<tr>
<td>ignition: IgnitionState</td>
</tr>
</tbody>
</table>

```plaintext
pred remove_AutosoftCar (ws0, ws1 : WS, ol : AutosoftCar) {
    ol in ws0.AutosoftCars
    ws1.AutosoftCars = ws0.AutosoftCars - ol
}

pred add_AutosoftCar (ws0, ws1 : WS, ol : AutosoftCar) {
    ol not in ws0.AutosoftCars
    ws1.AutosoftCars = ws0.AutosoftCars + ol
}

pred change_AutosoftCar_ignition (ws0, ws1 : WS, ol : AutosoftCar, v1 : IgnitionState) {
    ol in ws0.AutosoftCars
    ol.(ws1.AutosoftCar_ignition) = v1
}
```

Removing an object
Adding an object
Changing the ignition attribute
Transition Example

```plaintext
pred AutoSoft_BDS_main_t1 (ws0, ws1 : WS, al_v1 : IgnitionState, al_o1 : AutoSoftCar) {
  ws1.AutoSoftCars = ws0.AutoSoftCars
  ws1.AutoSoftCar_orientation = ws0.AutoSoftCar_orientation
  ws1.AutoSoftCar_acceleration = ws0.AutoSoftCar_acceleration
  ws1.Decelerate_value = ws0.Decelerate_value
  change_AutoSoftCar_ignition [ws0, ws1, al_o1, al_v1]  
}  
```
3 Kinds of Analysis

- Pairs of transitions which can remove and change the same object
- Pairs of transitions which can both change the same attribute on an object
- Single transitions which violate world state constraints
- Each method is encoded as an Alloy assertion
World State Constraints (WSC)

- **pred WSC(world_state){...}**
- Set of constraints over a single instance of the world model
- Implemented as a predicate in the Alloy model
- Contains:
  - Encodes cardinality constraints
  - Constraints specified by the user in their FORML model
Assertion Example

• An example of an assertion to check that a transition does not violate a World State Constraint

```plaintext
assert WSC_AutoSoft_BDS_main_t1 {
    all a1_v1 : IgnitionState, a1_o1 : AutoSoftCar |
    WSC [ws0] and AutoSoft_BDS_main_t1 [ws0, ws1, a1_v1, a1_o1] implies WSC [ws1]
}
```
World State Transition Constraints

- pred WSTC (world_state1, world_state2) { ... }

- Needed when a transition moves to a future state

- These encode constraints such as:
  - The parts of a composition must belong to the same whole over their lifetime

- Each transition must also encode its frame conditions
Demo

The Technology Demo:

The software isn’t 100% complete.

If it had a user interface you would see something here... here... and sometimes here.

And then you’d be saying, “I gotta get me some of that.”

Any questions?
FORML – Behaviour Model

SPL AutoSoft
feature BDS

feature-machine main

off

on

acceleration

\texttt{t3: Accelerate(value) / \}
\texttt{a1: AutoSoftCar.acceleration := acceleration()}

\textbullet\rightarrow \text{waitAccelerate}

deceleration

\texttt{t4: Decelerate(value) / \}
\texttt{a1: AutoSoftCar.acceleration := deceleration()}

\textbullet\rightarrow \text{waitDecelerate}

steering

\texttt{t5: Steer(value) / \}
\texttt{a1: AutoSoftCar.orientation := orientation()}

\textbullet\rightarrow \text{waitSteer}
Resulting Model

SPL AutoSoft feature BDS

feature-machine main

[Diagram]

on

acceleration

\[ t3: \text{Accelerate(value)} / \]
\[ a1: \text{AutoSoftCar.acceleration := acceleration()}) \]

\[ \bullet \rightarrow \text{waitAccelerate} \]

\[ t2: \text{IgniteOff() / a1: AutoSoftCar.ignition := off} \]

deceleration

\[ t4 > t3: \text{Decelerate(value)} / \]
\[ a1: \text{AutoSoftCar.acceleration := deceleration()}) \]

\[ \bullet \rightarrow \text{waitDecelerate} \]

steering

\[ t5: \text{Steer(value)} / \]
\[ a1: \text{AutoSoftCar.orientation := orientation()}) \]

\[ \bullet \rightarrow \text{waitSteer} \]
Results

- Translating the entire World Model and important parts of the Behaviour Model
- Implemented 3 methods of finding conflicts between pairs of transitions:
  - 2 transitions changing the same object
  - 1 transition removing, 1 transition changing the same object
  - 1 transition violating the World State Constraints
Editor Support

• Created a simple Emacs Major Mode for FORML
  • eLisp is just a DSL
• Provides syntax highlighting, code completion and (buggy) automatic indentation
Limitations

- Triggers and Guards on transitions are not used
  - The resulting model from the demo would still give a counterexample if re-translated and analyzed
  - This leads to false positives in an otherwise conflict free model
- Naming conflicts may occur
  - This is due to a limitation of TXL
Future Work

- Multiple actions per transition
- Simplified expressions (probably can't be done in TXL)
- Generalize the idea of an interaction to generalize the assertions
- (Refactoring)
TXL

• A fairly simple pattern matching Domain Specific Language

• Works with ambiguous grammars which is great

• Complex things need to be done in unintuitive ways
  • There is an extension called eTXL (however, it is no longer active and the only documentation is a Masters thesis)
Conclusion

- Using this method makes it possible to show correctness of a part of your model
  - This does not prove correctness of your entire model, only a facet of it
- By using a DSL like TXL to create the translator the initial learning overhead was lowered, but it complicated things later on