Experience and Insight from the INESS Project

Helle Hvid Hansen

Eindhoven University of Technology

RSTRC Workshop, 27 Sep 2011, York, UK
The INESS Project (EU FP7)

**INtegrated European Signalling System**

**Goal:**
Develop a harmonized and verified specification of common core functionality for a new generation of European interlockings.

**Purpose:**
Interoperability between countries, increase competition, faster certification process.

**Partners:**
- Coordinator: UIC (International Union of Railways)
- Railway operators: ProRail, DB Netz, ...
- Industry: Siemens, Alstom, Bombardier, Thales, ...
- Universities: Eindhoven, Twente, Southampton, York.

**Duration:** Oct 2008 - Sep 2011 (extended to Mar 2012).
Task of the Universities

**Task:**

*Formally verify Executable UML model of Common Core functional requirements against a set of safety properties.*

xUML lacks tooling for formal verification.

**Apply our general-purpose verification technology:**

- Model checking: mCRL2, LTSmin (TUE and UT).
- Theorem proving: UML-B/Event-B, Rodin (Southampton)

**Strategy:**

1. Translate xUML into formal language of our tools.
2. Formalise safety properties (logic formula / xUML).
3. Formally verify that safety properties hold using our tools.
Task of the Universities

Task:

Formally verify Executable UML model of Common Core functional requirements against a set of safety properties.

xUML lacks tooling for formal verification.

Apply our general-purpose verification technology:

- Model checking: mCRL2, LTSmin (TUE and UT).
- Theorem proving: UML-B/Event-B, Rodin (Southampton)

Strategy:

1. Translate xUML into formal language of our tools.
2. Formalise safety properties (logic formula / xUML).
3. Formally verify that safety properties hold using our tools.
Task of the Universities

Task:
Formally verify Executable UML model of Common Core functional requirements against a set of safety properties.

xUML lacks tooling for formal verification.

Apply our general-purpose verification technology:
- Model checking: mCRL2, LTSmin (TUE and UT).
- Theorem proving: UML-B/Event-B, Rodin (Southampton)

Strategy:
1. Translate xUML into formal language of our tools.

2. Formalise safety properties (logic formula / xUML).

3. Formally verify that safety properties hold using our tools.
Model Checking in INESS

Given formal model (of system behaviours) + property $P$, check that all system behaviours satisfy $P$.

- Model checking of xUML model is w.r.t. particular instance: model instance = generic xUML model + track layout.
- Formal model is expressed in process algebra mCRL2.
- Symbolic model checking using LTSmin.
Model Checking in INESS

Given formal model (of system behaviours) + property $P$, check that all system behaviours satisfy $P$.

- Model checking of xUML model is w.r.t. particular instance: model instance = generic xUML model + track layout.
- Formal model is expressed in process algebra mCRL2.
- Symbolic model checking using LTSmin.
General approach:

- Safety: “system cannot reach any bad states”.
- Verifying safety: reachability analysis.
- Safety property formalised as logic formula (μ-calculus).

Our approach:

- Express safety property as UML state machine ("observer"): send error signal when bad state is reached.
- Translate observers automatically with xUML model.
- Verification: search for error action.

Advantages:

- Scalability: larger systems can be verified.
- Use of UML facilitates communication with UML modellers.
Model Checking Safety Properties

**General approach:**
- Safety: *system cannot reach any bad states*.
- Verifying safety: reachability analysis.
- Safety property formalised as logic formula (mu-calculus).

**Our approach:**
- Express safety property as UML state machine (*“observer”*): *send error signal when bad state is reached*.
- Translate observers automatically with xUML model.
- Verification: search for error action.

**Advantages:**
- Scalability: larger systems can be verified.
- Use of UML facilitates communication with UML modellers.
Model Checking Safety Properties

**General approach:**
- Safety: *"system cannot reach any bad states"*.
- Verifying safety: reachability analysis.
- Safety property formalised as logic formula (mu-calculus).

**Our approach:**
- Express safety property as UML state machine (*"observer"*):
  - send error signal when bad state is reached.
- Translate observers automatically with xUML model.
- Verification: search for error action.

**Advantages:**
- Scalability: larger systems can be verified.
- Use of UML facilitates communication with UML modellers.
Our first experiment: Micro Interlocking

Common Core was only due late in project timeline. We started with a toy example manufactured by Markus Schacher of KnowGravity.

**Purpose of the experiment:**

- Learn how to translate xUML constructs into mCRL2
- Stimulate discussions about interpretation of xUML
- Make preliminary assessment of feasibility of our verification strategy
- Serve as a first test case for an automatic translation
Our first experiment: Micro Interlocking

Common Core was only due late in project timeline. We started with a toy example manufactured by Markus Schacher of KnowGravity.

**Purpose of the experiment:**
- Learn how to translate xUML constructs into mCRL2
- Stimulate discussions about interpretation of xUML
- Make preliminary assessment of feasibility of our verification strategy
- Serve as a first test case for an automatic translation
So what does Micro Interlocking look like?

Class diagram:

One of the state diagrams:

Track layout:
Translation from xUML to mCRL2

Interpreting the xUML:
- UML semantics underspecified (what did the modeller mean?)
- Different runtime semantics and event priorities possible.
  ⇒ Translation must resolve ambiguities.

Expressing xUML in mCRL2:
- UML states translate to process parameters.
- UML events translate to actions.
- ...
- Unbounded object event pools ⇒ infinite state space
- Non-local data access ⇒ extra communication
Translation from xUML to mCRL2

Interpreting the xUML:
- UML semantics underspecified (what did the modeller mean?)
- Different runtime semantics and event priorities possible.
  ⇒ Translation must resolve ambiguities.

Expressing xUML in mCRL2:
- UML states translate to process parameters.
- UML events translate to actions.
- ...
  - Unbounded object event pools ⇒ infinite state space
  - Non-local data access ⇒ extra communication
Translation from xUML to mCRL2

**Interpreting the xUML:**
- UML semantics underspecified (what did the modeller mean?)
- Different runtime semantics and event priorities possible.
  ⇒ Translation must resolve ambiguities.

**Expressing xUML in mCRL2:**
- UML states translate to process parameters.
- UML events translate to actions.
  
  ...  
- Unbounded object event pools ⇒ infinite state space
- Non-local data access ⇒ extra communication
Automated Translation from xUML to mCRL2

Implemented using model transformation technology Epsilon, developed in York.

**Architecture:**
- Parser for action and expression language (non-UML syntax)
- Transformation from UML to intermediate representation (iUML).
- Code generation from iUML to mCRL2.

**Input:**
- interlocking xUML model
- instance specification (track layout)
- safety property (as xUML model)

**Output:** mCRL2 specification
Automated Translation from xUML to mCRL2

Implemented using model transformation technology Epsilon, developed in York.

**Architecture:**
- Parser for action and expression language (non-UML syntax)
- Transformation from UML to intermediate representation (iUML).
- Code generation from iUML to mCRL2.

**Input:**
- interlocking xUML model
- instance specification (track layout)
- safety property (as xUML model)

**Output:** mCRL2 specification
We instantiated Micro for the following track layouts:

(a) Layout 1
(b) Layout 2
(c) Layout 3
(d) Layout 4
(e) Layout 5

Fig. 6: Several track layouts used to test the feasibility of the verification task.
Table 1: State spaces of the layouts from Figure 6, without observers, where each state machine is assumed to have at most one message in its event pool.

<table>
<thead>
<tr>
<th>Layout</th>
<th>Components</th>
<th>Routes</th>
<th>States</th>
<th>Runtime (s)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s,t,p</td>
<td>t</td>
<td>1</td>
<td>0.7 × 10^4</td>
<td>0.0161</td>
</tr>
<tr>
<td>2</td>
<td>s,t,p</td>
<td>t</td>
<td>1</td>
<td>0.3 × 10^9</td>
<td>0.2576</td>
</tr>
<tr>
<td>3</td>
<td>s,t,p</td>
<td>t</td>
<td>4</td>
<td>0.9 × 10^11</td>
<td>0.7348</td>
</tr>
<tr>
<td>4</td>
<td>s,t,p</td>
<td>t</td>
<td>8</td>
<td>0.9 × 10^13</td>
<td>1.3919</td>
</tr>
<tr>
<td>5</td>
<td>s,t,p</td>
<td>t</td>
<td>6</td>
<td>0.8 × 10^23</td>
<td>2.6052</td>
</tr>
<tr>
<td>6</td>
<td>s,t,p</td>
<td>t</td>
<td>7</td>
<td>0.0 × 10^30</td>
<td>&gt; 496 h</td>
</tr>
</tbody>
</table>

Resource consumption is for the LTSmin symbolic model checker run on an Intel Xeon X5550 machine with 148 GB of internal memory; a saturation-like exploration strategy was used. The running times exclude the time to load the model.
Verification of Micro Interlocking (2)

(f) Layout 6

![Diagram of a railway layout with nodes and edges labeled for verification purposes.]

Table 1: State spaces of the layouts from Figure 6, without observers, where each state machine is assumed to have at most one message in its event pool. Resource consumption is for the LTSmin symbolic model checker run on an Intel Xeon X5550 machine with 148 GB of internal memory; a saturation-like exploration strategy was used. The running times exclude the time to load the model.
Verification: feasibility

Resource consumption in state space exploration:

<table>
<thead>
<tr>
<th>Layout</th>
<th>Elts</th>
<th>Routes</th>
<th>States</th>
<th>Time</th>
<th>Mem (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>$1.7 \times 10^4$</td>
<td>0.01 sec</td>
<td>61</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>2</td>
<td>$1.3 \times 10^9$</td>
<td>0.25 sec</td>
<td>76</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2</td>
<td>$4.9 \times 10^{11}$</td>
<td>7.73 sec</td>
<td>86</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>3</td>
<td>$8.9 \times 10^{13}$</td>
<td>19.39 sec</td>
<td>115</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>6</td>
<td>$6.8 \times 10^{23}$</td>
<td>43 mins</td>
<td>3133</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>8</td>
<td>$7.0 \times 10^{31}$</td>
<td>1.7 days</td>
<td>&gt; 32 GB</td>
</tr>
</tbody>
</table>

Safety property:

“A point belonging to an established route should not move.”

Found violation already in layout 2.
Verification: feasibility

Resource consumption in state space exploration:

<table>
<thead>
<tr>
<th>Layout</th>
<th>Elts</th>
<th>Routes</th>
<th>States</th>
<th>Time</th>
<th>Mem (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>$1.7\times10^4$</td>
<td>0.01 sec</td>
<td>61</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>2</td>
<td>$1.3\times10^9$</td>
<td>0.25 sec</td>
<td>76</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2</td>
<td>$4.9\times10^{11}$</td>
<td>7.73 sec</td>
<td>86</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>3</td>
<td>$8.9\times10^{13}$</td>
<td>19.39 sec</td>
<td>115</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>6</td>
<td>$6.8\times10^{23}$</td>
<td>43 mins</td>
<td>3133</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>8</td>
<td>$7.0\times10^{31}$</td>
<td>1.7 days</td>
<td>&gt; 32 GB</td>
</tr>
</tbody>
</table>

Safety property:

“A point belonging to an established route should not move.”

Found violation already in layout 2.
Verification of Micro Interlocking: Remarks

Micro is too simple to be safe (we expect errors!)

We have demonstrated that

- we can find errors.
- errors can be found in small track layouts.
- error traces can be reported for diagnostics.

We have demonstrated a methodology.
Verification of Micro Interlocking: Remarks

Micro is too simple to be safe (we expect errors!)

We have demonstrated that

- we can find errors.
- errors can be found in small track layouts.
- error traces can be reported for diagnostics.

We have demonstrated a methodology.
Prototype Tool Chain

A generic verification methodology

- Based on Eclipse development platform.
- Works for xUML models similar to INESS example models.
- Safety properties supplied as UML state machines.
- Error trace is visualised as UML sequence diagram.
Conclusion

Challenges encountered:

- xUML semantics is underspecified and ambiguous.
- State space explosion (still, errors may be found in small layouts).
- In theorem proving, abstraction refinement needs much human insight ⇒ low degree of automation.

Future work:

- Verify Common Core xUML model (still to be delivered).
- Automatic translation from xUML to Promela (SPIN model checker) using iUML.
- Scalability (improve model checker, compositional verification, abstractions, ...)

Conclusion

Challenges encountered:

- xUML semantics is underspecified and ambiguous.
- State space explosion (still, errors may be found in small layouts).
- In theorem proving, abstraction refinement needs much human insight ⇒ low degree of automation.

Future work:

- Verify Common Core xUML model (still to be delivered).
- Automatic translation from xUML to Promela (SPIN model checker) using iUML.
- Scalability (improve model checker, compositional verification, abstractions, ...)

INES University Teams

Eindhoven (NL):
- Jos Baeten,
- Jan Friso Groote,
- Helle Hvid Hansen (PD)
- Bas Luttik,
- Mohammad Mousavi.

Twente (NL):
- Jeroen Ketema (PD),
- Jaco van de Pol.

(PD = PostDoc)

York (UK):
- Steve King,
- Richard Paige,
- Louis Rose (PD),
- Osmar Marchi dos Santos (PD),
- Jim Woodcock.

Southampton (UK):
- Vitaly Savicks (PD),
- Colin Snook.