Application of Model Checking for the Model-based Analysis of Interactive Systems
(An experience report)

Karsten Loer
Karsten.Loer@gi-group.com

Overview
• General idea: Using model-checking (MC) to discover latent errors in open system models
• Sample application: mobile appliances
  – using MC to assess safety implications of the introduction of a new mobile device (“Pucketizer”) into a processing plant
• Technical solutions:
  – Modelling interactive systems as open systems
  – Continuous analysis and model refinement
  – Introducing assumptions on behaviour of open variables
• Conclusions

Sample domain: A processing plant
Modelling Interactive Systems

CONTROL MECHANISM (=device core)

USER (TASKS) – ENVIRONMENT

Model 1: Controlled devices and environment
Analysis: Model “plausibility”

Does the model behave as intended?
- “sanity”: deadlock-freedom, state/event reachability
- “goal reachability”:
  • Can product C be produced?
  • What is the easiest way to produce product C?
  • What is the ‘best’ way to produce C under assumptions a...a?
  • Is it possible to reach unsafe states? ...

Temporal aspects of interest:
Characteristics of user tasks* in terms of temporal system-behaviour:
- task sequencing:
  • task interleaving, suspension and resumption
- task durations and optimisation:
  • e.g. best-case/worst-case execution times
  • multi-valued decision criteria
- task allocation
  • who needs to perform the task and when?

*note that we do not attempt to produce user models
Formulating System Requirements

- only input sequences containing “1-2-3” are accepted:
  - “all sequences containing “1-2-3” are accepted”
    \[ \text{AG} \left( \text{“1”} \land \text{AX} \left( \text{“2”} \land \text{AX} \text{“3”} \right) \right) \rightarrow \text{s3} \]
  - “any other sequence is rejected”
    \[ \text{AG} \left( \neg \left( \text{“1”} \land \text{AX} \left( \text{“2”} \land \text{AX} \text{“3”} \right) \right) \right) \rightarrow \neg \text{s3} \]
  - the accepting state can only be reached, if the inputs are made within a particular duration

![Diagram of system model, system property, and model checker]
Model-checking traces

• *trace* = sequence of execution steps that demonstrate how a state that violates (or demonstrates) a property can be reached from the initial system state.

• traces can point the analyst to:
  – violating user/device behaviour
  – task optimisations
  – recovery procedures

---

**Trace Comparison**

Goal/Property:
“Reachability of a state where end product C is released”

- a) Control room interface
- b) Pucketizer
- c) Pucketizer (forgetful operator)
- d) Pucketizer w. Interlock
- e) Pucketizer w. Interlock (reluctant operator)

---

Explorative application of model checking

1. starting from a device-centric model
   => all possible user inputs
2. gradually add assumptions about user and environment behaviour
   => sub-set of “sensible” user inputs

• formulation of assumptions:
  1. as part of the (temporal logic) property specification
  2. by model enhancements
     (e.g. “observer automata” or model decorations)
Influence of user task models on explored input space

- no task model
  - "all possible" user inputs

- restricted input space
  - only plausible behaviours are explored

- normative task model
  - only inputs that form a prescribed task

Restricting the input space:

Focus of analysis:

Given:
1. a device specification and
2. a desired target "situation" (= state of the device and environment)

Question: What assumptions can/need to be made about the user?

Restricting the input space:

- Goal: Limit search by adding restrictions (i.e. set of state machines) on the user behaviour
- Example: Prevent tank overflow
  "Whenever the user realises that pump 1 is operating full volume while its target tank is close to full the user will switch off the pump"
Normative task models

Focus of analysis:

Given: A specification of
1. the device under development,
2. relevant parts of the environment and
3. a normative task model

Question: *What states of the environment can be reached?*

Example task: “Once all pumps are off, switch pump 1 ON (after at most \( n \) steps)”

Adding assumptions about operator behaviour

- temporal logic assertions:
  “the operator always forgets to store pump controls”

```c
assert SAN1:
F ((PUMP5CTRLM.state=PMP5ON)
& (TANK1.state=HOLDS_C));
assert alwaysForget:
G!(savePmp1Controls| ... |savePmp5Controls);
assume alwaysForget;
using alwaysForget prove SAN1;
```
Adding assumptions about operator behaviour

• observer automata: the “forgetful” operator

• check properties under the assumption that violation states ("forget") are absent

Parametric search:

**Question:** "For what value(s) of variable x does property π hold?"

• iterative analysis:
  "Does π hold for x=n, for x=n-1, for x=n+1, ... ?"

• alternative: query checking
  [Chan 2000, Gurfinkel et al. 2002]

Conclusions:

Model checkers are good at...

• exhaustive and “automatic” analysis*
  (*provided that “appropriate input” is supplied)

• analysis of behavioural reachability properties
  – ordering/sequencing of tasks:
    e.g. Hollnagel's error phenotypes: repetition, reversal, omission, delay, premature action, replacement, insertion, and "intrusion"
  – mode complexity
  – dialogue control:
    visibility of action effects, visibility of available actions, recoverability, consistency, error prevention, flexibility, efficiency of use
  – timing
  – Usability heuristics (cf. [Loer & Harrision 2000])
Conclusions:
Model checking has limitations…

• deliver single, sometimes “trivial”, traces
• hard/impossible to determine tendencies, e.g. certain types of user behaviour, characteristics of components that contribute to potential errors …

• technique does not suggest corrections

• difficult/unsuitable to use for analysis of representational properties (layout, direct manipulation etc.)

Conclusions: Tool suitability

<table>
<thead>
<tr>
<th>feature</th>
<th>tool</th>
<th>SMV</th>
<th>Uppaal</th>
<th>HyTech</th>
</tr>
</thead>
<tbody>
<tr>
<td>temporal properties</td>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>simulation</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>parametric search</td>
<td>(0)₁</td>
<td>0₁</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>continuous variables</td>
<td>+</td>
<td>0</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>scalability</td>
<td>+</td>
<td>0</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>usability</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

⁠¹ via iteration over “parameter” variable(s)
⁠² execution step as unit of progress

Acknowledgements

• This work was performed in collaboration with Michael Harrison of the University of Newcastle upon Tyne.

• Many thanks for their support to the colleagues from:
  – BAE SYSTEMS Dependable Computing Systems Centre at the University of York, UK.
    http://www.cs.york.ac.uk/hise/dcsc
  – Interdisciplinary Research Collaboration in Dependability (DIRC project), UK.
    http://www.dirc.org.uk
Further Reading


