TEAMOD
Test and Model Checking

Bachelor and Master Project
Bachelor Project: Winter Semester 2018/19 – Summer Semester 2019
Master Project: Winter Semester 2019/20 – Summer Semester 2020
Jan Peleska and Wen-ling Huang
{jp,huang}@cs.uni-bremen.de
Background

- Embedded control systems are omnipresent in our daily life
Background

- The growing complexity of embedded applications leads to a dramatic increase of verification costs.
- These costs will further increase with autonomous systems deployment.
  - Robots
  - Vehicles
  - Cars
  - Ships
  - Trains
We can efficiently handle low complexity . . .

. . . but are not yet prepared for the complexity of future applications
TEAMOD Objectives

- Bounded model checking
- Model-based testing
Bounded model checking

Model

\[
I(s_0) \land \bigwedge_{i=1}^{k} R(s_{i-1}, s_i) \land G(s_0, \ldots, s_k)
\]

Verification Goal

\[ \Phi \]
Concrete test suite

1. \((v_{est}, v_{mrsp}) = (3.1, 100). (50, 100), (80, 100), (100.5, 100)\)
2. \((v_{est}, v_{mrsp}) = (90.1, 100). (110, 120), (110.5, 110), (110.5, 100)\) ...
3. \((v_{est}, v_{mrsp}) = (90.1, 200). (210, 220), (210.5, 210), (210.5, 200)\) ...
4. ...

Abstract test suite

\[ \mathcal{W} = P \left( \bigcup_{i=0}^{m-n} \mathcal{I} \cdot W \right) \]
TEAMOD Objectives

- Bounded model checking and model-based testing should be integrated in a common tool platform
  
  - Model checking is needed to verify the test model
  
  - Model-in-the-loop testing is a light-weight version of model checking
TEAMOD Bachelor Project

- Three sub-projects
  - Methods
  - System under test development
  - Modelling, bounded model checking, and model-based testing
Development

Märklin Modelleisenbahn

Märklin Central Station 6021
Hardware-in-the-loop test

Test engine

System under test
TEAMOD Master Project

- Four sub-projects
  - Algorithms
  - Autonomous train control
  - Safety monitor development
  - Graphical interfaces for scenario-based testing
Algorithms

1. Theory

![Diagram illustrating the satisfaction condition](image)

Given a pair \((T, T^*)\) fulfilling the satisfaction condition, this allows to translate complete testing theories existing in \(S_{g_1}\) to likewise complete testing theories in (sub-domains of) \(S_{g_1}\).

**Theorem 2.1** Suppose that \(T_{g_2} : F_2 \rightarrow P(TC(S_{g_2}))\) with \(F_2 \subseteq F(S_{g_2}, \leq)\) is a sound (respectively exhaustive, complete) testing theory. Define

\[
F_1 = \{ (S, \leq_1, Dom_1) \in F(S_{g_1}, \leq_1) \mid \exists Dom_2 \subseteq S_{g_2} : T(Dom_2) \subseteq Dom_2 \land (T(S), \leq_2, Dom_2) \in F_2 \}.
\]

Then \(T_{g_1} : F_1 \rightarrow P(TC(S_{g_1}))\) defined by

\[
T_{g_1}(S, \leq_1, Dom_1) = T^*(T_{g_2}(T(S), \leq_2, Dom_2)),
\]

such that \(T(Dom_1) \subseteq Dom_2\), is a sound (respectively exhaustive, complete) testing theory.

**Proof** Suppose \(T_{g_2}\) is sound (exhaustive). Let \(F_1 = (S, \leq_1, Dom_1) \in F_1\) and \(F_2 = (T(S), \leq_2, Dom_2) \in F_2\) be any fault models in \(F_1\) and \(F_2\) respectively, satisfying \(T_{g_1}(F_1) = T^*(T_{g_2}(F_2))\). Let \(S' \in Dom_1\), then \(T(S') \in Dom_2\), and

\[
S' \leq_1 T(S') \leq T(S) \quad \text{satisfaction condition SCI1}
\]

\[
\forall U \in T_{g_2}(F_2) : (S') \Rightarrow T(S) \text{ pass } U \quad [T_{g_2} \text{ is sound; } (\Rightarrow \text{ pass } U) \text{ SCI2}]
\]

\[
\forall U \in T_{g_2}(F_2) : S' \text{ pass } T^*(U) \quad \text{satisfaction condition SCI2}
\]

\[
S' \text{ pass } T_{g_1}(F_1) \quad T_{g_1}(F_1) = T^*(T_{g_2}(F_2))
\]

Hence \(T_{g_1}(F_1)\) is a sound (exhaustive) test suite for any fault model \(F_1 \in F_1\). Consequently, \(T_{g_1}\) is sound (exhaustive). Since completeness is the combination of soundness and exhaustiveness, this proves the theorem.

2. Algorithm design

8. For each \((i, j) \in J\), collect all disjuncts

\[
g_{i', i''} \leftarrow (d_{i'}, e_{i'}) \land (m', y') = (d_{i''}, e_{i''}) \land (x' = x)
\]

satisfying \(i', i'' \in RTR_{i,j}\) and consequently \(i' = i, i'' = j\) and merge them into a single disjunct

\[
g_{i,j} \land (m, y) = (d_i, e_i) \land (m', y') = (d_j, e_j) \land (x' = x)
\]

where

\[
g_{i,j} = \bigvee_{i', i'' \in RTR_{i,j}} g_{i', i''}
\]

9. Terminate by returning \(R\).

3. Programming

```cpp
void RttGenTestProcTree::generateTestCases()
{
    // The root of the test procedure tree carries the memory state before
    tprocRoot = &t;
    auto mSys = static_cast< RttGenConcreteLatticeMemory* > (system->get
    mSys->setParentSystem(system);
    return new RttGenTestProcTree(mSys, 0, true, 0);
}
```

// Initialise interpreters
```cpp
sim = new simlib::Simulator(*system);
sim->setAddGoalsOrder(additionalGoals->getOrdered());
sim->setAddGoalsOrder(additionalGoals->getUnordered());
sim->setTestCaseDb(tcDb);
sim->setParms(parms);
nextTimeTickFromSimulator = 0;
```
Autonomous train control & Safety monitor

Safety monitor

Generator

Φ_{Safe}
Graphical interfaces for scenario-based testing
Accompanying Lectures

• **Test automation** [highly recommended]

• Theory of reactive systems

• Systems of high quality, safety, and security

• Specification of embedded systems

• Operating systems

• Real-time operating systems development