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


+
32·44
M
42

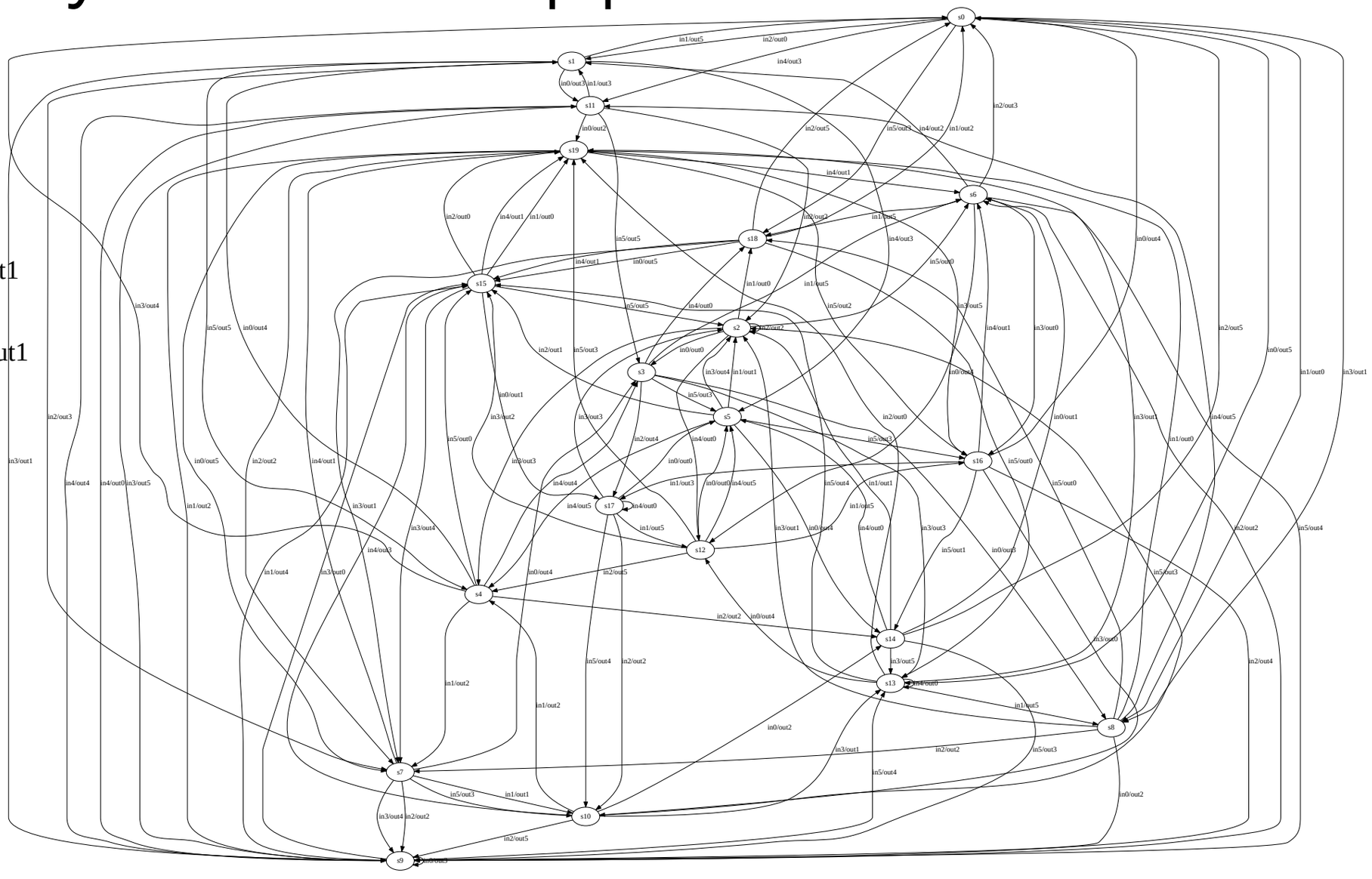
Background

- The growing complexity of embedded applications leads to an 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 . . .

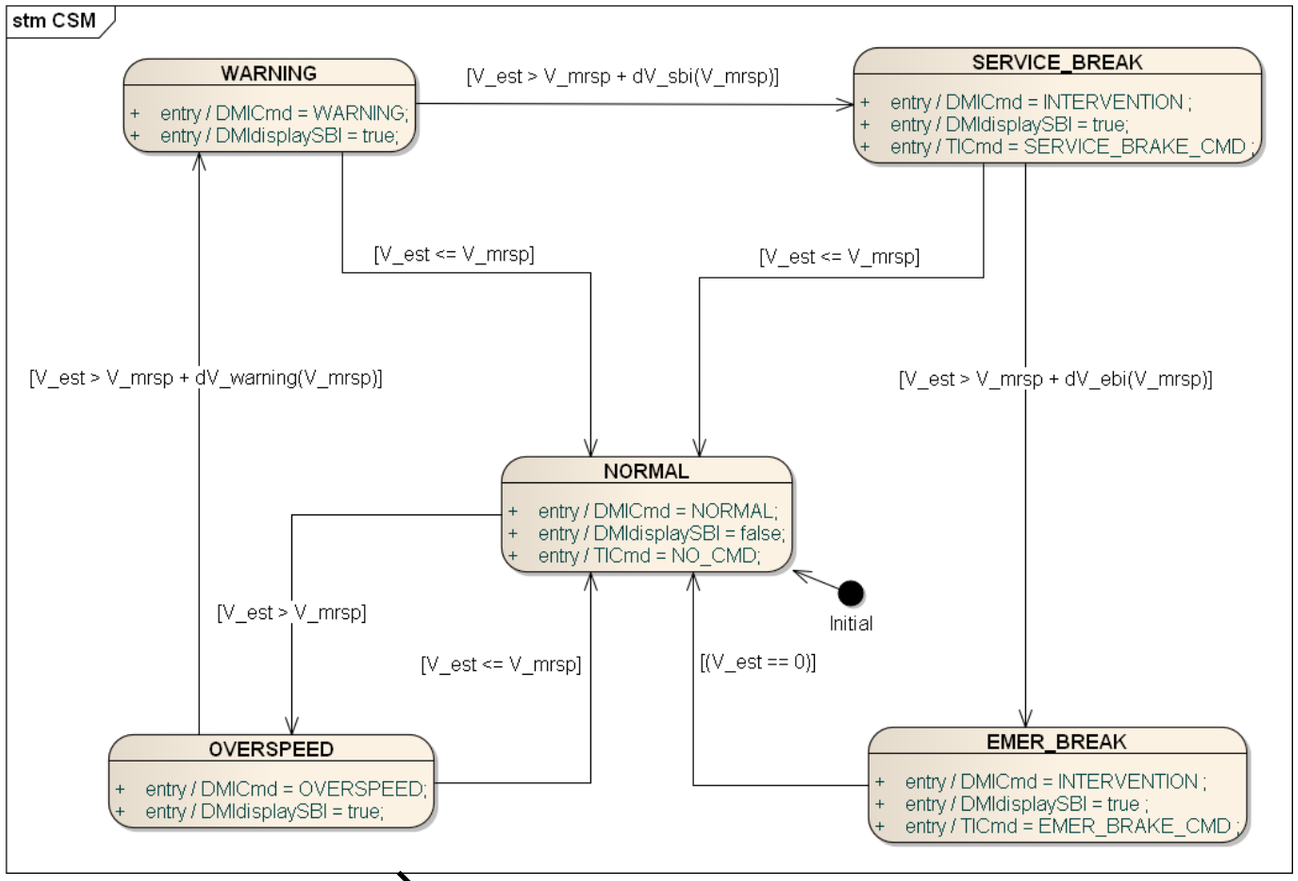
. . . but are not yet prepared for the complexity of future applications



TEAMOD Objectives

- Bounded model checking
- Model-based testing

Model



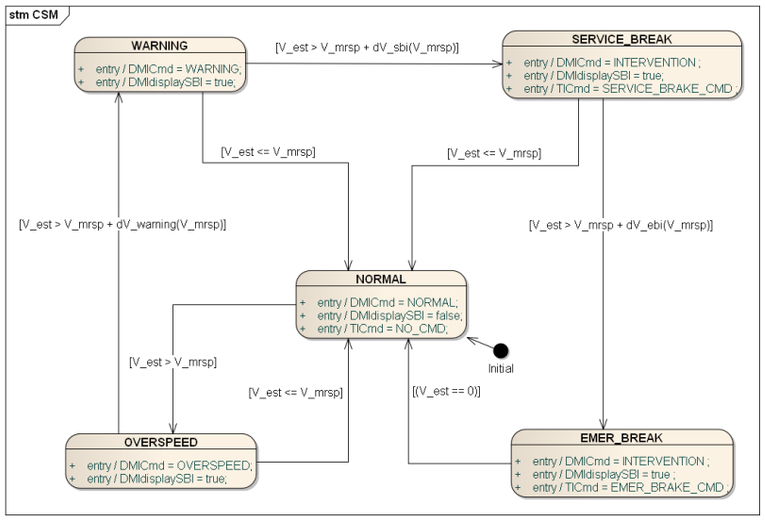
Verification Goal

Φ

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

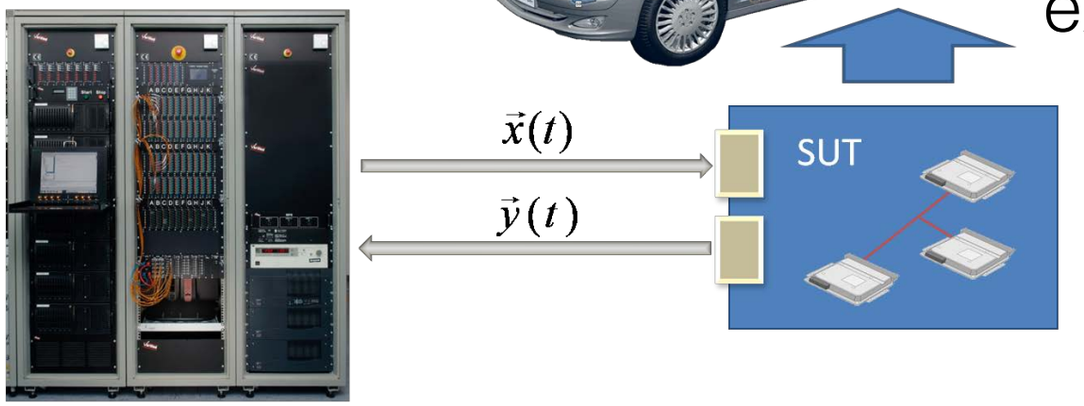
Bounded model checking

Test model (SysML)



Semantic representation

$$
\begin{aligned}
\mathcal{R} &\equiv \bigvee_{i \in \mathrm{IDX}} \left(\alpha_i \wedge (\vec{m}, \vec{y}) = (\vec{d}_i, \vec{e}_i) \wedge (\vec{m}', \vec{y}') = (\vec{d}_i, \vec{e}_i)\right) \\
&\vee \bigvee_{(i,j) \in J} \left(g_{i,j} \wedge (\vec{m}, \vec{y}) = (\vec{d}_i, \vec{e}_i) \wedge (\vec{m}', \vec{y}') = (\vec{d}_j, \vec{e}_j) \wedge \vec{x}' = \vec{x}\right) \\
\mathrm{IDX} &= \ldots \\
J &= \ldots \\
(\vec{d}_1, \vec{e}_1) &= \ldots \\
\ldots &
\end{aligned}
$$



executed against SUT

Concrete test suite

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

Abstract test suite

$$\mathcal{W} = P.\left(\bigcup_{i=0}^{m-n} \overline{\mathcal{I}}^{i}.W\right)$$

Model-based testing

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

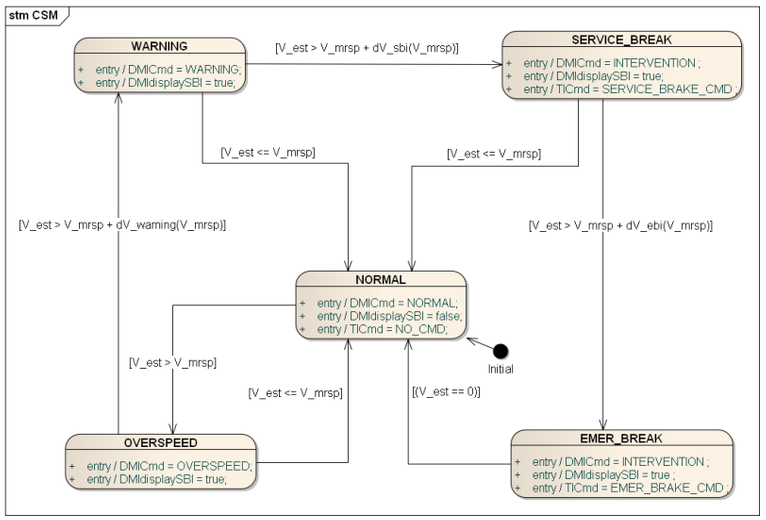
TEAMOD Bachelor Project

- 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

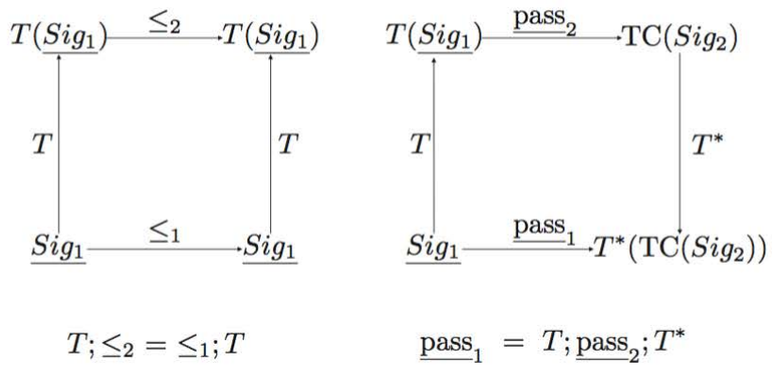
System under test

TEAMOD Master Project

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

Algorithms

1. Theory



$T; \leq_2 = \leq_1; T$ $\qquad$ $\underline{\text{pass}}_1 = T; \underline{\text{pass}}_2; T^*$

Fig. 1. Commuting diagrams reflecting the satisfaction condition

Given a pair (T, T^*) fulfilling the satisfaction condition, this allows to translate complete testing theories existing in Sig_2 to likewise complete testing theories in (sub-domains of) Sig_1.

Theorem 2.1 *Suppose that* $\text{TS}_2 : F_2 \to \mathbb{P}(TC(Sig_2))$ *with* $F_2 \subseteq F(Sig_2, \leq)$ *is a sound (respectively exhaustive, complete) testing theory. Define*

$$F_1 = \{(\mathcal{S}, \leq_1, Dom_1) \in F(\underline{Sig_1}, \leq_1) \mid \exists\, Dom_2 \subseteq Sig_2 : T(Dom_1) \subseteq Dom_2 \wedge (T(\mathcal{S}), \leq_2, Dom_2) \in F_2\}.$$

Then $\text{TS}_1 : F_1 \to \mathbb{P}(TC(Sig_1))$ *defined by*[2]

$$\text{TS}_1(\mathcal{S}, \leq_1, Dom_1) = T^*(\text{TS}_2(T(\mathcal{S}), \leq_2, Dom_2)),$$

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

Proof Suppose TS_2 is sound (exhaustive). Let $\mathcal{F}_1 = (\mathcal{S}, \leq_1, Dom_1) \in F_1$ and $\mathcal{F}_2 = (T(\mathcal{S}), \leq_2, Dom_2) \in F_2$ be any fault models in F_1 and F_2 respectively, satisfying $\text{TS}_1(\mathcal{F}_1) = T^*(\text{TS}_2(\mathcal{F}_2))$. Let $\mathcal{S}' \in Dom_1$. Then $T(\mathcal{S}') \in Dom_2$, and

$$\begin{aligned}
\mathcal{S}' \leq_1 \mathcal{S} &\Leftrightarrow T(\mathcal{S}') \leq_2 T(\mathcal{S}) && \text{[satisfaction condition \textbf{SC1}]}\\
&\Rightarrow \forall\, U \in \text{TS}_2(\mathcal{F}_2) : T(\mathcal{S}')\, \underline{\text{pass}}_2\, U && \text{[}\text{TS}_2\text{ is sound; (" } \Leftarrow \text{ " : } \text{TS}_2\text{ is exhaustive)]}\\
&(\Leftarrow)\\
&\Leftrightarrow \forall\, U \in \text{TS}_2(\mathcal{F}_2) : \mathcal{S}'\, \underline{\text{pass}}_1\, T^*(U) && \text{[satisfaction condition \textbf{SC2}]}\\
&\Leftrightarrow \mathcal{S}'\, \underline{\text{pass}}_1\, T^*(\text{TS}_2(\mathcal{F}_2))\\
&\Leftrightarrow \mathcal{S}'\, \underline{\text{pass}}_1\, \text{TS}_1(\mathcal{F}_1) && \text{[}\text{TS}_1(\mathcal{F}_1) = T^*(\text{TS}_2(\mathcal{F}_2))\text{]}
\end{aligned}$$

Hence $\text{TS}_1(\mathcal{F}_1)$ is a sound (exhaustive) test suite for any fault model $\mathcal{F}_1 \in F_1$. Consequently, TS_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

$$g_{i'.i'_1 \ldots i'_{n'}} \wedge (\mathbf{m}, \mathbf{y}) = (\mathbf{d}_{i'}, \mathbf{e}_{i'}) \wedge (\mathbf{m}', \mathbf{y}') = (\mathbf{d}_{i'_{n'}}, \mathbf{e}_{i'_{n'}}) \wedge (\mathbf{x}' = \mathbf{x})$$

satisfying $i'.i'_1 \ldots i'_{n'} \in \text{RTR}_{i,j}$ and consequently $i' = i$, $i'_{n'} = j$ and merge them into a single disjunct

$$g_{i,j} \wedge (\mathbf{m}, \mathbf{y}) = (\mathbf{d}_i, \mathbf{e}_i) \wedge (\mathbf{m}', \mathbf{y}') = (\mathbf{d}_j, \mathbf{e}_j) \wedge (\mathbf{x}' = \mathbf{x})$$

where

$$g_{i,j} \equiv \bigvee_{i' \cdot i'_1 \cdots i'_{n'} \in \text{RTR}_{i,j}} g_{i' \cdot i'_1 \cdots i'_{n'}}$$

9. Terminate by returning $\mathcal{R}$.

3. Programming

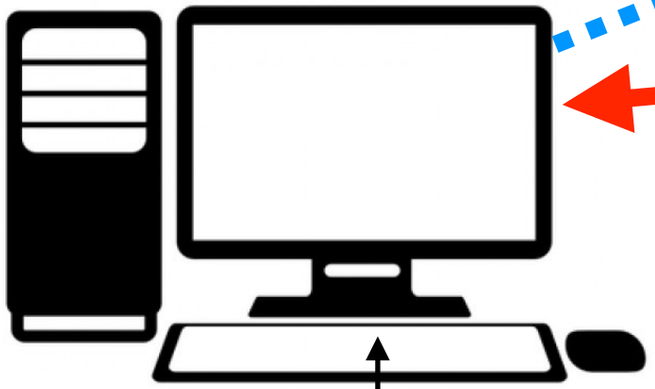
```
void RttTgenGenerator::generateTestCases() {

  // The root of the test procedure tree carries the memory state before
  tprocRoot = [&] {
    auto mSys = static_cast< RttTgenConcreteLatticeMemory* >(system->get
    mSys->setParentSystem(system);
    return new RttTgenTestProcTree(mSys, 0, true, 0);
  }();
  currentTprocNode = tprocRoot;

  remainingSimulationSteps = parms->getSimSteps();

  // Initialise interpreters
  sim = new simlib::Simulator(*system);
  sim->setAddGoalsUnordered(additionalGoals->getUnordered());
  sim->setAddGoalsOrdered(additionalGoals->getOrdered());
  sim->setTestCaseDb(tcDb);
  sim->setParms(parms);
  nextTimeTickFromSimulator = 0;
```

Autonomous train control & Safety monitor

Märklin Modelleisenbahn

Märklin Central Station 6021

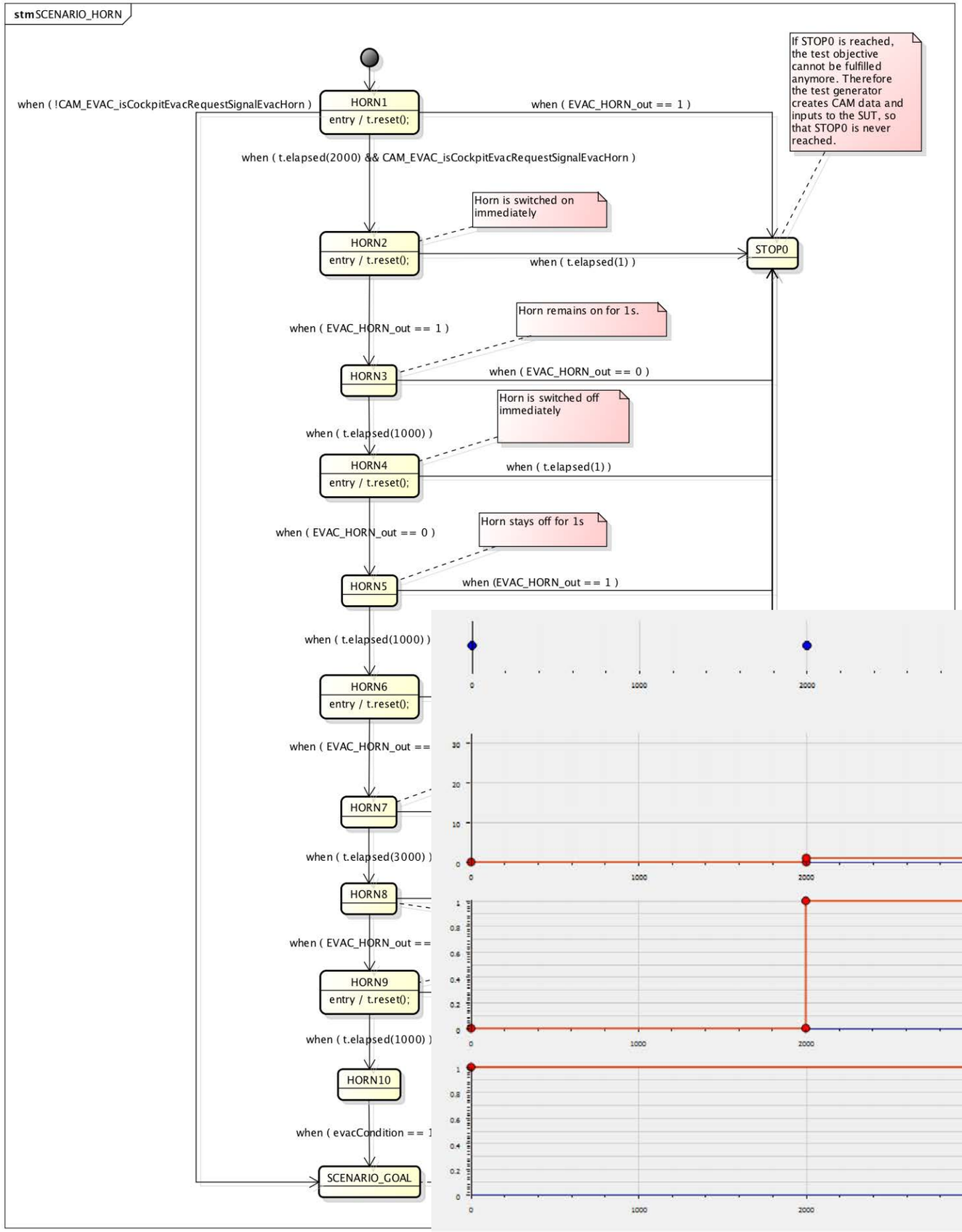
Safety monitor



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