Bachelor & Master Projects and Theses
Chair for Software and Systems Engineering

Prof. Dr. Stefan Leue
University of Konstanz
Chair for Software Engineering

Stefan.Leue@uni-konstanz.de
http://www.sen.inf.uni-konstanz.de

Winter Term 2018/2019
We offer

- ... integration in state of the art research projects
- ... if applicable, collaboration with our industrial partners
- ... interesting real-life applications and problems as well as more foundational work
Scope of Projects/Theses

♦ Project (Bachelor and Master)
  ‣ 1 Semester
  ‣ 9 ECTS (270h work)

♦ Thesis (Bachelor)
  ‣ 3-6 Months ("1 Semester")
  ‣ 12 (Thesis) + 3 (Colloquium) ECTS (450h work)

♦ Thesis (Master)
  ‣ 6 Months (1 Semester)
  ‣ 30 (Thesis + Colloquium) ECTS (900h work)
Formal Requirements, Deliverables

♦ Before you start your work
  ‣ Written proposal (≈ 1 page) containing (deadline: November 2, 2018)
    – the topic you want to choose
    – how well you fit the prerequisites
    – why you are suited for the task.
  ‣ Schedule for the project/thesis
    – What should be achieved at which point in time.

♦ During your preparation of the thesis or the project work
  ‣ Regular consultation with your supervisor.

♦ Deliverables
  ‣ Any models/code/data/binaries you created for the project.
  ‣ Report of ≥ 20 pages (Projects, Bachelor-Thesis) or ≥ 40 pages (Master-Thesis)
    – Discussing state of the art
    – Stating the problem
    – Presenting your approach and results
    – Critical discussion
Our Team

♦ Staff

♦ Prof. Dr. Stefan Leue
  – Email: Stefan.Leue@uni.kn.de, Room: PZ 902

♦ Dr. Georgiana Caltais
  – Email: Georgiana.Caltais@uni.kn.de, Room: PZ 913

♦ Martin Kölbl
  – Email: Martin.Koelbl@uni.kn.de, Room: PZ 912

♦ Dominik Winterer
  – Email: Dominik.Winterer@uni.kn.de, Room: PZ 912
Safety-Critical Cyber-Physical Systems

♦ Our Research Focus
  ▶ design methods in support of the development of dependable systems
Verification of Safety-Critical Systems

♦ A Railroad Crossing
  ▸ safety goal
    „It shall always be the case that there is never a car and a train in crossing at the same time.“
  ▸ can hazards happen? What is the cause?

Credits: Hanser / Südkurier
Possible Approach – Model Checking

- Model of The Real World – Transition Systems:

  - Train
    - \( \text{outside} \) \( \xrightarrow{T_a} \) \( \text{approaching} \)
    - \( \text{inside} \) \( \xleftrightarrow{T_l} \)
    - \( \text{inside} \) \( \xleftrightarrow{T_c} \)

  - Gate
    - \( \text{open} \) \( \xleftrightarrow{G_c} \) \( \text{closed} \)
    - \( \text{Go} \)

  - Car
    - \( \text{outside} \) \( \xrightarrow{C_a} \) \( \text{approaching} \)
    - \( \text{inside} \) \( \xleftrightarrow{C_l} \)
    - \( \text{closed} \) \( \xleftrightarrow{G_c} \) \( \text{open} \)
    - \( \text{Go} \)
Possible Approach – Model Checking

- Model Checking

\[ M \models S \]

**Model of the software** (transition system, Kripke structure)

**Model checking algorithm**

**Requirement specification** (assertions, temporal logic)

\[ \varphi = \Box \neg (T_c \land C_c) \]

"car and train never in crossing at the same time"
Model Checking

- **Explicit State Model Checking**
  - most common: automatic search of all reachable system states to find property violations using **Depth First Search**
  - the path into a property violating state is called an **error path** or **counterexample**
Causality Checking

“car and train never in crossing at the same time”

Model checking – (finite) counterexamples:

- \([Ta, Gf, Tc, Ca, Cc]\)
- \([Ca, Ta, Gf, Tc, Cc]\)
- \([Ta, Gf, Ca, Cc, Tc]\)
- \([Ta, Ca, Gf, Cc, Tc]\)
- \([Ca, Ta, Gf, Cc, Tc]\)
- \([Ta, Ca, Cc, Gf, Tc]\)
- \([Ca, Ta, Cc, Gf, Tc]\)
- \([Ta, Ca, Cc, Gc, Tc]\)
- \([Ca, Ta, Cc, Gc, Tc]\)
- \([Ca, Cc, Ta, Gc, Tc]\)
- \([Ca, Cc, Ta, Gc, Tc]\)
- \([Ca, Cc, Ta, Gc, Tc]\)
- \([Ca, Cc, Ta, Gc, Tc]\)
- \([Ca, Cc, Ta, Gc, Tc]\)
- \([Ca, Cc, Ta, Gc, Tc]\)
- \([Ca, Cc, Ta, Gc, Tc]\)

- but, what is a cause?
  - algorithmic solution

- representation well-known to engineers…
“Engineer Friendly”

- **Model of The Real World**
  - SysML / UML specification

![Diagram of a railroad crossing with a train, car, and gate.
The QuantUM Approach

(Failure Mode) Specification

(Safety-) Requirements

QuantUM Model

automatic

Quantitative Analysis of UML Models

Fault Trees
“Real World” Applications

- Experiments for safety properties include:
  - Airbag (with ZF/TRW Radolfzell)
  - Automotive Body Controller (AECU, with a German OEM)
  - Airspace Surveillance Radar (ASR), 3 variants (with Airbus Defense and Space)
Projects and Theses
Problem Statement

- Causality Checking in QuantUM relies on trace computation – bottleneck: memory
- LTSmin / PINS is interface to symbolic model checking engine – hope: more memory efficient

Project Tasks

- integrate QuantUM/Causality Checking into LTSmin / PINS
- direct encoding of causality checking in symbolic BDD data structures [MP/MT]

Prerequisites

- programming, discrete structures
- for [MP/MT]: (symbolic) model checking an advantage
Checking Real-Time Properties for SysML [BP/BT]

- **SysML and Real Time**
  - overview of real-time property expressiveness in SysML / MARTE
  - analysis of real-time primitives in SysML / Papyrus
  - definition of a semantics for selected primitives

- **Analysis**
  - translation into Timed Automata / UPPAAL input language using ATL model transformation technology
  - reverse interpretation of the analysis results
Causality Checking for Programs [MP/MT]

Programs

- the assignment of certain values to variables can cause a program to crash
- which variable assignments and which values are causal for a program failure?

Tasks

- selection of a program analysis framework (other than testing)
  - for instance, symbolic execution, static analysis
- development of a causality notion for program executions
- prototype implementation and case study

Prerequisites

- good understanding of logic, program semantics, foundations of computing
Causality Checking for Deadlock Properties [BP/BT] [MP/MT]

- Deadlocks
  - circular wait, no more progress
- Causality Checking for Deadlocks
  - deadlock is reachability [BP/BT]
    - adopt causality checking to deadlock
    - implement in SpinJa
  - consider different deadlocks
    - extend the algorithmics and implementation of causality checking to multiple deadlocked states
  - implementation and case studies
- Prerequisites
  - preferably, one of the model checking courses
  - good programming skills
Run-Time Causality Checking [MP/MT]

♦ Runtime Verification
  ‣ observe and assess running system
  ‣ often: monitoring

♦ Run-Time Causality Checking
  ‣ observe system behavior
  ‣ detect occurrence of events at run-time as causal for undesired system behavior
  ‣ learn for the future

♦ Tasks
  ‣ study various run-time verification approaches, in particular run-time model checking
  ‣ analyze, what causality can mean in this context
  ‣ adapt causality checking

♦ Prequeuisites
  ‣ one model checking course
  ‣ advantageous: machine learning, data mining
Model Transformation

- UML-RT models edited in Papyrus Real Time (EMF)
- understand semantics of state machine diagrams, inter-object communication and meta-models
- define transformation rules in ATL model transformation framework
- implementation and case studies

Benefits

- exposure to practically very relevant model based design language
- interest in Papyrus community

Prerequisites

- Software Engineering
- interest in semantics and model transformation
Model Transformation to LTSmin / PINS

- SysML models edited in Papyrus Real Time (EMF)
- understand semantics of state machine diagrams, inter-object communication and meta-models
- define transformation rules in ATL model transformation framework
- implementation and case studies

Benefits

- exposure to practically very relevant model based design language
- interest in Papyrus community

Prerequisites

- Software Engineering
- interest in semantics and model transformation
Explaining Faults with Machine Learning [MP/MT]

- Explaining Faults with Machine Learning
  - Model Checking: thousands of counterexamples with hundreds of events
  - Use Machine Learning (ML) to learn & explain counterexamples
    - Learn compact representation of counterexamples
    - Extract patterns
- Goals
  - Tool to explain a set of counterexamples of Model Checker (SPIN)
    - Implementation in Python
    - Decision Tree Learning (scikit learn)
  - Case study on selected Promela Models
- Requirements
  - Strong programming skills (e.g. C++, Java, Python)
  - Ability to work independently
  - Basic Machine Learning knowledge is a plus
Causality Checking Concurrent Code [MP / MT]

CHESS

- tool for finding and producing bugs in concurrent programs (C,C++,C#)

Task

- transfer the idea of Causality Checking to testing / code analysis
- apply it to the CHESS tool / environment
- implement prototype
- perform case studies
Develop a graphical representation of EOL (event order logic)

- EOL (event order logic) is a subset of LTL (linear temporal logic) including ordering constraints.
- Currently, EOL is represented by formulae.
  \[(T_a \land (C_a \land C_c)) \land < \neg C_l \land > T_c\]
- Graphical representations are better suited for non-specialists, e.g.:

Prerequisite:

- Knowledge of LTL, e.g., course: Model Checking of Software and Systems, is an advantage but not strictly required
Interested? Contact...

- ... either one of us any time!
  - Prof. Dr. Stefan Leue
    - Email: Stefan.Leue@uni.kn, Room: PZ 902
  - Dr. Georgiana Caltais
    - Email: Georgiana.Caltais@uni.kn, Room: PZ 913
  - Martin Kölbl
    - Email: Martin.Koelbl@uni.kn, Room: PZ 912
  - Dominik Winterer
    - Email: Dominik.Winterer@uni.kn, Room: PZ 912