Universität Bremen

Systeme hoher Sicherheit und Qualität

WS 2019/2020

# Lecture 12:

# Tools for Model Checking

Christoph Lüth, Dieter Hutter, Jan Peleska

# Organisatorisches

- Prüfungstermine
  - 06.03.2020, 12- 18 Uhr
  - 02.04.2020, ganztägig

- Scheinbedingungen:
  - Note aus der mündlichen Prüfung
  - Benotung der Übungsblätter: A = 1.3, B = 2.3, C = 3.3
  - Kann als Bonus (nicht Malus) mit 20% hinzugerechnet werden.

# Where are we?

- 01: Concepts of Quality
- 02: Legal Requirements: Norms and Standards
- 03: The Software Development Process
- 04: Hazard Analysis
- 05: High-Level Design with SysML
- 06: Formal Modelling with OCL
- 07: Testing
- 08: Static Program Analysis
- 09: Software Verification with Floyd-Hoare Logic
- 10: Verification Condition Generation
- 11: Foundations of Model Checking
- 12: Tools for Model Checking
- 13: Conclusions

# Introduction

- In the last lecture, we saw the **basics of model-checking**: how to model systems on an abstract level with **FSM** or **Kripke structures**, and how to specify their properties with **temporal logic** (LTL and CTL).

- This was motivated by the promise of "efficient tool support".

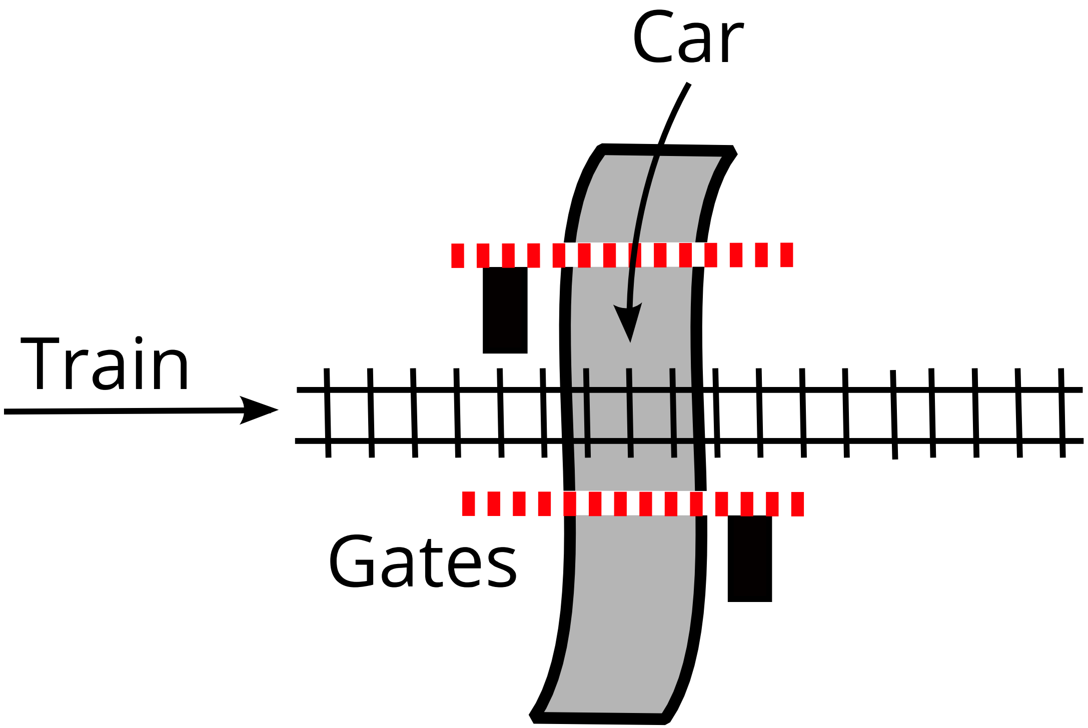
- So how does this tool support look like, and how does it work? We will hopefully answer these two questions in the following…

- Brief overview:
  - An **Example**: The Railway Crossing.
  - Modelchecking with **NuSMV** and **Spin**.
  - Algorithms for Model Checking.

# The Railway Crossing

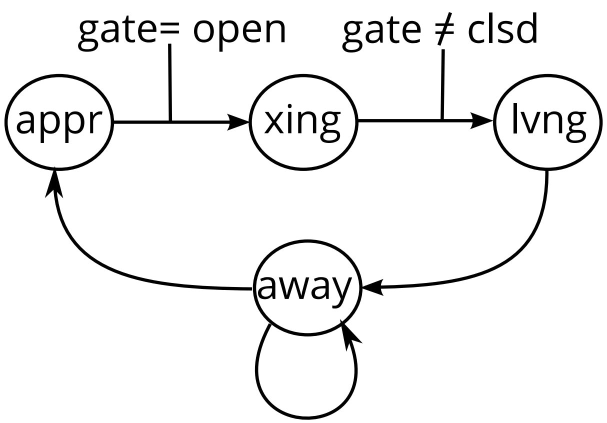
Quelle: Wikipedia

# First Abstraction

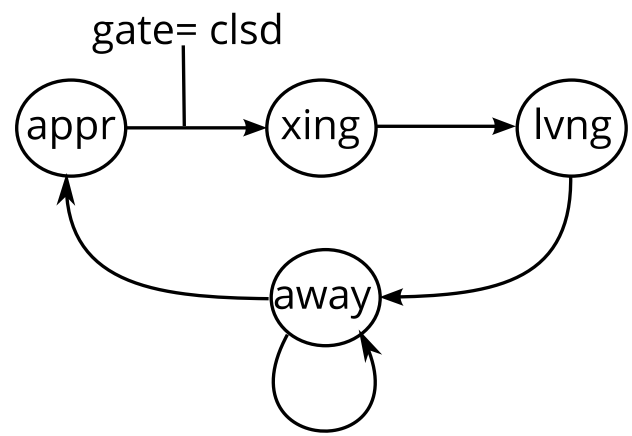


# The Model

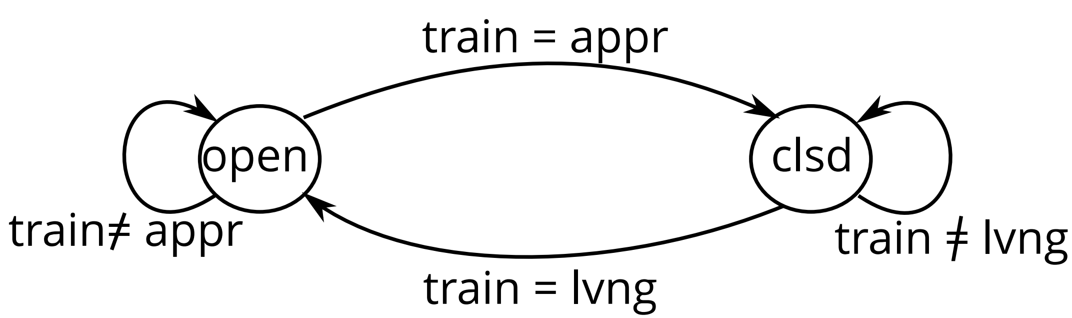
States of the car:




States of the train:




States of the gate:



# The Finite State Machine

- The states of the FSM is given by mapping variables $car, train, gate$ to the domains

$$\begin{aligned} \Sigma_{car} &= \{appr, xing, lvng, away\} \\ \Sigma_{train} &= \{appr, xing, lvng, away\} \\ \Sigma_{gate} &= \{open, clsd\} \end{aligned}$$

- Or alternatively, states are a 3-tuples

$$s \in \Sigma = \Sigma_{car} \times \Sigma_{train} \times \Sigma_{gate}$$

- The transition relation is given by

$$\begin{aligned} &\langle away, away, open\rangle \rightarrow \langle appr, away, open\rangle \\ &\langle appr, away, open\rangle \rightarrow \langle xing, away, open\rangle \\ &\langle appr, appr, clsd\rangle \rightarrow \langle appr, xing, clsd\rangle \\ &\langle appr, xing, clsd\rangle \rightarrow \langle appr, lvng, clsd\rangle \\ &\langle appr, lvng, clsd\rangle \rightarrow \langle appr, away, open\rangle \\ &\ldots \end{aligned}$$

# Properties of the Railway Crossing

- We want to express properties such as
  - Cars and trains may never cross at the same time.
  - The car can always leave the crossing.
  - Approaching trains may eventually cross.
  - It is possible for cars to cross the tracks.
- The first two are **safety properties**, the last two are **liveness properties**.
- To formulate these in temporal logic, we first need the **basic propositions** which talk about the variables of the state.

# Basic Propositions

- The basic propositions $Prop$ are given as equalities over the state variables:

  $(car = v) \in Prop$ mit $v \in \Sigma_{car}$,
  $(train = v) \in Prop$ mit $v \in \Sigma_{train}$,
  $(gate = v) \in Prop$ mit $v \in \Sigma_{gate}$

- The Kripke structure valuation $V$ maps each basic proposition to all states where this equality holds.

# The Properties

- Cars and trains never cross at the same time:
$$G\neg(\ car = xing \wedge train = xing)$$

- A car can always leave the crossing:
$$G\left(car = xing \rightarrow F\ (car = lvng)\right)$$

- Approaching trains may eventually cross:
$$G\left(train = appr \rightarrow F\ (train = xing)\right)$$

- There are cars which are crossing the tracks:
$$EF\ (car = xing)$$

    - Not expressible in LTL, $F\ (car = xing)$ means something stronger („there is always a car which eventually crosses“)

# Model-Checking Tools: NuSMV2

- NuSMV is a reimplementation of SMV, the first model-checker to use BDDs. NuSMV2 also adds SAT-based model checking.
- Systems are modelled as synchronous FSMs (Mealy automata) *or asynchronous processes**.
- Properties can be formulated in LTL and CTL.
- Written in C, open source. Latest version 2.6.0 from Oct. 2015.
- Developed by Fondazione Bruno Kessler, Carnegie Mellon University, the University of Genoa and the University of Trento.

* This is apparently depreciated now.

# Model-Checking Tools: Spin

- Spin was originally developed by Gerard Holzmann at Bell Labs in the 80s.
- Systems modelled in Promela (Process Meta Language): asynchronous communication, non-deterministic automata.
- Spin translates the automata into a C program, which performs the actual model-checking.
- Supports LTL and CTL.
- Latest version 6.4.7 from August 2017.
- Spin won the ACM System Software Award in 2001.

# Conclusions

- Tools such as **NuSMV2** and **Spin** make model-checking feasible for moderately sized systems.

- This allows us to find errors in systems which are hard to find by testing alone.

- The key ingredient is **efficient state abstraction**.
  - But careful: **abstraction** must **preserve properties**.