Stepwise Development and Model Checking of a Distributed Interlocking System – using RAISE

Signe Geisler, Anne E. Haxthausen
DTU Compute, Technical University of Denmark

Outline

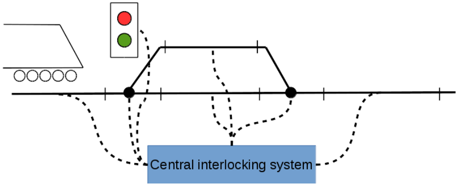
1. Background & Objectives

2. The Engineering Concept

3. Modelling and Verification by Model Checking

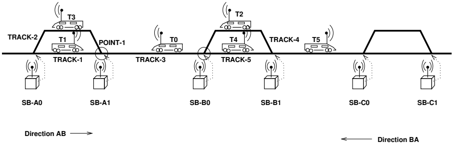
4. Conclusions and Future Work

Background: Interlocking Systems



- The **task** of an *interlocking system* is to ensure *safe* train movements in the railway network under its control.
- Interlocking systems are typically *centralised*.

Background: A Distributed Interlocking System



- **INSY GmbH Berlin** developed in the late nineties an engineering concept and a prototype of *distributed* railway control system (**RELIS 2000**) for local railway networks.
- The system was *generic* and could be configured with data depending on the railway network under control.
- In 1997-98, Jan Peleska and Anne Haxthausen *formally verified* the system with respect to safety properties. The **RAISE** *theorem prover* was used for this.

  Anne E. Haxthausen and Jan Peleska: *Formal Development and Verification of a Distributed Railway Control System*. In: *IEEE Transaction on Software Engineering*, 26(8):687--701, 2000.

- In recent years, other suggestions for distributed interlocking systems have been given. For a survey, see *Fantechi and Haxthausen, FMICS 2018.*

Objectives

- *Theorem proving* has the advantage that it verifies *once-and-for-all* that all admitable system instances are safe, but has the disadvantage that it is very *time consuming*. In contrast to that *model checking* has to be *repeated for each system instance*, but has the advantage of being *fully automated*.
- Our **goal** has been to formally verify the RELIS 2000 system by *model checking*.
- To tackle this challenge, we investigated how *stepwise modelling and model checking* of distributed systems could be done in a RAISE setting.

Outline

1. Background & Objectives

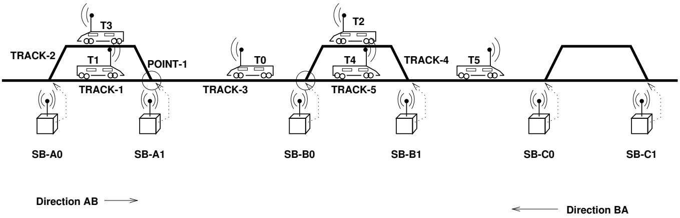
2. The Engineering Concept

3. Modelling and Verification by Model Checking

4. Conclusions and Future Work

The RELIS 2000 System Architecture

Example railway network: tracks, points, sensors, no signals



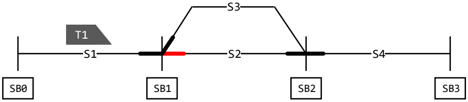
Control components:

- *a train control computer* in each train having the task to: give movement authorities to the train and issue reservation and switch commands to switchboxes
- *a switchbox (computer)* for each point having the task to: control the point, monitor the status of the associated sensor, record track segment reservations for trains
- *communication* via mobile telephone networks

must collaborate to ensure *safety* (no collisions, no derailments).

Main Idea of the RELIS 2000 Control Protocol

T1	
ROUTE SEGs	S1,S2,S4
ROUTE SBs	SB1,SB2,SB3
POSITION	S1
RESERVATIONS	[SB1:{S1,S2},SB2:{S2}]
LOCKS	[SB1]



SB1	
CONNECTED	S1-~~S3~~S2
LOCKED BY	T1
SENSOR	passive
RES S1	T1
RES S2	T1
RES S3	-

SB2	
CONNECTED	S2-S4
LOCKED BY	-
SENSOR	passive
RES S2	T1
RES S3	-
RES S4	-

- *Safety conditions* for train T1 to pass SB1 and enter the next segment (S2):
  - T1 must have a reservation for S2 at SB1.
  - T1 must have a reservation for S2 at SB2.
  - The point at SB1 must be switched in correction position and locked for T1.
- In order to achieve these conditions, the train sends requests to SB1 and SB2.
- When a train has passed a point/switchbox, its reservations and locks at that switchbox are *released*.

Outline

1. Background & Objectives

2. The Engineering Concept

3. Modelling and Verification by Model Checking

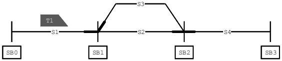
4. Conclusions and Future Work

Overview

- Development of a *generic state transition system (STS) model* in three steps:
    1. abstract model of the system behaviour (no explicit communication)
    2. the model is extended with communication
    3. the model is refined to a just-in-time allocation principle
       (it could be refined to other principles as well)
- Specification of *generic properties*.
- For each model: *model instances* are *model checked* against the properties.

- Models are expressed in RSL$\star$, in a *guarded command* style.
- Properties are specified as LTL-formulas.
- Verification is done using the SAL toolset.

Generic Model 1: abstract STS model



T1	
ROUTE SEGs	S1,S2,S4
ROUTE SBs	SB1,SB2,SB3
DIRECTION	->
POSITION	S1
RESERVATIONS	[SB1:{S1}]
LOCKS	-

SB0	
CONNECTED	S1
LOCKED BY	-
SENSOR	passive
RES S1	-

SB1	
CONNECTED	S1-S3
LOCKED BY	-
SENSOR	passive
RES S1	T1
RES S2	-
RES S3	-

SB2	
CONNECTED	S2-S4
LOCKED BY	-
SENSOR	passive
RES S2	-
RES S3	-
RES S4	-

SB3	
CONNECTED	S4
LOCKED BY	-
SENSOR	passive
RES S4	-

- *Model parameters*:

  type TrainID, SwitchboxID, SegmentID
  value network: Network $/*$ describes how segments are connected $*/$

- *Type and function declarations*, e.g.:

  type Network = ...

- *Declaration of variables* to keep the states of all objects, e.g.

  sbReservations[sb : SwitchboxID] : (SegmentID $\xrightarrow[m]{}$ TrainID)

- *State transition rules* for system events: *move*, *reserve*, *switch* and *lock*.

  E.g. for *reserve*, the generic rule takes the form:

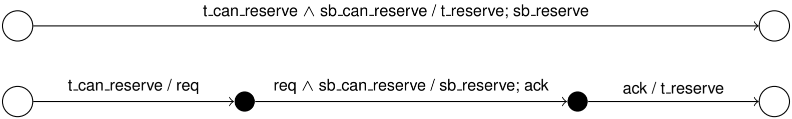
  $(\,[\!]\ \mathrm{sb} : \mathrm{SwitchboxID}, \mathrm{t} : \mathrm{TrainID}, \mathrm{seg} : \mathrm{SegmentID} \bullet$
  $\quad [\,\mathrm{reserve}\,]\ \mathrm{t\_can\_reserve} \wedge \mathrm{sb\_can\_reserve} \longrightarrow \mathrm{t\_reserve}; \mathrm{sb\_reserve}\ )$

  where e.g.

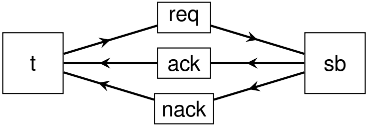
  $\mathrm{sb\_can\_reserve} \equiv \mathrm{seg} \in \mathbf{dom}(\mathrm{sbReservations[sb]}) \wedge \mathrm{sbReservations[sb](seg)} = \mathrm{t\_none}$
  $\mathrm{sb\_reserve} \equiv \ \mathrm{sbReservations}'\mathrm{[sb]} = \mathrm{sbReservations[sb]} \dagger [\,\mathrm{seg} \mapsto \mathrm{t}\,]$

Generic Model 2: added communication

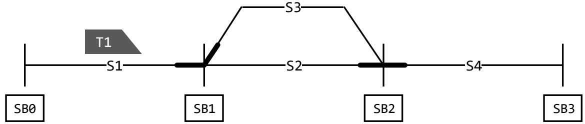
- Decomposition of events.



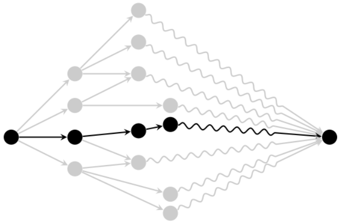
- Communication variables for each pair of TrainID, *t*, and SwitchboxID, *sb*.



Generic Model 3: refined to a more restricted protocol



- A *just-in-time order* of allocation is enforced by strengthening the guards for the train control computers – $t\_can\_reserve$ and $t\_can\_lock\_and\_switch$.
- The restricted state transition system should correspond to a *subset* of the transition system of $M2$.



Generic Properties

- **Invariants:**
  - *Safety conditions*, e.g. no train collisions

    $\forall$ t1, t2 : TrainID •
    $\quad$ G(t1 $\neq$ t2 $\wedge$ t1 $\neq$ t_none $\wedge$ t2 $\neq$ t_none $\Rightarrow$ no_collide(pos[t1], pos[t2]))

    where

    no_collide(pos[t1], pos[t2]) $\equiv$
    $\quad$ segments_of_pos(pos[t1]) $\cap$ segments_of_pos(pos[t2]) $=$ {}

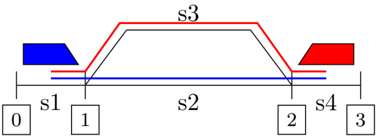
    and pos[t : TrainID] : Position is a state variable keeping track of train positions.
  - *Distributed data consistency*.
- **Progress:**
  - Absence of deadlocks.
  - *Actions are completed* (if a request is sent, then a reply comes back).
  - *Trains can reach their destination* is proved by contradiction: i.e. by disproving *not all trains arrive*

    G($\sim$ ($\forall$ t • TrainID • pos[t] $=$ dest(t)))

    where dest(t) is the destination for train t.

Verification: Model Checking Model Instances

- To perform *verification*, the generic models and properties are *instantiated with configuration data* (a concrete network and train routes) and translated to SAL and checked by the SAL model checker.
- Example instantiation:



Outline

1. Background & Objectives

2. The Engineering Concept

3. Modelling and Verification by Model Checking

4. Conclusions and Future Work

Conclusions and Future Work

Results:

- A method for automated verification of distributed railway interlocking systems using RAISE:
  - Generic state transition models are stepwise refined, instantiated and model checked.
  - This is a novelty for RAISE.
- Applications:
  - The method has successfully been applied to a real-world distributed railway interlocking system.
  - Experience: The stepwise approach was very useful.
  - The method can also be applied to other domains.

Future work:

- Apply techniques to reduce the state space, e.g.
  - compositional methods, see e.g. *Fantechi, Haxthausen, and Macedo: SEFM'17*
  - SAT/SMT based k-induction, see e.g. *Vu, Haxthausen, and Peleska: Formal modelling and verification of interlocking systems featuring sequential release. Sci. Comput. Program. 133, 2017.*
- Try other RAISE backend model checkers, e.g. RT-Tester and nuXMV, and compare with the SAL model checker.
- Try other modelling and verification frameworks, e.g. UPPAAL, and compare with RAISE.

Thank you for your attention!

Many thanks to Jan Peleska, from whom the case study originates, for helpful comments and for the continuing collaboration.