Int J Softw Tools Technol Transfer (2014) 16:713–726
DOI 10.1007/s10009-013-0295-9

FMRCS

Automated generation of formal safety conditions from railway interlocking tables

Anne E. Haxthausen

Published online: 22 December 2013

Abstract This paper describes a tool for extracting formal safety conditions from interlocking tables for railway interlocking systems. The tool has been applied to generate safety conditions for the interlocking system at Stenstrup station in Denmark, and the SAL model checker tool has been used to check that these conditions were satisfied by a model of the relay circuits implementing the interlocking system at Stenstrup station.

Keywords Railways · Interlocking systems · Formal methods · Safety · Verification · Model checking · Interlocking tables · Signal control tables

1 Introduction

1.1 Background and problem

With more than 170 million passengers going by train on a yearly basis in Denmark, the safety of the railway traffic is a top priority for Railnet Denmark. As in other countries, railway interlocking systems are used to prevent trains from colliding and derailing. Many interlocking systems in Denmark are still relay based, i.e. implemented by complex electrical circuits containing relays. These systems are documented by track layout diagrams, relay circuit diagrams and interlocking tables (also sometimes called signal control tables or train route tables). The interlocking tables serve as design specifications for relay circuit implementations,[1] and the latter are verified to satisfy the design requirements by manual inspection of the circuit diagrams and the tables. Such a manual verification is very challenging, time consuming, and error prone. For these reasons, Railnet Denmark asked us to research a better verification method.

1.2 Solution

Our solution has been to develop a set of tools [12] supporting automated formal verification of relay interlocking systems. We decided that the verification method should be *formal* as formal verification has been recognised as one of the best ways of avoiding errors and is for that reason strongly recommended by the CENELEC standard EN50128 [9] for software for railway control and protection systems. Furthermore, we decided that the method should be *automated* as much as possible to reduce the time consumption. We chose the model checking approach [6] to formal verification as this allows for full automation. However, the model checking approach requires as input a formal model of the system behaviour and a formal specification of the required properties, and it is not a trivial task to create this input. To overcome this problem, we decided also to create tools for generating verifiable formal models and for generating formal requirements, respectively.

The tools are centred around a domain-specific language (DSL) for digitised representations of track layout diagrams, interlocking tables, and circuit diagrams used for documenting a relay interlocking system. We chose to centre the tools around a domain-specific language rather than a general purpose modelling language, as it is easier for railway engineers

A. E. Haxthausen (✉)
DTU Compute, Technical University of Denmark,
2800 Lyngby, Denmark
e-mail: aeha@dtu.dk

[1] They are also used for some computer based interlocking systems.

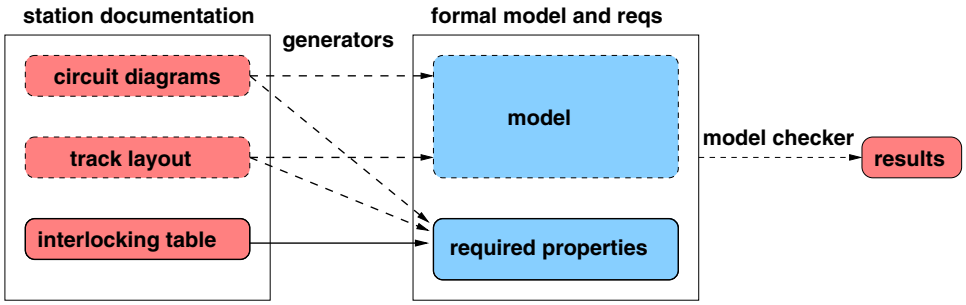

Fig. 1 Overview of generator tools. The tool described in this paper is shown by a *solid arrow*

to use a language that facilitates concepts already known and used in the railway domain.

The tools comprise:

- *data validators* for checking that the documentation (in DSL) follows certain general wellformedness rules,
- *generators* that from a DSL description produce input to the SAL model checker [20]:
  - a formal, behavioural model (state transition system) of the described interlocking system and its environment and
  - required properties expressed as formulae in the temporal logic LTL [16].

Figure 1 shows an overview of the generator tools. As it can be seen, the model is generated from the circuit diagrams and the track layout diagram. Additional generators can be used to derive required properties from the circuit diagrams, the track layout diagram, and the interlocking table, respectively. The generated properties include the following:

1. *High-level safety conditions* expressing that there are no derailments and no collisions. These are generated from the track layout.
2. *Low-level safety conditions* (also called *signalling conditions*) expressing that general signalling rules of Railnet Denmark are obeyed. These are generated from the interlocking table.
3. *Circuit confidence conditions* expressing that the circuits are well-designed in a general sense (e.g. not giving rise to race conditions). These are generated from the circuit diagrams.
4. *Model consistency conditions* expressing consistency between related state variables of the model (used for model validation). These are generated from the track layout.

Prototypes of the generator tools have been developed using the RAISE formal method [21,22] due to previous good experience in using that method. Details on these tools and their development can be found in [1,4,14].

The whole collection of tools can be used to verify an interlocking system in the following number of steps:

- Write a DSL description of the interlocking system.
- Validate the description using the data validators.
- Apply the generators to generate input to a model checker.
- Apply the model checker to that input to investigate whether the model satisfies the required properties.

1.3 Focus of this paper and relation to past papers

In a series of papers and technical reports our approach and tools have been described:

- The article [12] *Towards a Framework for Modelling and Verification of Relay Interlocking Systems* gives an overview of our approach and tools without going into technical details.
- The article [13] *Modelling and Verification of Relay Interlocking Systems* has main focus on describing how a behavioural model of a relay interlocking system can be extracted from the circuit diagrams describing the system.
- The article [14] *Formal Development of a Tool for Automated Modelling and Verification of Relay Interlocking Systems* explains how the model generator tool (making the extraction described in [13]) was formally developed using the RAISE formal method.
- The development of a domain model for circuit diagrams, the interlocking system model generator and the circuit confidence conditions generator was done in a sub-project described very detailed in the technical project report [4]. This report also gives a first suggestion for how the environment can be modelled and which other conditions to consider.
- Another sub-project, described in the technical report [1], continued the first sub-project by developing a domain model for railway networks and interlocking tables, and generators for extracting behavioural environment models, and three property generators for extracting high-level safety conditions, low-level safety conditions, and model consistency conditions, respectively.

The current paper describes how one of the *property generators* takes interlocking tables as input and generates low-level safety conditions expressing that the signalling rules

of Railnet Denmark are obeyed. This generator utilises the fact that interlocking tables serve as design specifications and contain data that can be used to instantiate generic signalling rules to concrete instances that can serve as safety requirements. More details on this tool and its development can be found in the technical report [1], except for a description of the conditions of Principle 7 in Sect. 6.5 as these were not yet included in the tool when the report was written.

1.4 Related work

The railway domain has been identified as a grand challenge for computer science, and the modelling, development and verification of interlocking systems has been investigated by many researchers. Different types of interlocking systems (for instance relay based versus computer based, functional versus geographical, etc.) have been modelled using different modelling formalisms and verified using different verification techniques (e.g. theorem proving and model checking).

An overview of results and trends in 2003 can be found in [3], and more recent results can be found in proceedings like [19] and book chapters like [10,24].

Several other research groups [2,5,11,15,17,18,25] have also investigated interlocking systems having interlocking tables as design specifications. One of their goals is to verify interlocking tables. Their approach for verification is to translate the tables into execution/design models for interlocking systems (typically by instantiating generic models with data from the tables) and verify by model checking that these models satisfy high-level safety requirements, such as no collisions and no derailments.

Hence, our goal of model checking is different from that of the above-mentioned research groups: their goal is to verify the interlocking tables, while our goal is to verify circuit diagrams. Consequently, a main difference between their and our verification approach is that their interlocking models are derived from the interlocking tables (i.e. from the design specification) while our models are derived from the relay circuit diagrams for the implementation. Instead of using interlocking tables for generating interlocking models, we use them for generating requirements (LTL formulas) in terms of signalling. Like the others, we also check for no collisions and no derailments.

Eriksson [8] has also formally verified relay based interlocking systems by deriving a model from the relay circuits, but he used theorem proving and not model checking for the verification.

1.5 Paper overview

First, in Sect. 2, an informal introduction to the domain of the considered interlocking systems is given. Then, in Sect. 3 the notions of Kripke structures (used as behavioural models) and LTL formulas (used to express safety conditions) are introduced for the convenience of readers who are not familiar with these notions. In Sect. 4, the state space of models and conditions is introduced; in Sect. 5, it is shortly explained how the models are created; and in Sect. 6, it is explained how the considered conditions generator extracts safety conditions from relay interlocking tables. Section 7 reports on how the tool has been applied in the verification of the interlocking system for Stenstrup station in Denmark, and finally, in Sect. 8 some conclusions are drawn.

2 Train route based interlocking systems

In this paper, we consider a class of interlocking systems (DSB type 1954) used for many Danish stations. These systems are based on a concept of train routes and implemented by relay based electrical circuits. In this section, a short introduction to these systems and their documentation is given.

2.1 The physical domain of a station

The physical domain under control consists of the railway tracks, points and signals. The tracks are divided into sections, each having a track circuit for detecting whether or not it is occupied by a train. The points can be switched between two positions: plus (i.e. straight) and minus (i.e. branching), and the signals can give proceed and stop indications by lights in coloured lamps.

Figure 2 shows a (simplified) track layout diagram for a typical station (Stenstrup station in Denmark). The track layout diagram outlines the geographical arrangement of the tracks and track-side equipment such as track circuits, points, and signals. From the diagram, it can be seen that Stenstrup has six track circuits (named A12, 01, 02, 03, 04, and B12), two points (named 01 and 02), and six signals (named A, B, E, F, G, and H).

2.2 Train route based interlocking

The task of an interlocking system is to control points and signals such that train collisions and derailments are avoided. The interlocking systems we are considering use a *train route based* approach to achieve that. The basic ideas of this approach are:

- Trains should drive on predefined *routes* through the network.
- Each route is covered by an entry signal that indicates whether it is allowed for a train to enter the route or not. Trains are assumed to respect the signals.

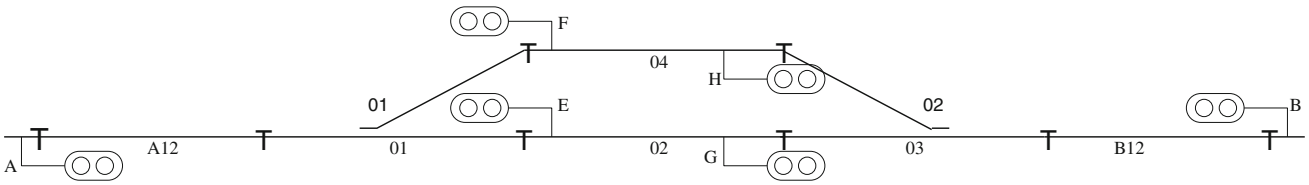

Fig. 2 Track layout diagram for Stenstrup station

– Two trains must never be allowed to drive on conflicting (e.g. overlapping) routes at the same time. *(To prevent collisions.)*
– Before a train is allowed to enter a route, the points in the route must be locked in positions making the route connected (i.e. it is physically possible to go from one end of the route to the other end without derailing), and the route must be empty (i.e. there are no trains on the route). *(To prevent derailments and collisions, respectively.)*
– The points of a route must not be switched while a train is driving on the route. *(To prevent derailments.)*

2.3 Relay circuit implementations and diagrams

The interlocking systems we are considering are implemented by electrical relay circuits. The circuits are made up of components, such as power supplies (each having a positive and a negative pole), relays, contacts, lamps inside signals, and operator buttons, connected by wires. A *relay* is an electrical switch operated by an electromagnet to connect or disconnect a number of contacts in a circuit. When current flows through the relay, the magnet is *drawn* and some of the associated contacts are connected (these contacts are said to be *upper contacts*) while others (the *lower contacts*) are disconnected. When no current flows through the relay, the magnet is *dropped* and the associated upper and lower contacts will be disconnected and connected, respectively. When contacts are connected/disconnected this may imply that sub-circuits containing these contacts become live/dead. This again may imply that relays of these sub-circuits are drawn or dropped, and so on.

An interlocking system can get input from the environment: buttons in the circuits can be pushed by an operator and some of the relays (called *the external relays*) change state (are drawn or dropped) when points change positions or trains enter or leave track sections. The external relays are hence controlled by the environment. All other relays are said to be *internal* and are controlled by the circuits.

2.3.1 Relay circuit diagrams

The electrical circuits implementing a relay interlocking system are documented by *relay circuit diagrams*.

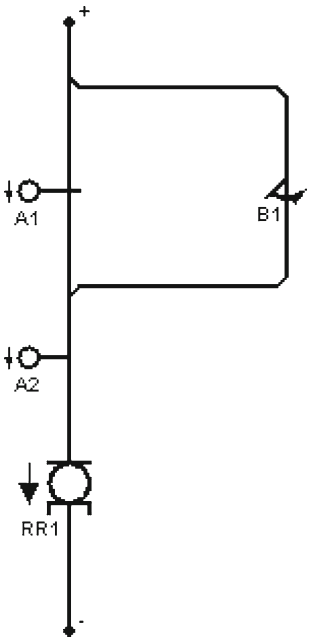

Fig. 3 A simple circuit diagram

For each internal relay one of the diagrams shows the sub-circuit that controls that relay. Figure 3 shows an example of a (simplified) relay circuit diagram. This diagram shows the sub-circuit controlling a relay named $RR1$. The circuit consists of a number of components connected by wires. The wires are depicted as black lines. At the top is the positive pole and at the bottom is the negative pole of the power supply. Relay $RR1$ is shown using this signature:

The downwards arrow informs that in the initial state this relay is dropped. (If it had been drawn, the arrow would have been upwards.) A number of contacts belonging to other relays occur in this circuit, e.g. a contact belonging to a relay named $A1$ is shown using this signature:

The downwards arrow informs that in the initial state relay $A1$ is dropped. The horizontal bar breaks the wire—this indicates that the contact is disconnected in the initial state (and

Table 1 Interlocking table (divided in two parts) for Stenstrup station

Route			Signals						Track sections						Points		Stop	Route release	
Id	From	To	A	B	E	F	G	H	A12	01	02	04	03	B12	01	02		Init	Final
2	A	G	g			r	r		↑	↑	↑		↑	↑	+	+	A:A12	↓01,↑02	↓02,↑01
3	A	H	g		r			r	↑	↑		↑	↑	↑	–	–	A:A12	↓01,↑04	↓04,↑01
5	B	E		g	r			r	↑	↑	↑		↑	↑	+	+	B:B12	↓03,↑02	↓02,↑03
6	B	F		g		r	r		↑	↑		↑	↑	↑	–	–	B:B12	↓03,↑04	↓04,↑03
7	E	A			g	r			↑	↑					+		E:01	↓01,↑A12	↓A12,↑01
8	F	A			r	g			↑	↑					–		F:01	↓01,↑A12	↓A12,↑01
9	G	B					g	r					↑	↑		+	G:03	↓03,↑B12	↓B12,↑03
10	H	B					r	g					↑	↑		–	H:03	↓03,↑B12	↓B12,↑03

Route conflicts

	2							
3	○	3						
5	○	○	5					
6	○	○	○	6				
7	○	○		○	7			
8	○	○	○		○	8		
9		○	○	○			9	
10	○		○	○			○	10

in all states where $A1$ is dropped). If the bar had not been breaking the wire, it would have indicated that the contact had been connected in the initial state, as it is the case for the contact of $A2$ (which is connected in all states where relay $A2$ is dropped).

Also a button $B1$ is shown in the diagram using this signature:

In the initial state the button is not pushed.

Current will flow through a relay if there is a path from the positive pole to the negative pole that goes through the relay and all contacts within this path are connected and all buttons are pushed. Therefore, from the diagram in Fig. 3 it can deduced that for current to flow through relay $RR1$, button $B1$ must be pushed and relay $A2$ must be dropped, or relay $A1$ must be drawn and relay $A2$ must be dropped. When that condition becomes fulfilled, relay $RR1$ will be drawn, and when it is not anymore fulfilled, it will be dropped.

2.4 Interlocking tables

For each station an *interlocking table* specifies the train routes of the station and for each of these routes

- the conditions for when the train route can be locked (reserved),
- the conditions for when the entry signal of the route is set to show a proceed aspect,
- the conditions for when the entry signal of the route is set back to show a stop aspect, and
- the conditions for releasing the train route again.

The interlocking table serves as a design specification of the interlocking system. Hence, it is used by the engineers who design the electrical circuits of the interlocking system, and it is used by the test team who tests that the implicit requirements of the table hold for the implemented interlocking system.

The aim of the generator tool we describe in this paper is to derive explicit, formal requirements from an interlocking table such that they can be formally verified to hold for a formal model of the behaviour of the implemented interlocking system. The formal model is generated from the circuit diagrams and track layout diagram by other generators of our tool set.

Table 1 shows a (simplified) interlocking table for Stenstrup station. The interlocking table has one row for each train route. For each route

- the *Route* sub-columns contain basic information about the train route such as its identification number,
- the *Signals* sub-columns state (1) which signals (the entry signal and any distant signal for this) should be set to a proceed aspect when the conditions for entering the route are met, and (2) which signals must be set to a stop aspect (to provide flank or front protection) before the entry sig-

nal can be set to proceed (g means green light (indicating proceed) and r means red light (indicating stop)),
- the *Points* sub-columns state required positions of points (+/- means straight/branching position) for the route to be connected (and possibly also flank protected),
- the *Track sections* columns state with an ↑ which track sections must be unoccupied for the route and its safety distance to be empty,
- the *Stop* column specifies that a certain signal (the entry signal of the route) should be switched to a stop aspect when a certain track section (the first section of the route) becomes occupied,
- the *Route release* columns define conditions for when the train route can be released (to be explained in Sect. 6), and
- the *Route conflicts* marks with the symbol ∘ which routes are conflicting.

2.4.1 Data validation

One of our data validator tools can be used to check that such an interlocking table contains suitable data with respect to a given track layout diagram, e.g. that the track sections of a route constitute a connected path in the track layout, that the signal in the Stop column is an entry signal for that path and the section in the Stop column is the first section of the route.

3 Background on models and assertions

According to our method, to verify an interlocking system, the generator tools should be applied to the documentation of the system (expressed in the domain-specific language) in order to derive (1) a behavioural model M of the system and its environment, and (2) formal safety conditions ϕ that the system must fulfil. The verification problem is then to check that each condition ϕ is satisfied by the model M. This is written $M \models \phi$.

In order to use the SAL model checker [20] to perform this check, we have chosen the conditions ϕ to be LTL formulas and the models M to be Kripke structures represented in the input language [7] of the SAL model checker.

This section gives a short introduction to LTL and Kripke structures, and it defines the satisfaction relation $\models$. More details on these topics can for instance be found in [6,16]. The section also shortly explains how the Kripke structures are represented in SAL.

3.1 LTL formulas

The generated conditions are LTL formulas built over a finite set of propositional state variables V that characterise the state of the system. The set of LTL formulas over V is the least set satisfying the following rules:

- If $v \in V$, then v is an LTL formula.
- If ϕ, ψ are LTL formulas, then $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \Rightarrow \psi$, $X(\phi)$, $G(\phi)$, $F(\phi)$, $U(\phi, \psi)$, and $W(\phi, \psi)$ are LTL formulas.

3.2 State transition system models

As models we use Kripke structures that describe how the state of a system can evolve over time.

A *Kripke structure* over a finite set of propositional[2] variables V is a four tuple (S, s_0, R, L), where

- S is a finite set of states,
- $s_0 \in S$ is an initial state,[3]
- $R \subseteq S \times S$ is a total,[4] binary relation on the state space describing possible state changes, and
- $L : S \to 2^V$, where 2^V denotes the power set of V, is a function that labels each state with variables that are true in that state.

3.3 Satisfaction relation between models and LTL formulas

An (execution) *path* in a Kripke structure (S, s_0, R, L) is an infinite sequence p of states $p(1), p(2), \ldots$ such that for each $i \geq 1$, $(p(i), p(i+1)) \in R$. In the following, we use the notation p^i for the suffix of p starting at $p(i)$, i.e. $p^i = p(i), p(i+1), \ldots$.

The *satisfaction relation* $\models$ between paths p in a Kripke structure (S, s_0, R, L) and LTL formulas ϕ over the same set of variables V is the least relation satisfying:

- $p \models v$, if $v \in L(p(1))$
- $p \models \neg\phi$, if it is not the case that $p \models \phi$
- $p \models \phi \wedge \psi$, if $p \models \phi$ and $p \models \psi$
- $p \models \phi \vee \psi$, if $p \models \phi$ or $p \models \psi$
- $p \models \phi \Rightarrow \psi$, if $p \models \psi$ whenever $p \models \phi$
- $p \models X(\phi)$, if $p^2 \models \phi$ (from the next time step ϕ must be true)
- $p \models G(\phi)$, if for all $i \geq 1$, $p^i \models \phi$ (ϕ must hold on the entire path)
- $p \models F(\phi)$, if there is some $i \geq 1$ such that $p^i \models \phi$ (ϕ eventually holds, i.e. holds somewhere on the path)

[2] More general Kripke models allow non-propositional variables provided these have finite domains. However, for the models presented in this article, propositional variables are sufficient.

[3] In the models we are considering, there is only one possible initial state. More general Kripke structures allow for a set of initial states.

[4] R is *total* means that for all $s \in S$ there is a state $s' \in S$ such that $(s, s') \in R$

- $p \models U(\phi, \psi)$, if there exists $i \geq 1$ such that $p^i \models \psi$ and for all $1 \leq k < i$, $p^k \models \phi$ (ϕ must remain true until ψ becomes true)
- $p \models W(\phi, \psi)$ iff $p \models U(\phi, \psi) \vee G(\phi)$ (ϕ must remain true forever or until ψ becomes true)

The satisfaction relation $\models$ between Kripke structures $M = (S, s_0, R, L)$ and LTL formulas ϕ over the same set of variables V is defined as follows: $M \models \phi$, iff for all paths p starting in the initial state s_0 of M (i.e. for which $p(1) = s_0$), $p \models \phi$ holds.

3.4 SAL representations of Kripke structures

This subsection shortly explains how Kripke models are represented in SAL.

A *generated* SAL specification consists of the following elements:

- a declaration of a finite set V of propositional (Boolean) state variables,
- an initialisation assigning initial values (true or false) to the variables in V,
- a set of transition rules of the form $cond \rightarrow update$ where
  - $cond$ (called the enabling condition) is a propositional formula over the variables in V using the connectives $\neg$, $\wedge$, $\vee$, and $\Rightarrow$, and
  - $update$ is a multiple assignment of Boolean values (true and false) to the primed versions v' of some of the variables v in V.

Intuitively, a transition rule states that for states in which the enabling condition is true, the system can change state to a new state that is obtained from the current state by performing the assignments in the $update$.

Such a specification represents the following Kripke structure (S, s_0, R, L) over V:

- $S = V \rightarrow \{true, false\}$ is the finite set of states, where a *state* is a truth valuation of the variables in V,
- $s_0 \in S$ is an initial state deduced from the initialisation,
- $R \subseteq S \times S$ is a binary relation[5] on the state space induced by the transition rules: $(s, s') \in R$ if there is a transition rule $cond \rightarrow update$ such that (i) the enabling condition $cond$ is true when evaluated in s, (ii) for each assignment $v' = e$ in $update$, $s'(v) = e$, and (iii) for all variables v in V not having an assignment in $update$, $s'(v) = s(v)$,
- $L : S \rightarrow 2^V$ is defined by: $L(s) = \{v | v \in V \wedge s(v) = true\}$ for $s \in S$.

[5] Note that we only consider SAL specifications for which the relation defined in this way is total (as required for Kripke structures). We use the SAL deadlock checker to check that.

4 State space

For a given interlocking system, our tools can be used to generate a model and safety conditions for the system. The model and the safety conditions are defined over a common set of variables describing the state space. This section first describes informally the common state space and then it defines the set V of variables capturing the possible states of the system. In Sects. 5 and 6, it is described how models and conditions, respectively, are generated by the generator tools.

4.1 State space

The relays and buttons in the circuits implementing an interlocking system change state over time as reaction to input from the environment as described in Sect. 2.3.

In the relay circuits of an interlocking system there are relays monitoring the states of the track side equipment:

- For each point P, there are two relays $plus\,P$ and $minus\,P$ that are drawn when and only when P is in the plus and the minus position, respectively.
- For each track section t, there is a relay t that is drawn when and only when the track section is unoccupied.
- For each signal S, there are two relays $Red\,S$ and $Green\,S$ that are drawn when and only when there is a red light and a green light in S, respectively.

The two first classes of relays are said to be *external* as they are controlled by the environment. All other relays, including the last class of relays, are controlled by the circuits and are said to be *internal*.

The remaining internal relays store the internal state of the interlocking system. For instance, there are relays keeping track of which routes are locked. For some interlocking systems, there is one locking relay for each route, however, for systems of DSB type 1954, some routes share a relay. We will use the notation $l(x)$ to denote the locking relay associated with a route x. A locking relay r is dropped, when and only when one of the train routes x associated with r is locked. Which of the routes is locked is determined by the point settings: route x is locked, when $l(x)$ is dropped and the points settings are as required for route x according to the interlocking table.[6]

As an example, for Stenstrup station there are four locking relays, ia (for routes 2 and 3), ib (for routes 5 and 6), ua (for routes 7 and 8), and ub (for routes 9 and 10).

[6] Note: two routes can only share a locking relay when at least one point is required to be set in different positions for the two routes.

The safety requirements that will be formalised in Sect. 6 can be expressed in terms of the states of the relays mentioned above.

4.2 State variables and initial states

The set V of variables, over which the models and the conditions are defined, includes:

- A Boolean variable r for each relay r in the circuit diagrams. When a relay variable r is *true*/*false*, it models that the associated relay is drawn/dropped.
- A Boolean variable b for each button in the circuit diagrams. When a button variable b is *true*/*false*, it models that the associated button is pushed/released.
- A Boolean auxiliary variable $idle$. When it is true, it models that the interlocking system is in an idle state waiting for new input.

The initial state of buttons is *false*, the initial state of track section relay variables is *true* (modelling that the track sections are initially unoccupied), the initial states of point relay variables $plus\,P$ and $minus\,P$ are *true* and *false*, respectively, and the initial states of internal relays are derived from the circuit diagrams.

5 Behavioural models

Our framework provides tool components that from the circuit diagrams and the track layout diagram of an interlocking system can be used to create a behavioural model M of the interlocking system and its interface to the environment. In Sect. 4, we described the state variables of M and stated the initial values of these variables. In this section, we will shortly outline which transition rules are generated. The transition rules describe how the internal relays, the external relays, and the buttons can change state over time.

5.1 State transition rules for internal relays and buttons

The state transition rules for internal relays and buttons are generated from the circuit diagrams.

For each internal relay r in the circuit diagrams two rules are generated:

$$\neg \mathrm{r} \wedge \mathrm{isConducting}_r \rightarrow \mathrm{r}' = \mathbf{true}$$

for drawing r and

$$\mathrm{r} \wedge \neg \mathrm{isConducting}_r \rightarrow \mathrm{r}' = \mathbf{false}$$

for dropping r. Here $isConducting_r$ is a condition for current to flow through the relay. It expresses that there is a path from the positive pole to the negative pole that goes through the relay and all contacts within this path are connected and all buttons are pushed.

As an example, from the diagram in Fig. 3 in Sect. 2.3.1 it can deduced that for current to flow through relay $RR1$, button $B1$ must be pushed and relay $A2$ must be dropped, or relay $A1$ must be drawn and relay $A2$ must be dropped. Therefore, the following two rules for relay $RR1$ are generated:

$$\neg \mathrm{RR1} \wedge ((\mathrm{A1} \wedge \neg \mathrm{A2}) \vee (\mathrm{B1} \wedge \neg \mathrm{A2})) \rightarrow \mathrm{RR1}' = \mathbf{true}$$
$$\mathrm{RR1} \wedge \neg((\mathrm{A1} \wedge \neg \mathrm{A2}) \vee (\mathrm{B1} \wedge \neg \mathrm{A2})) \rightarrow \mathrm{RR1}' = \mathbf{false}$$

Similarly, for each button in the relay circuit diagrams a rule for pressing it is generated, and a rule for releasing buttons is also generated.

More details on these transition rules can be found in [4, 14].

5.2 State transition rules for external relays

The state transition rules for external relays are generated from the track layout diagram.

A point P can be switched between three positions: the plus position, the minus position, and the intermediate position between the plus position and the minus position. For each point P in the track layout diagram, four transition rules are generated describing how the relays $plus\,P$ and $minus\,P$ associated with P, change state when the point is switched between its three possible positions.

For each track section t in the track layout diagram, transition rules for drawing and dropping the corresponding track relay are generated. The rules reflect the possible train movements and therefore depend on the track layout.

More details on these transition rules can be found in [1].

6 Safety requirements

This section first describes which formal requirements the generator tool derives from an interlocking table (that has been checked by the data validator tool) and then it informs how the tool was formally specified.

The formal requirements are formulas in LTL, expressed as conditions on the relay variables keeping the state of points, track sections, signals, and route lockings. They express safety conditions at the design level (i.e. concrete instances of the general signalling principles) that an interlocking system must satisfy.

In each of the Subsects. 6.1–6.6 below, first a general signalling principle is stated informally, then it is explained how formal, concrete instances of this can be generated by instantiating a formal condition pattern with data from a given inter-

locking table, and finally an example of this is given for the interlocking table for Stenstrup in Table 1.

In the formal condition patterns the following notation will be used for a route x:

- L_x: the locking relay $l(x)$ of x.
- $RouteLocked_x$: the condition $\neg L_x \wedge PointsSet_x$ expressing that route x is locked.
- $PointsSet_x$: a condition expressing that the points of x are set as required according to the "Points" fields for x in the interlocking table.
- $TracksFree_x$: a condition expressing that the track sections of x are unoccupied as required according to the "Track sections" fields for x in the interlocking table.
- $SignalsSet_x$: a condition expressing that the covering signals of x are set to a stop aspect as required according to the "Signals" fields for x in the interlocking table.
- $StopSection_x$: the track section (relay) specified in the "Stop field" for x in the interlocking table.
- $Init_x$: a condition expressing that the second last track section of x is occupied and the last track section of x is unoccupied as specified in the "Init" field for x in the "Route release" columns in the interlocking table.
- End_x: a condition expressing that the second last track section of x is unoccupied and the last track section of x is occupied as specified in the "Final" field for x in the "Route release" columns in the interlocking table.

6.1 No locking of conflicting routes

Principle 1 *When a train route x is locked, none of its conflicting routes y must be locked.*

For each route x, this is expressed by a condition of the following form:

$$G(RouteLocked_x \Rightarrow \bigwedge_{y \in ConflictingRoutes(x)} \neg RouteLocked_y) \quad (1)$$

where $ConflictingRoutes(x)$ is the set of routes that are in conflict with x according to the interlocking table.

Example 1 Applying this principle to train route 2 for Stenstrup, the generated condition will be in the following form as the route is in conflict with train routes 3, 5, 6, 7, 8 and 10 according to the interlocking table for Stenstrup:

$$\begin{aligned} &G(RouteLocked_2 \Rightarrow \\ &\quad \neg\, RouteLocked_3 \wedge \\ &\quad \neg\, RouteLocked_5 \wedge \\ &\quad \neg\, RouteLocked_6 \wedge \\ &\quad \neg\, RouteLocked_7 \wedge \\ &\quad \neg\, RouteLocked_8 \wedge \\ &\quad \neg\, RouteLocked_{10}) \end{aligned}$$

Expanding each of the expressions $RouteLocked_y$ using the data in the interlocking table, this gives

$$\begin{aligned} &G(\neg \text{ ia} \wedge \text{plus01} \wedge \text{plus02} \Rightarrow \\ &\quad \neg\, (\neg \text{ ia} \wedge \text{minus01} \wedge \text{minus02}) \wedge \\ &\quad \neg\, (\neg \text{ ib} \wedge \text{ plus01} \wedge \text{plus02}) \wedge \\ &\quad \neg\, (\neg \text{ ib} \wedge \text{ minus01} \wedge \text{minus02}) \wedge \\ &\quad \neg\, (\neg \text{ ua} \wedge \text{ plus01}) \wedge \\ &\quad \neg\, (\neg \text{ ua} \wedge \text{ minus01}) \wedge \\ &\quad \neg\, (\neg \text{ ub} \wedge \text{minus02}) \\ &) \end{aligned}$$

6.2 Locking and points positions

Principle 2 *When a locking relay r is dropped, one of the routes x, which is controlled by r, must have the points of that route set as required for route x according to the interlocking table. (This implies that a route can't be locked before its points are set.)*

For each locking relay r, this is expressed by a condition of the following form:

$$G(\neg r \Rightarrow \bigvee_{x \in Routes(r)} PointsSet_x) \quad (2)$$

where $Routes(r)$ is the set of routes x controlled by r, i.e. for which $l(x) = r$.

Example 2 Applying this principle to locking relay ia for Stenstrup, the following condition is generated as routes 2 and 3 are the routes controlled by ia:

$$G(\neg \text{ ia} \Rightarrow (\text{plus01} \wedge \text{plus02}) \vee (\text{minus01} \wedge \text{minus02}))$$

The condition expresses that when ia is dropped, points 01 and 02 are either both set in the plus position or both set in the minus position as required by the interlocking table for routes 2 and 3, respectively.

6.3 Signal aspects

Only certain combinations of lights are allowed aspects of the signals.

Principle 3 *A signal must never display a red light and green light at the same time.*

For each signal S, this is expressed by a condition of the following form:

$$G(idle \Rightarrow \neg(RedS \wedge GreenS)) \quad (3)$$

Example 3 Applying this principle to signal A for Stenstrup, the following condition is generated:

$$G(\text{idle} \Rightarrow \neg\, (\text{RedA} \wedge \text{GreenA}))$$

Principle 4 *When the green light is turned off in a signal S, the red light must be turned on.*

For each signal S, this is expressed by a condition of the following form:

$$G(idle \wedge \neg\, GreenS \Rightarrow RedS) \tag{4}$$

Example 4 Applying this principle to signal A for Stenstrup, the following condition is generated:

$$\text{G(idle} \wedge \neg\, \text{GreenA} \Rightarrow \text{RedA)}$$

6.4 Proceed signal

Principle 5 *When a signal S shows a proceed aspect, one of the routes x, starting from S, must be ready for use, i.e. (1) the route x must be locked, (2) all the track sections of the route must be unoccupied as stated in the interlocking table, and (3) all covering signals of the route must show a stop aspect as stated in the interlocking table.*

For each signal S, this is expressed by a condition of the following form:

$$G(idle \wedge GreenS \Rightarrow \bigvee_{x \in Routes(S)} (RouteLocked_x \wedge TracksFree_x \wedge SignalsSet_x)) \tag{5}$$

where *Routes(S)* is the set of routes starting from signal S.

From the condition, it can be derived that the green light must be turned off when the right-hand side becomes false. As it takes time for the system to turn the green light off, *idle* has been included on the left-hand side of the implication.

Example 5 Applying this principle to signal A, a condition of the following form is generated as train routes 2 and 3 start from signal A:

$$\begin{aligned}&\text{G(idle} \wedge \text{GreenA} \Rightarrow\\ &\quad (RouteLocked_2 \wedge TracksFree_2 \wedge SignalsSet_2) \vee\\ &\quad (RouteLocked_3 \wedge TracksFree_3 \wedge SignalsSet_3))\end{aligned}$$

Expanding each of the sub-formulae using the data in the interlocking table, this gives:

$$\begin{aligned}&\text{G(idle} \wedge \text{GreenA} \Rightarrow\\ &\quad ((\neg\text{ia} \wedge \text{plus01} \wedge \text{plus02}) \wedge\\ &\qquad (\text{A12} \wedge 01 \wedge 02 \wedge 03 \wedge \text{B12}) \wedge\\ &\qquad (\text{RedF} \wedge \text{RedG})\\ &\quad)\\ &\quad \vee\\ &\quad ((\neg\text{ia} \wedge \text{minus01} \wedge \text{minus02}) \wedge\\ &\qquad (\text{A12} \wedge 01 \wedge 04 \wedge 03 \wedge \text{B12}) \wedge\\ &\qquad (\text{RedE} \wedge \text{RedH})\\ &\quad)\\ &)\end{aligned}$$

6.5 Stop signal

Principle 6 *When track section, $StopSection_x$, specified in the "Stop" field for route x in the interlocking table, is occupied in an idle state, the signal S_x in the same field must show a stop aspect (i.e. the red light must be on).*

For each route x, this is expressed by a condition of the following form:

$$G(idle \wedge \neg\, StopSection_x \Rightarrow RedS_x) \tag{6}$$

In the condition it is necessary to include *idle* on the left-hand side of the implication in order to give the system time to change the setting of the signal as a reaction on the occupation of $StopSection_x$.

Note that condition (6) is a consequence of conditions (5) and (4) if it has been generated from a well-formed interlocking table as $\neg\, StopSection_x$ implies $\neg\, TracksFree_y$ for all routes y starting from S_x (due to the fact that for a well-formed interlocking table $StopSection_x$ is included in the track sections of any route y starting from S_x), and this implies $\neg GreenS_x$ [due to (5)], which implies $RedS_x$ [due to (4)].

Example 6 Applying this principle to route 2 for Stenstrup, the following condition is generated:

$$\text{G(idle} \wedge\ \neg\, \text{A12} \Rightarrow \text{RedA)}$$

It expresses that when track section A12 (the first section of the route) is occupied by a train (or another object), then the entry signal, A, must show a stop aspect (i.e. the red light must be on).

Principle 7 *When the setting of the entry signal S of a locked route x is changed to stop (i.e. the red light is turned on), it must keep this setting as long as the route is still locked.*

This principle prohibits that the signal is changed to proceed in the case where a train, that has entered the route, reverses direction and leaves the route before the route has been released.

For each signal S and each route $x \in Routes(S)$, this principle is expressed by a condition of the following form:

$$G(\neg\, L_x \wedge \neg\, RedS \wedge X(RedS) \Rightarrow X(W(RedS, L_x))) \tag{7}$$

where $L_x = l(x)$ is the locking relay of x, and W is the LTL weak until operator and X is the next state operator.

Example 7 Applying this principle to signal A and route 2 (or route 3) for Stenstrup, the following condition is generated

$$G(\neg ia \wedge \neg aRed \wedge X(aRed) \Rightarrow X(W(aRed,ia)))$$

It expresses that if the red light in signal A is off (i.e. ¬aRed is true) while route 2 (or route 3) is locked (i.e. ¬ia is true) and if the red light is turned on in the next state (i.e. X(aRed) is true), then the red light must be kept as long as the route is still locked, i.e. the red light can't be turned off before a release (where ia becomes true). Next subsection states the conditions for when a release can happen.

6.6 Train route release

Before a locked train route can be released, the two last sections $t1$ and $t2$ of the route must first have been in a state (called the *release start state*) where $t1$ is occupied and $t2$ is unoccupied, and then in a state (called the *release end state*) where $t1$ is unoccupied and $t2$ is occupied. This sequence of states is called the release sequence. This sequence will happen when a train passes the second last track section and ends on the last track section of the route. The "Route release" columns of the interlocking table state the release start and end states for each train route.

Principle 8 *When a train route has been locked, the route must not be released before the release sequence for the route has taken place.*

For each route x, this is expressed by a condition of the following form:

$$\begin{array}{l} G(L_x \wedge X(RouteLocked_x \wedge F(L_x)) \Rightarrow \\ \quad X(\\ \quad\quad U(\neg L_x, \\ \quad\quad\quad \neg L_x \wedge Init_x \wedge \\ \quad\quad\quad\quad X(U(\neg L_x, \neg L_x \wedge End_x)) \\ \quad\quad) \\ \quad) \\ ) \end{array}$$

where L_x is the locking relay of x, $Init_x$ is a condition expressing the release start state for x, End_x is a condition expressing the release end state for x, U is the LTL until operator, X is the next state operator and F is the eventually operator.

If condition L_x is true, it implies that $RouteLocked_x$ is false, as $RouteLocked_x = \neg\ L_x \wedge PointsSet_x$. Therefore, the left-hand side of the implication is true if route x is not locked in the current state, it becomes locked in next state and later it becomes released (unlocked) again. The right-hand side of the implication expresses the condition that from the next state (i.e. the state in which the route became locked according to the left-hand side condition), the route will not be released, i.e. L_x will not become true, until after the release sequence.

Example 8 Applying this principle to train route 2 for Stenstrup Station, the following condition is generated:

```
G(ia ∧ X((¬ia ∧ plus01 ∧ plus02) ∧ F(ia)) ⇒
  X(
    U(¬ia,
      ¬ia ∧ (¬01 ∧ 02) ∧
        X(U(¬ia, ¬ia ∧ (01 ∧ ¬02))
    )
  )
)
```

The left-hand side of the implication says that route 2 is not locked (i.e. ia is true) in the current state, it becomes locked (i.e. ¬ ia ∧ plus01 ∧ plus02) in the next state and later on it becomes released (unlocked) again (i.e. F(ia)). The right-hand side says that in the next state the route will stay locked at least until the release start state (where track section 01 is occupied and track section 02 is unoccupied) and in the state after this release start state, the route will continue being locked until the release end state where track section 01 is again unoccupied and track section 02 has been occupied.

6.7 Development of the tool

A prototype of the generator tool was developed by creating an executable specification in the RAISE specification language RSL [21]. In this section the overall structure of the specification is outlined.

The main components of the specification are:

- a specification of a data type $Table$ for representing interlocking tables,
- a specification of a data type $Assertion$ for representing conditions, i.e. LTL assertions (formulas), and
- a function gen for generating conditions from interlocking tables.

The generator function gen is specified to take a $Table$ value as input and to return an $Assertion{-}$**set** value, i.e. a set of $Assertion$ values (representations of LTL formulas). For each signalling principle i stated above, gen instantiates the formal condition pattern of principle i with data from the interlocking table obtaining a set set_i of assertions, and then it returns the union of all these sets.

More details about the tool and its development can be found in [1].

7 Experiments

We applied the developed signalling conditions generator to the interlocking table (shown in Table 1) for Stenstrup station in Denmark. In this way 52 conditions were generated. Table 2 shows for each signalling principle stated in Sect. 6, the number of conditions generated from the interlocking table.

We also applied other condition generators from our tool set to generate 152 other desired properties from the station documentation. Furthermore, we applied yet other generator tools from our tool set to generate a state transition system model for the behaviour of the implemented relay interlocking system (described by 18 circuit diagrams) and its environment (allowing operator input and an arbitrary number of trains driving according to the traffic rules). This state transition system model had 71 Boolean variables (i.e. 2^{71} states) and 141 transition rules.

We then used the SAL symbolic model checker [20] to verify that the generated model satisfied the 204 generated conditions. All conditions turned out to be valid. The SAL symbolic model checker is a BDD-based model checker for finite state systems.

Details on the elapsed execution time for checking the 52 signalling conditions (generated from the interlocking table) and the 152 other conditions can be found in Table 3. Each class of conditions were verified separately and without utilising intermediate results among queries. The elapsed time was measured with the LinuxMint12 `time` command on a Lenovo T420.

According to the signalling engineers it would take about a month to validate the circuit diagrams for Stenstrup station using their traditional manual inspection, and they would only check a small part of our 204 conditions. So it is really much faster to use our tools.

Table 2 For each signalling principle stated in Sect. 6, the number of signalling conditions generated from the interlocking table for Stenstrup station

Signalling principle	Number of generated conditions
1 (No locking of conflicting routes)	8
2 (Points set correctly while locked)	4
3 (Not red and green signal)	6
4 (Red signal when not green)	6
5 (Only green signal when allowed)	6
6 (Red signal when required)	8
7 (Keep red signal until release)	6
8 (Only allowed train route release)	8
Total	52

Table 3 For each of the four classes of conditions introduced in Sect. 1: (1) the number of conditions generated from the documentation of Stenstrup station and (2) elapsed execution time in seconds for checking these conditions

Kind of conditions	Number of generated conditions	Execution time
Signalling	52	409
No collisions, no derailments	12	20
Circuit confidence	102	5,139
Model consistency	38	20

We also tried to introduce some design flaws in the relay circuits to demonstrate that these can be found by using our tools; e.g. we introduced flaws such that a signal could reach a state where both the red light and the green light were turned on at the same time. In this case, the model checker detected that the signal aspect condition in formula (3) was broken for that signal.

Furthermore, we made some validation of the model. For instance, we model checked that the behavioural model allows trains to move through the network.

The safety conditions all take the form $G(a \Rightarrow b)$. If for such a condition, a is always false, the condition is trivially true. For a given condition and model, one can check that this is not the case by checking that $G(\neg a)$ is violated. If it turns out that a is always false, it can be an indication that there may an error in the model or that the condition a has been formulated wrongly. By making such checks, we actually found a flaw in an earlier formulation of the left-hand side condition of Principle 7.

8 Conclusions

8.1 Summary

This paper has described a tool component of a tool set that supports formal verification of relay interlocking systems.

Given the interlocking table of a relay interlocking system, the tool can automatically generate formal safety requirements for the implementation of the relay interlocking system. The requirements express that the signalling rules are followed. Other tool components of the tool set can be used to generate a formal model of an interlocking system and its environment. Having generated the requirements and the model, a model checker can be used to verify that these requirements always hold for the formal model.

To use such an automated, formal verification approach is a great improvement compared to manual inspections of interlocking tables, track layout diagrams and circuit diagrams: it is much faster and less error prone, it is much

more complete with respect to what is being checked, and the checking itself is exhaustive considering all possible scenarios. The approach has successfully been applied to the relay interlocking system for Stenstrup station.

Although the signalling conditions generator tool has been developed for a certain type of interlocking systems (the relay based DSB type 1954), it is expected that it can easily be adapted to other DSB types of interlocking systems that are based on similar interlocking tables, as the safety conditions for these systems are basically the same. With respect to the input of the generator (i.e. the interlocking tables), there may be small variations in the concrete syntax, but at least for the Danish systems, the content is basically the same. For other systems there may, apart from variations in the concrete syntax of the tables, also be minor variations in the signalling principles and therefore in the actual content of the tables. For instance, if the signalling principle for releasing a train route does not require the same release sequence of track sections as in the Danish systems, the release conditions in the tables will be different and the same holds for the formal release conditions that should be generated. In such a case, the generator tool should be changed accordingly.

8.2 Future work

The current tool set has been used for a proof-of-concept. To be used in industry, further development needs to be done, e.g. a better user interface should be provided.

We plan to apply the tools to larger stations to test to what extent the method is scalable without state space explosion problems. Experience by other research groups shows that the use of standard model checking techniques for verifying similar systems is only feasible for small railway stations. For instance, in [11] a systematic study of applicability bounds of the symbolic model checker NuSMV and the explicit model checker SPIN showed that these popular model checkers could only verify small railway stations. So it is likely that the application of our method to larger stations would also lead to state space explosion. If this happens, more advanced verification techniques must be investigated. Our safety conditions are independent of the model checking technique so the conditions generator tool described in this paper can be used in connection with more advanced model checking techniques. Several domain specific techniques to push the applicability bounds for model checking interlocking systems have been suggested. For instance, one could combine bounded model checking with inductive reasoning, as done in [15]. In [23], Winter pushes the applicability bounds of symbolic model checking (NuSMV) by optimising the ordering strategies for variables and transitions using domain knowledge about the track layout. In [18], it is suggested to reduce the state space using several abstraction techniques: reduction of the number of track sections and the number of trains, and compositional reasoning by decomposition of the railway network into several smaller networks.

We also plan to make a similar tool set for the new ERTMS-based signalling systems that are to be implemented in Denmark over the next decade.

Acknowledgments I would like to thank Kirsten Mark Hansen for providing the initial idea for this project and for many valuable discussions when she was employed at Railnet Denmark. Special thanks go to my former students Morten Aanæs and Hoang Phuong Thai who developed the first version of the generator tool described in this paper in their master thesis project supervised by me. The functionality of the tool was inspired by another master thesis made by my former students Marie Le Bliguet and Andreas A. Kjær. Finally, I would like to thank the reviewers for comments to a previous version of this paper. The work has been partially supported by the RobustRailS project granted by the Danish Council for Strategic Research.

References

1. Aanæs, M., Thai, H.P.: Modelling and verification of relay interlocking systems. Technical Report IMM-MSC-2012-14, DTU Informatics. Master thesis supervised by Anne Haxthausen, ah@imm.dtu.dk, Technical University of Denmark (2012)
2. Banci, M., Fantechi, A., Gnesi, S.: Some experiences on formal specification of railway interlocking systems using Statecharts (2005)
3. Bjørner, D.: New results and current trends in formal techniques for the development of software for transportation systems. In: Tanai, G., Schnieder, E. (eds.) Proceedings of the Symposium on Formal Methods for Railway Operation and Control Systems (FORMS'2003), Budapest/Hungary. L'Harmattan Hongrie, 15–16 May 2003, pp. 3–22
4. Bliguet, M.L., Kjær, A.A.: Modelling interlocking systems for railway stations. Technical Report IMM-M.Sc.-2008-68, DTU Informatics. Master thesis supervised by Anne Haxthausen, ah@imm.dtu.dk, Technical University of Denmark (2008)
5. Cao, Y., Xu, T., Tang, T., Wang, H., Zhao, L.: Automatic generation and verification of interlocking tables based on domain specific language for computer based interlocking systems (DSL-CBI). In: Proceedings of the IEEE International Conference on Computer Science and Automation Engineering (CSAE 2011), pp. 511–515. IEEE (2011)
6. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. The MIT Press, Cambridge (1999)
7. de Moura, L., Owre, S., Shankar, N.: The SAL language manual. Technical Report SRI-CSL-01-02, SRI International, 2003. Available from http://sal.csl.sri.com
8. Eriksson, L.-H.: Using formal methods in a retrospective safety case. In: Computer safety, reliability, and security—23rd International Conference, SAFECOMP 2004, volume 3219 of Lecture Notes in Computer Science. Springer, Berlin, Heidelberg (2004)
9. European Committee for Electrotechnical Standardization: EN 50128:2011—Railway applications—communications, signalling and processing systems—software for railway control and protection systems. CENELEC, Brussels (2011)
10. Fantechi, A.: The role of formal methods in software development for railway applications. In: Railway safety, reliability and security: technologies and system engineering, pp. 282–297. IGI Global, USA (2012)
11. Ferrari, A., Magnani, G., Grasso, D., Fantechi, A.: Model checking interlocking control tables. In: Schnieder, E., Tarnai G. (eds.)

Proceedings of formal methods for automation and safety in railway and automotive systems (FORMS/FORMAT 2010). Springer, Berlin, Heidelberg (2011)
12. Haxthausen, A.E.: Towards a framework for modelling and verification of relay interlocking systems. In: 16th Monterey Workshop: modelling, development and verification of adaptive systems: the grand challenge for robust software, number 6662 in Lecture Notes in Computer Science. Springer, Berlin, Heidelberg (2011) (Invited paper)
13. Haxthausen, A.E., Bliguet, M.L., Kjær, A.A.: Modelling and verification of relay interlocking systems. In: Choppy, C., Sokolsky, O. (eds.) 15th Monterey Workshop: foundations of computer software, pp. 141–153. Future trends and techniques for development, number 6028 in Lecture Notes in Computer Science. Springer, Berlin, Heidelberg (2010) (Invited paper)
14. Haxthausen, A.E., Kjær, A.A., Bliguet, M.L.: Formal development of a tool for automated modelling and verification of relay interlocking systems. In: 17th International Symposium on Formal Methods (FM 2011), number 6664 in Lecture Notes in Computer Science, pp. 118–132. Springer, Berlin, Heidelberg (2011)
15. Haxthausen, A.E., Peleska, J., Kinder, S.: A formal approach for the construction and verification of railway control systems. Formal aspects of computing, 23(2):191–219, (2011). The article is also available electronically on SpringerLink: http://www.springerlink.com/openhbreakhbreakurlss.asp?genre=article&id=doi:10.1007/s00165-009-0143-6
16. Manna, Z., Pnueli, A.: The temporal logic of reactive and concurrent systems. Springer, New York (1992)
17. Mirabadi, A., Yazdi, M.B.: Automatic generation and verification of railway interlocking control tables using FSM and NuSMV. Transp. Probl. **4**, 103–110 (2009)
18. Moller, F., Nguyen, H.N., Roggenbach, M., Schneider, S., Treharne, H.: Defining and model checking abstractions of complex railway models using CSP‖B. In: Hardware and Software: Verification and Testing, 8th International Haifa Verification Conference, number 7857 in Lecture Notes in Computer Science. Springer, Berlin, Heidelberg (2013)
19. Schnieder, E., Tarnai, G. (eds.): Proceedings of formal methods for automation and safety in railway and automotive systems (FORMS/FORMAT 2010). Springer, Berlin, Heidelberg (2011)
20. Symbolic Analysis Laboratory, SAL, home page: http://sal.csl.sri.com (2001)
21. The RAISE Language Group: The RAISE specification language. The BCS Practitioners Series. Prentice Hall Int., UK (1992)
22. The RAISE Method Group: The RAISE development method. The BCS Practitioners Series. Prentice Hall Int., UK (1995)
23. Winter, K.: Optimising ordering strategies for symbolic model checking of railway interlockings. In: 5th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA'2012), Part II, number 7610 in Lecture Notes in Computer Science, pp. 246–260. Springer, Berlin, Heidelberg (2012)
24. Winter, K.: Symbolic model checking for interlocking systems. In: Railway safety, reliability and security: technologies and system engineering, pp. 298–315. IGI Global, USA (2012)
25. Winter, K., Johnston, W., Robinson, P., Strooper, P., van den Berg, L.: Tool support for checking railway interlocking designs. In: Proceedings of the 10th Australian workshop on Safety Critical Systems and Software, vol. 55, SCS '05, pp. 101–107. Australian Computer Society Inc., Darlinghurst (2006)