SOFTWARE VERIFICATION RESEARCH CENTRE

SCHOOL OF INFORMATION TECHNOLOGY

THE UNIVERSITY OF QUEENSLAND

Queensland 4072
Australia

TECHNICAL REPORT

No. 02-29

Modelling Large Railway Interlockings and Model Checking Small Ones

Kirsten Winter and Neil J. Robinson

Version 1, September 2002
Phone: +61 7 3365 1003
Fax: +61 7 3365 1533
http://svrc.it.uq.edu.au

Submitted to *Australian Computer Science Conference (ACSC'2003)*.

Modelling Large Railway Interlockings and Model Checking Small Ones

Kirsten Winter **Neil J. Robinson**

Software Verification Research Centre
University of Queensland
Email: kirsten@svrc.uq.edu.au njr@svrc.uq.edu.au

Abstract

This paper describes the results to date of a feasibility study on model checking applied to railway interlockings. Our approach, in contrast to others, targets a high-level description of interlocking systems, namely the logical view of its operation. The result is a formal model that can be discussed with and validated by our industry partners and, moreover, provides a formal semantics for the notation that is used in practice. We suggest optimisations on the formal model and a decomposition technique for large railway layouts that is easy to apply. This renders our approach feasible for use in industrial practice.

Keywords: Railway interlockings, automated verification, model checking, Abstract State Machines

1 Introduction and Motivation

Railway signalling interlockings are safety critical systems. They are designed to permit the safe movement of trains along a railway system. Therefore special attention has to be given to the correctness of the design and the implementation of an interlocking system. We aim to provide automated tool support for the analysis during early stages of the design by means of model checking.

Railway interlocking systems, next to hardware designs, have been shown to be a suitable application for automated analysis techniques, such as model checking and automated theorem proving. Borälv and Stålmark (Borälv & Stålmarck 1999) employ automated theorem proving based on propositional logic for formal verification of interlocking systems. The formal model is derived from program code (written using the language Sternol) that implements the signalling interlocking. Eisner (Eisner 1999) uses the symbolic model checker RuleBase (Beer, Ben-David, Eisner, Geist, Gluhovsky, Heyman, Landver, Paanah, Rodeh, Ronin & Wolfstahl 1997) to verify the safety of railway interlocking software, which implements the control cycles. Huber (Huber 2001) describes a framework based on the model checker NuSMV (Cimatti, Clarke, Giunchiglia & Roveri 1999) for verifying interlockings specified using the Geographic Data Language (GDL), a language used in the British railway industry for defining a track layout and the corresponding signalling control. Simpson, Woodcock and Davies (Simpson, Woodcock & Davies 1997) use CSP, a formal language based on process algebras, to model geographical data of interlockings and apply the model checker FDR (For 1996).

In our current approach we apply the NuSMV model checker for automatically verifying safety requirements. However, we model the interlocking at a higher level of abstraction than the other approaches. Queensland Rail (QR), the main operator of railway systems in Queensland, Australia, is the industry partner in our project. QR, as most railways, uses a table notation for describing the functionality of an interlocking for a particular *track layout*. These tables are called *control tables*. The task is to support the analysis of these tables in order to find erroneous or incomplete entries (see Robinson et al. (Robinson, Barney, Kearney, Nikandros & Tombs 2001) for an overview of the project). In earlier work we used the language CSP as a modelling language and applied the model checker FDR. The results are published in Winter (Winter 2002).

In our new approach we model the semantics of control tables by means of the formal notation ASM (Abstract State Machines) (Gurevich 1995). Experience with our industry partners has shown that the resulting formal model is easier to read and understand than our corresponding CSP model (Winter 2002). It provides a formal semantics to the control tables (whose meaning is fairly difficult to grasp) as well as a platform for discussions with our industry partner. The formal model is automatically transformed into NuSMV code (which is equivalent to SMV code) using the tool interface introduced by Del Castillo and Winter (Castillo & Winter 2000). The safety requirements to be checked are modelled in CTL (Emerson 1990), the temporal logic supported by NuSMV. Due to the high-level of abstraction that is given through our control table model, the safety requirements can be stated in terms of train movement and train collision. The requirements specification therefore becomes easy to write and to understand. Also the counter-examples provided by the NuSMV tool when an error is found are easy to analyse by railway engineers.

In this paper, we introduce our formal model of interlockings based on control tables, propose several optimisations to the model to improve efficiency of the model checking process, and provide a procedure for decomposing large interlocking systems into smaller ones without decreasing the scope of our verification approach. Optimisation and decomposition techniques render our approach feasible for realistic case studies.

The paper is organised as follows: Section 2 explains the basic concepts used in QR's interlocking design process. Section 3 describes briefly the formal notation of ASM and presents our formal model of interlockings and the requirements. Section 4 introduces some optimisations that helped improve the model and describes a decomposition procedure for splitting up large systems. We conclude the paper in Section 5.

2 Railway signalling and control tables

Railway signalling permits the safe movement of trains over a *track layout*. An example track layout is shown in Figure 1. *Signals*, e.g. `he2`, use colour light indications (e.g. green for go, red for stop), to give authorities for trains to travel a particular *route* through

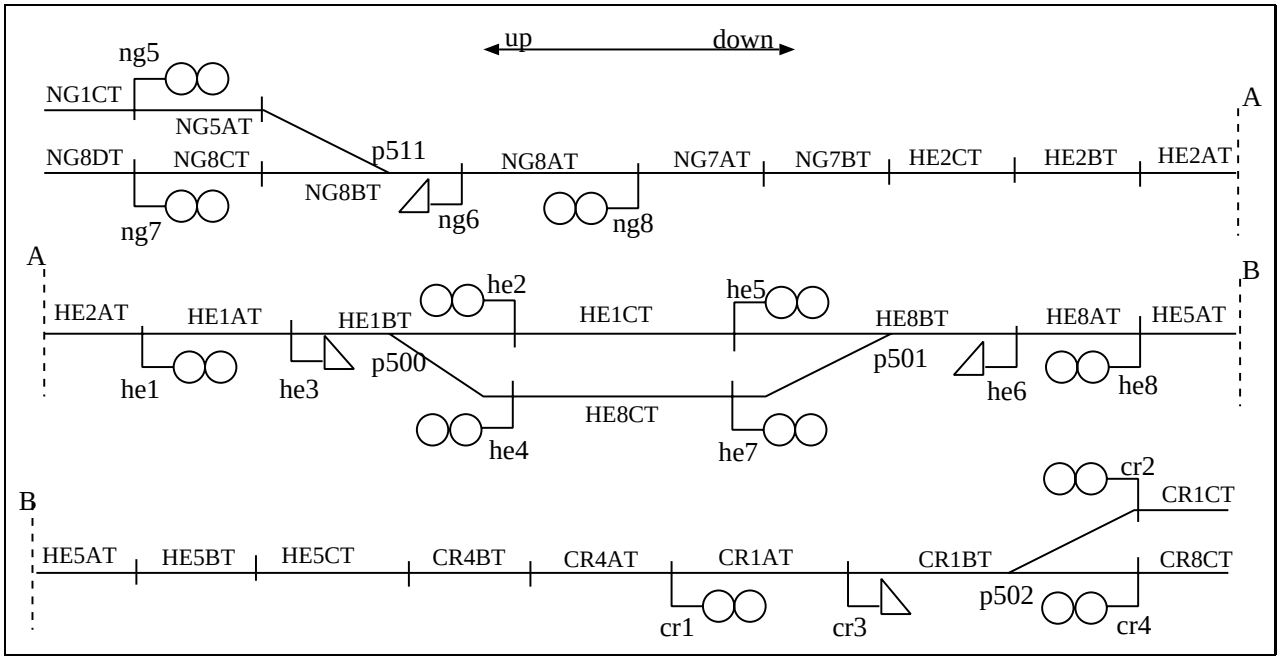

Figure 1: Track layout of the Helensvale section and its surrounding

the layout (triangles in Figure 1 represent shunt signals that operate similarly). *Points*, e.g. `p500`, are switches in the track that permit the train to continue at high speed in the current direction *normal* or to 'switch' to another track, generally at a lower speed *reverse*.

Railway interlockings control and monitor the signals and points, and monitor the position of trains on the layout via *track circuits*. Track circuits can either be *clear* indicating that there is no train on the track section, or *occupied* indicating the possible presence of a train. Track circuits are shown in Figure 1 by labels adjacent to track sections, e.g. `HE1CT`.

The functionality of QR's railway interlockings is generically specified by the QR *signalling principles*, which state in general terms how track layouts should be signalled. To specify the functionality of a particular railway interlocking, QR uses *control tables*. These are functional specifications for the signalling of a particular track layout and are presented in a tabular form. It is the control tables that we aim to verify with our model checking approach. In the following we refer to the signal control table simply as the *control table*. We will also mention the *points table* which is responsible for the control of points.

Table 2 shows a part of the control table for Helensvale (see Figure 1), which specifies the conditions for route `he2_1m`, from signal `he2` to signal `ng8`. The meaning of the conditions can be informally understood from the column headings. For example, the "Tracks clear" column (column ⑧) specifies those tracks which must be clear before route `he2_1m` can be cleared for a train to pass through. As described in Section 3, we define the semantics of QR's control tables with a formal model.

3 The Formal Model

ASM (Abstract State Machines) (Gurevich 1995) is a formal notation with an operational semantics similar to ordinary state transition systems. The state space is specified through domains and functions over these domains.

The transition behaviour is given through a set of transition rules for which a minimal set of rule constructors is provided.

A computation (*run*) of an ASM model is given as a sequence of states S_i, obtained from a given initial state S_0 by repeatedly executing transitions δ_i:

$$S_0 \xrightarrow{\delta_1} S_1 \xrightarrow{\delta_2} S_2 \ \ldots \ \xrightarrow{\delta_n} S_n \ \ldots$$

Each transition δ_i is a group of transition rules applied (*firing*) *simultaneously* to the current state S_{i-1} of the model.

Domains are specified as types in the typed version of ASM that is used here. The functions over these domains can be *static*, *dynamic*, or *external*. Static functions simply define constants. Their evaluation is predefined and they do not change their value. In contrast, dynamic functions are not predefined. They are only initialised but their evaluation can change throughout a run of the system due to *updates* specified in the transition rules (see below). Dynamic functions play the role of state variables. External functions may change their value during a run, however, they are not controlled by the model. Their value is arbitrarily given through the environment. External functions model oracles or inputs to the system.

The following transition rule constructors are used in our model:

- Simple update:
  $$f(t_1, \ldots, t_n) := t$$
  where t_i are terms and f is a dynamic function; updates are comparable to simple value assignments in programming languages;

- Guarded command rule:
  `if` G `then` R_T `else` R_F `endif`
  where G, the *guard*, is a Boolean expression and R_T and R_F are transition rules; the semantics is that of a simple guarded command "if guard is true then apply rule R_T otherwise apply R_F".

- Block rule:
  `block` $R_1 \ldots R_n$ `endblock`
  where R_i are transition rules; the block rule groups a set of transition rules and fires them simultaneously;

- Do-forall rule:
  `do forall` v `in` A `with` $G(v)$ $R(v)$ `enddo`
  where v is a name of a new variable (bound in this rule), A is a set, $G(v)$ is a Boolean condition

Signal	Route Number	Route to	Route Indication	Requires									Replaced by Tracks Occ
				Points Locked		Routes		Route Holding	or Until		Tracks		
				Normal	Reverse	Normal	Reverse	Maintained by Tracks occ	Tracks occ	for Time secs	Clear	Occ	
he2	1m	ng8	-	p500							HE1BT HE1AT HE2AT HE2BT HE2CT NG7BT NG7AT NG8AT		HE1BT
						he3(1s)		HE1BT					
						he1 (1M) he1 (2M)		HE1BT HE1AT					
						ng5 (1M) ng7 (1M)		HE1BT HE1AT HE2AT HE2BT HE2CT NG7BT NG7AT					
ⓐ	ⓑ	ⓒ	ⓓ	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⓪

Figure 2: Control Table for Helensvale (shown is the row for route `he_2m` with its sub-rows)

over v, and $R(v)$ is a transition rule depending on v; semantically the do-forall rule collects all rule instances $R(a)$ with values $a \in A$ such that $G(a)$ is satisfied; all those instances are fired simultaneously; the restrictions on the values of parameter v, namely the phrase `with` $G(v)$, is optional and can be omitted.

3.1 The ASM Model of Control Tables

We use ASM as a notation to model the logic of interlockings at a high level of abstraction. Our formalisation comprises

a.) the *static part* (namely data types and static function definition as well as dynamic/external function declarations) that contains all information about the particular track-layout that is considered, and the corresponding control tables;

b.) the *behavioural part* (namely a set of transition rules) that specifies the logic of interlocking control as it is described through the control table.

The framework of our model, meaning the function declarations and the set of transition rules, is generic. For analysing a particular layout and its control table, the generic model is "instantiated" with the static information of the layout and the table, a process that can easily be automated.

In contrast to other approaches, we also model the movement of trains in the track-layout. Although our model of trains is fairly simplistic, e.g. we do not specify the speed of a train, it prevents unrealistic behaviour like the jumping of trains. The benefits of this approach are that the safety requirements are easier to specify and to read. They relate directly to the hazards the signalling system is designed to avoid. Also the counterexamples are easier to follow in that they provide a hazardous scenario.

The Static Part.

As domains of our model, we define the sets of tracks, signals, points, and routes that are given in the track-layout. Additionally, we define a set of trains. These sets are specified as simple enumerated types, for example the type *TrackId*: `datatype` $TrackId$ `==` $\{HE1AT, HE1BT, HE1CT, HE8CT, \ldots\}$.

Furthermore, we specify a number of value domains, e.g. the state of a track with the values clear and occupied [1]: `datatype` $TrackState$`==`$\{clr, occ\}$.

We also have to model a set of request-values. For each route, we need a value for requesting this route and for cancelling a request. For each point we need a value for requesting to set the point normal and one to set it reverse: `datatype` $RequestType$`==`$\{req\_he2\_1m, reqC\_he2\_1m, \ldots, reqN\_p500, reqR\_p500, \ldots\}$.

We define a list of static functions to model the columns in the control table and the point table. For instance, the function

`static function` $tracksClear$:
$RouteId \rightarrow$ `SET`$(TrackId)$ `==`
`MAP_TO_FUN` $\{he2\_1m \rightarrow \{HE1BT, \ldots\}, he3\_1s \rightarrow \ldots\}$

models column ⑧ of the table in Figure 2. The function maps each route to the set of tracks that are listed in the corresponding field in the table. Note that a related project aims to generate the mappings for these static functions automatically from a control table.

Another category of static functions reflects the information given in the track-layout, e.g. a function *next* that provides for each track in each direction the next track. These functions are necessary in order to control the train movement.

A number of dynamic functions describe those entities in our model that can change their value during a run. We dynamically model the *trackState* (each track becomes occupied when a train is on it), the *routelock* (a route is either set reverse, i.e. locked or reserved for an approaching train, or set normal, i.e. free), the *signalaspect* (each signal indicates either proceed or stop), the *pointset* (a point is set normal or reverse), the $front$ position of a train, the *rear* position of a train, and its direction *dir*. The dynamic function *route_used* indicates for each route how far a train has proceeded on this route so that parts of the route can be released as soon as the train has passed certain points. We assign numerical values for the "degree" of usage. With this model of route-use, we avoid introducing dynamic functions for all subroutes (which would increase the complexity of the model checking process). The maximal degree of usage coincides with the number of sub-rows for each route.

We declare some external functions to represent input to our model. For instance,

[1] Although most of these domains could also be modelled in terms of Boolean, we thought it prudent to adapt the railway terminology to improve readability.

```
transition Route_Reverse ==
   do forall r in Routes
      if
         guard(r)
      then
         routelock(r) := rtR
      endif
   enddo
```

Figure 3: Transition Rule for Route_Reverse

`external function` *request* : *RequestType* models a request for a route or a point that is input by the signaller. The value of this nullary function can change arbitrarily and is not controlled by the model. The external function *long_enough_occ* is specified as a Boolean. It is used for now to model those conditions that need some time to elapse before being satisfied. If a train is occupying the track in column ⑥ longer than the number of seconds in column ⑦, then the route locking applied by the points and routes in columns ① to ④ can be released. This condition is modelled by the Boolean *long_enough_occ*.

The Behavioural Part.

The behavioural part of our formal model specifies the actual meaning of the control table, i.e. the actual control of routes, points and signals. It consists of a set of transition rules, each of which describe the control of one of the dynamic functions that are described above.

All transition rules have the same structure, which we will explain using *Route_Reverse* as an example. Transition rule *Route_Reverse* sets a route r reverse (i.e. locks or reserves it for an approaching train) if a particular condition for r, *guard(r)*, is satisfied. This checking of the condition and setting the route is done simultaneously for all routes.

The rule has a structure as shown in Figure 3. The guard that needs to be satisfied in order to set the routelock of a particular route r reverse (rtR) is a conjunction of simple conditions on r. All these conditions can be read from the control table by railway engineers. Since the static functions in our model reflect the content of the control table, the simple conditions are given in terms of these static functions. They read as follows:

- *member(request, RouteReq)*
  `and` *convertRouteReq(request)=r*
  The current request must be a request for a route and, in particular, it must be a request for route r, the route that is currently considered.
- `forall` *t* `in` *tracksClear(r) :*
  trackState(t)=clr
  All tracks in the entry of tracksClear column (⑧) for route r must be clear.
- `forall` *p* `in` *pointLockDetN(r) :*
  pointset(p)=setN
  All points in pointsLockedNormal column (①) must be set normal.
- `forall` *p* `in` *pointLockDetR(r) :*
  pointset(p)=setR
  All points in pointsLockedReverse column (②) must be set reverse.
- `forall` *r2* `in` *routesNormal(r) :*
  routelock(r2)=rtN
  All routes in routesNormal column (③) must be set normal.
- `forall` *r3* `in` *routesReverse(r) :*
  routelock(r3)=rtR
  All routes in routesReverse column (④) must be set reverse.
- `forall` *pair* `in` *routeNormalIndex(r) :*
  (routelock(first(pair))=rtN `and`
  (route_used(first(pair)) >= second(pair))
  All routes, for which the current route r appears in a sub-row i in column ③, must be normal and not used (not used implies that the tracks in column ⑤ are clear).

If *guard(r)*, the conjunct of the simple conditions that are listed above, is not satisfied, then the value of routelock(r) will not change.

All other transition rules are specified similarly. For all those transition rules that update the same dynamic function we have to make sure that in every state the guards of both rules exclude each other in order to prevent a clash of two updates happening simultaneously. This is easy to show in our model since all related rules have contradicting conditions.

The transition rule that models the movement of trains is based on the following notion of trains: Stationary trains occupy one track, a moving train never occupies more than two tracks and they move at a constant speed, i.e. they move their front or rear at each step onto the next track. They obey signals in that they always stop in front of a red signal. A stationary train will move in the direction of a clear signal (this permits a train to change direction in the case its rear faces a clear signal). Track layouts are not circular, they simply end on the left and right side. As a consequence, trains disappear at the boundaries of the track-layout. The front and rear position become undefined. Trains can also appear on the track-layout from beyond the boundaries. Since our model has a fixed set of trains a "new" train can only appear if it has disappeared before.

As a train moves, tracks clear or become occupied. The dynamic function *trackState* indicates this status. This status is only evaluated one step after a train has moved from or onto a track, i.e. the indication is delayed. This is a fairly realistic model since also in a real implementation track-circuits first have to detect their status and a delay should be expected.

3.2 The Safety Requirements to be verified

The general safety requirements of railway interlocking systems are explained in full detail in Tombs, Robinson and Nikandros (Tombs, Robinson & Nikandros 2002). The formalisation of these requirements are easy to model in our case since they can be expressed in terms of trains rather than route, point or signal settings as in the related work (e.g. (Simpson et al. 1997) and (Huber 2001)). We use the temporal logic CTL (Emerson 1990) as a specification language which is supported by the NuSMV tool. Two examples follow.

No Collision.

A train collision is simply specified as two trains (CR and FS) occupying (either with their front or rear) the same track, unless the track is outside the track-layout (i.e.$undef$). The CTL formula for non-collision is given as follows:

```
AG (((front(CR) != front(FS))
    & (front(CR) != rear(FS)))
   | (front(CR) = undef))
 &
   (((rear(CR) != front(FS))
    & (rear(CR) != rear(FS)))
   | (rear(CR) = undef))
```

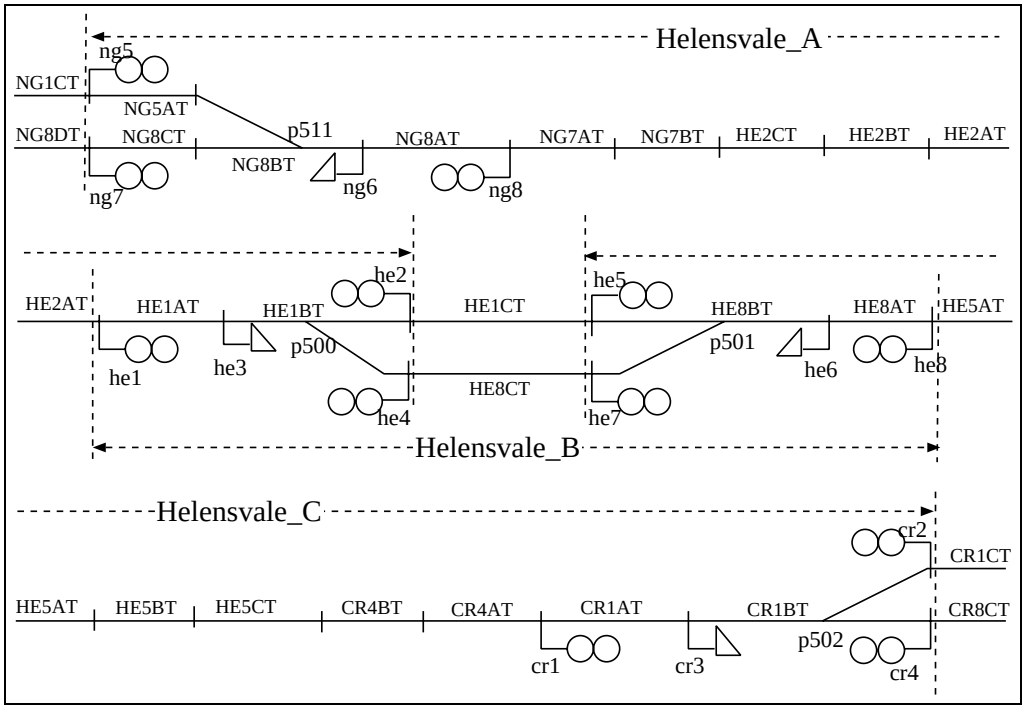

Figure 4: Segments of the Helensvale track-layout

where `AG` can be read as *always*, `|` as *or*, `&` as *and*, and `!=` as *not equal*.

No Derailment.

The notion of derailment caused by a point that is moving under a train can be specified in CTL as follows: If one of the trains (*CR* or *FS*) has either its front or rear on the track where the point is located (provided by static function *homeTrack(p)*) then the point should not change its position. That is, the point is either normal and stays normal in all next states or it is reverse and it stays reverse.

We give the CTL formula for the case that point p is normal:

$$\begin{aligned}&\texttt{AG}\ ((front(CR) = homeTrack(p)\\ &\quad \texttt{|}\ rear(CR) = homeTrack(p)\\ &\quad \texttt{|}\ front(FS) = homeTrack(p)\\ &\quad \texttt{|}\ rear(FS) = homeTrack(p))\\ &\quad \texttt{\&}\ pointset(p) = setN\\ &\texttt{->}\ \texttt{AX}\ (pointset(p) = setN))\end{aligned}$$

where `AX` can be read as *always in the next state*. This has to be proved for each point in the track-layout for both directions.

4 Model Checking the Formal Control Table Model

Model checking as a fully automatic approach is limited with respect to the state space of the model. With the formal model described so far we were able to check only very small track-layouts. In order to get better results for our feasibility study on model checking, we exploited a number of optimisations on our formal model.

4.1 Optimising the ASM Model

We agreed with our industry partner on a number of simplifications and optimisations that could be exploited. So far we have not attempted any optimisations of the model checking procedure itself, only on how our model is formalised. For example:

- All those tracks in the layout that always occur as a group in the control table can be collapsed into one track. For example, the tracks HE2AT, HE2BT, HE2CT, NG7BT, NG7AT in Figure 1 are replaced by one track HE2AT_NG7AT in our model. These changes also affect the safety requirements.
- One part of the control table logic describes the functionality of *approach locking*, i.e. the locking of routes in advance for an approaching train. We decided to restrict our checking to a model without approach locking in order to decrease the state space to be checked. This also allowed us to simplify our train movement model.
- The dynamic function *route_used* was originally computed for all routes in the layout. However, it is only ever required for those routes which occur in the columns ③ or ④ or in the point table. Restricting the range of the transition rule *Route_Usage* appropriately helped saving up to eight extra state variables in our example.
- The external function *long_enough_occ* models if a train occupies a track long enough so that the locking of the corresponding route can be released. Since only those cases are safety critical in which the route locking is released, it is sufficient to check what happens if *long_enough_occ* evaluates to true. Hence, we changed this external function into a static function that is always true.

These optimisations were all discussed with railway engineers. The discussion provided some validation for the optimised model. This was only possible because the formal model was understood by everybody and changes could be communicated.

All the suggested optimisations above have helped to speed up the model checking process quite remarkably. However, a track-layout of the size shown in Figure 1 (i.e. 24 routes, 16 signals, 18 tracks (after combining tracks) and 4 points) is without further op-

```
...

State 1.8:
incoming_train = FS
request = reqC_he2_1m
route_used(he2_1m) = 0
signalaspect(he2)
              = proceed
State 1.9:
dir(FS) = up
front(FS) = HE1BT
incoming_train = CR
request = req_ng5_1m
routelock(he2_1m) = rtN
signalaspect(he2) = stop

State 1.10:
rear(FS) = HE1BT
route_used(he2_1m) = 1
trackState(HE1BT) = occ

State 1.11:
front(FS) = HE1AT
route_used(he2_1m) = 2

State 1.12:
rear(FS) = HE1AT
trackState(HE1AT) = occ

State 1.13:
front(FS) = HE2AT
route_used(ng8_1m) = 2
trackState(HE1BT) = clr

State 1.14:
move(FS) = 0
rear(FS) = HE2AT
route_used(he2_1m) = 3
trackState(HE2AT) = occ

   ...

State 1.19:
incoming_route = ng7_1m
route_used(ng7_1m) = 0
signalaspect(ng7)
              = proceed
State 1.20:
front(CR) = NG8CT

State 1.21:
rear(CR) = NG8CT
trackState(NG8CT) = occ

State 1.22:
front(CR) = NG8BT

State 1.23:
rear(CR) = NG8BT
trackState(NG8BT) = occ

State 1.24:
front(CR) = NG8AT
trackState(NG8CT) = clr

State 1.25:
move(FS) = 1
rear(CR) = NG8AT
trackState(NG8AT) = occ

State 1.26:
front(CR) = HE2BT_NG7AT
front(FS) = HE2BT_NG7AT
trackState(NG8BT) = clr
```

Figure 5: A Counter-example output by NuSMV

timisations still beyond our computational capacity[2]. The solution to this problem lies in the decomposition of large layouts. The procedure is described in the following.

4.2 Practical Approach for Decomposing Interlockings

In order to deal with large track-layouts, we use the following decomposition technique. We divide the track-layout into a number of segments such that the following is guaranteed:

- On both sides (left and right) the layout is cut *behind* a signal that faces towards the part that is focussed. These incoming signals allow us to safely let trains appear from the outer world onto the layout segment without causing collisions.
- All signals, points and routes between left and right boundary have to be considered in the control table model. Even those routes that lie only partly in the segment must be checked for the part within the boundaries of the segment.
- Each route of the full layout and its opposing route must be contained fully in at least one of the checked sub-layouts, i.e. we have to check overlapping segments.

If the safety requirements are satisfied by each segment then we can conclude that they are also satisfied for the track-layout as a whole. This decomposition technique conforms with the approach used in industrial practise and it can thus be regarded as validated by use.

Following these rules for decomposition, we are able to check the Helensvale layout in three smaller steps. The proposed segments are

1. Helensvale_A: between signals `ng5` and `ng7` and signals `he2` and `he4`
2. Helensvale_B: between signal `he1` and signal `he8`
3. Helensvale_C: between signals `he5` and `he7` and signals `cr2` and `cr4`.

[2]The main problem was the limitation of memory space which was exceeded by the process (up to 2173MB) and swapping became necessary.

They lead to the decomposition of the Helensvale layout as shown in Figure 4. Note that there are overlapping parts (e.g. between signals `he1` and `he2`, `he4`), which are contained in two segments.

4.3 Model Checking Results

Helensvale, cut by vertical boundaries into three parts as described above, became a feasible task for the model checker. Each part could be checked within one hour.

In order to check the scope of errors the model checking approach can find in a control table, we introduced errors into the table. We give an example for the model Helensvale_A (see Figure 4): In order to check the entries of column ⑤ we need to delete all entries of column ⑧. This is necessary since with our simple train movement model the columns carry redundant controls. Assume that in the row for route *he2_1m* (as shown in the control table in Figure 2) the entry *HE2AT* is missing in column ⑤ in the fourth sub-row. As a consequence this track is not collapsed with the tracks *HE2BT, ..., NG7AT* since to collapse a number of tracks they always have to appear together. With this missing entries in column ⑤ and ⑧ the route locking against *ng5_1m* and *ng7_1m* will not be maintained (i.e. kept locked) when a train is on track HE2AT.

Checking the erroneous ASM model results in a counter-example output by the NuSMV. A counter-example is a list of states that lead to a state that violates the checked safety requirements (i.e. in this case a front-to-front collision). For each state only the changes from the previous state are given. Figure 5 shows the key parts of the states that finally lead to a collision of trains *CR* and *FS* on the track next to the omitted track, *HE2BT_NG7AT* (see **State 1.26**).

If an error exists a counter-example is often generated in a relatively short period of time as the full state space does not need to be explored. Finding this error took less than 20 minutes user time. Our experience suggests that the initial debugging of the control table is very fast and that counter-examples are fairly quickly computed by the NuSMV tool. It is only when no error can be found that the model checker needs a long time to complete.

5 Conclusion and Future Work

This work describes an approach for debugging interlocking specifications, provided as a high-level description in tabular form, the so called control tables. We introduced our formal model of the control tables specified in ASM notation. Additionally, we introduced a model for train movement, so that trains could use the modelled interlocking system. As a consequence, the requirements of the system can be easily formalised in terms of trains, namely no collision and no derailment should occur. In case a requirement is violated the output counter-example provides a meaningful scenario of possible train behaviour that would lead to an accident. This is easy to read by railway engineers.

In contrast to work suggested by others (e.g. (Borälv & Stålmarck 1999), (Eisner 1999), (Simpson et al. 1997) and (Huber 2001)) our approach focuses on the modelling side of the analysis. We choose a formal language, namely ASM, that seemed to be closest to the construct to be modelled, namely control tables. Our optimisations are based on the model formalisation rather than on the checking algorithms. We formulated and tested an approach to allow the decomposition of large layouts.

It was essential to have QR railway engineers validate our model including our simplifications that optimised the model. For any simplification the consequences for the model checking results needed to be discussed so that we were aware about how a simplification would decrease the scope of errors that can be found.

Our approach would certainly benefit from an optimisation within the model checking algorithms. Huber reports in his thesis (Huber 2001) attractive results gained by tailoring the NuSMV algorithms towards checking of geographical signalling data. Although both control tables and geographical data model interlockings, the table notation used by QR is more abstract. As a consequence our formal model has a different structure. Most significantly, our model is a synchronous model whereas Huber's model is asynchronous. Nevertheless, it appears to be promising to follow his approach in our future work, namely mapping the ASM model onto the internal data structure of the NuSMV tool and adapting algorithms and heuristics to our particular needs. However, we do not expect that this will remove the need for decomposition of the track layouts.

Moreover, we are planning to develop an animator for control tables based on our formal model. This will help to check liveness properties which cannot easily be checked by the model checker since the input values can easily obstruct liveness of the system. It will also allow us to run the counter-examples provided by the checker. In the future we may extend our model in order to check, e.g., the functionality of approach locking, a feature that is currently not checked. This would also involve the use of a more sophisticated train model.

To complete the feasibility study on automated tool support for control tables, we will apply the automated theorem prover NP-Tools (Borälv 1998) to our verification task and compare the results with our results from NuSMV.

Acknowledgements

This work is a result of joint work with and funded by Queensland Rail (QR). It greatly benefited from the knowledge and insight into railway interlocking systems provided by David Barney from QR. We would also like to thank George Nikandros from QR for his interest and support of our work.

References

Beer, I., Ben-David, S., Eisner, C., Geist, D., Gluhovsky, L., Heyman, T., Landver, A., Paanah, P., Rodeh, Y., Ronin, G. & Wolfstahl, Y. (1997), Rulebase: Model checking at IBM, *in* O. Grumberg, ed., 'Proc. of Int. Conf. on Computer Aided Verification, CAV'97', Vol. 1254 of *LNCS*, Springer-Verlag, pp. 480–485.

Borälv, A. (1998), 'Case study: Formal verification of a computerized railway interlocking', *Formal Aspects of Computing* **10**, 338–360.

Borälv, A. & Stålmarck, G. (1999), Formal verification in railways, *in* M. Hinchey & J. Bowen, eds, 'Industrial-Strength Formal Methods in Practice', Springer-Verlag.

Castillo, G. D. & Winter, K. (2000), Model checking support for the ASM high-level language, *in* S. Graf & M. Schwartzbach, eds, 'Proc. of Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2000', Vol. 1785 of *LNCS*, Springer-Verlag.

Cimatti, A., Clarke, E., Giunchiglia, F. & Roveri, M. (1999), NuSMV: A new symbolic model verifier, *in* 'Proc. of Int. Conf. on Computer Aided Verfication, CAV'99', Vol. 1633 of *LNCS*, Springer-Verlag, pp. 495–499.

Eisner, C. (1999), Using symbolic model checking to verify the railway stations of hoornkersenboogerd and heerhugowaard, *in* 'Proc. of Conf. on Correct Hardware Design and Verification Methods (CHARME'99)', Vol. 1703 of *LNCS*, Springer-Verlag.

Emerson, E. A. (1990), Temporal and modal logic, *in* J. van Leeuwen, ed., 'Handbook of Theoretical Coomputer Science', Vol. B, Elsevier Science Publishers.

For (1996), *Failure Divergence Refinement, FDR 2.0, User Manual*.

Gurevich, Y. (1995), Evolving Algebras 1993: Lipari Guide, *in* E. Börger, ed., 'Specification and Validation Methods', Oxford University Press.

Huber, M. (2001), Towards an industrially applicable model checker for railway signalling data, Master's thesis, University of York.

Robinson, N., Barney, D., Kearney, P., Nikandros, G. & Tombs, D. (2001), Automatic generation and verification of design specification, *in* 'Proc. of Int. Symp. of the International Council On Systems Engineering (INCOSE)'.

Simpson, A., Woodcock, J. & Davies, J. (1997), The mechanical verification of solid state interlocking geographic data, *in* 'Proc. of Formal Methods Pacific (FMP'97)', Vol. vii+320, Springer-Verlag, pp. 223–243.

Tombs, D., Robinson, N. J. & Nikandros, G. (2002), Signalling control table generation and verification, *in* 'Proc. of Conference on Railway Engineering (CORE 2000)', Railway Technical Society of Australasia.

Winter, K. (2002), Model checking railway interlocking systems, *in* M. Oudshoorn, ed., 'Proc. of Australasian Computer Science Conference (ACSC02)', Vol. 24 of *Austalian Computer Science Communications*, Australian Computer Society Inc., pp. 303–310.