Tool Support for Checking Railway Interlocking Designs

K. Winter **W. Johnston** **P. Robinson** **P. Strooper** **L. van den Berg**

School of Information Technology and Electrical Engineering
University of Queensland
Email: kirsten@itee.uq.edu.au

Abstract

The development of railway interlocking systems is currently very labour-intensive. Specialists develop the interlocking design for a particular area and manually check for completeness and consistency. The interlocking is implemented in either software or using electrical relays. The interlocking has to be tested against the safety requirements for signalling systems, i.e., the signalling principles.

The whole process can be supported by various tools, ranging from editors to animators. In this paper we focus on exploiting model checking to automatically check the design with respect to safety. The main concerns of this technique are the problem size and the efficiency of available tools. We have investigated both of these problems: seeking to work with a minimal model of the interlocking design and to improve efficiency of the model-checking process by exploiting domain knowledge of our particular application.

Keywords: Railway interlockings, automated verification, symbolic model checking, Binary Decision Diagrams

1 Introduction

Model checking (Clarke, Grumberg & Peled 2000) is an automatic technique used to support the validation and verification of system designs. It is of particular interest to industry since its application does not rely on any expertise in the underlying verification technique.

A model checker explores the full state space of a given model of the system. Similar to a complete test, every possible behaviour is investigated. The tool provides the user with an answer indicating whether the model violates a given property or requirement. Most tools provide a *counter-example* that shows a possible scenario in the case when a violation occurs, which proves to be very useful when debugging the model.

The following tasks have to be solved when setting up a development process that integrates model checking to support debugging and verifying a system design.

1. The system design has to be modelled formally in the input language of the tool.
2. Requirements or properties to be checked have to be identified and also formalised in the tool's input notation for requirements.
3. Both model and requirements have to be carefully validated. That is, the user has to make sure that both formalisations correspond with the actual system design and requirements that ought to be checked.
4. A thorough analysis has to clarify which problems in the system design can be detected with model checking and which cannot be detected due to the given formalisation of system design and requirements.

Within a specific application, tasks 1 and 2 of the list above can often be automated once the domain of interest is well enough understood. This is possible, if (a) a system design is always provided in a standard format and (b) the requirements can be derived in a standard fashion for each particular case.

Railway signalling interlockings are safety critical systems. They are designed to permit the safe movement of trains along a railway system. We are currently investigating the use of model checking for the verification of railway interlocking designs within a collaborative project with Queensland Rail (QR). It is planned that the interface to the model checker will become part of a *Signalling Design Toolset* (Robinson, Barney, Kearney, Nikandros & Tombs 2001), which includes also a track-layout editor, a control-table generator, and a control-table editor (CTE). For this application, both conditions above are satisfied: the system design (i.e., the interlocking design) is specified by experts at QR as tables, called *control tables*. The *track layout* and *route table* (Winter & Robinson 2003) provide additional information about the position of signals, points, tracks, and routes between the signals. From these documents, our tool support automatically generates a formal model of the interlocking design. The properties we want to check are the safety requirements of an interlocking system as specified in the *Signalling Principles*[1] (SAOS Standards 1999). They are generic for our particular model and can be summarised as (a) avoidance of train collision and (b) avoidance of train derailment.

We use the model checker NuSMV (Cimatti, Clarke, Giunchiglia & Roveri 1999) to check the safety properties for interlocking designs. NuSMV is a software tool for the formal verification of finite state transition systems. It is a reimplementation and a re-engineering of the *Symbolic Model Verifier* (SMV) developed by McMillan at Carnegie Mellon University (McMillan 1993). The tool checks whether temporal logic properties are satisfied by a given model. That is, the model has to be specified using (typed) state variables to model the state space and guarded transitions that capture the behaviour of the model. The requirements have to be specified in *Computation Tree Logic* (CTL), a propositional branching type

Copyright ©2005, Australian Computer Society, Inc. This paper appeared at the *10th Australian Workshop on Safety Related Programmable Systems (SCS'05)*, Sydney. Conferences in Research and Practice in Information Technology, Vol. 55. Tony Cant, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

[1]This is a document of the principles to be applied to all signalling works in the Brisbane suburban area.

temporal logic (Emerson 1990). Both input notations are well suited for our problem. Additionally, NuSMV is a *symbolic* model checker which means model and requirements are internally represented by graph structures, called *Binary Decision Diagrams* (BDDs) (Bryant 1986). Generally, BDD-based model checking has proved to be very efficient (see e.g., (Burch, Clarke, McMillan, Dill & Hwang 1992)).

However, to make our approach of model checking interlocking designs feasible for use in practise, we have to target the issue of efficiency. Whereas a small design can be automatically checked quite fast, as design size increases, the time taken to check the design increases at a rapid rate and may not return a result at all. This is often referred to as the *state explosion problem*. Run-time and memory usage of the process have to be improved. This can be done in two ways: (a) reducing the models of design and requirements by stripping information that is not necessary for the model checker and (b) by improving the model checking process itself for this particular application.

This paper addresses both issues. In Section 2, we describe our particular model of the system design that allows for a generic requirements specification and how this model could be optimised in terms of its size. Section 3 shows how the characteristics of the model checker we use can be exploited to gain a significant speed-up in run-time by using domain knowledge. We report on related work in Section 4 and conclude the paper in Section 5.

2 The Model of the Interlocking Design and its Requirements

Run-time and memory-usage of the model checking process depend on the size of the model and the complexity of the requirements to be checked. Factors that determine the model's size are the number of state variables, the size of their (enumerated) type, and the number of transitions that model the behaviour. The complexity of the requirements can be measured in terms of the length of the CTL formula and the number of nested temporal operators. To optimise the complexity of the model-checking process we have to minimise these factors.

2.1 The Model

Unlike other approaches for verifying interlockings (see Section 4), our model not only includes a model of the interlocking design but also of (one or two) trains moving along the tracks (Winter & Robinson 2003). As a consequence, the safety requirements become generic and very easy to validate because they can be modelled in terms of trains. Trains must not collide and they must not derail. The checking process verifies that trains, that are moving according to the constraints permitted by the control tables, do not violate these safety requirements. The model therefore consists of a model of the behaviour of trains, in terms of how they move from one track to the next and how they react to signalling equipment, and a model of signalling equipment behaviour, instantiated by the behaviour prescribed in the control tables.

Our model is instantiated for a specific *verification area* which describes a local part of the railway network. Figure 1 depicts the *track layout* of a small verification area showing the location of points, signals and tracks within that area. Each verification area should be large enough to include at least one route and all its opposing routes. Ideally, a verification area would include all the routes and opposing routes for a particular interlocking. For each verification area,

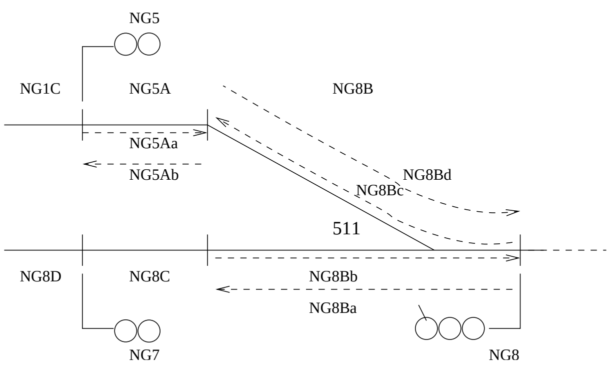

Figure 1: Example of a Verification Area

an extract of the control table which includes data relevant to the area is produced.

Figure 1 is an example track layout. Signals, e.g. NG5, use colour indications (e.g. green for go), to give authorities for trains to travel a particular *route* through the layout. The difference between signals that have two circles (e.g. NG5) and those with three and a dash (e.g. NG8) is not important in this context. *Points*, e.g. 511, are movable components in the track that permit a train to move from one track to another. The position of the points is referred to by the railway signalling industry as points normal or points reverse. A route is a path between two facing signals (a facing signal is a signal that is 'facing' towards an approaching train), a route can be locked *reverse* meaning it is reserved for use, or *normal* meaning it is free. NG1C, NG5A, NG8D and NG8C are tracks, while NG5Aa, NG5Ab and NG8Ba-d are track-segments, the latter are introduced purely for the purpose of modelling and are not based on actual hardware design.

Train behaviour is such that trains only proceed past a facing signal if that signal is showing proceed. The trains otherwise move from track to track according to data extracted automatically from the track layout. Signal equipment behaviour is generically described in the Signalling Principles. For example, points can only change state if the tracks which contain the points are not occupied, and for every route crossing the points, the conditions for holding the points in their current lie do not apply. The precise data, giving which routes and what locking conditions are needed, is extracted (automatically) from the data for the verification area that is under investigation.

Our model comprises the following entities and entity values: trains and their positions, signals and their aspects (either `stop` or `proceed`), points and their lie (either `normal` or `reverse`), and routes and their locking and route-usage. These entities are modelled as state variables. Their values can be changed at each state if the conditions (corresponding to the general description in the Signalling Principles and according to the control table entries for the specific verification area) are satisfied. We adopt a *synchronous model*, i.e., at each step all possible changes to the state variables are conducted at the same time. This model of synchronous conditional updates reflects the behaviour of an interlocking system of a specific verification area as it is permitted by the corresponding control tables.

A position of a train is given in terms of a segment. Each track comprises one or more segments, where each segment represents a unique way of traversing the track. For tracks NG5A and NG8B figure 1 shows the corresponding segments, namely NG5Aa and b, and NG8Ba-d are shown. Note that for the track that contains a point, NG8B, we can find four unique ways to traverse it and therefore get four segments, NG8Ba-d.

A route may cross a number of tracks. The number of tracks very much depends on the operational requirements of the railway. A route is in use when one of its track sections is occupied by a train. As the train proceeds into the route, tracks are progressively occupied and subsequently unoccupied, and in so doing progressively release parts of the route behind the train for other trains to use. Thus a route may have various *stages of usage* depending on the number of tracks. This is modelled by an additional state variable called *route_usage*.

For the operation of a real interlocking, route and point settings are requested by the signaller. This is modelled by an input variable *request*. When model checking, the value of this input variable is set arbitrarily at each step. Of course, this model includes quite "unconventional" behaviour of a signaller, since every possibility is investigated and no assumptions have been made on the behaviour of the signaller. It is reasonable to proceed this way since the control table has to guarantee safe operation in every scenario.

We are able to show by exhaustive testing that any element missing from the data in the route holding section of the control of the control tables, leads to a violation of the safety requirements (that is, a collision or derailment occurs).

2.1.1 A Minimal Model

For our purpose a simple model of trains and their movement is sufficient. We consider trains to behave well, i.e., they do not speed or overrun red signals. They move according to the state of points and signals. We abstract from the speed and length of a train. A train just occupies one segment at a time and can stop instantly. The direction of a train is determined through its position, which is a particular segment that carries information about direction.

Signals can show only two aspects, *stop* and *proceed*. This reduces the specified aspect type but it does prevent us from checking the *aspect sequencing* of the interlocking design. Aspect sequencing ensures that the train driver will see a safe sequence of signal aspects, for example, a yellow aspect before a red one. This mechanism, however, can also be checked statically within the Control Table Editor (CTE) (Robinson et al. 2001).

One part of the control table logic describes the functionality of *approach locking* which is the function that prevents a route that has been set for a train from changing until it is deemed safe to do so. We decided to restrict our checking to a model without approach locking in order to decrease the model's state space and behaviour. This also allowed us to simplify the train movement and signal model as described above. Approach locking is a safety concern, but the corresponding entries in the table can be checked statically by the CTE.

Our model does not distinguish between normal routes and shunt routes. Shunting is a low speed operation in which trains are joined together. In terms of our model, however, this describes a train collision, i.e., a hazard, since we do not consider the speed of a train. For simplicity, the shunting behaviour of trains is currently ignored. This can be justified in so far that shunting does not provide a high safety concern due to the low speed that is involved. Shunting is certainly a hazardous operation for those directly involved in the coupling and uncoupling of items of rolling stock i.e. a significant workplace hazard. It is not however considered significantly hazardous in the railway signalling context, as the low speeds involved should allow trains to stop short of any obstruction, thus causing either none or minimal damage.

2.1.2 An improved Initialisation

We also improve the initialisation of our model. When setting the route-usage initially to the lowest values, the model checking process reveals that the first few iterations are used only to increase the value of the route-usage. To avoid these iterations, we initially set route-usage of each route to its maximal value.

A less restrictive initialisation that leaves values unspecified where possible can also help to reduce the checking time due to the fact that the internal representation of the initial states becomes smaller (see also (Huber & King 2002)). In our application we can leave out the initialisation of the points setting.

2.2 Consequences for the Verification Task

Reducing the model of the interlocking design comes at a cost and carries two consequences. Firstly, the model and its behaviour is less intuitive for railway signal interlocking designers. The counter-examples that are output by the NuSMV tool, although revealing real errors in the control table, show in some cases unexpected or unusual behaviour for the trains due to our simplified model of train movement. In that sense, the model checking approach is very different to testing using simulation that aims at realistic scenarios. Our approach is not inclined to do that but rather to check that the entries in the control tables prevent train collision and derailment.

This is a problem that requires resolution in order to achieve acceptance of the tool support by interlocking designers. We propose to provide the user with an *interpretation* of the counter-examples produced. In most cases, the necessary information, on what the cause of the problem and where the hazard is, can be automatically derived from the counter-example. This enables us to generate an interpretation that points the user directly to the right place in the control table where an entry is missing or flawed, without inspection of the counter-example. We are currently discussing and testing this approach with practitioners from QR.

Secondly, the scope of the verification task is reduced. As already discussed in Section 2.1.1, certain parts of the control tables cannot be checked using our simplified model. In some cases, e.g., approach locking and aspect sequencing, it seems reasonable and more efficient to check those parts using other approaches, e.g., doing static checks using the CTE. In other cases, e.g., shunt routes, the benefit of including checks on those entries does not outweigh the benefit of a more efficient model checking process because they do not carry a significant safety concern.

However, there are issues that we want to include into our model in the future, like the notion of *overlaps* and *level crossings*. Overlaps are tracks beyond a signal and are introduced as a safety buffer for trains that overrun a red signal. Since the trains in our model always stop at a red signal, missing overlaps in the control table cannot be detected in our current approach. Moreover, including the concept of overlaps into our model would also allow us to check for certain liveness conditions on setting signals and routes.

Level crossings also carry a safety concern. They are not present in every area but when they are, the corresponding part of the control table should be checked. Future work will be to include necessary concepts, such as gates and gate movement, into the model.

All the changes on our model are thoroughly discussed with our industry partners from QR. The changes and their impact are well documented, especially the scope of the verification that is provided by the model checking process.

2.3 The Requirements

Since our model comprises a model of moving trains, the requirements on an interlocking design are generic. Rather than expressing, for example, a possible train collision in terms of routes, signals and points, we can state this in terms of trains that use the tracks according to the control table entries.

We check the following safety hazards:

- collisions between trains travelling on the same track and in the same direction
- collisions between trains travelling on the same track but in different directions
- derailments caused by points moving underneath a train
- derailments when a train crosses incorrectly set trailing points
- trains passing signals with routes set in the opposite direction.

For checking collisions on trains, we obviously need a model with at least two trains. However, a careful analysis of our approach shows that no more than two trains are necessary to find all possible errors in the control tables. Derailment and trains running into wrongly set routes, on the other hand, can be checked using one train only. Hence, we run different checks with different models: two-train models and one-train models, of which the latter run significantly faster.

We translate the safety hazards into requirements formalised in CTL, e.g., it is always the case that the position of train $tr1$, $pos(tr1)$, is different to the position of train $tr2$, $pos(tr2)$. In CTL syntax (note that **AG** models *always, in every state*):

$$\mathbf{AG}\ (pos(tr1) \neq pos(tr2))$$

To check derailment caused by points moving underneath a train we want to check that whenever a train tr is on a track section with a point p (i.e., $pos(tr) = homeTrack(p)$) it should not be possible to move point p, i.e., to change its setting $pointset(p)$. Using CTL this can be formalised as follows:

$$\begin{array}{l} \forall p \in Points, \forall val \in dom(pointset): \\ \mathbf{AG}\ (pos(tr) = homeTrack(p) \wedge pointset(p) = val \\ \quad \rightarrow \mathbf{AX}\ (pointset(p) = val)) \end{array}$$

The quantification on points p and values val have to be unfolded: val ranges over $\{setN, setR\}$ and the set of points $Points$ is specific for the verification area under investigation. (Note that **AX** models *always in the next state*).

All other requirements can be formalised in CTL in a similar fashion. However, a close inspection reveals that all CTL requirements in our model can also be specified as simple *invariants*. NuSMV not only supports model checking for CTL formulas but also for simple invariant checking. Since the algorithms for the latter are much more efficient, the use of invariants over temporal logic where possible is preferable.

If a CTL formula contains only the temporal operators **AG** then this formula is equivalent to an invariant (leaving out the temporal operators). In our case the formula on checking derailment due to moving points (as shown above) can also be stated as

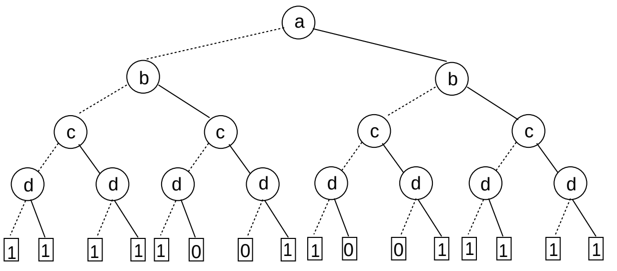

Figure 2: OBDD for $f = (a \leftrightarrow b) \vee (c \leftrightarrow d)$ with ordering $a < b < c < d$

invariant if we exploit the knowledge from our interlocking model: A point only moves if certain conditions are satisfied, i.e., if the guard $pointNGuard$, for setting a point normal (to value $setN$), or the guard $pointRGuard$, for setting a point reverse (to value $setR$), is true. The following invariant is equivalent to the CTL formula above:

$$\begin{array}{l} \forall p \in Points: \\ pos(tr) \neq homeTrack(p)\ \vee \\ \quad \neg((pointset(p) = setN \wedge pointRGuard(p)) \\ \qquad \vee (pointset(p) = setR \wedge pointNGuard(p))) \end{array}$$

Again, the quantification on point p has to be unfolded and the parameters $pointRGuard(p)$ and $pointNGuard(p)$ replaced according to the verification area under investigation. This can be done automatically.

3 The Model Checking Process

Our model checking process is based on a technique called *symbolic model checking*. Symbolic model checking uses *ordered binary decision diagrams* (OBDDs) as a data structure for the internal representation of the model and the temporal logic formula to be checked. OBDDs are a canonic representation for boolean formulas. They can be reduced into *reduced* OBDDs (ROBDDS). ROBDDs provide for most functions a more concise representation than other normal forms (e.g., KNF and DNF) (Bryant 1986). Very efficient algorithms for building and combining ROBDDs are available (Somenzi 1998).

3.1 Variable Ordering of ROBDDs

The possibilities for reducing an OBDD depend on the chosen *ordering* of variables. Figure 2 shows the OBDD for the boolean function $f = (a \leftrightarrow b) \vee (c \leftrightarrow d)$.[2] Nodes of the graph are labelled with the variable names occurring in f. Nodes on each level are labelled with the same variable, i.e. the graph is ordered. A dotted edge from a node marks the evaluation to 0 (or `false`) of the variable the node is labelled with. A solid edge marks its evaluation to 1 (or `true`). The leaves of the graph are labelled with 0s and 1s indicating the evaluation of the formula f depending on the evaluation of the variables as represented by the path in the graph that leads to the leaf.

In the OBDD shown in Figure 2 the variables are ordered according to their appearance in the formula, namely $a < b < c < d$. This ordering is reflected in the graph through the levels on which a variable appears as a node label. The reduction algorithm for OBDDs allows us to eliminate *redundant tests on a variable*, *isomorphic subgraphs*, and *leaf nodes with*

[2]Note the the symbols in the formula read as *if and only if* ($\leftrightarrow$), and *or* ($\vee$).

the same label (and redirecting remaining edges accordingly). The amount of reduction that can be applied is obviously essential for the resulting size of the ROBDD: the more we can reduce the better. In our example in the figure, we can find two isomorphic subgraphs as well as several redundant tests on variables. For instance, if variable a and variable b evaluates to 0, then we know that f evaluates to 1; if both variables evaluate to 1, f evaluates to 1 too. In these two cases we do not have to test the evaluations of variables c and d.

The OBDD in Figure 3 shows a different variable ordering: we evaluate variable c before b. This graph shows a different pattern of subgraphs and leaf nodes. It allows for less reduction. In Figure 4 we show the reduced OBDDs for both orderings. As can be seen the resulting ROBDD for ordering $a < b < c < d$ is significantly smaller (given the fact that we are looking at a very small example). It has six (non-leaf) nodes instead of nine. This number of nodes determines the complexity of the algorithms used when model checking.

3.2 Application specific Variable Ordering

The size of the OBDDs influences the time taken for model checking and the memory usage. In general, finding an optimal ordering for the variables is infeasible (Clarke et al. 2000).

The insight into the issue of variable orderings can be exploited for generating variable orderings automatically. According to the rules for building and reducing an OBDD as described above, the following principles can be observed: When ordering the variables that occur in a formula, it is beneficial to

- group variables together that are closely interrelated; often the *locality* within the formula is characteristic for close interrelation between variables;
- place groups of variables, that determine the overall value of the formula, at the top of the ordering.

As a default, the NuSMV tool generates a variable ordering according to the order of appearance of variables within the SMV code. We call this the *default ordering.* The NuSMV tool also has a user input option to generate an ordering and optimise it during the run of a model checking process, referred to as *dynamic re-ordering* (Cavada, Cimatti, Olivetti, Pistore & Roveri 2001). However, those automated orderings did not prove to be successful for our application (see results below).

Therefore, we additionally use knowledge from our application domain, namely interlocking design, to propose alternative ordering strategies. These strategies are not based on the order of appearance of the variables in the SMV code but rather on the information provided through the track layout and the control tables.

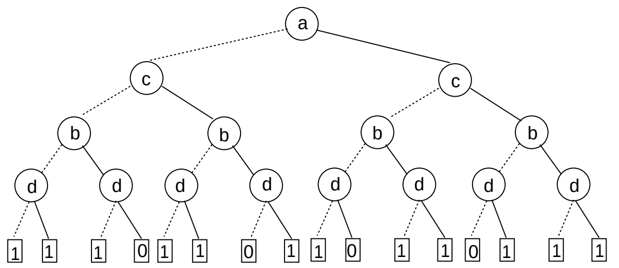

Figure 3: OBDD for f with ordering $a < c < b < d$

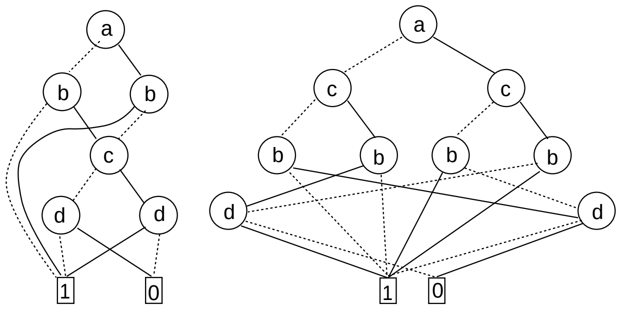

Figure 4: Two ROBDDs for $(a \leftrightarrow b) \vee (c \leftrightarrow d)$ with different orderings. Left: $a < b < c < d$, Right: $a < c < b < d$

a.) **Geographical ordering**: For a model of a specific verification area we group variables according to the locality within the track layout. Signalling equipment, which determines the variables of the OBDDs, is placed together in the ordering if it is *geographically* close. That is, we collect the signalling equipment occurring in the track layout from left to right and place them in the order of their position. We call this ordering a *geographical ordering.*

b.) **Causal ordering**: We group the variables for a specific verification area according to *causal dependencies* between the variables. More specifically, we group each point with those routes that cross the point in a facing direction and with the signals from which those routes are entered. Routes that do not cross points in a facing direction are grouped with routes that oppose them. This results in groups of variables, in which all members determine the state of all other members. The position of a group of entities within the overall ordering is geographically. We call this ordering the *causal ordering.*

Interestingly, this strategy corresponds with the way mechanical interlocking design used to be done, as QR railway engineers pointed out. To them, our causal ordering strategy seem to be the natural strategy to choose.

The position of the train is related to the behaviour of all signalling entities. In both application-specific orderings we therefore place variables on the train position and train movement at the beginning of the ordering. Input variables to the model, like the *request* of a route or point, are placed in the middle of the variable ordering, preferably between two neighbouring groups of entities. This coincides with the suggestions in (Moon, Hachtel & Somenzi 2000).

We have tested the different strategies for variable orderings on various verification areas using an UltraSPARC II 450 MHz processor with 2GByte of RAM under the operating system Solaris version 8. For example, on a medium-sized verification area (29 routes, 13 signals, 22 tracks, and 9 points) we get a results as shown in Table 1.

ordering strategy	run-time (in hours)	memory usage (in MByte)
default	13.6	1246
geographical	4.5	1105
causal	2.4	732

Table 1: Statistics for medium-sized problem

The dynamic re-ordering was tested on a small verification area only (24 routes, 16 signals, 18 tracks, and 4 points)[3] and the results were discouraging: the model checking process with a dynamic re-ordering runs for 31.9 hours whereas using the causal ordering on the same example reduces the run-time to 40 min.

These results illustrate that the strategy of choosing a variable ordering has significant impact on the applicability of model checking to larger problems.

3.3 Setting the Maximum Cache Size Limit

The NuSMV tool integrates the *Colorado University Decision Diagram* (CUDD) package (Somenzi 1998) which provides a library of efficient algorithms for all BDD operations. The efficient recursive manipulation of BDDs uses a cache to store computed results. This cache provides fast access to BDDs, enables re-usability of graphs and supports an efficient garbage collection if graphs are not used any more. The CUDD package starts by default with a small cache, and increases its size until either no further benefit is achieved, or a limit size is reached. The user can set the initial and the limit value for the cache size. The impact of these figures is twofold. Too small a cache size will lead to a frequent overwriting of useful results. Too large a cache size will lead to a bigger overhead used for garbage collection. The CUDD manual recommends the following: "The optimal parameters depend on the specific application. The default values work reasonably well for a wide spectrum of applications" (Somenzi 1998).

Although this parameter cannot be set as a user option to the NuSMV tool we changed its value within the NuSMV code. Instead of using the default limit for the cache size (104 MByte), we changed this value to 512 MByte.

These experiments were done on a new machine with two Intel 3192 MHz processors, and 4GByte RAM, running Red Hat Enterprise Linux AS release 3. On the medium size verification area, using the causal ordering and a cache size of 104 MByte, the statistics are 2.1 hours run-time and 578 MByte memory usage. Increasing the size of the cache to 512MByte, the run-time reduced to 1.2 hours and the memory usage increased marginally to 596 MByte.

Although the memory usage is slightly increased we gain a significant speed-up in processor run-time. We are currently analysing the optimisation of this value in more detail.

4 Related Work

Model checking has been applied before to the analysis of interlocking systems: Gnesi et. al (Gnesi, Lenzini, Latella, Abbaneo, Amendola & Marmo 2000), Bernardeschi et. al (Bernardeschi, Fantechi, Gnesi & Mongardi 1996), and Cleaveland et. al (Cleaveland, Luettgen & Natarajan 1996), for instance, have addressed the problem of fault-tolerance in interlocking systems. In their work, the checking task is focused on communication issues between components of the system rather than the control logic of the interlocking. The preferred modelling language for formalising the systems are based on process algebras (e.g., *Communicating Sequential Process* (CSP), *Calculus of Communication Systems* (CCS), *Process Meta Language* (PROMELA)). These languages provide suitable features for modelling communication between components.

[3]Although the numbers of routes, signals, tracks and points in our small and medium-size models do not differ much, the number of resulting state variables and values in the SMV model are significantly bigger in the medium-sized model.

The work of Simpson, Woodcock and Davies (Simpson, Woodcock & Davies 1997) describes another approach that uses a process algebra for modelling. The paper describes how the control logic of an interlocking system is modelled using CSP. The refinement checker *Failure Divergence Refinement* (FDR) (For 1996) is used to check the safety properties. However, their model is at a lower level of abstraction than ours. The safety invariants, namely no collision of trains and no derailment, are modelled in terms of the signalling entities such as points, signals, routes, and segments. This formalisation of safety invariants has to be manually derived from the track-layout (in the paper it is not explained how) and, therefore, it is not obvious if a given set of invariants is complete and covers all eventualities.

Closer to our approach are the contributions by Eisner (Eisner 1999) and Huber et. al (Huber & King 2002). Both use a symbolic model checker to analyse the interlocking logic of a given track layout and discuss strategies for optimisation. In both works, however, the model is significantly different from our model.

Eisner starts her analysis with a model given as *Vital Logic Code* (VLC) (essentially of a set of Boolean expressions), to specify railway interlocking software which is then translated into a dialect of the SMV input notation. Therefore, optimising the model is not an issue discussed in the paper. Her optimisations relate to the way in which the safety requirements are formalised in a sub-language of CTL, called AGAX formulas. She shows that the model used has certain general characteristics (called robustness and locality) that render the application particularly suitable to symbolic model checking of AGAX formulas. Although this is generally a very interesting observation, since it allows predictions for other applications too, in our case the requirements are even simpler than AGAX formulas. For our model the requirements can be stated as invariants.

Huber et. al model and check the Geographical Data of a Solid State Interlocking program using NuSMV. That is, their approach for verification is placed at program level rather than design level. Moreover, their model does not contain a model for train position and movement. Consequently, the requirements have to be formalised based on the signalling entities. The paper suggests an automated approach for generating the CTL formulas from the given principles. In this approach the number of requirements to be checked is rather large. The general template for the formulas has to be instantiated for all tracks, and all points, and all routes. To optimise the variable ordering, the paper suggests using the dynamic re-ordering of the NuSMV tool. In our case, however, we were able to significantly improve on this option by using an application specific ordering.

This work is of particular interest because it suggests a number of ways to optimise the model-checking process. In contrast to our work, the input data is not translated into SMV code but rather into BDD structures (circumventing the compilation of the NuSMV tool). This provides more direct access to the BDD structures. Some of the suggestions can be applied to our approach too. For example, the optimisation of initialisation of the model (see Section 2). Other suggestions will be further investigated in our future work (e.g., the potential of splitting the transition relation).

5 Conclusion

This work describes an approach to checking the safety requirements of interlocking designs using a symbolic model checker. In order to minimise the state explosion problem and to improve the performance of the model checker for larger examples, we suggest a number of optimisations. We reduce the model to be checked, where this is possible, without loss of credibility regarding safety issues. We describe a strategy for finding a very good variable ordering based on domain knowledge and we suggest on an improvement of parameter settings of the NuSMV tool and the CUDD package for our specific application.

In future work we will continue to investigate further improvements to the model as well as further optimisations to the settings of the tool's parameters. To automate the overall process, we are aiming to develop an automated generator for variable orderings for specific verification areas, and to provide the user with support for comprehensive counter-example interpretation.

Acknowledgements:

This work has been supported by the ARC Linkage Grant LP0455155. This work benefited greatly from the knowledge and insight into railway interlocking systems provided by George Nikandros, David Barney and David Tombs from QR. We also wish to thank the reviewers for their helpful comments.

References

Bernardeschi, C., Fantechi, A., Gnesi, S. & Mongardi, G. (1996), Proving safety properties for embedded control systems, *in* 'Procs. of Conference on Dependable Computing (EDCC-2)', Vol. xvi+440, Springer-Verlag, pp. 321–332.

Bryant, R. E. (1986), 'Graph-based algorithms for boolean function manipulation', *IEEE Transactions On Computers* **C-35**(8).

Burch, J., Clarke, E., McMillan, K., Dill, D. & Hwang, L. (1992), 'Symbolic model checking 10^{20} states and beyond', *Information and Computation* **98**(2), 142–170.

Cavada, R., Cimatti, A., Olivetti, E., Pistore, M. & Roveri, M. (2001), *NuSMV 2.0 User Manual*, IRST Trento, `http://nusmv.irst.itc.it`.

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.

Clarke, E., Grumberg, O. & Peled, D. (2000), *Model Checking*, MIT Press.

Cleaveland, R., Luettgen, G. & Natarajan, V. (1996), Modeling and verifying distributed systems using priorities: A case study, *in* 'Procs. of Int. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)', Vol. 1055 of *LNCS*, Springer-Verlag, pp. 287–297.

Eisner, C. (1999), Using symbolic model checking to verify the railway stations of Hoorn-Kersenboogerd 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 Computer Science', Vol. B, Elsevier Science Publishers.

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

Gnesi, S., Lenzini, G., Latella, D., Abbaneo, C., Amendola, A. & Marmo, P. (2000), An automatic SPIN validation of a safety critical railway control system, *in* 'Procs. of IEEE Conference on Dependable Systems and Networks', IEEE Computer Society Press, pp. 119–124.

Huber, M. & King, S. (2002), Towards an integrated model checker for railway signalling data, *in* L.-H. Eriksson & P. Lindsay, eds, 'Proc. on Formal Methods Europe (FME'2002)', Vol. 2391, Springer-Verlag, pp. 204–223.

McMillan, K. (1993), *Symbolic Model Checking*, Kluwer Academic Publishers.

Moon, I.-H., Hachtel, G. & Somenzi, F. (2000), Border-block triangular form and conjunction schedule in image computation, *in* W. A. Hunt & S. D. Johnson, eds, 'Int. Conference on Formal Methods in Computer Aided Design (FMCAD 2000)', Vol. 1954 of *LNCS*, Springer-Verlag, pp. 73–90.

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)'.

SAOS Standards, (1999), 'Signalling principles - Brisbane suburban area'.

Simpson, A., Woodcock, J. & Davies, J. (1997), The mechanical verification of solid state interlocking geographic data, *in* L. Groves & S. Reeves, eds, 'Proc. of Formal Methods Pacific (FMP'97)', Discrete Mathematics and Theoretical Computer Science Series, Springer-Verlag, pp. 223–243.

Somenzi, F. (1998), *CU Decision Diagram Package: Release 2.3.0*, Department of Electrical and Computer Engineering, University of Colorado at Boulder, `http://vlsi.colorado.edu/~fabio/{CUDD}`.

Winter, K. & Robinson, N. J. (2003), Modelling large railway interlockings and model checking small ones, *in* M. Oudshoorn, ed., 'Proc. of Australasian Computer Science Conference (ACSC2003)'.