Optimising Ordering Strategies for Symbolic Model Checking of Railway Interlockings

Kirsten Winter

School of Information Technology and Electrical Engineering, The University of Queensland, St.Lucia, QLD 4072, Australia

Abstract. Interlockings implement Railway Signalling Principles which ensure the safe movements of trains along a track system. They are safety critical systems which require a thorough analysis. We are aiming at supporting the safety analysis by automated tools, namely model checkers.
Model checking provides a full state space exploration and is thus intrinsically limited in the problem's state space. Current research focuses on extending these limits and pushing the boundaries. In our work we investigate possible optimisations for symbolic model checking. Symbolic model checkers exploit a compact representation of the model using Binary Decision Diagram. These structures provide a canonical representation which allows for reductions. The compactness of this data structure and possible reductions are dependent on two orderings: the ordering of variables and the ordering in which sub-structures are manipulated. This paper reports on findings of how a near to optimal ordering can be generated for the domain of interlocking verification.

1 Introduction

Railway signalling interlockings are safety critical systems. Therefore special attention has to be given to the correctness of the design and the implementation of an interlocking system. The development of such systems is very labour intensive and prone to error. It requires specialised skills. Moreover, possible errors in the design are detected very late in the design process. To mitigate these problems Queensland Rail (QR), the major railway operator and owner in Queensland, Australia, intended to support its design process by a specialised tool set called the Signalling Design Toolset (SDT) [1]. Parts of this toolset were intended for supporting the verification task.

In this paper we summarise our findings on how to formally model a functional specification of an interlocking system, the control table, and how to optimise symbolic model checking of these interlocking models. Our results render the approach feasible for industrial practise.

Others have applied model checking to analyse railway interlocking systems: Gnesi et al. [2], Bernadeschi et al. [3], and Cleaveland et al. [4], for instance, have analysed fault tolerance of interlocking systems. The main focus in their work were communication aspects between components rather than the control logic as in our work.

The first approaches on applying model checking to verify railway interlocking systems represented as control tables or equations were reported by Groote et al. [5], using μCRL and its tools, and Eisner [6, 7] using SMV. Also Simpson et al. [8], and Huber et al. [9] are focusing on the control logic similar to our approach. In particular, the approach by Eisner and Huber et al. is very close to ours as they also use a symbolic model checker. The interlocking systems, however, are modelled on a lower level of abstraction. This leads to significantly different models and in particular a more complex model of the requirements which in our case can be simply expressed as train collision and train derailment.

In [10] Ferrari et al. attempt a comparison of different model checkers when applied to interlocking control tables. Although the work is very interesting it is lacking to take into account the potential of optimising symbolic model checking as it has been proposed in this work.

While most existing work on signalling design verification targets the later interlocking design phase, the focus of our approach was to enable verification of the interlocking design at an early stage, when specified as control tables. The results presented here consolidate work published in [11, 12] and [13, 14].

2 The Context

The research this paper reports on has been conducted in the context of the Signalling Design Tools projects (SigTools), a collaboration of Queensland Rail (QR), the Software Verification Research Centre (SVRC), and later the University of Queensland. Aim of these projects was to design tool support for the creation and verification of control tables[1, 15].

Control Tables provide the functional specification for railway signalling interlockings and contain the key safety requirements. They act as (1) an agreement between the railway administration and the train operators on when moves will be permitted on a track layout, (2) a specification for designing the interlocking itself, and (3) a test specification for use by testers.

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					

Fig. 1. Control Table for a small verification area (as shown in Figure 2); depicted is the row for the route from signal HE2 to NG8

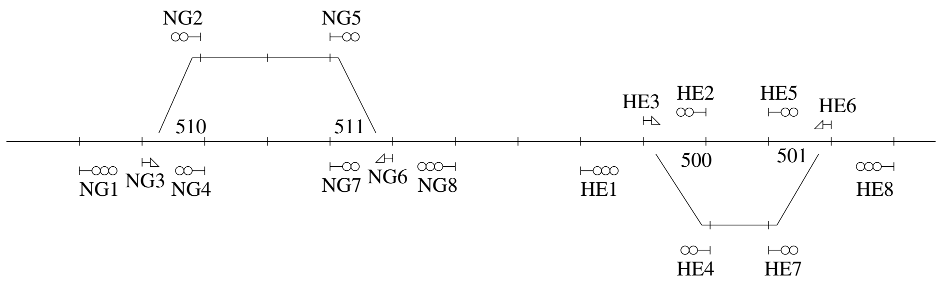

Fig. 2. Track Layout of a small verification area

A control table is a structured, tabular presentation of the rules governing route setting on a railway track layout. An example of a QR (signal) control table is shown in Figure 1. A *route*, a sequence of tracks from an entry signal to an exit signal, is a key concept in the table. One row of the table indicates the conditions needed to "set" a single route (i.e., reserve it for a train to safely use it).

The control table is specific to a *track layout* which represents the local arrangement of the railway equipment on a selected part of the railway network while abstracting from the actual length of the tracks and distances between the signalling equipment. In our context we refer to the area covered by the track layout as the *verification area*. Figure 2 shows the track layout for the above control table. It contains the signals HE2, NG8, etc., the points 500, 501, etc., and tracks which are not named in the figure but indicated as sections on the railway.

2.1 The Signalling Design Tools Project

Intention of the SigTools project was to design a tool suite to enable generation of (parts of) signalling control tables, the editing of generated control tables, and the verification of control tables against safety principles for railway signalling, the Signalling Design Tool (SDT) suite. Figure 3 gives an overview of the architecture.

It comprises a *Track Layout Editor* [16, 17] to manually edit track layouts, a *Control Table Generator* [18] for automatically generating all entries of the table that can be inferred from the layout, a *Control Table Editor* [19] for manually editing control tables, and a *Counter-example Interpreter* [20] which automatically interprets counter-examples output by the verification tool, as well as the *Control Table Verification* itself which is the focus of the following sections.

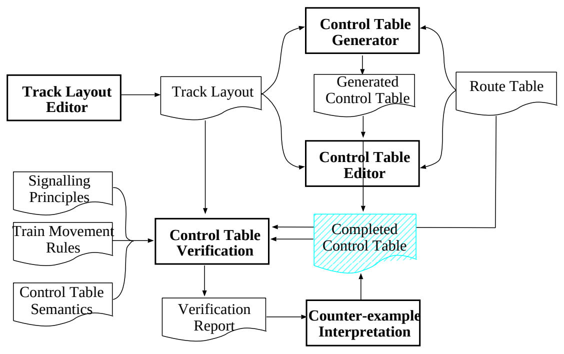

Fig. 3: Architecture of the Signalling Design Tools

2.2 Modelling Control Tables

We target an automated verification of the completed control tables utilising the model checker NuSMV [21]. This requires to provide a formal model of the control table. To this end we designed a *generic* formal model that captures the control table semantics using ASMs [22] as a vehicle. For each particular control table to be verified the generic model is to be instantiated with the data from the accompanying track-layout, the route table, and the control table itself. This instance constitutes the formal model of the control table to be checked. It is automatically translated into the model checker's input notation, here the NuSMV language [23]. The ASM model serves as a useful tool to communicate, formalise and document (our understanding of) the control table semantics. More detail on the model can be found in [11, 24].

At later stages of the project we omitted the "intermediate" ASM model as we derived a generic NuSMV model from the NuSMV code that was generated by the ASM-to-NuSMV translator. The generic NuSMV model is to be instantiated similarly to the ASM model used earlier. The instantiation process has been automated within the SDT suite [25, 1]. In the following we sketch the simplifying assumptions of our control table model as these have an impact on the scope of our analysis.

Train Movement. Unlike other approaches our model also includes (one or two) trains moving along the tracks. As a consequence, the safety requirements become generic and very easy to validate. They are modelled in terms of train *collision* and *derailment*. It could be shown through tests that even a very simplistic model of train movement suffices to show missing entries in the control table. The train data is limited to an identifier, which route the train is on, and which track it is occupying. The simplifications are carried by the following assumptions: (a) Trains move at a constant speed or stop (i.e., we do not model the speed of a train or its braking capacity). (b) We assume that trains always stop at red signals (excluding possible overruns). (c) Trains move according to the conditions given by the state of points and signals from track to track. (d) Trains can stop at any time. (e) The direction of a train is determined through its position, which is a particular track segment each of which carry direction information.

It suffices to consider only two trains in the system to check for collision, and only one train to detect possible derailment. The reasoning for this simplification, which is in agreement with the railway engineers, is based on the fact that the more trains are running through a particular verification area the more the movement of other trains is restricted. Moreover, trains can appear on the tracks of the verification area in an arbitrary fashion in our model. Therefore, every possible combination of two potentially colliding trains is investigated. Considering only two trains at maximum limits the additional complexity that stems from adding trains to the model to a tolerable level.

Aspect Sequencing. Signals can show only two aspects, stop and proceed. This reduces the specified aspect type (two values instead of three) but it does prevent us from checking the aspect sequencing of the interlocking design. Aspect sequencing ensures that the train driver will see a yellow aspect before a red one. This mechanism, however, can also be checked statically within the Control Table Editor [1].

Approach Locking. One part of the control table logic describes the functionality of approach locking. Approach locking 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 verification to a smaller problem 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 Control Table Editor.

Shunt Routes. Our initial model did not distinguish between normal routes and shunt routes. Shunting is a low speed operation in which trains are joined together. In terms of our initial model, however, the scenario of shunting equals a train collision since we do not consider the speed of a train. For simplicity, the shunting behaviour of trains was then ignored. As justification we used the fact that shunting does not provide a high risk safety concern due to the low speed that is involved. At a later stage of the project however it was found that the models can be extended to include shunt routes as distinguished instances. This required to change the some of the rules for train movement as well as the specification of the safety requirements for train collision.

Overlaps. Overlaps are tracks behind 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 (see above), missing overlaps in the control table cannot be detected in our initial model. Moreover, including the concept of overlaps into our model would also allow us to check for certain liveness conditions on setting signals and routes. A later version of the model included the model of overlapping tracks and it could be shown that the modification extended the scope of the verification sufficiently.

Level Crossings. 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. These checks require to introduce new concepts, such as gates and gate movement, as well as the notion of time into the model. This can be supported by either the use a modelling framework that supports time (e.g., timed automaton and UPPAAL [26] or timed CSP [27]), or to integrate an explicit timer variable into a "standard" model like ours. The latter approach has been proposed in [28].

Keeping the model of the interlocking design at a more abstract and more simplified level reduces the complexity of the checking process. However, it comes at a cost of retrieving counterexamples that are less intuitive for the railway engineer for whom the formal model is an internal representation he/she is not familiar with. Although revealing real errors in the control table the given path might show unusual behaviour for the trains due to our simplified model of train movement. To alleviate this problem the project designed a Counter-example Interpretation which provides the user with hints as to which entry in the table is missing [20].

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

3 Symbolic Model Checking

Simplifying the model as described in the previous section helped to scale-up the verification to some extend but the improvement was not sufficient to target medium- or large-sized models. Therefore we aimed at optimising the model checking process itself.

3.1 Variable Ordering

Central to symbolic model checking is the idea of internally representing the model (i.e., its states and state transitions) using Bryant's *Binary Decision Diagrams* (BDDs) [29]. BDDs are a graph structure that allows the canonical representation of Boolean functions. In most cases a BDD representation is substantially more compact than other canonical representations in normal form, and it furthermore allows for signification reductions avoiding the representation of duplicate information.

A BDD consists of a set of nodes that are labelled with a variable. Each node is linked via two directed edges to its successor nodes. These edges represent the evaluation of the variable to *false* and *true*, respectively. The leaf nodes of the BDD are labelled with the truth values. If the labels of the nodes in the BDD are ordered then we call the graph an *ordered* BDD (OBDD).

The possible reduction of an OBDD depends on its structure which is determined by the ordering of its node variables. An OBDD for the same function but with the changed variable ordering has a different shape and consequently reduces differently.

It is infeasible to compute an optimal variable ordering in general, however, much research effort has been focused on heuristics for finding a good variable ordering automatically. These heuristics are based on dependencies between variables and early evaluation of the represented formula. When a formula represents a transition that updates one variable we refer to it as *single transition*. It will reference variables on which the transition depends. We call these variables *support variables* of the transition. Variables that are referenced by all transitions are called *global variables*. The following guidelines help finding a good ordering:

1. Declare closely related variables together. In the variable ordering, each variable should be close to the support variable of its transition [30, 31].
2. For each transition, having the support variables closer to the top of the order than the variable being transformed, gives the smallest ROBDD [30].
3. Place global variables at the top of the order [31].

3.2 Transition Orderings

Due to the nature of the application our model is similar to synchronous hardware circuits. The transition relation is described as the conjunct of single transitions of which each describes the update of one state variable. That is, the value for a state variable v can be given as $v = f(v_1, \dots v_n)$ where f is a function that captures the logical dependencies between variable v and the support variables $v_1, \dots, v_n$. All transitions are fired simultaneously which leads to a *next state* in which some variables have changed their values while others have not. Therefore, we can use results that originated in the domain of synchronous circuit verification, in particular [32].

If we assume a set V of n state variables, $v_1, \dots v_n$, we denote a state of the model as $S(v_1, \dots v_n)$ and abbreviate this as $S(V)$. The transition relation, modelling the transition from current state to next state, is captured by a function N which depends on variables in V as well as their primed counterparts, V', capturing the evaluation in the next state. The function $N(v_1, \dots, v_n, v'_1, \dots, v'_n)$ is abbreviated as $N(V, V')$.

In symbolic model checking the state space is explored by iteratively applying the transition relation to the state. This is done in a forward fashion starting with the initial state and is called *image computation*. The operation on ROBDDs used for image computation is called *relational product*. For synchronous systems (as in our case) it is defined as
$S'(v'_1, \ldots, v'_n) = \exists v_1 \in V[\ldots \exists v_n \in V[S(v_1, \ldots, v_n) \wedge N(v_1, \ldots, v_n, v'_1, \ldots, v'_n)] \ldots]$
We abbreviate this formula using the notion from [32] as follows

$$S'(V') = \exists_{v \in V}[S(V) \wedge N(V, V')] \tag{1}$$

where V' is the set of primed state variables and $S'(V')$ describes the set of next states reachable from $S(V)$ via one transition step.

The transition relation $N(V, V')$ can be applied to the state as one big transition or it can be envisaged as a conjunct of smaller *partitions* of N, $N_i(V, V')$, where each partition is either a single transition (that updates one variable) or a conjunct of several single transitions, referred to as *cluster* in the following. In practise each $N_i(V, V')$ can often be represented by a small BDD whereas the whole of $N(V, V)$ becomes very large. The aim is to compute the image without building the whole of $N(V, V')$ by using the following formula:

$$S'(V') = \exists_{v \in V}[S(V) \wedge N_0(V, V') \wedge \ldots \wedge N_{n-1}(V, V')]. \tag{2}$$

Unfortunately, existential quantification (e.g., $\exists_{v \in V}$ in the formula above) does not distribute over conjunction and we cannot simply split up the operation into single steps. Burch et al. in [32], however, developed a method to overcome this problem. It is based on two observations. Firstly, our model of interlockings exhibits locality (in a similar fashion to circuits), that is, most single transitions will depend on only a small number of variables in V and V'. Secondly, sub-formulas from the relational product can be moved out of the scope of the quantification if they do not depend on the variables being quantified. Therefore, it is beneficial to conjoin the $N_i(V, V')$ with $S(V)$ one at a time moving out those variables from the scope of the quantification that none of the remaining partitions depends on. To do so we want to *order* the partitions $N_i(V, V')$ in such a way that the number of variables that can be eliminated early is maximised. This will lead to smaller intermediate results of the image computation.

Assume the chosen order of clusters is given by the permutation p that permutes the indices $\{0, \ldots, n-1\}$. That is, cluster $N_{p(i)}(V, V')$ will be applied in the ith step of the image computation. Let $D_{p(i)}$ be the variables that $N_{p(i)}(V, V')$ depends on and E_i the set of variables that can be eliminated after the ith step. That is, E_i is the subset of $D_{p(i)}$ that is not contained in any of the other dependency sets. Then $S'(V')$ can be computed in a number of steps each eliminating the variables in E_i when building the corresponding *intermediate products* $S_{i+1}(V, V')$:

$$\begin{array}{rcl} S_1(V, V') & = & \exists_{v \in E_0}[S(V) \wedge N_{p(0)}(V, V')] \\ S_2(V, V') & = & \exists_{v \in E_1}[S_1(V, V') \wedge N_{p(1)}(V, V')] \\ & & \ldots \\ S'(V') & = & \exists_{v \in E_{n-1}}[S_{n-1}(V, V') \wedge N_{p(n-1)}(V, V')] \end{array} \tag{3}$$

The chosen order p has an impact on how early variables can be quantified out and therewith affects the size of the BDDs constructed. The aim is to group those transitions together into one cluster that have the same support variables. Selective grouping of transitions into clusters, and the order p of application of the clusters leads to smaller and fewer intermediate products that are manipulated faster [33]. The following heuristics can be proposed: Transitions that are supported by the maximal number of variables should be grouped together in a cluster and applied first. Subsequent transitions that are supported by fewer variables should be grouped into clusters so that as many of their support variables as possible do not support transitions in clusters yet to be applied. This enables some of the support variables to be quantified out progressively from the intermediate products giving smaller intermediate products.

If transitions do not naturally fall into clear-cut divisions, the grouping of transitions within clusters and the order of application of the clusters should be such that early elimination of support variables is maximised. We will see in the next section how a good ordering for the transitions can be generated for the domain of railway interlockings.

4 Optimised Orderings for Railway Interlockings

Checking medium-sized control tables utilising the standard user options provided by NuSMV [23] would in many cases lead to memory overflow and a non-acceptable runtime of the process. The approach needed to be taylored for the application domain.

Three optimisation were taken into consideration: (1) Generating the variable ordering on the basis of domain knowledge. (2) Computing an ordering in which the transition clusters are to be applied to maximise the early quantification of support variables. (3) Determining the threshold size for clusters based on experimental results. The following subsection report on our findings.

The domain knowledge on which our approach is based stems from the following characterisation of the domain data. Variables in our model can be divided into three groups: global, local and input variables. Similarly, single transitions can be characterised as global transitions if they update a global variable or local transitions if they update local variables.

Global Variables represent train attributes like the current position (given in terms of a track) and the currently used route. In the SMV code this is modelled by four variables of enumerated type. Typically 30 -130 different values, depending on the number of tracks and routes in the interlocking, are required. The larger and more complex the verification area, the larger becomes the set of Booleans necessary to represent the values. Typically five to seven Booleans are required for each attribute in the implementation (see [30] for details on implementing enumerated types efficiently).

Local Variables model the lie of the individual points, the current aspect of the signals, and the lock and usage of routes. Mostly, this information can be represented by simple Booleans (e.g., points are set normal or reverse, signals are set proceed or stop, routes are locked normal or reverse), only the route usage is encoded by a typically small enumerated set.

Input Variables represent signalling and train control commands (i.e., requests). They are not controlled by the interlocking but change their values randomly (to capture every possible behaviour). This can be modelled using a number of simple Booleans variables and one variable of enumerated type. The number of enumerated values again depends on the size of the verification area (number of routes, signals, etc.). The implementation of the enumerated input variable typically requires 5 to 7 Booleans.

Transitions in our context are modelled using the `next` operator of the SMV input language [34, 23]. For each variable the evaluation in the next state is modelled depending on the previous values of the support variables. The size of the ROBDD representing the transition for each variable depends on the variable ordering (see Section 3.1).

Global transitions are supported by all the variables. Local transitions depend on a limited number of variables. Specifically they are supported by the global variables, the input variables and some of the local variables, e.g., only the occupation of nearby tracks and the input command variable are relevant to the movement of a point.

Generally, the transitions are such that if $\{v_1, v_2, v_3\}$ is the set of support variables for transition $N_{v_1}(V_1, v_1')$ then the set of support variables for transition $N_{v_3}(V_2, v_3')$ is likely to include v_1. That is, there is a cross-dependency between transitions. An analysis of the dependencies between all the variables using a dependency matrix (see [35]) resulted in a very dense matrix. This made the application of the standard heuristics which are based on the characteristics of a less dense dependency matrix unsuccessful in finding a good ordering.

Size of the Verification Area. An increase in the complexity of models (more signals, points and tracks), introduces more local variables, and maintains the same number of global and input variables but adds more values to the enumerated types. Adding more values to the enumerated types does not impact significantly on the number of Booleans used to implement them but does impact on the size of the ROBDD used to distinguish particular values of the variables.

Let $\mathit{Var} = \{v_1, \ldots, v_{m+n+p}\}$ be the set of state variables in our model with $\{g_1, \ldots, g_m\} \subset \mathit{Var}$ the set of global variables, $\{l_1, \ldots, l_n\} \subset \mathit{Var}$ the set of local variables and $\{req_1, \ldots, req_p\} \subset \mathit{Var}$ the set of input variables. Let $N_{v_i}(V, v_i')$, $1 \leq i \leq (m + n)$, be the transitions, local or global, that changes (local or global) variable v_i dependent on the support variables $V \subseteq Var$.

4.1 Optimising the Variable Ordering

We have implemented an algorithm that performs an ordering of local, global and input variables using the available data from the track layout. The strategy, and the reasoning for this strategy, on which the algorithm is based are explained in detail in the following.

Local Variables Single transitions which model the update of signals, points, etc. depend on the support of signalling equipment that is in the close vicinity on the track-layout graph. The dependencies between the state variables are therefore related to the *geographical* arrangement that can be read from the track layout (see, for example, Figure 2).

Mechanical interlocking design suggests further considerations for this ordering strategy. Building the relays for the mechanical interlockings starts usually with points, followed by the signals whose routes crossed those points. Typically each signal is associated with routes from that signal. This leads to a strategy which associates the signals and routes with a particular point. The associated entities form a *group*.

However, there are several different ways of associating signals and points if a route comprises several in-route points. It was noted that moving a signal and its routes from a group with its first in-route *trailing* point to a group with its first in-route *facing* point made a significant difference to the model checking time and the memory used.

These consideration led to the following first ordering heuristics:

a) A signal and its routes are associated with the first facing point in the route or
b) with the first trailing point in-route if this is the only point.

While variables within groups are related by the transition relations, there is significant cross dependencies between the groups e.g. routes are related to routes that oppose them and the opposing routes are likely to be associated with different points. This leads us to the second ordering heuristics:

c) The groups are best ordered according to their arrangement on the track layout rather than in a random order.

The heuristics a) to c) form the basis of our ordering strategy, called *geographic order* which defines a permutation γ on the *local* variables. The layout is viewed as a grid and the signals and points are read in order from left to right. Where signals or points are in the same vertical grid, elements are ordered from top to bottom.

Ordering the local variables $\{l_1, \ldots, l_n\} \subset \mathit{Var}$ according to the geographic order following the heuristics above leads to an ordering of local variables of the form $l_{\gamma(1)} < \ldots < l_{\gamma(n)}$. For each local variable $l_{\gamma(j)}$, $1 \leq j \leq n$, the corresponding transition $N_{l_{\gamma(j)}}(V, l'_{\gamma(j)})$ then depends on local variables in reasonably close proximity to $l_{\gamma(j)}$ in the order, e.g., $l_{\gamma(j-1)}$ and $l_{\gamma(j+1)}$, etc.

Global Variables The local transitions also depend on the global variables. Experimentation shows that putting the global variables higher in the variable order than all the local variables gives the smallest local transitions (i.e., those transitions that update local variables) supporting heuristics 3 in Section 3.1. The global transitions for the four global variables of enumerated type depend on all the variables and are large.

Input Variables Placement of the input variables in the variable order is problematic. Input variables are in the support variables for all transitions. When they are placed at the beginning of the order, the ROBDDs representing the transitions $N_{v_i}(V, v'_i)$, $1 \leq i \leq (m + n)$, are smaller than ROBDDs for an order in which the input variables are placed lower in the order. However, this does not necessarily lead to smaller intermediate products. Experimentation has shown that placing the large input variable lower in the order increases the size of the local transitions and the size of the clusters. However, this gives smaller intermediate products and uses less memory overall. There are time and memory efficiency penalties for manipulating large transitions, large clusters, and large intermediate products and for our data, experimentation has shown that the best results are obtained by placing the large input variable about 2/3 down the order. For a detailed discussion see [13].

4.2 Improving the Transition Ordering

NuSMV did not have provision for the user to supply a transition order at the time we started our project. It has its own generic algorithm for estimating the *affinity* of transitions [35] (which describes their degree of similarity) and by default progressively builds clusters based on this affinity. A cluster is closed off when its size reaches a threshold (defined by the user or a default value) which results in evenly sized clusters.

For railway interlockings the dependency matrix on which the affinity is based is very dense. Therefore, computing the affinity between variables by itself does not provide the necessary information to improve efficiency. However, examining the railway interlocking model and its semantics has enabled us to define an order in which transitions can be conjoined and the points in the order at which to cut the conjunctions to form clusters. When these clusters are applied in turn in the image computation, the variables are quantified efficiently from the intermediate product.

We use the same argument of vicinity of symbols on the track layout that is used for finding a good variable ordering for ordering the partitioned transition relation. The global transitions, $N_{g_1}, \ldots, N_{g_m}$, are supported by all the other variables including the input variables. This suggests that global transitions should be applied first. The local transitions, $N_{l_1}, \ldots, N_{l_n}$, depend on global variables and other local variables associated with nearby symbols in the track layout. A transition order that reflects the geographic order of variables γ for the local transitions results in a permutation $N_{l_{\gamma(1)}} < \ldots < N_{l_{\gamma(n)}}$ of local transitions which then can be progressively grouped into clusters with some overlap of support variables.

Eliminating variables that are at the leaf end of an ROBDD (lowest in the variable order) favours BDD reduction and results in smaller diagrams than removing variables from the middle or root end of the diagram (higher in the variable order). Therefore, we order the local transition in such a way that transitions for variables of lower order will be applied first. If the local variables indexed progressively by $\gamma(1), \ldots, \gamma(n)$ using the geographic order γ then the aim is that the transition for the $\gamma(n)$th variable is applied before the transition for the $(\gamma(n-1))$th variable to facilitate early elimination of the $\gamma(n)$th variable. While the $\gamma(n)$th variable may not be eliminated immediately after application of its transition, it should be soon after since all transitions using it will be within close range. This leads to an ordering of local transitions that is reverse to the ordering of local variables as introduced in Section 4.1.

In summary, a good order of application of transitions is the global transitions followed by the local transitions in the order $\gamma(n)$ to $\gamma(1)$. That is, assuming the NuSMV principle of prepending the cluster list and applying the transitions from the back to the front of the list, a good transition order for railway interlockings is the local transitions in the order $\gamma(1)$ to $\gamma(n)$, followed by the global variable transitions:

$$N_{l_{\gamma(1)}} < \ldots < N_{l_{\gamma(n)}} < N_{g_1} < \ldots < N_{g_m} \tag{4}$$

The NuSMV code was extended so that the user could provide a transition order in terms of an ordered list of the corresponding variables, $l_{\gamma(1)} < \ldots < l_{\gamma(n)} < g_1 < \ldots < g_m$.

4.3 Clustering

Transitions are conjoined in order according to the transition order. Having defined a good transition order that supports the elimination of variables as early as possible, the question becomes where to cut the transition conjunction and form a cluster. If all transitions are in one cluster, no elimination of variables can occur and the ROBDD representing the cluster becomes very large. If the clusters are too small then many intermediate products $S'(V')$ (see equation 3 in Section 3.2) have to be computed. The issue is to find the balance between size and number of clusters and intermediate products.

Using a transition order and the default threshold to form the clusters resulted in between ten and fourteen clusters for our models. This number is too large and we had to re-define the cut-off points for the clusters.

The global transitions are applied first and it is logical to put all of these into the first cluster. After application of the global cluster the *next* values for the global variables can be quantified out.

When clustering the remaining local transitions, which are ordered using the geographical ordering, we used the insight that by referencing the track layout it is possible to nominate where in the transition order the dependencies change. This observation was confirmed by the railway engineers. For example, Figure 2 shows us that variables related to symbols to the right of signal *HE1* will be supported mostly by variables lower in the transition order than the variable for signal *HE1* since we ordered the variables inspecting the track layout from left to right. Similarly, variables related to symbols to the left of signal *HE1* will be mostly supported by variables higher in the variable ordering than the variable for signal *HE1*. Thus, for this verification area the local transitions fall naturally into two clusters at this point. Including the global cluster gives three clusters for this track layout.

From our experimentation it is clear that with a good transition order, few clusters are required. We found that often the models fell naturally into three or four clusters. However, for large models these clusters can become very big and the model checker spent significant time building them. In this case the performance was best using a clustering based on the threshold.

Another way to achieve few clusters is to specify a large threshold. The clusters will not be cut as precisely as before but because the order is good, progressive elimination of variables will occur. This approach is not as efficient as the customised formation of clusters described above, but is a worthwhile improvement on the default threshold used by standard NuSMV. The result (shown in Table 1 in Section 4.4) suggest that this approach is a reasonable alternative as it requires no specialist knowledge of the model or the application domain.

4.4 Experimental Results

We conducted our experiments with real data provided by QR. Table 1 compares our results for three different sized models: The large model consists of 41 routes, 9 points, 19 signals, and 31 track circuits. The medium model comprises 29 routes, 9 points, 13 signals, and 22 tracks. The small model comprises 12 routes, 2 points, 8 signals, and 8 tracks. The experiments were conducted using the options as indicated below.

The figures show that a significant improvement of run-time and memory usage was achieved by choosing a good variable ordering that was based on geographical information from the track layout, i.e., domain knowledge over the dependencies. This result is

	User Options	Time(s)	Memory
Small model	1	4081	655Mb
	2	651	98Mb
	3	124	42Mb
	4	61	29Mb
	5	88	36Mb
Medium model	1	9620	1098Mb
	2	734	114Mb
	3	321	78Mb
	4	152	49Mb
	5	222	63Mb
Large model	1	N/A	N/A
	2	N/A	N/A
	3	68872	3.6Gb
	4	33641	980Mb
	5	29357	1160Mb

Option 1:	using NuSMV defaults for variable and transition orders and clustering
Option 2:	using user-defined variable order with default transition order and clustering
Option 3:	using user-defined variable ordering and user-defined transition orders with default clustering
Option 4:	using user-defined variable ordering, user-defined transition order and clusters selected by user
Option 5:	using user-defined variable order, user-defined transition order and clusters selected by threshold.

Table 1. Comparison of various sized models using the discussed options

not surprising as this correlation is often stated in the literature. Improvements of similar scale could also be achieved by customising the order of transition partitions and by forming the clusters. Both parameters were chosen using the same reasoning as was used for choosing the variable ordering — in our case geographic order of dependencies.

The NuSMV tool (from version 2.4.1) has been extended by the user option `-t <tv_file>` which allows the user to specify an alternative variable ordering to be used for clustering of the transition relation [23]. The grammar of the ordering file is the same as for the variable ordering file.

5 Conclusion

In this paper we provide a strategy for improving efficiency of symbolic model checking when applied in the domain of railway interlockings. Based on domain knowledge we show how to compute optimised variable and transition orderings and report on our findings of how to set the threshold for clusters. The results from our experiments are encouraging and render the approach feasible for use in industrial practise.

For future work it would be interesting to include an *optimised* symbolic model checking approach into a comparative study as it has been done in [10]. Moreover, we would like to apply the proposed strategies to a model on the level of code or a geographical data model of an interlocking system to confirm whether similar improvements as for the control table model can be achieved.

Acknowledgements. The work has been conducted in close cooperation with Queensland Rail, in particular we would like to thank George Nikandros, David Barney and David Tombs. The domain expertise of Neil Robinson helped to shape the model and its simplifications, and the persistence of Wendy Johnston when analysing BDD size and structures was essential in producing the ordering strategies. This work was supported by Australian Research Council (ARC) Linkage Grant LP0882479 and the Australian Safety Critical Systems Association (aSCSa).

References

1. Robinson, N., Barney, D., Kearney, P., Nikandros, G., Tombs, D.: Automatic generation and verification of design specification. In: Proc. of Int. Symp. of the International Council On Systems Engineering (INCOSE 2001). (2001)
2. Gnesi, S., Lenzini, G., Latella, D., Abbaneo, C., Amendola, A., Marmo, P.: 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 (2000) 119–124
3. Bernardeschi, C., Fantechi, A., Gnesi, S., Mongardi, G.: Proving safety properties for embedded control systems. In: Procs. of Conference on Dependable Computing (EDCC-2). Volume xvi+440., Springer-Verlag (1996) 321–332
4. Cleaveland, R., Luettgen, G., Natarajan, V.: 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). Volume 1055 of LNCS., Springer-Verlag (1996) 287–297
5. Groote, J., Koorn, J., van Vlijmen, S.: The safety guaranteeing system at station Hoorn-Kersenboogerd. In: Proceedings 10th IEEE Conference on Computer Assurance (COMPASS95), IEEE Computer Society Press (1995) 131–150
6. Eisner, C.: 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). Volume 1703 of LNCS., Springer-Verlag (1999)
7. Eisner, C.: Using symbolic CTL model checking to verify the railway stations of Hoorn-Kersenboogerd and Heerhugowaard. Software Tools for Technology Transfer **4**(1) (2002) 107124
8. Simpson, A., Woodcock, J., Davies, J.: The mechanical verification of solid state interlocking geographic data. In Groves, L., Reeves, S., eds.: Proc. of Formal Methods Pacific (FMP'97). Discrete Mathematics and Theoretical Computer Science Series, Springer-Verlag (1997) 223–243
9. Huber, M., King, S.: Towards an integrated model checker for railway signalling data. In Eriksson, L.H., Lindsay, P., eds.: Proc. on Formal Methods Europe (FME'2002). Volume 2391., Springer-Verlag (2002) 204–223
10. Ferrari, A., Magnani, G., Grasso, D., Fantechi, A.: Model checking interlocking control tables. In Schnieder, E., Tarnai, G., eds.: Proceedings of Conference on Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 2010). Volume 2., Springer-Verlag (2011) 107 – 115
11. Winter, K., Robinson, N.J.: Modelling large railway interlockings and model checking small ones. In Oudshoorn, M., ed.: Proc. of Australasian Computer Science Conference (ACSC2003). (2003)
12. Winter, K., Johnston, W., Robinson, P., Strooper, P., van den Berg, L.: Tool support for checking railway interlocking designs. In Cant, T., ed.: Proc. of the 10th Australian Workshop on Safety Related Programmable Systems (SCS'05). Volume 55., Australian Computer Society, Inc. (2005) 101–107
13. Johnston, W., Winter, K., van den Berg, L., Strooper, P.A., Robinson, P.: Model-based variable and transition orderings for efficient symbolic model checking. In Misra, J., Nipkow, T.,

Sekerinski, E., eds.: Proc. of 14th Int. Symposium on Formal Methods (FM 2006). Volume 4085 of LNCS., Springer-Verlag (2006) 524–540
14. Winter, K.: Symbolic Model Checking for Interlocking Systems. In: Railway Safety, Reliability, and Security: Technologies and Systems Engineering. IGI Global (2012) 298–315
15. Robinson, N.: Operation concept document. SigTools-004, version 1.9 (May 2002)
16. Robinson, N.: Design specification – Track Layout Editor. SigTools-032, version 1.1 (April 2004)
17. McComb, T., Robinson, N.J.: Assuring graphical computer aided design tools. Technical Report TR02-18, Software Verification Research Centre, University of Queensland (2001)
18. Tombs, D., Robinson, N.J., Nikandros, G.: Signalling control table generation and verification. In: Proc. of Conf. on Railway Engineering (CORE 2002), Railway Technical Society of Australasia (2002)
19. Johnston, W.: Design specification – Control Table Editor. SigTools-044, version 0.1 (January 2003)
20. van den Berg, L., Strooper, P., Johnston, W.: An automated approach for the interpretation of counter-examples. In R, B., Roveri, M., Somenzi, F., eds.: Proceedings of 1st Workshop on Verification and Debugging. (2006) 6–25
21. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: an OpenSource tool for symbolic model checking. In Brinksma, E., Larsen, K.G., eds.: Proc. of Int. Conference on Computer Aided Verification (CAV 2002). Volume 2404 of LNCS., Springer Verlag (2002) 359–364
22. Egon Börger, R.S.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer-Verlag (2003)
23. Cavda, R., Cimatti, A., Jochim, C.A., Keighren, G., Olivetti, E., Pistore, M., Roveri, M., Tchaltsev, A.: NuSMV 2.5 User Manual, `http://nusmv.irst.itc.it.` (2010)
24. Winter, K.: Model checking control tables: the ASM-NuSMV approach. SigTools.039, version 0.1 (October 2002)
25. Johnston, W.: Design specification - Verification Manager and NuSMV Driver. SigTools-051, version 0.4 (April 2004)
26. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2) (1994) 183–235
27. Schneider, S.: An operational semantics for timed CSP. Information and Computation **116**(2) (1995) 193–213
28. van den Berg, L., Strooper, P., Winter, K.: Introducing time in an industrial application of model checking. In: Proc. of 12th Int. ERCIM Workshop on Formal Methods for Industrial Critical Systems (FMICS'07). Volume 4916 of LNCS., Springer-Verlag (2008) 56–67
29. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions On Computers **C-35**(8) (August 1986)
30. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press (2000)
31. Lewis, G., Comella-Dorda, S., Gluch, D., Hudak, J., Weinstock, C.: Model-based verification: Analysis guidelines. Technical Report CMU/SEI-2001-TN-028, Carnegie Mellon Software Engineering Institute (2001)
32. Burch, J., Clarke, E., Long, D.: Symbolic model checking with partitioned transition relations. In: Int. Conf. on Very Large Scale Integration. (1991)
33. Geist, D., Beer, I.: Efficient model checking by automated ordering of transition relation. In David L. Dill, ed.: Proc. of Int. Conference on Computer-Aided Verification (CAV'94). Volume 818 of LNCS., Springer-Verlag (1994) 299–310
34. McMillan, K.: Symbolic Model Checking. Kluwer Academic Publishers (1993)
35. Moon, I., Hachtel, G.D., Somenzi, F.: Border-block triangular form and conjunction schedule in image computation. In: Proc. of Formal Methods in Computer-Aided Design (FMCAD 2000). Volume 1954 of LNCS. (2000) 73–90