Towards an Integrated Model Checker for Railway Signalling Data

Michael Huber[1] and Steve King[2]

[1] Siemens Transportation Systems***, Industriestrasse 42, CH-8304 Wallisellen, Switzerland
michael.mh.huber@siemens.com

[2] The University of York, Dept of Computer Science, York YO10 5DD, UK
king@cs.york.ac.uk

Abstract. Geographic Data for Solid State Interlocking (SSI) systems detail site-specific behaviour of the railway interlocking. This report demonstrates how five vital safety properties of such data can be verified automatically using model checking. A prototype of a model checker for Geographic Data has been implemented by replacing the parser and compiler of NuSMV. The resulting tool, gdlSMV, directly reads Geographic Data and builds a corresponding representation on which model checking is performed using NuSMV's symbolic model checking algorithms.
Because of the large number of elements in a typical track layout controlled by an SSI system, a number of optimisations had to be implemented in order to be able to verify the corresponding data sets.
We outline how most of the model checking can be hidden from the user, providing a simple interface that directly refers to the data being verified.

Keywords: Data verification, model checking, hidden formal methods.

1 Introduction

The basic purpose of railway interlockings is to provide trains with safe routes to their destinations. A route can be any part of track between two signals. The first signal, often referred to as the entry signal, allows a train to proceed once the route is locked. This means that all points have been moved and locked to the required position, and protections such as gates at level-crossings are appropriately deployed. By locking all elements necessary for safely passing a route before opening the entry signal, the interlocking makes sure that no two routes with conflicting requirements will be allowed at the same time [1].
The Solid State Interlocking (SSI) system is a modular computer interlocking of the first generation [5]. It consists of interlocking modules which communicate with signals, points and track circuits via track-side data links. In that way, each SSI module can control an area with up to 40 signals and about 40 points [9].

*** The first author joined Siemens after the work reported in this paper had been completed. Siemens did not take part in this research.

For larger control areas, several SSI modules can be grouped and operated from a single control panel.
Whether it is safe to set a route depends on the site-specific track layout. Therefore, the generic SSI program needs to rely on configuration data specifying the conditions under which routes can be set for the specific track layout controlled by the installation. Therefore, each SSI interlocking module keeps a database with the necessary geographic information, which is referred to as Geographic Data.
Errors in the Geographic Data could lead to conflicting routes being set at the same time, which would allow trains to proceed to the same track segment. Other data errors could allow points to be moved while a train is passing over them, which could lead to derailment of the train. Therefore, Geographic Data are safety-critical. For each installation, the data set has to be verified carefully.
The aim of the work reported here is to provide a data verification tool based on model checking, which is both efficient enough to verify full Geographic Data sets, and easy to use for signal engineers without special knowledge of formal verification techniques. In order to achieve the latter, one of the design goals was to hide the model checking from the user, and to present him with a simple interface.

The rest of this paper starts with a section on Geographic Data and its verification. Then the model checker for Geographic Data, which has been developed based on the NuSMV [2] system, is described and the safety properties to be verified are introduced. We then set out how both the efficiency and the usability of the resulting model checker have been improved. Finally, the results are compared to previous work and conclusions are drawn.

2 SSI's Geographic Data

2.1 Geographic Data Specifying Routes

There are three types of data sets specifying how routes can be set and released:

Panel Route Request (PRR) data define the conditions under which route requests can be granted, and the actions necessary in order to lock the route.

Points Free to Move (PFM) data specify the conditions under which points are allowed to be moved from one position to the other.

Sub-Route Release (SRD) data define the conditions under which subroutes which have been locked for a route can be released again.

The following subsections present brief descriptions of a slightly simplified version of these data. [12] contains more detailed definitions.

PRR Data. PRR data define the response of the interlocking to route requests issued by the signalman. The following data relates to the track layout shown in Fig. 1. The highlighted part is the route from signal S10 to signal S14, for which the PRR data is shown below:

```
*QR10B if   R10B a, P201 cfn, UAB-BC f, UAC-AB f
       then R10B s, P201 cn,  UAB-CB l, UAC-BA l
```

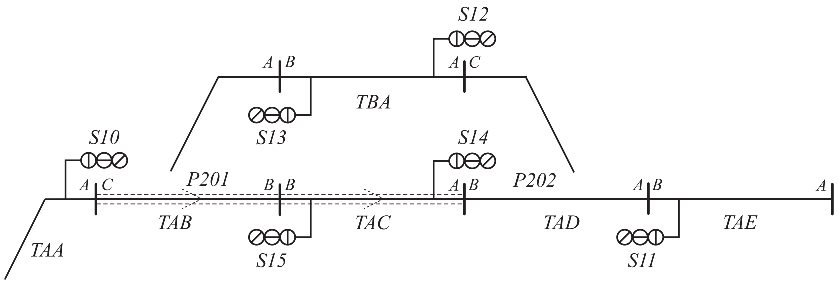

Fig. 1. Route R10B from signal S10 to S14

The data for the request to set route R10B above starts with the label `*QR10B`. This is recognised as a unique identifier for the route setting data (`*Q`) for route R10B (`R10B`). This is followed by the conditions under which the route may be set, indicated by the key word `if`. In this case, the conditions are as follows:

`R10B a`	Route R10B is available, i.e. not barred from being set;
`P201 cfn`	Point P201 is either already in its normal position, or it is free to go there;
`UAB-BC f`	Sub-route UAB-BC[1]is free; and
`UAC-AB f`	Sub-route UAC-AB is free — these last two conditions ensure that no route in the opposite direction is set over any part of the route currently requested.

If these conditions are met, than the route can be set. The necessary steps for that are specified after the key word `then`. In the example, they are as follows:

`R10B s`	The status of route R10B is ‘set’.
`P201 cn`	Point P201 is moved to its normal position.
`UAB-CB l`	Sub-route UAB-CB is locked.
`UAC-BA l`	Sub-route UAC-BA is locked.

PFM Data. In the PRR data described above, the acronyms `cfn` and `cfr` refer to points being free to move to their normal position, or their reverse position respectively. The conditions under which points are free to move are defined in the PFM data . An example for point P201 is shown below.

```
*P201N  TAB c, UAB-AC f, UAB-CA f
*P201R  TAB c, UAB-BC f, UAB-CB f
```

[1] The names of subroutes start with the same identifier as the track-circuit to which they are associated, except for the first letter which is T for track-circuits but U for sub-routes. The two characters following the dash give the direction of the sub-route.

The labels for PFM data contain the identifier of the point for which the data applies, and N or R depending on whether the conditions are for moving to the normal or reverse position. Therefore, the statement starting with `*P201N` above gives the condition under which point P201 is free to move to its normal position:

`TAB c`	Track-circuit TAB is clear.
`UAB-AC f,` `UAB-CA f`	The two sub-routes over the reverse branch of the point are free.

In other words, point P201 can move to its normal position if there is no train occupying the point and if no route is set over the reverse position of the point.

SRD Data. PRR and PFM data contain all the information necessary for setting routes. SRD data are responsible for freeing the individual sub-routes locked by a route as soon as it is safe to do so. SRD data are periodically checked, unlike PRR and PFM data which are executed on request.

```
UAB-CB f    if TAB c, R10B xs
UAC-BA f    if TAC c, UAB-CB f
```

The first sub-route of the route can be released as soon as

`TAB c`	Track-circuit TAB is clear, and
`R10B xs`	Route R10B is unset.

Following sub-routes depend on the release of their predecessors, therefore the second sub-route can be released when

`TAC c`	Track-circuit TAC clear is, and
`UAB-CB f`	the previous sub-route UAB-CB is free, i.e. it has already been released.

2.2 Verification of Geographic Data

The Geographic Data specifying when and how routes are locked, and when sub-routes are allowed to be released again, are safety-critical. An error in the data could mean that conflicting routes are allowed to be set at the same time, which would allow trains to proceed on conflicting paths. Furthermore, an error in PFM data could allow points to move under a passing train.
For example, route R11A in Fig. 2 conflicts with route R10B shown earlier in Fig. 1, as both routes set at the same time would allow two trains from opposite directions to proceed to the same track segment, namely TAC. However, a simple mistake in the PRR data can allow these two routes to be set at the same time. In the data shown below, sub-route UAC is checked in the wrong direction for route R10B, i.e. UAC-*BA* is checked instead of the opposing sub-route UAC-*AB* (last statement in the first line):

```
*QR10B if R10B a, P201 cfn, UAB-BC f, UAC-BA f
       then R10B s, P201 cn, UAB-CB l, UAC-BA l
*QR11A if R11A a, P202 cfn, UAD-BA f, UAC-BA f
       then R11A s, P202 cn, UAD-AB l, UAC-AB l
```

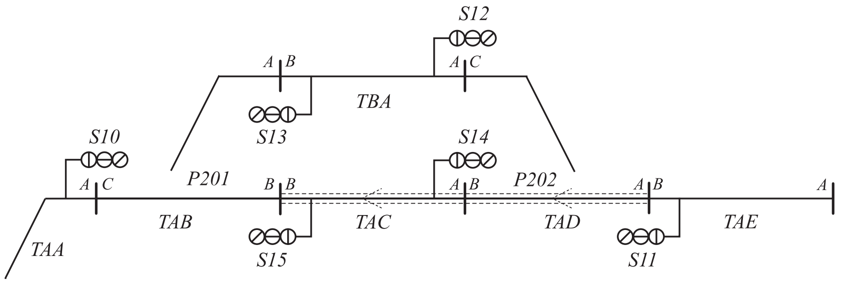

Fig. 2. Route R11A from signal S11 to S15

Mistakes like this one are sufficient to allow conflicting routes to be set at the same time, which could have catastrophic consequences. Therefore, Geographic Data have to be verified carefully before they can become operational on an installation. Currently, this verification is mostly done manually [16]. There are two main verification steps:

Checking of Geographic Data is carried out after all the data have been prepared. An independent signal engineer who has not been involved in the preparation of the data inspects print-outs of the complete data and verifies them against design information.

Testing involves actually running a simulated interlocking with the checked data. A third not previously involved engineer carries out the tests. In a first stage, correspondence between the simulated states of trackside equipment and the display is systematically tested. After that, it is tested whether signalling principles are correctly followed. Extensive tests are carried out to gain confidence in the data. For example, attempts are made to set all possible opposing and conflicting routes for each route in order to confirm that this is not possible.

Finally, an on-site test on the complete installation checks for timing problems and for interoperation with neighbouring interlockings. Task and error analyses of the SSI data design process have been carried out ([16,17]). It has been reported that the bulk of errors detected shifts from checking to testing with increasing complexity of the data: in simple schemes most errors are detected early in the checking phase, but in more complex data sets the majority of problems are discovered during testing. Furthermore, some categories of errors were particularly resistant to the checking phase. Most problems with opposing subroutes, for example, slipped through the checking phase and were only detected by testing [17].

2.3 Automated Verification of Geographic Data

As manual verification of Geographic Data is time-consuming and demanding, the potential for automation has been assessed earlier. [6] noted that there are two levels at which automated data checking could be applied:

- At a simpler level, conformance to general rules could be checked. This would include, for example, verifying that commands to move points are always preceded by checks whether these points are free to move to the required positions.
- A higher level of checking would verify that basic safety properties always hold. This would attempt to show that there exists no series of inputs that can violate any of these safety properties.

[6] further pointed out that the first level of checking is to some extent similar to the rules applied when writing the data. Therefore, it is susceptible to common mode errors: the errors that are likely to be made in the data preparation are at the same time unlikely to be detected by that checking approach. The paper concluded that the second approach "is potentially the best means of detecting any unexpected side-effects of data errors which cannot be guaranteed to be found by testing" [6, p 62].
In order to obtain such an extensive verification of safety properties, model checking has been deployed in previous work. [11] modelled Geographic Data data with CCS, and the Concurrency Workbench (CWB) was used for model checking. However, because of CWB's explicit representation of the modelled system as a graph, it was found to be unsuitable for large scale applications. [14] obtained better results by using CSP for modelling the data and FDR2 for model checking it. This work also describes a formal translation from SSI data to CSP. [7] developed a tool for the automated translation from Geographic Data to machine readable CSP as input to the FDR2 model checker. [13] investigated both the use of CSP and SMV for the verification of Geographic Data. Emphasis was set on devising a library of general SMV modules that could be used for any railway network described by Geographic Data.
The work described here builds on the experience gained from [13]. The goal was to build a tool for the verification of five vital safety properties for locking and releasing routes. The tool should be efficient enough to handle Geographic Data sets of realistic sizes. Furthermore, model checking technicalities should be hidden from the user as far as possible, in order to allow signal engineers without special knowledge in model checking to work with the tool.

3 gdlSMV: A Symbolic Model Checker for Geographic Data

For reasons of both efficiency and usability, it was decided not to translate Geographic Data to an SMV input script, but to integrate a Geographic Data parser with an SMV model checker. In that way, Geographic Data input is translated directly to Ordered Binary Decision Diagrams (OBDDs), which are the model checker's internal representation. It was assumed that abstractions and optimisations would be easier to apply at the OBDD level. Furthermore, an integrated tool that reads Geographic Data and produces verification output directly without the need for an intermediate SMV representation can be better tailored to the user's needs concerning the interface.

3.1 Adapting NuSMV for the Verification of Geographic Data

Fig. 3 shows the basic concept of adapting an existing SMV model checker for the verification of Geographic Data. Instead of SMV scripts, the inputs are Geographic Data files. Therefore, the parser and compiler of the model checker are replaced. Furthermore, optimisations are performed on the OBDD representation. As the basis model checker to be adapted, NuSMV [2] has been chosen.

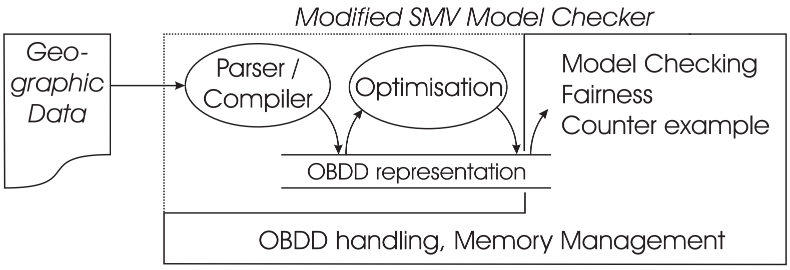

Fig. 3. NuSMV's parser and compiler are replaced in order to obtain a model checker which directly reads Geographic Data. This makes it possible to perform optimisations on the internal OBDD representation

NuSMV is a modular and well-documented re-implementation of the SMV system [10]. In the design of NuSMV, it is explicitly acknowledged that the SMV input language is not optimal for all applications. Therefore, NuSMV provides a clear distinction between the SMV input front end and the model checking algorithms: only the parser, instantiation, encoder, semantic checking and parts of the compiler are dependent on the format of the model description, i.e. the SMV language. The actual model checking, the computation of fairness constraints, and the generation of counter examples operate on the OBDD representation of the system and therefore independent of the initial input format. Also the kernel providing OBDD handling routines and memory management functions, and the user interfaces are, with minor exceptions, independent.
Converting NuSMV with its SMV input format to gdlSMV which directly reads Geographic Data files necessitated the following changes:

Parser: a new parser has been written for reading Geographic Data files and storing their content in an appropriate form. However, parts of the original NuSMV parser are still necessary, as it is also used to parse CTL formulas, which are still required in gdlSMV.

Instantiation: this module is obsolete in gdlSMV, as there are no hierarchical definitions or module declarations. There is, however, information necessary to build the transition relation which is not explicitly declared in Geographic Data. For example, there is no variable declaration in Geographic Data, nor are initial states specified. Therefore, the instantiation module of NuSMV has been replaced with a set of functions that extract implicit information from Geographic Data and store it in an adequate way for use in later stages.

Encoder: all variables in Geographic Data have to be encoded in the same way that variables in SMV scripts are encoded by NuSMV. However, the variables in Geographic Data are of a small number of pre-defined types; there is no need to deal with type declarations. That means variable encoding in gdlSMV is static and much simpler than in NuSMV.

Semantic checking: the semantic checks performed in NuSMV for SMV scripts are obsolete. Therefore semantic checking has, for now, been abandoned. However, it might be beneficial to introduce some semantic checks for Geographic Data, as some classes of errors could be discovered that way, without the need to perform model checking.

Compiler: the transition relation is built from the list of Geographic Data statements rather than from SMV code describing next state assignments. This again relies on implicit information in Geographic Data, but the task is simplified by the fact that there are a small number of basic Geographic Data constructs that have to be catered for.

We now describe how exactly Geographic Data is used to build an OBDD representation of a corresponding finite state machine.

3.2 Modelling the Operations Specified by Geographic Data

The three parts of the finite state machine for performing model checking, i.e. the initial states, the transition relation, and the fairness constraints are built as follows:

Initial States. gdlSMV considers four types of variables, namely points, track-circuits, routes, and sub-routes. The set of initial states is defined as:

- points are either controlled normal or controlled reverse,
- track-circuits are either clear or occupied,
- all routes are unset, and
- all sub-routes are free.

This results in a set of 2^{p+t} initial states, where p and t are the numbers of points and track-circuits respectively. These initial states differ slightly from the initial state of a real Solid State Interlocking when it is reset. However, since the modelled initial states are less restrictive than what is possible in practice, it is safe to use this initialisation (see [8] for details).

Transition Relation. In NuSMV, both synchronous and asynchronous systems can be modelled. In synchronous systems, all variables change their value at the same time. Each transition evaluates all expressions for the new values of all variables depending on the present values. In asynchronous systems, there are a number of processes. In each process, one or several variables can change their state. For each individual transition, one process is chosen non-deterministically and its variables updated according to the specified transitions. All other variables not changed by that process retain their previous value. This second behaviour is what is needed for gdlSMV: each PRR and SRD statement from the Geographic Data is modelled as a process. Therefore, at any time at most one statement will be executed. However, if in this statement several variables change their value, then this is done synchronously as all variables belong to the same process. Such a behaviour closely resembles SSI, as in SSI at most one Geographic Data statement can be executed at any time, and all the variables affected by a statement are updated in one atomic operation.

SRD statements. The SRD statements specify the conditions under which a sub-route can be released, and they easily translate to corresponding transition expressions, as shown below.[2]

```
UAB-CB f  if TAB c, R10B xs      IF(TAB = c AND R10B = xs) THEN
                                    UAB-CB':=f
                                 ELSE
                                    UAB-CB':=UAB-CB
```

As there are individual SRD statements for all sub-routes, there is always exactly one variable that can change its value in an SRD transition. Each SRD statement is modelled as a process that can be executed at any time. The ELSE clause needs to state explicitly that the variable does not change, as a variable that is not mentioned in a transition can take any value.

PRR and PFM statements. PFM statements just represent conditions under which points are free to move. They do not themselves represent transitions, but the PFM conditions will be incorporated in the transitions constructed from PRR statements. The PFM and PRR statements

```
/ Points free to move (PFM)
*P201N   TAB c, UAB-AC f, UAB-CA f

/ Panel Route Request (PRR)
*QR10B if   P201 cfn, UAB-BC f, UAC-AB f
       then P201 cn, R10B s, UAB-CB l, UAC-BA l
```

translate to the following transition:

```
IF((P201 = cn OR (TAB = c AND UAB-AC = f AND UAB-CA = f)) AND
   UAB-BC = f AND UAC-AB = f) THEN
  P201':=cn;   R10B':=s;    UAB-CB':=l;      UAC-BA':=l;
ELSE
  P201':=P201; R10B':=R10B; UAB-CB':=UAB-CB; UAC-BA':=UAC-BA
```

As described in section 2.1, the expression `P201 cfn` in the PRR statement is an implicit reference to the PFM statement. The attribute `cfn` requires that P201 is

[2] Dashed variable names (`UAB-CB'`) indicate the next state as opposed to the current state (`UAB-CB`).

either in controlled normal, or that it is free to go to controlled normal. Whether the latter is the case is specified in the PFM statement `*P201N`, which sets out the condition under which P201 might change to normal. In the pseudo-code for the transition above, it can be seen that the PFM condition has been integrated in the condition of the PRR statement. Each PRR statement is extended in that way with the corresponding PFM statement(s), and for each PRR statement a process is created.

Track-circuits. Finally, the behaviour of track-circuits is modelled by allowing them to change their states at any time. For reasons of efficiency, this is not implemented as a further transition, but it is dealt with in the frame axiom. Letting track-circuits change their states unconditionally at any time is an unrealistic behaviour, as it represents trains jumping across the track layout. However, as [11] has shown, if setting routes and releasing sub-routes is safe with this behaviour than it is also safe with the more orderly behaviour of real trains.

Fairness Constraints. Presently, there are no fairness constraints generated, as they are not necessary for checking our safety properties (see [8] for details).

3.3 Verifying Safety Properties

[14] lists the following safety properties that could potentially be violated by errors in Geographic Data:

(i) At any time, at most one sub-route over a given track-circuit is locked.
(ii) Whenever a sub-route over a point is locked, the point is in alignment with that sub-route.
(iii) Whenever a route is set, all its sub-routes are locked.[3]
(iv) If a sub-route is locked for a route, then all sub-routes ahead of it on that route are also locked.
(v) It is never the case that points are moved while the corresponding track-circuit is occupied.

Assuming that trains obey signals and therefore only enter routes that are locked for them, properties (i), (iii) and (iv) guarantee that there is never more than one train on the same track segment. This is achieved by safe handling of the sub-routes (i), and by verifying that routes are properly composed of their sub-routes (iii), (iv). Property (ii) assures that trains never pass over points that are in the wrong position. Finally, (v) makes sure that points are prevented from moving while trains are passing over them.
These five properties can easily be translated to CTL specifications which can be verified with gdlSMV:

[3] Note that the first sub-route of a route can not be released before the route is unset. Then the sub-routes are released in sequence, as it is safe to do so.

(i) *At most one sub-route over the same track-circuit is locked at any time.*

```
AG !(UAA-AB = 1 & UAA-BA = 1)
```

i.e. it is never the case (`AG !`: in all reachable states it is not true) that both UAA-AB and UAA-BA are locked. For each track-circuit, there is a CTL expression required, stating that any two of the sub-routes over that circuit can not be locked at the same time. There can be more than two sub-routes over the same track-circuit, in which case the property expands.

(ii) *Whenever a sub-route over a set of points is locked, then the points are in alignment with that sub-route.*

```
AG ((UAB-AC = 1 | UAB-CA = 1) -> P201 = cr)
AG ((UAB-BC = 1 | UAB-CB = 1) -> P201 = cn)
```

i.e. whenever a sub-route over the reverse position of point P201 is locked, then P201 is in that position, and when a sub-route over the normal position is locked, it is in the normal position.

(iii) *Whenever a route is set, all its sub-routes are locked*

```
AG ((R13  = s) -> (UAA-AB = 1 & UAB-AC = 1 & UAK-AB = 1))
```

i.e. it is always the case that when route R13 is set, all its sub-routes UAA-AB, UAB-AC, and UAK-AB are locked.

(iv) *Sub-routes are released in correct order*

```
AG ((R13  = s) -> (!E[UAB-AC = 1 U UAA-AB = f] &
                   !E[UAA-AB = 1 U UAK-AB = f]))
```

i.e. from any state where route R13 is set, there exists no path along which UAB-AC remains locked until UAA-AB is free, nor exists a path along which UAA-AB remains locked until UAK-AB is free. By asserting that no sub-route can be freed as long as its immediate predecessor is still locked, this property enforces the sub-routes to be released in correct order.

(v) *Occupied points never move*

```
AG (TAB = o -> ((P201 = cr -> AX P201 = cr) &
                (P201 = cn -> AX P201 = cn)))
```

i.e. in any state where track-circuit TAB is occupied, if point P201 is in its reverse position in that state then it will remain so in all next states. Correspondingly, if it is in its normal position it will remain there in all next states. It is sufficient to consider next states only: if the track-circuit is still occupied in that next state, then the property applies again for all next states reachable from that state, and so on as long as TAB is occupied.

Such CTL specifications need to be generated and verified for all track-circuits, points and routes in a set of Geographic Data. See section 4.2 on how these CTL expressions can be hidden from the user.

4 Improving Efficiency and Usability

4.1 Efficiency

Before gdlSMV was tested on a large data set, some optimisations were implemented in order to reduce run-time. The results were tested on a small data set called "OpenAlvey" (Data from [11]).

Variable ordering [13] pointed out that the order of the variables for constructing the OBDDs representing SSI data can have a huge impact on run-time requirements. It has therefore been attempted to extract a satisfactory variable ordering from the data. Generally, it seems that orderings where related variables are kept close together tend to yield good results [4]. In SSI data, variables occurring in the same statement clearly are related. Therefore, it was expected that ordering the variables as they are found in a traversal of the data should yield satisfactory results. Experimenting with all possible permutations, the order PRR - SRD - PFM obtained the best results. Within each data block, the individual statements are examined in the order in which they occur in the source file. The quality of the results obtained therefore depends on the order of the statements. However, as this order is not usually random but follows a geographical logic, the obtained result is satisfactory.

Modified frame axiom.[4] Initially, there was a separate transition in which track-circuits could change their state, and routes which are set could become unset. However, this leads to unnecessary iterations in reachability analysis, and it makes the representation of intermediate sets of states found needlessly complicated. In the case of track-circuits which can change their states unconditionally at any time, it is clear that for each reachable state, the same state with any possible combination of track-circuits being occupied and clear is also reachable. However, if track-circuits need an explicit transition to change their state, then there are intermediate results which specify in detail that certain track-circuits must be clear and others occupied — only to find in the next iteration that all other permutations are also permitted. Similar situations occur for routes which would be explicitly specified to be set, while whenever a state with a route set is reachable, the same state with that route no longer set is also reachable.

Moving the specification of the behaviour of track-circuits and the un-setting of routes to the frame reduced both the number of iterations necessary in reachability analysis, and the size of the representation of intermediate sets of states found after each iteration. For the track-circuits, describing their behaviour in the frame axiom is particularly easy as they are simply not mentioned. This allows them to take any value. Technically, they will not be mentioned in the OBDD representing the set of states after each transition,

[4] The frame axiom is the expression specifying what does *not* change in a transition. For each transition, it explicitly assigns present state to next state for all variables not affected by the transition.

therefore reducing the size of that OBDD and allowing the variables to take either of their values. As for routes, as soon as they are set they will disappear from the expression describing that state, as whenever a route is set the same state with that route unset is also reachable.

Compared to the version with a separate transition, the modification to the frame axiom significantly decreased run-time (by almost 90% for a simple data set).

Expanded set of initial states. At first, all points had been initialised to their normal position. However, this implied that many reachable states were only found after a relatively large number of iterations. Consider the state where no routes are set, all sub-routes are free, and all points are reverse. Except for the position of the points, this is the same state as the initial state. Nevertheless, it can only be reached after routes have been set which change the points to reverse, and after these routes have been unset and all sub-routes released again. Throughout the necessary iterations to reach that state, the OBDD describing the set of states reached so far has to exclude states which have not yet been found. Therefore, the size of that OBDD is likely to be large.

The set of initial states has therefore been expanded to include all possible combinations of point positions. Apart from placing no unnecessary restrictions on the initial states, this leads to a greater number of states being found in earlier iterations, and it can reduce the size of the OBDDs representing the sets of states found after each iteration.

For the small data example, the above change reduced run-time by almost 50%. As the impact of the optimisation depends on the number of points in a scheme and the complexity of routes leading over these points, it is assumed that its benefits are even bigger on larger data sets.

The optimisations described so far have reduced the verification time for a small data example from around 10 minutes to around 7 seconds — slightly faster than a model for the same data written in SMV verified by the original NuSMV, which included the optimisation of the initial set, and had been run with the same variable ordering. However, when a larger data set (Leamington Spa railway station) was made available for tests, an as-yet-unexpected problem occurred. With 64 route requests, 98 sub-routes, 15 points and 35 signals, Leamington Spa is a fairly large scheme. 223 binary variables are necessary to model it, spanning a space of $1.3 \cdot 10^{67}$ states, $4.7 \cdot 10^{27}$ of which are reachable. While reachability analysis in a state space of such dimensions was expected to take much longer than it did for the small data example used so far (64 binary variables; $2.8 \cdot 10^{9}$ reachable states out of a total of $1.8 \cdot 10^{19}$), no thought had been given to the fact that memory usage too could become a problem. However, the first verification attempt of Leamington Spa used up all 256M of RAM after just over an hour, and in a few more hours it also exhausted the 306M of available swap memory. By that time, it had completed only the first five iterations of reachability analysis. While run-time optimisation remained a goal, memory usage become the dominating concern. The following optimisations were then introduced:

Disjunctive partitioning. While early tests had not suggested disjunctive partitioning to be more efficient than a monolithic representation of the transition relation, its implementation in gdlSMV led to a remarkable reduction in memory usage. In a simplified version of Leamington Spa with 139 binary variables, maximum memory usage could be reduced from 160M to 95M, and run-time from 2 hours 40 minutes to 1 hour 55 minutes.

Dynamic variable re-ordering. The importance of a suitable variable ordering has been mentioned before. However, a variable ordering which allows for representing the transition relation in a compact form does not necessarily also result in optimal representations of the set of states found after each iteration in reachability analysis. Since, in general, different variable orderings are optimal for different stages in model checking, NuSMV offers dynamic variable re-ordering (as do other model checkers). In NuSMV, as soon as dynamic variable re-ordering is enabled it will be applied to find a compact representation of the OBDDs currently in use. According to their size, a limit is set for memory usage. As soon as this limit is reached, the re-ordering process is started again, and a new memory limit is set. As the model checking part of NuSMV has not been changed in any way in gdlSMV, the option for dynamic re-ordering is also available in gdlSMV. Enabling it not only greatly reduces the amount of memory required, but also decreases run-time. Although almost half of the verification time for Leamington Spa is spent re-ordering variables, the total run-time is still much lower than without re-ordering, because operations on compact OBDDs are much faster than on large OBDDs.

Although the above measures did decrease both run-time and memory usage, they were still not sufficient to reduce resource requirements to an acceptable level for the full verification of Leamington Spa. On smaller examples it could be seen that the first few iterations run fairly quickly, as do the last iterations. Both the OBDD size of the set of reachable states found and run-time requirements drastically increase in the middle iterations. As this is a common problem in model checking, it was expected that there is previous work addressing it. [3], although concerned with hardware verification, provided the vital clue. It suggested decomposing complex hardware circuits into loosely coupled modules. Then, instead of performing a breadth-first search with the whole transition relation, an initial search is restricted to one of these modules. As before, this search will have peak resource requirements in its middle iterations, but as it is restricted to a smaller state space, that peak will be smaller. Such local searches are carried out individually for all modules. However, these searches will not initially find all reachable states if modules depend on each other. Therefore, the reachability analysis for all modules has to be repeated with the states found in the previous searches of the individual modules until no new states are found anymore. This results in more iterations being required to find all reachable states, but in small peak sizes of the intermediate OBDDs.

Splitting the transition relation. The principle described in [3] and outlined above has been incorporated in gdlSMV by splitting the disjunctively partitioned transition relation into two parts, one containing all transitions for route setting, and one containing all sub-route release transitions. The route setting transitions are applied iteratively, first to the initial states, and then to the set of reachable states found previously. The latter is repeated until no new states are found anymore: all combinations of routes that can be set from the initial states are found. Now the sub-route releasing data are applied iteratively, again until no new states are found anymore. After that the route setting transitions have to be applied again, as there are reachable states where a route can be set after a previous route has been partially released. Therefore, iterating over route setting and sub-route releasing transitions is repeated until the set of reachable states remains constant.

For the simplified data of Leamington Spa, splitting the transition relation in the above manner reduced run-time from about an hour (with all the previous optimisations including dynamic re-ordering) to under 10 minutes, and memory usage from 31M to 10M. For the first time it now became possible to complete reachability analysis for the original Leamington Spa data. However, this still took more than 70 hours and up to 360M of RAM. Further improvement was clearly needed. As splitting of the transition relation had been successful, it was an obvious choice to apply the same technique again:

Second splitting of the transition relation. There is no second partitioning as obvious as separating route setting from sub-route releasing. However, there are two more classes in which data can easily be grouped, namely whether the route or sub-route they apply to are in up- or in down-direction. The naming convention set out in [15] ensures that the direction of sub-routes can be determined by the last two characters of their name: if these characters are in alphabetical order, the sub-route is in down direction, otherwise up. Therefore, sub-route release data can be immediately grouped by the corresponding direction using the name of the sub-route concerned. For route setting data, the name of the first sub-route to be locked is used. Combined with the previous splitting, this results in four categories of transitions: route setting for up direction, sub-route releasing for up direction, route setting for down direction, and sub-route releasing for down direction. As before, reachability analysis is done over these groups of statements individually, and all four sub-analyses are repeated until the full reachable state space is explored.

This final version reduced run-time of reachability analysis for Leamington Spa to 9 hours, and peak memory during that time to below 70M. Fig. 4 summarises data for the verification of the simplified data set for Leamington Spa with three versions of gdlSMV: without splitting the transition relation for reachability analysis, all reachable states are found in 54 iterations. However, the second graph shows that intermediate OBDDs are of large size, and the iterations on these large OBDDs are very slow, as can be seen on the third graph. For the first splitting of the transition relation, the top graph clearly shows how reachable states are now found stepwise (first with route setting, then sub-route release,

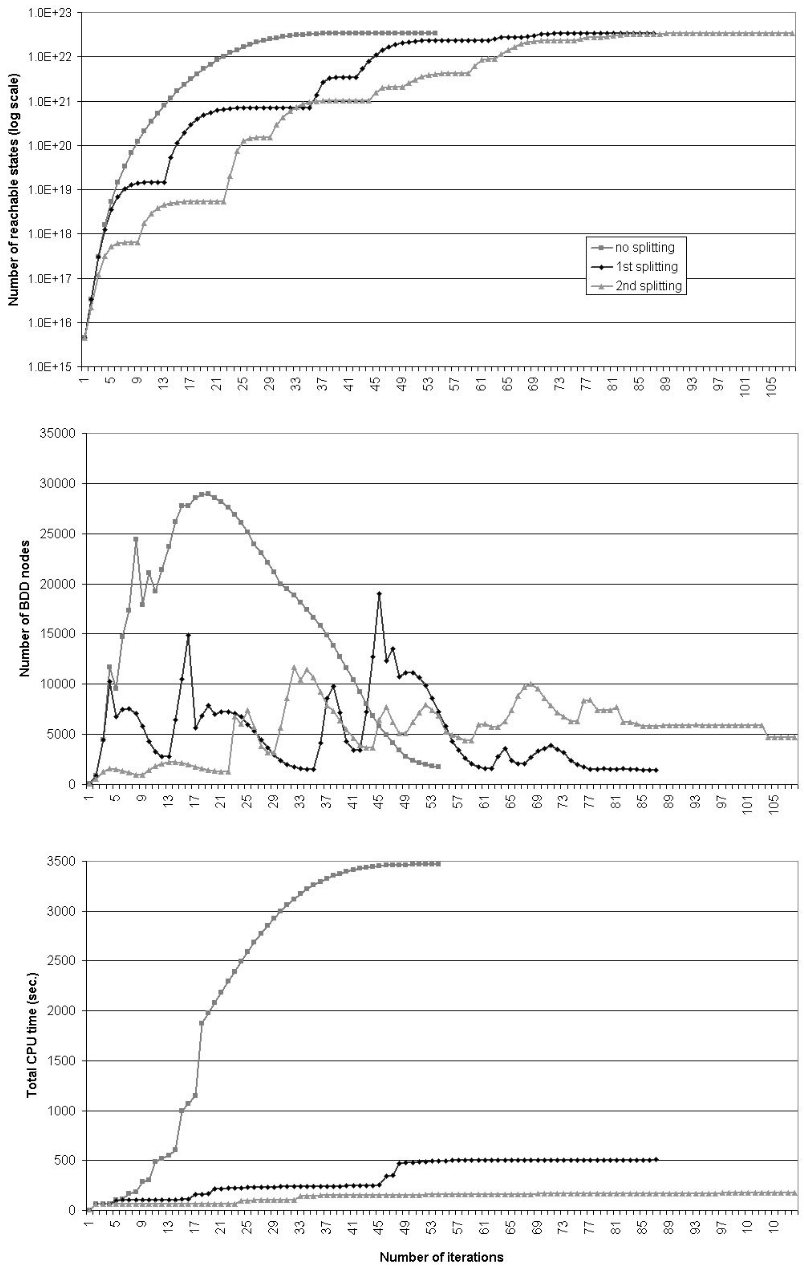

Fig. 4. From top to bottom: number of states found, size of the OBDD representing the set of these states, and total execution time after each iteration of reachability analysis. The three data series are for the original transition relation; with transitions split into route setting and sub-route release; and with additional splitting according to the direction.

next route setting again, and so on). Although the total number of iterations necessary to find all reachable states grew to 87, OBDD sizes have been reduced resulting in individual iterations executing much faster. Finally, the second splitting of both route setting and sub-route releasing transitions according to direction further decreases peak OBDD size and run-time, although it increases the number of iterations to 108. Concerning the OBDD size, it is important to take into account that computations during reachability analysis combine two OBDDs. The complexity of these operations is proportional to the product of the sizes of the two OBDDs. After each iteration, there is an operation on the OBDDs representing the previous and the current set of reachable states in order to determine whether still new states were found, or whether the search can be terminated. Therefore, runtime is a function of OBDD size squared. Seemingly modest reductions of OBDD sizes, as shown in the middle graph in Fig. 4, can considerably improve run-time, as well as memory requirements while the operations take place.

4.2 Usability

Although the current version of gdlSMV still refers to model checking in many places, the potential to hide the underlying technique in gdlSMV clearly exists. In particular, the following improvements could be implemented easily:

Hiding CTL. In the current version of gdlSMV, the specific instantiation of the safety properties for a given data set have to be expressed in CTL. As the structures of these properties follow simple principles, it would be easily possible to generate them automatically. Simple lists of sub-routes for each track-circuit; routes with their sub-routes in correct order; and points with corresponding track-circuit and sub-routes for each branch would be sufficient as input. Such data could be extracted from design specifications.

Managing counter-examples. There are exceptional circumstances when it is permitted to violate a safety property. For example, opposing routes might be set for shunting. In such situations, the question is no longer whether there exists a state that violates a property, but whether there is such a state other than the permitted exceptions. Currently gdlSMV would just report the property to be violated and would produce one arbitrary counter-example. If this example is a permitted exception, then this result is not satisfactory in that it does not specify whether there are other, unintended violations besides it. For that reason, it would be necessary to verify whether there is a counter-example which does not incorporate the permitted violation. This could be implemented by repeating the backward search from states that violate the safety property with a modified transition relation from which the transitions causing the permitted exceptions are excluded. If this search still leads back to an initial state, then another unintended violation is found and a counter example for it can be generated.

Further improvements to gdlSMV's usability could include the provision of additional information relating results back to the SSI data, such as line numbers in the source file when referring to route requests and sub-route release transitions.

5 Evaluation and Conclusions

Our aim was to develop an integrated verification tool for Geographic Data, which is efficient enough for verifying large data sets, and easy to use by signal engineers who are not experts in the underlying model checking techniques. Compared to previous work, it seems that we made important steps in this direction.

As mentioned earlier, there are two main previous works on model checking SSI data. [11] used CCS for the verification of SSI data. However, as the corresponding model checker in the Concurrency Workbench uses an explicit internal representation of the state transition graph, it was impossible to use it for other than very small data examples. An optimised version using HOL has been tested with data for Leamington Spa. However, no full verification has been carried out. It is understood that the correctness of individual statements has been proved. For the most complex route request, this took almost half an hour. Therefore, a full verification would be unlikely to be more efficient than what has been achieved with gdlSMV. Furthermore, [11] does not mention any attempt to develop the verification techniques towards a tool that could be used in the SSI data production process.

In contrast, [14] did suggest the integration of work on translating Geographic Data to CSP and model checking it, with FDR2 in an automated SSI verification tool [7]. The approach differs from gdlSMV in that it does not construct the full model but only extracts data relevant to the property currently verified. No indication is given of the efficiency of a full verification for the small data set "OpenAlvey", except that checking individual properties is said to complete 'in a matter of seconds'. Compared to these results, the full verification of the five safety properties for all elements in the OpenAlvey in under 1.6 seconds, and the ability to fully verify a scheme as complex as Leamington Spa in gdlSMV seem a considerable improvement.

These results suggest two main conclusions in the wider context of verifying configuration data, and for the application of model checking in general.

Verifying Configuration Data. Model checking has seen successful applications in the verification of hardware designs. Although previous work on model checking configuration data exists, it seems not yet to be in industrial use.

One reason for the success of model checking in hardware verification is that properties of particular types of hardware verification problems often allow for specific optimisation techniques to increase efficiency of model checking. The verification of SSI data in this work has followed the same pattern in that it applied optimisations which are particularly suited to the general characteristics of SSI data. Consider, for example, the fact that from any state, a large number

of states are reachable in one transition, and any other state is reachable in a relatively small number of transitions. Whereas the latter is an advantage in reachability analysis as it limits the number of iterations necessary to find all states, the former is a problem as it leads to complicated representations of the intermediate set of states found during reachability analysis. This characteristic is the reason for the success of splitting the transition relation during reachability analysis, as the increased number of iterations necessary is still small whereas the sizes of intermediate sets are drastically reduced. Systems with a different characteristic could not equally benefit from the technique — in contrast, it might even have adverse effects on some systems.
Since configuration data such as Geographic Data can be expected to have general characteristics which hold for all data sets of that specific type, it is likely that techniques can be found which result in efficient model checking for these particular types of systems. Therefore, as was the case in this work, many encouraging results from hardware verification could potentially be applied to model checking configuration data.

Purpose-Built Model Checkers. As mentioned earlier, essential improvements to gdlSMV's efficiency required direct access to the internal representation of the transition relation. The decision to integrate translation from Geographic Data with the model checker, rather than building a separate translator which would generate an SMV input script, has made it possible to apply these improvements. Furthermore, the same decision opened opportunities for considerable improvements to the tool's usability, as the format of input and output to the model checker can be adapted as required. The model checking technique can be hidden from the user to a great extent
NuSMV has proven to be well-documented, and replacing its parser and compiler has been relatively easy, as its architecture clearly separates them from the actual model checking modules.

The results achieved with gdlSMV suggest that moderate efforts allow an open model checker such as NuSMV to be turned into a purpose-built adaptation for a particular input format. In contrast to translating the verification input to a standard model checking language such as SMV, purpose-built model-checkers have two main advantages:

– Access to internal representations allows for specific optimisations appropriate for the general structure of the systems to be verified. Although recent model checkers do provide techniques to increase their efficiency, some relatively simple and powerful techniques such as splitting of the transition relation for reachability analysis in gdlSMV are not applicable without direct access to the internal representation of the transition relation.
– Both input of verification properties and output of results and counter-examples can be tailored to the application area if the model checker itself is modified. Modest efforts in this area can significantly improve usability.

For these reasons, it is suggested that purpose-built model checkers should be considered for areas where an efficient, self-contained and usable tool for the verification of inputs with particular characteristics is required.

References

1. Bailey, C. (ed.): European Railway Signalling. Institution of Railway Signal Engineers. A & C Black, London (1995)
2. Cimatti, A., Clarke, E., Giunchiglia, F. and Roveri, M.: NuSMV: a new Symbolic Model Checker. Springer: International Journal on Software Tools for Technology Transfer, Volume 2 Issue 4, pp410–425 (2000)
3. Cabodi, G., Camurati, P. and Quer, S.: Reachability analysis of large circuits using disjunctive partitioning and partial iterative squaring. Elsevier: Journal of Systems Architecture, Volume 47, Issue 2, pp163–179 (February 2001)
4. Clarke, M.E., Grumberg, O. and Peled, D.A.: Model Checking. MIT Press (1999, 2^{nd} printing 2000)
5. Cribbens, A.H.: Solid State Interlocking (SSI): an Integrated Electronic Signalling System for Mainline Railways. IEE Proceedings, Part B: Electric Power Applications, Volume 134, Issue 3, pp148–158. IEE (1987)
6. Cribbens, A.H. and Mitchell, I.H.: The Application of Advanced Computing Techniques to the Generation and Checking of SSI Data. Institution of Railway Signal Engineers. IRSE Proceedings, Volume 1991/92, pp 54–64 (1991)
7. Gurukumba, T.: From GDL to CSP: towards the full formal verification of solid state interlockings. Oxford University, MSc dissertation (1998)
8. Huber, M.: Towards an Industrially Applicable Model Checker for Railway Signalling Data. York University, MSc Dissertation, Department of Computer Science (2001)
9. Leach, M. (ed.): Railway Control Systems. Institution of Railway Signal Engineers. A & C Black, London (1991, Reprint 1993)
10. McMillan, K.L.: Symbolic Model Checking. Carnegie Mellon University, PhD thesis CMU-CS-92-131 (1992)
11. Morley, M.J.: Safety Assurance in Interlocking Design. Edinburgh University, PhD thesis ECS-LFCS-96-348 (1996)
12. Morley, M.J.: Semantics of Geographic Data Languages. In Proceedings of the 1^{st} FMERail Workshop; Breukelen, Netherlands (June 1998)
13. Raili, E.L.: The Verification of the Design of Railway Networks. York University, MSc Dissertation, Department of Computer Science (1996)
14. Simpson, A.: Model Checking for Interlocking Safety. In Proceedings of the 2^{nd} FMERail Workshop, Canary Wharf, London (October 1998)
15. SSI 8003: SSI Applications Manual. London: Railtrack, Head of Corporate Standards, electronic copy (February 1999)
16. Westerman, S.J., Shryane, N.M., and Sauer, J.: Task Analysis of the Solid State Interlocking Design Process. Human Factors in the Design of Safety Critical Systems project, Work package 1.1, Report No. SCS-01. University of Hull, Department of Psychology (April 1994)
17. Westerman, S.J., Shryane, N.M., Crawshaw, C.M. and Hockey, G.R.J.: Error Analysis of the Solid State Interlocking Design Process. Human Factors in the Design of Safety Critical Systems project, Work package 1.2, Report No. SCS-04. University of Hull, Department of Psychology (June 1995)