Electronic Communications of the EASST
Volume X (2012)

Proceedings of the 12th International Workshop on Automated Verification of Critical Systems (AVoCS 2012)

Railway modelling in CSP||B: the double junction case study

Faron Moller, Hoang Nga Nguyen, Markus Roggenbach,
Steve Schneider and Helen Treharne

15 pages

Guest Editors: Gerald Lttgen, Stephan Merz
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Railway modelling in CSP||B: the double junction case study

Faron Moller[1], Hoang Nga Nguyen[1], Markus Roggenbach[1], Steve Schneider[2] and Helen Treharne[2]

[1] Swansea University, Wales

[2] University of Surrey, England

Abstract: This paper extends recent work in verifying railway systems through CSP||B modelling and analysis. Our motivation is to develop a modelling and verification approach accessible to railway engineers: it is vital that they can validate the models and verification conditions, and — in the case of design errors — obtain comprehendable feedback. In this paper we run through a full production cycle on a real double junction case study, supplied by our industrial partner, who contributed at every stage. As our formalization is, by design, near to their way of thinking, they are comfortable with it and trust it. Without putting much effort on optimization for verification, the scale of the models analyzed is comparable with the work of other groups.

Keywords: Railway verification, CSP||B, modelling and analysis, ProB.

1 Introduction

Formal verification of railway control software has been identified as one of the "Grand Challenges" of Computer Science [Jac04]. But in respect of this challenge, a question has been asked by the community: "Where do the axioms come from?" [Fia12]. Bluntly expressing a view common to the Formal Methods community, Paulson states, "I have seen many pieces of work spoilt by unrealistic models, incorrect axioms or proofs of irrelevant properties" [Pau12]. The modelling of systems, as well as of proof obligations, needs to be *faithful*.

Our paper reports on a case study in which railway engineers and computer scientists together undertake the exercise of domain engineering and formal modelling. By involving the railway engineers from the start, we benefit twofold: they provide a realistic case study – a double junction – and they guide the modelling approach, ensuring that it is natural to the working engineer. In the context of railways, the double junction case study represents a typical railway node, commonly found on any railway network. The rules defining its operation are of moderate size. The exercise helps to confirm the naturalness of our approach, beyond the previous toy examples we have carried out, e.g., the mini-alvey benchmark from the literature, see [MNR+12].

Here, we use CSP||B [ST05], which combines event-based with state-based modelling. This reflects the double nature of railway systems, which involves events such as train movements and, in the interlocking, state based reasoning. In this sense, CSP||B offers the means for the natural modelling approach we strive for: the formal models are close to the domain models. To the domain expert, this provides traceability and ease of understanding. Having defined models

which the railway engineers are confident are faithful, the verification tool ProB [LB08] allows us to analyze the obtained models for safety and liveness and provides meaningful counter example traces if appropriate.

Outline We first introduce the double junction. Then we briefly discuss our modelling language CSP||B. In Section 4 we recall our generic modelling approach as described in [MNR+12] and discuss its advancement for the double junction. This generic model is then instantiated with the double junction, see Section 5. Finally, we carry out a number of verification experiments: these establish that the double junction is safe, and that mistakes in the tables describing its control lead to error traces meaningful to railway engineers. In Section 7 we put our contribution in the context of related approaches, and Section 8 concludes the paper.

2 The double junction challenge

In this paper, we model the double junction which was set as a challenge by our industrial partner. Figure 1 depicts the *scheme plan* for the double junction, which comprises of a track plan, a control table, and release tables. This case study contains features that have not been present in our previous CSP||B railway modelling: the related points (the flank protection, and the treatment of P103 and P104 as a unit) and the fact that the system is open (i.e., contains entries and exits). These aspects provide a number of new modelling challenges for our application of CSP||B.

The *track plan* provides the topological information of the double junction. As given to us by our industrial partner, it consists of 20 tracks (e.g., the track AA), six signals (S2, S3, S4, S5, S16, and S17), and four points (P101, P102, P103, and P104). In order to protect its open ends, we add three further signals (S1, S8, and S19). Furthermore, we add entry and exit tracks on which trains can "appear" and "disappear". This extended track plan, though more complex, allows us to study safety covering *all* tracks of the original plan.

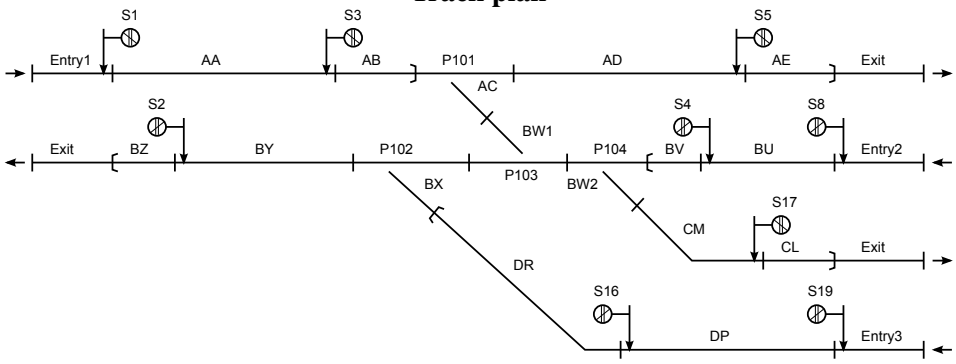
An interlocking system gathers train locations, and sends out commands to control signal aspects and point positions. The *control table* determines how the double junction interlocking system sets signals and points. For each signal, there is one row describing the condition under which the signal can show proceed. There are two rows for signal S3: one for the main line (Route 3A) and one for the side line (Route 3B). For example, signal S3 for the main line can only show proceed when point P101 is in normal (straight) position and tracks AB, AC, AD and AE all are clear.

The interlocking also allocates *locks* on points to particular route requests to keep them locked in position, and releases such locks when trains have passed. For example, the setting of Route 3B obtains locks on all points, namely P101, P102, P103, and P104, and sets them all to reverse. The locks are released after the train has passed. *Release tables* store the relevant track. For example, the lock 3B can be removed on point P101 when the train has arrived on track BW1.

In this setting, we consider two safety properties: *collision-freedom* excludes two trains occupying the same track; and *no-derailment* says that whenever a train enters a point, the point is set to cater for this; e.g., when a train travels from track DR to track BY, point P102 is set so that it connects DR and BY (and not BW1 and BY).

The correct design for the control table and release tables is safety-critical: mistakes can lead to collision or derailment. Our verification approach provides counter-example traces in these

Track plan



Control table

Route	Normal	Reverse	Clear
3A	P101		AB, AC, AD, AE
3B		P101 P102* P103 P104	AB, AC, BW1, BW2, CM, CL
4A	P101* P102 P103 P104		BV, BW2, BW1, BX, BY, BZ
16A		P102	DR, BX, BY, BZ

** flank protection*

Release tables

P101	Occupied
3A	AD
3B	BW1
4A	BX

P102	Occupied
3B	CM
4A	BY
16A	BY

P103	Occupied
3B	CM
4A	BX

P104	Occupied
3B	CM
4A	BX

Figure 1: Scheme plan of the double junction.

cases – see Section 6.

3 CSP||B

The CSP||B approach [ST05] is an approach that allows us to specify communicating systems using a combination of the B-Method [Abr96] and the process algebra CSP (Communicating Sequential Processes) [Hoa85]. The overall specification of a combined communicating system is comprised of two separate specifications: one given by a number of CSP process descriptions and the other by a collection of B machines. Our aim when using B and CSP is to factor out as much of the "data-rich" aspects of a system as possible into B machines. The B machines in our CSP||B approach are classical B machines, which are components containing state and operations on that state. The CSP||B theory allows us to combine a number of CSP processes

Ps in parallel with machines *Ms* to produce $Ps \parallel Ms$ which is the parallel combination of all the controllers and all the underlying machines. Such a parallel composition is meaningful because a B machine is itself interpretable as a CSP process whose event-traces are the possible execution sequences of its operations. The invoking of an operation of a B machine outside its precondition within such a trace is defined as divergence [Mor90]. Therefore, our notion of consistency is that a combined communicating system $Ps \parallel Ms$ is *divergence-free* and also *deadlock-free*.

A B machine consists of a collection of clauses and a collection of operations. The MACHINE clause declares the abstract machine and gives its name. The VARIABLES clause declares the variables that are used to carry the state information within the machine. The INVARIANT clause gives the type of the variables, and more generally it also contains any other constraints on the allowable machine states. The INITIALISATION clause determines the initial state of the machine. Operations are given in the format

$$oo \longleftarrow op(ii) = \textbf{PRE}\ P\ \textbf{THEN}\ S\ \textbf{END}$$

The declaration $oo \longleftarrow op(ii)$ introduces the operation: it has name *op*, a (possibly empty) output list of variables *oo*, and a (possibly empty) input list of variables *ii*. The precondition of the operation is predicate *P*. This must give the type of any input variables, and can also give conditions on when the operation can be invoked. If it is invoked outside its precondition then divergence results. Finally, the body of the operation is *S*. This is a *generalised substitution*, which can consist of one or more assignment statements (in parallel) to update the state or assign to the output variables. Conditional statements and nondeterministic choice statements are also permitted in the body of the operation. In combined communicating systems we also define B machines that do not have operations and only contain sets, constants and invariants. These are included in order to provide contextual information to a system.

The language we use to describe the CSP processes for B machines is as follows:

$$P ::= e?x!y \rightarrow P(x) \mid P_1 \,\Box\, P_2 \mid P_1 \sqcap P_2 \mid \textbf{if}\ b\ \textbf{then}\ P_1\ \textbf{else}\ P_2\ \textbf{end} \mid N(exp) \mid P_1 \parallel P_2 \mid P_1\ {}_A\|_B\ P_2 \mid P_1 \,|||\, P_2$$

The process $e?x!y \rightarrow P(x)$ defines a channel communication where *x* represents all data variables on a channel, and *y* represents values being passed along a channel. Channel *e* is referred to as a *machine channel* as there is a corresponding operation in the controlled B machine with the signature $x \longleftarrow e(y)$. Therefore the input of the operation *y* corresponds to the output from the CSP, and the output *x* of the operation to the CSP input. Here we have simplified the communication to have one output and one input but in general there can be any number of inputs and outputs. The external choice, $P_1 \,\Box\, P_2$, is initially prepared to behave either as P_1 or as P_2, with the choice being made on occurrence of the first event in the environment. The internal choice, $P_1 \sqcap P_2$, is similar, however, the choice is made by the process rather than the environment. Another form of choice is controlled by the value of a boolean expression in an **if** expression. $N(exp)$ is a call to a process where *N* is the process name and *exp* is an expression. The synchronous parallel operator, $P_1 \parallel P_2$, executes P_1 and P_2 concurrently, requiring them to synchronize on all events. The alphabetized parallel operator, $P_1\ {}_A\|_B\ P_2$, requires synchronisation only in $A \cap B$, allowing independent performance of events outside this set. Finally, the interleaving operator, $P_1 \,|||\, P_2$, allows concurrent processes to execute completely independently.

4 The general modelling approach

Together with railway engineers we developed a common view on the information flow in railways: physically, a railway consists of (at least) four different components; see Figure 2. The *Controller* selects and releases routes for trains. The *Interlocking* serves as a safety mechanism with regards to the Controller and, in addition, controls and monitors the Track equipment. The *Track equipment* consists of elements such as signals, points, and track circuits: signals can show proceed or halt; points can be in normal position (leading trains straight ahead) or in reverse position (leading trains to a different line) and track circuits detect if there is a train on a track. Finally, *Trains* have a driver who determines their behaviour. For simplicity, we make the (unrealistic) assumption that track equipment reacts instantly and is free of defects.

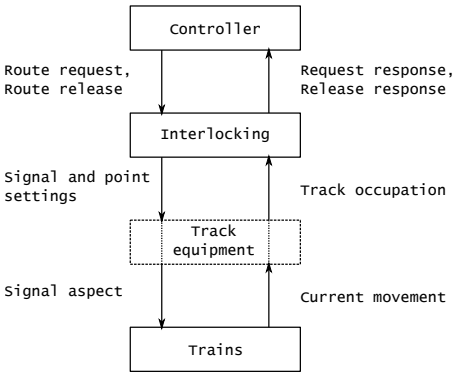


Figure 2: Information flow.

The information flow shown in Figure 2 suggests that railways should be modelled in an *event-based* way: the controller sends a request message to the interlocking to which the interlocking responds; the interlocking sends signalling information to the trains; and the trains inform the interlocking about their movements. The interlocking serves as the system's clock: messages can be exchanged once per cycle.

$$
\begin{array}{l}
RW\_CTRL = \\
\quad \sqcap_{r \in ROUTE}\ (request!r?b \rightarrow RW\_CTRL) \\
\quad \sqcap \\
\quad \sqcap_{r \in ROUTE}\ (release!r?b \rightarrow RW\_CTRL) \\
\\
TRAIN\_OFF(t) = enter!t?newp \rightarrow TRAIN\_CTRL(t,newp) \\
\\
TRAIN\_CTRL(t,pos) = \\
\quad pos \notin EXIT \wedge pos \in SIGNALHOMES\ \&\ nextSignal!t?aspect \rightarrow \\
\qquad \textbf{if}\ aspect == green \\
\qquad \textbf{then} \\
\qquad\quad move!t.pos?newp \rightarrow TRAIN\_CTRL(t,newp) \\
\qquad\quad \sqcap \\
\qquad\quad stay!t.pos \rightarrow TRAIN\_CTRL(t,pos) \\
\qquad \textbf{else} \\
\qquad\quad stay!t.pos \rightarrow TRAIN\_CTRL(t,pos) \\
\quad \Box \\
\quad pos \notin EXIT \wedge pos \notin SIGNALHOMES\ \& \\
\qquad move!t.pos?newp \rightarrow TRAIN\_CTRL(t,newp) \\
\qquad \sqcap \\
\qquad stay!t.pos \rightarrow TRAIN\_CTRL(t,pos) \\
\quad \Box\ \ldots \\
\\
ALL\_TRAINS = |||_{t \in TRAIN}\ TRAIN\_OFF(t) \\
\\
CTRL = RW\_CTRL\ |||\ ALL\_TRAINS
\end{array}
$$

Figure 3: CSP control processes for Controller and Trains.

The control and release tables as shown in Figure 1 as well as their processing in the interlocking – see the discussion of Figure 4 below – however suggest that railways should be modelled in an *state-based* way: if points are in a certain state and tracks are in the state "clear," then the signal controlling a route can be set to proceed; if some track is "clear", then a lock can be removed from a point.

CSP||B caters for this double nature of railways: the interlocking as the "data-rich" component is modelled as a single, dynamic B machine, the *Interlocking* machine. It represents the centralized control logic of a rail node, which reacts to its environment without taking any initiative. The *Interlocking* machine offers to perform events in the form of operations to the two active system components: the controller and the trains, both of which are modelled as CSP processes.

Trains and Controller run independently of each other, on the CSP level expressed with an interleaving operator – see Figure 3 (lines 24 and 26). It is an internal decision of the controller which routes are requested to be set or to be released (lines 2, 3, and 4). Similarly, it is an internal decision of the train (driver) to stay or to move in front of a green signal (line 13) or when there is no signal (line 20). However, a train has to stop in front of a red signal (line 16). This logic is sometimes referred to as the *driving rules* of a train.

```
1   bb ⟵ release(route) =
2    PRE route ∈ ROUTE THEN
3        LET emptyTracksBEemptyTracks = TRACK \ ran(pos) IN
4            IF
5                /* the signal of the route is green */
6                signalStatus(signal(route)) = green∧
7                /* points locked for the route */
8                currentLocks[route] = lockTable[route]∧
9                /* the route is clear */
10               clearTable(route) ⊆ emptyTracks∧
11               /* no train is in the track preceding the route (i.e. nothing close enough to go through the red light ) */
12               homeSignal(signal(route)) ∈ emptyTracks
13           THEN
14               /* signal of route to red */
15               signalStatus(signal(route)) := red ||
16               /* release the locks associated with the route */
17               currentLocks := route ⩤ currentLocks ||
18               /* release is successful */
19               bb := yes
20           ELSE
21               bb := no
22           END
23       END
24   END
```

Figure 4: *release* operation from *Interlocking*.

The *Interlocking* machine captures information about the location of trains on tracks using the *pos: TRAIN ⇸ ALLTRACK* function. In Section 6 we will discuss the reason for this weak invariant and its impact on safety verification. The machine also captures the current information about successor tracks through a dynamic function *nextd* which is dependent upon the position of the points. Furthermore, the machine captures information about signal settings using the function *signalStatus* and point settings using the sets: *normalPoints* and *reversePoints*. Finally, the current locks on points are modelled using *currentLocks*. The initial state of the model sets all tracks to being empty, all signals to red, all points to the normal position and no locks are made

on points. This dynamic state is then updated and queried, respectively, in the four operations of the *Interlocking* machine.

Figure 4 shows the full B code of a typical operation of the *Interlocking* machine. It describes how a release request from the controller is processed. The release is granted provided a number of conditions is fulfilled (the signal of the route is green, line 6, the are points locked for the route, line 8, etc.). In such a case, a number of state changes are made (the signal of the route is set to red, line 15, etc.) and the controller is notified with a "yes" (line 19). Otherwise, the state does not change and the controller is notified with a "no".

$$\begin{array}{rcll} normalTable & \in & ROUTE \leftrightarrow POINTS & \wedge \\ reverseTable & \in & ROUTE \leftrightarrow POINTS & \wedge \\ clearTable & \in & ROUTE \rightarrow \mathbb{P}(TRACK) & \end{array}$$

Figure 5: Generic control table definition from *ControlTable*.

The CSP controller and the *Interlocking* machine are generic and independent of any particular scheme plan. They are supported by a *Topology*, a *ControlTable*, a *ReleaseTable*, and a *Context* machine. These four supporting machines are stateless, and provide generic domain definitions; see Figure 5 for a typical example from the *ControlTable* machine which splits up the modelling of a control table into two relations and one function. Figure 6 shows the overall architecture of our modelling.

As the CSP||B code is easy to read and moreover short, it is actually possible to discuss and to validate it with railway engineers. This is especially useful for discussing the algorithms underlying the four operations of the Interlocking machine which they confirmed to be correct. On their request, we removed an event from our model that should inform the train (driver) that there was no signal ahead. They also confirmed our insight of the dual nature of railways by stating that they actually developed and still use a programming language for interlockings which offers primitives for manipulating both events and states.

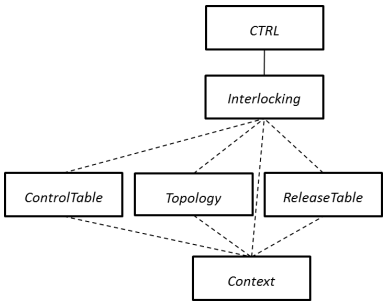


Figure 6: Architecture.

5 Modelling the double junction

Using the generic model presented in Section 4 we now instantiate the domain specific definitions with the scheme plan of the double junction. Technically, this means instantiating the set, relation and function definitions of the four supporting B machines *ControlTable*, *Topology*, *ReleaseTable* and *Context*. We encourage readers to download the complete CSP||B model [1].

We begin with the simplest machine *ClosedContext*, which provides simple set definitions, as shown in Figure 7. We specifically include the *nullTrack* in order to be able to identify the derailment of a train. As seen above in Figure 5, the *ControlTable* machine splits up modelling the control table into two relations and a function. These are now instantiated for the double

[1] CSP||B model download: http://www.csp-b.org/avocs2012-double-junction.zip

$$\begin{array}{rcl} ALLTRACK & = & \{AA, AB, AC, AD, AE, BZ, BY, BX, BW1, BW2, BV, BU, \\ & & \quad CM, CL, DR, DP, Entry1, Entry2, Entry3, Exit, nullTrack\}; \\ TRACK & = & ALLTRACK - \{nullTrack\} \end{array}$$

Figure 7: Datatype definitions for tracks defined in *ClosedContext*.

junction, e.g., see the instantiation of clearTable in Figure 8[2]. The *ReleaseTable* machine is instantiated in a similar way.

$$\begin{array}{rcl} clearTable & = & \{A1 \mapsto \{AA, AB\}, A2 \mapsto \{BZ\}, A3 \mapsto \{AB, AC, AD, AE\}, \\ & & \quad B3 \mapsto \{AB, AC, BW1, BW2, CM, CL\}, \\ & & \quad A4 \mapsto \{BV, BW1, BW2, BX, BY, BZ\}, \\ & & \quad A5 \mapsto \{AE\}, A8 \mapsto \{BU, BV\}, A16 \mapsto \{DR, BX, BY, BZ\}, \\ & & \quad A17 \mapsto \{CL\}, A19 \mapsto \{DP, DR\}\} \end{array}$$

Figure 8: The clear table information as defined in *ControlTable*.

The *Topology* machine models which signals are associated with a route, the track where a signal is situated, the track where points are situated and relations between tracks and possible successor tracks. For example, Figure 9 illustrates the relation *homeSignal* and instantiates it for the double junction example stating the signal *S3* is associated with routes *3A* and *3B* and that it is situated on track *AA*. The topology also provides all possible pairs of tracks and their successor tracks.

Comparing the instantiation of our model with the double junction with the scheme plan given in Figure 1 is straightforward. The railway engineers can easily work with the generic domain definitions. The concrete data needed for the instantiation could, in principle, be automatically derived from scheme plans in data formats used in the rail industry.

6 Verification

In this section we focus on the safety verification approach of our method. We rely on the ProB toolset [LB08] to verify our CSP||B models as it supports B models that are controlled by CSP controllers.

In carrying out our verification, our starting point is:

- that the generic descriptions in the B machines are correct;
- that the driving rules dictating how trains can move in the railway system in the CTRL are correct; and
- that the safety conditions identified by CTL formulae faithfully represent the assumptions identified in Section 2.

However, we make no assumptions about the correctness of the instantiations of the scheme plan in the *Topology*, *ControlTable* and *ReleaseTable*. As such, our verification strategy resonates closely with that of Haxthauhen *et al.* [HPK11].

[2] In ProB [LB08] *8A* is represented as *A8* since ProB does not allow names to begin with a numeral.

$$
\begin{array}{rcl}
signal & \in & ROUTE \rightarrow SIGNAL \quad \wedge \\
signal & = & \{(A1 \mapsto S1), (A2 \mapsto S2), (A3 \mapsto S3), (B3 \mapsto S3), (A4 \mapsto S4), \\
 & & \quad (A5 \mapsto S5), (A8 \mapsto S8), (A16 \mapsto S16), (A17 \mapsto S17), (A19 \mapsto S19)\} \wedge \\
homeSignal & \in & SIGNAL \rightarrowtail TRACK \quad \wedge \\
homeSignal & = & \{(S1 \mapsto Entry1), (S2 \mapsto BY), (S3 \mapsto AA), (S4 \mapsto BU), (S5 \mapsto AD), \\
 & & (S8 \mapsto Entry2), (S16 \mapsto DP), (S17 \mapsto CM), (S19 \mapsto Entry3)\} \quad \wedge \\
next & \in & TRACK \leftrightarrow TRACK \quad \wedge \\
next & = & \{(Entry1 \mapsto AA), (AA \mapsto AB), (AB \mapsto AC), (AC \mapsto AD), (AD \mapsto AE), \\
 & & (AE \mapsto Exit), (AC \mapsto BW1), (BW1 \mapsto BW2), (BW2 \mapsto CM), \\
 & & (CM \mapsto CL), (CL \mapsto Exit), (Entry2 \mapsto BU), (BU \mapsto BV), (BV \mapsto BW2), \\
 & & (BW2 \mapsto BW1), (BW1 \mapsto BX), (BX \mapsto BY), (BY \mapsto BZ), (BZ \mapsto Exit), \\
 & & (Entry3 \mapsto DP), (DP \mapsto DR), (DR \mapsto BX), (Exit \mapsto Exit)\}
\end{array}
$$

Figure 9: Definitions from *Topology*.

Since we do not assume that the tables are correct, we cannot rely on a universal "once-and-for-all" verification; each railway system (such as the double junction) has to be verified with respect to its particular scheme plan – reflecting current practice in rail industry, where a control table and release tables are individually designed for each track plan.

This notwithstanding, we can nonetheless outline a general strategy which can be followed for any specific railway system. This strategy involves the following steps:

1. We first review the *CTRL* process in order to produce the most compact control process possible, which we refer to as *RESTRICTED_CTRL*. This means that we should remove all the events which do not update the B state and hence have no impact on safety verification. Typically, these events will be those that are in the alphabet of *CTRL* and not in the alphabet of *Interlocking* and those events that report a response of *no* or *false*. This step also involves replacing the internal choice operators with external choice operators in order to make the *RESTRICTED_CTRL* process deadlock free – which is justified as they have the same semantics within the traces model which we employ for our safety analysis.

2. We then model check *RESTRICTED_CTRL* $\|$ *Interlocking* for deadlock-freedom and invariant violations. Since the invariants in the B machine do not capture our complete safety check, this step simply gives confidence in the combination.

$$
\begin{array}{l}
collision = \textbf{PRE}\ exists t_1, t_2 \bullet t_1 \in TRAIN \wedge t_2 \in TRAIN \wedge t_1 \in \mathrm{dom}(pos) \wedge t_2 \in \mathrm{dom}(pos) \wedge \\
\qquad pos(t_1) \notin EXIT \wedge pos(t_2) \notin EXIT \wedge t_1 \neq t_2 \wedge pos(t_1) = pos(t_2) \\
\quad \textbf{THEN}\ skip \\
\quad \textbf{END} \\
\\
derailment = \textbf{PRE}\ \exists t \bullet t \in TRAIN \wedge t \in \mathrm{dom}(pos) \wedge pos(t) = nullTrack \\
\quad \textbf{THEN}\ skip \\
\quad \textbf{END}
\end{array}
$$

Figure 10: new *collision* and *derailment* operations in *Interlocking* machine.

3. Finally, we model check *RESTRICTED_CTRL* $\|$ *Interlocking* with respect to safety. In order to do this, we must augment our model with specific error events, *e.g.*, *collision*

	Description	States Checked	Result
1	B machines alone. No CSP: no driving rules, so trains can always move	x	Simple violation of collision
2	B machines and RESTRICTED_CTRL, constrained to reduce state space	240,655	No violation
3	Faulty clear tracks for a route in control table	352,367	Violation trace with 22 events found, leading to a collision
4	Faulty points in control table	20,109	Violation trace with 10 events found, leading to a derailment
5	Alteration to Release Table	85,052	Violation trace with 9 events found, leading to a derailment

Figure 11: Scenarios checked for the double junction.

and *derailment* (as depicted in Figure 10). These are events which should only be enabled when safety is violated. We achieve this by first introducing these operations in the *Interlocking* machine, and then by augmenting the control process *RESTRICTED_CTRL* to be $RESTRICTED\_CTRL \;|||\; RUN(\{collision, derailment\})$. Then safety verification is achieved by checking a CTL formula on the combination. We use the following CTL formulae: $AG(not\ e(collision))$ and $AG(not\ e(derailment))$.

Failure of the CTL checking will enable us to identify errors in control and release tables, and provide meaningful traces to demonstrate this violation.

An alternative approach that comes to mind for verifying safety is to strengthen the *pos* function in the invariant of *Interlocking* to $pos \in TRAIN \rightarrowtail\!\!\!\!\!\!\rightarrow TRACK$; that is, to assert *pos* to be an injective function, thus allowing at most one train on any one track and allowing trains to be moved only to valid tracks. The reason we take the approach of using temporal logic formulae is as follows. When combining CSP processes and B machines in parallel, we require the B machines themselves to be consistent with respect to their invariants. However, the *Interlocking* machine on its own would not be valid with respect to such a stronger invariant. In practice, the temporal logic check gives us the same safety guarantee.

6.1 Safety Verification of the Double Junction

We considered four scenarios, and in each case used ProB to check the CTL formulae representing safety in the model. The analysis results for these four scenarios are summarised in Figure 11.

Scenario 1 - B model only: The B machine contains $2^{20} \times 4^4 \times 2^{10} \times 2^9$ states, since we have 20 tracks, 4 points, 10 routes, and 9 signals. It is not appropriate to consider verifying the safety of a railway system by examining the *Interlocking* machine in isolation because the B model alone does not place any constraints on train movements. Therefore it is no surprise that a collision occurs. The following counter-example trace illustrates that both trains start on $Entry1$ and move through the red light associated with signal $S1$ and then leads *bertie* to collide with *albert* on AA.

$$\langle enter.albert.Entry1, move.albert.Entry1.AA, enter.bertie.Entry1, move.bertie.Entry1.AA \rangle$$

$$removedEvents = \{|\ stay\ |\} \cup \{nextSignal.t.red \mid t \in TRAIN\} \cup \{request.r.no, release.r.no \mid r \in ROUTE\}$$
$$RESTRICTED\_CTRL = (CTRL[\alpha(CTRL) \,||\, removedEvents]Stop) \,|||\, RUN(\{collision, derailment\})$$

Figure 12: Restriction of the *CTRL* process.

Scenario 2 - CSP||B model: We begin by identifying an appropriate restricted controller as shown in Figure 12. Steps 2 and 3 of the verification strategy are then applied, resulting in *all* nodes being checked and the CTL formulae being true. In this case the train driving rules ensure that collisions and derailments do not occur.

Scenario 3 - CSP||B model with faulty clear tracks for a route in control table: Suppose the control table is adjusted to contain a mistake by omitting AC the tracks that should be clear to grant route $3A$. The clear table information in the B machine *ControlTable*, then, is as follows:

$$\begin{array}{rcl} clearTable & = & \{A1 \mapsto \{AA, AB\}, A2 \mapsto \{BZ\}, A3 \mapsto \{AB, AD, AE\}, \\ & & B3 \mapsto \{AB, AC, BW1, BW2, CM, CL\}, \\ & & A4 \mapsto \{BV, BW1, BW2, BX, BY, BZ\}, \\ & & A5 \mapsto \{AE\}, A8 \mapsto \{BU, BV\}, A16 \mapsto \{DR, BX, BY, BZ\}, \\ & & A17 \mapsto \{CL\}, A19 \mapsto \{DP, DR\}\} \end{array}$$

Then a trace with 22 events is produced as a counter-example for the CTL *collision* check, which leads to a collision of the two trains on AD. Since $clearTable(A3)$ does not take AC into consideration, the fact that *albert* is already on AC is ignored. Consequently, as AB, AD and AE are not occupied, the route is granted, then signal $S3$ is set to green and this enables *bertie* to make the moves which lead to the collision.

Scenario 4 - CSP||B model with faulty points in control table: If the control table contains a mistake on the directions of points, then this may also impact on safety. For example, suppose $P101$ is the wrong way around in the control table for route $3A$:

$$\begin{array}{rcl} normalTable & = & \{A4 \mapsto P101, A4 \mapsto P102, A4 \mapsto P103, A4 \mapsto P104\} \quad \wedge \\ reverseTable & = & \{A3 \mapsto P101, B3 \mapsto P101, B3 \mapsto P102, B3 \mapsto P103, B3 \mapsto P104, \\ & & A16 \mapsto P102\} \end{array}$$

Then, by checking the CTL formula for no derailment, ProB gives a counter-example trace showing that *albert* derails when moving from $BW1$, *bertie* is not involved in this trace. *Albert* requests route $3A$ but the points $P101$ direct him to $BW1$, however $P103$ is set to normal causing *albert*'s derailment. Less of the state space needed to be explored before this became apparent to discover generate this trace.

Scenario 5 - CSP||B model with alteration to release table: If the release table contains a mistake so that a lock is released too early, then this may also impact on safety. For example, suppose the lock $P102$ for route $16A$ is now released when a train occupies DR, i.e., $(DR \mapsto (A16, P102)$ is included in the *releaseTable* instead of $(BY \mapsto (A16, P102))$. The CTL derailment check yields a counter-example trace comprised of 9 events where *albert* moves from DR into *nullTrack* as $P102$ is not in the reverse position. This is caused by the fact that (i) $P102$ is released

too early when *albert* is on *DR* and, hence, (ii) there is a request for $A4$ which is successfully granted and moves $P102$ into the normal position.

The railway engineers consider the counter-example traces above as informative, especially in combination with the possibility to explore the whole state space along these traces in ProB. From the engineer's point of view, the above scenarios represent design errors typical in the production of scheme plans.

7 Related work

Our work builds upon prior approaches to the modelling and verification of railways. [BG00, LFFP11, Sab12] are prominent examples from the B community, [SWD97, Mor93] are classical contributions from process algebra, [HP00, JR11b] use techniques from Algebraic Specification. On a lower abstraction layer [FH98, FMGF11, JR11a] verify the safety of interlocking programs with logical approaches.

7.1 Modelling comparison

In this paper we have applied our modelling strategy to the double junction. Our modelling is most related to Winter's approach in CSP [Win02] and Abrial's modelling in Event-B [Abr10].

Winter [Win02] presents a generic, event-based railway model in CSP as well as generic formulations of two safety properties: CollisionFreedom and NoMovingPoints. Overall, this results in a generic architecture and a natural representation of two safety properties. Traceability, however, is limited. There are relations in the model which are *derived* from the control table. For example, the driving rule "trains stop at a red signal" is distributed over different parts of the model: it is a consequence of the fact that (1) the event "move to the first track protected by a signal" belongs to a specific synchronziation set and (2) a red signal does not offer this event. Purely event-based modelling leads to such decentralized control. Consequently, the model has no interlocking cycle.

Chapter 17 of the book by Abrial [Abr10] gives an excellent detailed description and analysis of the railway domain, deriving a total of 39 different requirements. The modelling approach is generic, even though no concrete model is proven to be correct. Traceability in a tower of specifications can be complex for various reasons. For instance, a requirement can be the consequence of invariants from different levels. The relation between intended properties and the model remains an informal one. This is in contrast to other approaches (including Winter's and our own) which directly represent the intended property in the formal world and then prove that the modelled property is a mathematical consequence of the formal model. Furthermore, the approach is monolithic: behaviour is not attached to different entities to which they relate.

The model of the double junction means that a train always occupies one track (segment) only. We recognise that in more complex systems we will need to consider the length of the train, as has been done by Winter *et al.* [WR03], so that a train can occupy more than one segment at the same time.

7.2 Verification comparison

The focus of our paper has been on safety verification using model checking in ProB. Model checking is becoming more recognised as an industrial technique [FG11] and therefore it is important to discuss it in the context of scalability. Ferrari *et al.* [FMGF11] state that model checking large interlocking systems is unfeasible with current state-of-the-art model checkers, in particular SPIN and NuSMV. However, James *et al.* [JR11a] clearly demonstrate better results and the feasibility of the lower level approach involving program slicing. A detailed comparison with these approaches is not appropriate since our approach is at a higher level of abstraction. The justification for this higher level of abstraction is that the industrial partners wish to have feedback on interlocking systems already during the design stage.

Nonetheless, as is the case in [FMGF11], we successfully use CTL formulae to capture safety requirements. The example given in [FMGF11] illustrates that a point should not be moved while a train occupies the track containing the point. In this paper we have not modelled moving points but we would expect to use a similar property in future work.

We recognise that compositional verification will need to be considered when the models do become very large. Winter *et al.* [WR03] already propose some decomposition techniques; optimising an ASM model and using SMV model checking. Our current verification strategy also recognises the need for optimisation. We will also consider, in the future, composing track plans together to define larger ones. To this end, we will explore the possibility of developing new compositional verification techniques in our CSP||B approach to establish that: if different scheme plans invidually preseve safety then their composition will also preserve safety.

8 Conclusion

Through our association with Invensys Rail, we are working towards deriving railway models which are formal and analysable by current verification technologies, yet are fully faithful; we do not want to hide the engineering understandings held by our industrial partners in clever abstract encodings. Despite being expressed in the mathematical language of formal methods, our models must be immediately understandable — and verifiable — by our industrial partners.

This has proven to be a challenge, as we find that the extant approaches to railway modelling have been hindered in this respect by the framework in which they have been carried out. As explained above, modelling in the railway domain involves event-based components as well as state-based components. Using a solely-event-based framework or a solely-state-based framework succeeds in faithfully representing the relevant components, yet suffers in representing other components through encodings which — whilst clever feats of abstract modelling — are not easily appreciated by the working railway engineer.

Future work will extend the analysis to handle emergency stops (trains passing red signals and stopping in the following track segment) by extending the driving rules. We will also include points moving under trains more explicitly in the model. As well as safety, the model is suitable for analysing the capacity of the track plan: the maximum number of trains it can hold without compromising safety, and this will lead to the investigation of extending CSP||B to handle time.

Acknowledgement: The authors would like to thank Simon Chadwick and Dominic Taylor from the company Invensys Rail for their support, contribution and encouraging feedback.

Bibliography

[Abr96] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. CUP, 1996.

[Abr10] J.-R. Abrial. *Modeling in Event-B*. Chapter 17 – Train System. CUP, 2010.

[BG00] J. Boulanger, M. Gallardo. Validation and verification of METEOR safety software. In *Advances in Transport Vol 7*. WIT Press, 2000.

[FG11] A. Fantechi, S. Gnesi. On the adoption of model checking in safety-related software industry. *Computer Safety, Reliability, and Security*, pp. 383–396, 2011.

[FH98] W. Fokkink, P. Hollingshead. Verification of Interlockings: from Control Tables to Ladder Logic Diagrams. In *Proceedings of FMICS'98*. Pp. 171–185. 1998.

[Fia12] J. Fiadeiro. 2012. Personal communication.

[FMGF11] A. Ferrari, G. Magnani, D. Grasso, A. Fantechi. Model Checking Interlocking Control Tables. *FORMS/FORMAT 2010*, pp. 107–115, 2011.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HP00] A. E. Haxthausen, J. Peleska. Formal Development and Verification of a Distributed Railway Control System. *IEEE Trans. Software Eng.* 26(8):687–701, 2000.

[HPK11] A. Haxthausen, J. Peleska, S. Kinder. A formal approach for the construction and verification of railway control systems. *Formal Aspects of Computing* 23(2):191–219, 2011.

[Jac04] R. Jacquart (ed.). *IFIP 18th World Computer Congress, Topical Sessions*. Chapter TRain: The Railway Domain - A Grand Challenge. Kluwer, 2004.

[JR11a] P. James, M. Roggenbach. Automatically Verifying Railway Interlockings using SAT-based Model Checking. In *AVoCS'10*. EASST, 2011.

[JR11b] P. James, M. Roggenbach. Designing Domain Specific Languages for Verification: First Steps. In *ATE-2011*. CEUR-WS.org, 2011.

[LB08] M. Leuschel, M. Butler. ProB: an automated analysis toolset for the B method. *Int. J. Softw. Tools Technol. Transf.* 10(2):185–203, Feb. 2008.

[LFFP11] M. Leuschel, J. Falampin, F. Fritz, D. Plagge. Automated property verification for large scale B models with ProB. *Formal Asp. Comput.* 23(6):683–709, 2011.

[MNR+12] F. Moller, H. Nguyen, M. Roggenbach, S. Schneider, H. Treharne. Combining event-based and state-based modelling for railway verification. Technical report CS-12-02, University of Surrey, 2012.

[Mor90] C. C. Morgan. Of wp and CSP. In Feijen et al. (eds.), *Beauty is our Business: a birthday salute to Edsger J. Dijkstra*. Springer-Verlag, 1990.

[Mor93] M. J. Morley. Safety in Railway Signalling Data: A Behavioural Analysis. In *6th International Workshop on HOLTPA*. Springer, 1993.

[Pau12] L. Paulson. 2012. Isabelle user-list.

[Sab12] S. Sabatier. Formal proofs for the NYCT line 7 (Flushing) modernization project. In *ABZ*. P. to appear. 2012.

[ST05] S. Schneider, H. Treharne. CSP theorems for communicating B machines. *Formal Asp. Comput.* 17(4):390–422, 2005.

[SWD97] A. Simpson, J. Woodcock, J. Davies. The mechanical verification of solid-state interlocking geographic data. In *Formal Methods Pacific 97*. Springer, 1997.

[Win02] K. Winter. Model checking railway interlocking systems. *Australian Computer Science Communications* 24(1), 2002.

[WR03] K. Winter, N. Robinson. Modelling large railway interlockings and model checking small ones. In *Proceedings of the 26th Australasian computer science conference-Volume 16*. Pp. 309–316. 2003.