Science of Computer Programming 96 (2014) 315–336

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

On modelling and verifying railway interlockings: Tracking train lengths

Phillip James[a], Faron Moller[a,*], Hoang Nga Nguyen[a], Markus Roggenbach[a], Steve Schneider[b], Helen Treharne[b]

[a] Swansea University, Wales, UK
[b] University of Surrey, England, UK

ARTICLE INFO

Article history:
Received 22 March 2013
Received in revised form 29 March 2014
Accepted 7 April 2014
Available online 18 April 2014

Keywords:
Railway verification
CSP||B
Modelling and analysis

ABSTRACT

The safety analysis of interlocking railway systems involves verifying freedom from collision, derailment and run-through (that is, trains rolling over wrongly-set points). Typically, various unrealistic assumptions are made when modelling trains within networks in order to facilitate their analyses. In particular, trains are invariably assumed to be shorter than track segments; and generally only a very few trains are allowed to be introduced into the network under consideration.

In this paper we propose modelling methodologies which elegantly dismiss these assumptions. We first provide a framework for modelling arbitrarily many trains of arbitrary length in a network; and then we demonstrate that it is enough with our modelling approach to consider only two trains when verifying safety conditions. That is, if a safety violation appears in the original model with any number of trains of any and varying lengths, then a violation will be exposed in the simpler model with only two trains.

Importantly, our modelling framework has been developed alongside – and in conjunction with – railway engineers. It is vital that they can validate the models and verification conditions, and – in the case of design errors – obtain comprehensible feedback. We demonstrate our modelling and abstraction techniques on two simple interlocking systems proposed by our industrial partner. As our formalization is, by design, near to their way of thinking, they are comfortable with it and trust it.

1. Introduction

Formal verification of railway control software has been identified as one of the Grand Challenges of Computer Science [1]. As is typical with Formal Methods, this challenge comes in two parts: the first addresses the question of whether the mathematical models considered are legitimate representations of the physical systems of concern. The modelling of the systems, as well as of proof obligations, needs to be *faithful*. The second part is the question of how to utilize available technologies, for example model checking or theorem proving. Whichever verification process is adopted, it needs to be both *effective* and *efficient*.

In a series of papers [2–5] we have been developing a new modelling approach for railway interlockings. This work has been carried out in conjunction with railway engineers drawn from our industrial partner Invensys Rail. By involving the railway engineers from the start, we benefit twofold: they provide realistic case studies, and they guide the modelling approach, ensuring that it is natural to the working engineer.

* Corresponding author.

http://dx.doi.org/10.1016/j.scico.2014.04.005
0167-6423/© 2014 Elsevier B.V. All rights reserved.

We base our approach on CSP‖B [6], 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 by design close to the domain models. To the domain expert, this provides traceability and ease of understanding. This addresses the first of the above stated challenges: *faithful* modelling. The validity of this claim was demonstrated in particular in [2] where a non-trivial case study – a complex double junction – was provided which was understandable and usable by our industrial partners.

In [3] we addressed the second challenge: that of how to *effectively* and *efficiently* verify safety properties within our CSP‖B models. To this end we developed a set of abstraction techniques for railway verification that allow the transformation of complex CSP‖B models into less involved ones; we proved that these transformations are sound; and we demonstrated that they allow one to verify a variety of railway systems via model checking. The first set of abstractions allows us to prove safety of a scheme plan which involves an unbounded number of trains by considering only a bounded number of trains with the number dependent only on the number of routes in the scheme plan. Their correctness proof involves slicing of event traces. Essentially, these abstractions provide us with finite state models. The second set of abstractions simplifies the underlying track topology. Here, the correctness proof utilizes event abstraction specific to our application domain similar to the ones suggested by Winter in [7]. These abstractions make model checking faster.

Still present in these approaches, however, are unrealistic assumptions about trains within networks: namely that the trains are shorter than the track segments in the network, and that only a very few trains will ever enter the network. In this paper we address these unrealistic assumptions. Firstly, we develop a modelling approach which incorporates train and track lengths, allowing trains to span any number of track segments. Secondly, we provide an abstraction technique which allows us to detect safety violations in networks involving an arbitrary number of trains by considering only two trains (thus markedly improving on our previous result).

The paper is organised as follows. In Section 2 we discuss our modelling language CSP‖B. In Section 3 we introduce railway concepts and our two case studies, and describe how they are modelled in CSP‖B. In particular, we outline in detail the modelling of train and track lengths. In Section 4 we present our main result that considering two trains suffices in our analyses for safety properties. The application of our approach is presented in Section 5 via verification of our example scenarios. Finally, in Section 6 we put our work in the context of related approaches.

2. Background to CSP‖B

The CSP‖B approach allows us to specify communicating systems using a combination of the B Method [8] and the process algebra CSP (Communicating Sequential Processes) [9]. The overall specification of a combined communicating system comprises 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 [6] allows us to combine a number of CSP processes *Ps* in parallel with machines *Ms* to produce *Ps* ‖ *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 [10]. Therefore, our notion of consistency is that a combined communicating system *Ps* ‖ *Ms* is *divergence-free* and also *deadlock-free* [6].

A B machine consists of a collection of clauses and a collection of operations that query and modify the state. 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 of a B machine are given in one of the following formats:

preconditioned operation – $oo \longleftarrow op(ii) =$ **PRE** P **THEN** S **END**: if this is called when P holds then it will execute S, otherwise it will diverge.
guarded event – $op =$ **SELECT** P **THEN** S **END**: this will execute S when P holds, and will *block* when P is false.

The declaration $oo \longleftarrow op(ii)$ for preconditioned operation 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. The guarded event simply has a name *op*. If its condition fails, then its execution is blocked rather than leading to a divergence.

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:

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

The process Stop does not engage in any events, it represents deadlock. 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 B operation y corresponds to the output from the CSP, and the output x of the B 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. The synchronous parallel operator, $P_1 \parallel P_2$, executes P_1 and P_2 concurrently, requiring them to synchronise 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. The interleaving operator, $P_1 ||| P_2$, allows concurrent processes to execute completely independently. Finally, $N(exp)$ is a call to a process where N is the process name and exp is an expression.

It should be noted that the syntax we present for the purpose of this work only allows communication events between a CSP process (modelling the controller) and a B machine (which it is controlling). CSP processes cannot communicate between themselves apart from the possibility of synchronising when communicating with a B machine. (In general CSP‖B, the CSP processes can communicate with each other.)

For reasoning with CSP‖B models we require the following notation:

- A system run σ (of a CSP‖B model) of length $n \geq 0$ is a finite sequence

  $$\sigma = \langle s_0, e_0, s_1, e_1, \ldots, e_{n-1}, s_n \rangle$$

  where the s_i, $i = 0 \ldots n$, are states of the B machine, and the e_i, $1 \leq i \leq n-1$, are events – either controlled by CSP and enabled in B when called, or B events. Here we assume that s_0 is a state after initialisation. Given a system run σ, we can extract its trace of events:

  $$events(\sigma) = \langle e_0, \ldots, e_{n-1} \rangle.$$

  To demonstrate consistency of the combined CSP‖B model we must consider every sequence of events in a system run σ that correspond to a single pass through the recursive definition of the CSP processes and verify that the matching sequence of B operations are called within their preconditions. In [6] we provided a general proof obligation that characterised this notion of successful termination for sequences of operations. When this obligation is discharged for a particular CSP‖B model this verifies the divergence-freedom of the combined system. In practice the proof obligation requires the identification of a control loop invariant which is a predicate between the variables of the B model and the parameters within the CSP processes and also predicates which must hold of the B model. Proof obligations in CSP‖B have also been defined to characterise the condition for deadlock freedom [6]. In this paper we need not concern ourselves with ensuring deadlock freedom of the combined model since we only use events/operations which could give rise to a deadlock in the encoding of safety in Section 3.4.
- Given a trace of events tr we define its projection to a given set A: $\langle\rangle \upharpoonright A = \langle\rangle$; and

  $$\big(\langle e \rangle \frown t\big) \upharpoonright A = \begin{cases} \langle e \rangle \frown (t \upharpoonright A); & e \in A \\ t \upharpoonright A; & e \notin A \end{cases}$$

3. Modelling railways in CSP‖B

Together with railway engineers, we have developed a common view of the information flow in railways. In physical terms, for our purposes we consider a railway as consisting of (at least) the four different components shown in Fig. 1.

- 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 the aspects *green* or *red*; 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.

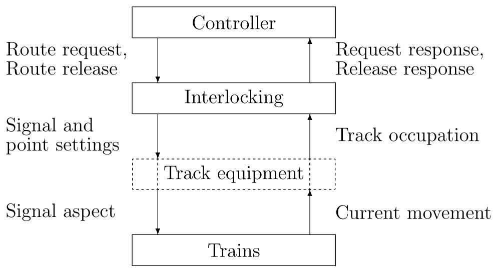
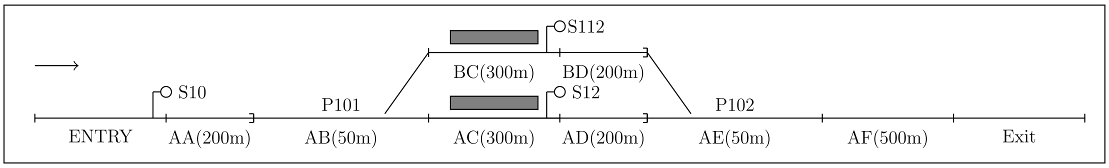


Fig. 1. Information flow.



Control table

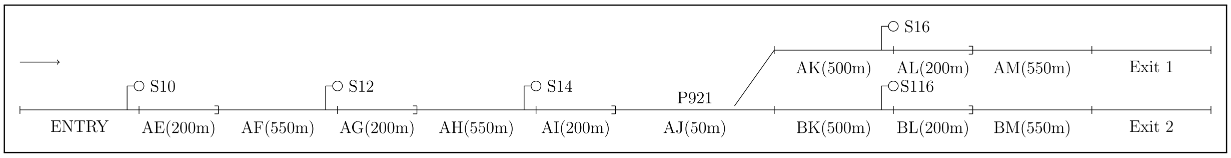
Route	Normal	Reverse	Clear
$R10A$	$P101$		AA, AB, AC, AD
$R10B$		$P101$	AA, AB, BC, BD
$R12$	$P102$		AD, AE, AF
$R112$		$P102$	BD, AE, AF

Release tables

$P101$	Occupied
$R10A$	AC
$R10B$	BC

$P102$	Occupied
$R12$	AF
$R112$	AF

Fig. 2. Station scheme plan.



Control table

Route	Normal	Reverse	Clear
$R10$			AE, AF, AG
$R12$			AG, AH, AI
$R14A$		$P921$	AI, AJ, AK, AL
$R14B$	$P921$		AI, AJ, BK, BL
$R16$			AL, AM
$R116$			BL, BM

Release table

$P921$	Occupied
$R14A$	AK
$R14B$	BK

Fig. 3. Single junction scheme plan.

For the purposes of modelling, we make the assumption that track equipment reacts instantly and is free of defects.

The information flow shown in Fig. 1 is as follows: 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.

In this paper, we study two example track plans, one of which is a station illustrated in Fig. 2, the other being a single junction illustrated in Fig. 3. In both cases, the figures depict the *scheme plans* for the examples, each comprising of a track plan, a control table, and release tables. (Scheme plans and the various tables are provided as standard entities by the railway industry, and it is our task to provide models which faithfully capture the behaviour associated with these.) We explain our modelling approach here with reference to the station example of Fig. 2. In general, we adhere closely to the established principles laid out in [11]. The *track plan* provides the topological information of the station which consists of 8 tracks (e.g., the track AA), three signals (e.g., S10), and two points (e.g., P101). Note that the tracks include entry and exit tracks on which trains can "appear" and "disappear". These two kinds of tracks are specially treated during verification.

```
1   RW_CTRL =
2         ⊓r∈ROUTE (request!r?b → RW_CTRL)
3       ⊓
4         ⊓r∈ROUTE (release!r?b → RW_CTRL)
5
6   TRAIN_OFF(t) = enter!t?newp → TRAIN_CTRL(t, newp)
7
8   TRAIN_CTRL(t, pos) =
9       pos ∉ EXIT ∧ pos ∈ SIGNALHOMES & nextSignal!t?aspect →
10          if aspect == green
11          then
12              move!t.pos?newp → TRAIN_CTRL(t, newp)
13              ⊓
14              stay!t.pos → TRAIN_CTRL(t, pos)
15          else
16              stay!t.pos → TRAIN_CTRL(t, pos)
17              ⊓
18              move!t.pos?newp → Stop
19      □
20      pos ∉ EXIT ∧ pos ∉ SIGNALHOMES &
21          move!t.pos?newp → TRAIN_CTRL(t, newp)
22          ⊓
23          stay!t.pos → TRAIN_CTRL(t, pos)
24      □ ...
25
26  ALL_TRAINS = |||t∈TRAIN TRAIN_OFF(t)
27
28  CTRL = RW_CTRL ||| ALL_TRAINS
```

Fig. 4. CSP control processes for Controller and Trains.

An interlocking system gathers train locations, and sends out commands to control signal aspects and point positions. The *control table* determines how the station interlocking system sets signals and points. For each route, there is one row describing the condition under which that route can be granted, and hence the corresponding signal can be set to show proceed. For example, there are two rows corresponding to signal S10: one for the main line (Route R10A) and one for the side line (Route R10B); signal S10 for the main line can only show proceed when point P101 is in normal (straight) position and tracks AA, AB, AC, AD are all clear.

Note that we do not assume that trains are equipped with an Automatic Train Protection system which prevents trains from moving over a red light; thus overlaps are needed, e.g., the overlap for Route R10A is AD, and hence AD is included in the clear table.

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 R10A obtains a lock on point P101, and sets it to normal. The lock is released after the train has passed the point. The *release tables* store the relevant track.

In this setting, we consider three safety properties:

1. *collision-freedom* excludes two trains occupying the same track;
2. *run-through* says that whenever a train enters a point, the point is set to cater for this; e.g., when a train travels from track AD to track AE, point P102 is set so that it connects AD and AE (and not BD and AE);
3. *no-derailment* says that whenever a train occupies a point, the point doesn't move.

The correct design for the control table and release tables is safety-critical: mistakes can lead to a violation of any of the three safety properties.

3.1. Modelling short trains

As outlined in [2], CSP‖B caters for the double nature of railways by addressing the (control) state and data aspects separately: 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.

The Trains and Controller processes run independently of each other, on the CSP level expressed with an interleaving operator – see Fig. 4 (lines 26 and 28). It is an internal decision of the controller which routes are requested to be set or to be released (lines 2–4). Similarly, it is an internal decision of the train (driver) to stay or to move in front of a green signal (lines 12–14) or when there is no signal (lines 21–23). If there is a red signal (lines 16–18) then it is an internal decision of the train (driver) to stay or to overrun the signal onto an overlap but then to stop. This dynamic operation is sometimes referred to as the *driving rules* of a train.

```
 1  b ⟵ release(route) =
 2   PRE route ∈ ROUTE THEN
 3        LET emptyTracks = 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
12                     (i.e. nothing close enough to go through the red light ) */
13                  homeSignal(signal(route)) ∈ emptyTracks
14             THEN
15                  /* signal of route to red */
16                  signalStatus(signal(route)) := red||
17                  /* release the locks associated with the route */
18                  currentLocks := route ⩤ currentLocks||
19                  /* release is successful */
20                  b := yes
21             ELSE
22                  b := no
23             END
24        END
25   END
```

Fig. 5. *Release* operation from *Interlocking*.

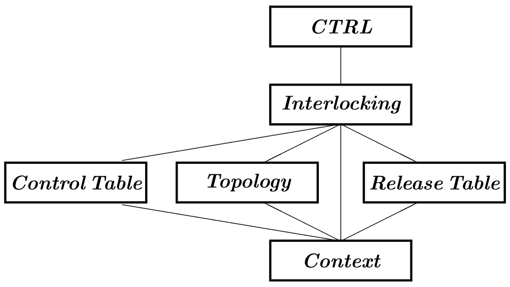


Fig. 6. Architecture.

The *Interlocking* machine captures information about the location of trains on tracks using the function $pos : TRAIN \to ALLTRACK$. 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 is a safe state. This dynamic state is then updated and queried, respectively, in the four operations of the *Interlocking* machine.

Fig. 5 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 are fulfilled (the signal of the route is green, line 6, there 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 16, etc.) and the controller is notified with a “yes” (line 20). Otherwise, the state does not change and the controller is notified with a “no”.

Fig. 6 shows the overall architecture of our modelling. The CSP controller *CTRL* and the *Interlocking* machine are independent of any particular scheme plan. They are supported by a *Topology*, a *Control Table*, a *Release Table*, and a *Context* machine. These four machines encode the scheme plan and are the parameters in our generic approach. Seen as B machines, these four supporting machines are stateless (i.e., without behaviour), and provide generic domain definitions.

A typical example from the *ControlTable* machine which splits up the modelling of a control table into three functions would be given as follows:

$$\begin{aligned} normalTable &\in ROUTE \to \mathbb{P}(POINTS) \quad \wedge \\ reverseTable &\in ROUTE \to \mathbb{P}(POINTS) \quad \wedge \\ clearTable &\in ROUTE \to \mathbb{P}(TRACK) \end{aligned}$$

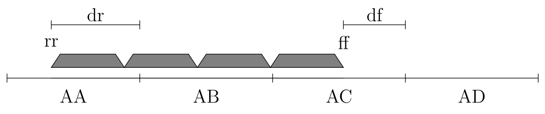


Fig. 7. A long train.

These three functions capture the data in the relevant columns from the control table. The *Release Table* is modelled as a function

$$releaseTable \in TRACK \to \mathbb{P}(ROUTE \times POINTS)$$

which indicates, for given track t, the route/point pairs (r, p) such that occupancy of track t releases point p on route r. This information is drawn directly from the release table.

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. Indeed, our industrial partners were able to contribute to the development of the model at the CSP||B level so as to ensure it reflected real-world concerns. 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.

To ensure consistency, the relationship between the *Interlocking* machine on the one hand and the *CTRL* process on the other is captured by an invariant which relates the *pos* function within the *Interlocking* machine to the parameters t and *pos* of the *TRAIN_CTRL* process. This invariant is required to hold at each recursive call, and hence the system is divergence-free.

3.2. Modelling long trains

Until now we have relied on the assumption that trains are shorter than track segments. Whilst unrealistic, this assumption allows much smaller models to be devised and, hence, analysed. Here we provide an approach which encompasses train and track lengths, making no assumptions about trains having to fit on track segments. For example, Fig. 7 depicts a train spanning the three tracks AA, AB and AC. Specifically, the front of the train sits on track segment AC ($ff = \text{AC}$), and has a distance $df \geq 0$ to the next track segment AD; and the rear of the train sits on track segment AA ($rr = \text{AA}$), and has a distance $dr \geq 0$ to the next track segment AB.

This approach allows fine-grained modelling of the distances that trains travel, as well as the times it takes to do so, and we have carried out such studies in the context of Timed CSP [12]. However, for the purposes of this paper – that is, verifying the safety of the rail network – we restrict attention to an untimed model in which state changes reflect the front or rear of the train either reaching or passing the end-points of track segments. There are thus four variables (ff, rr, df and dr) which define the state of a train.

There are the following four situations in which a state change occurs, depending on a partitioning of the values of the distances df and dr. (The track segments named are in reference to Fig. 7.)

(a) $\underline{df = 0 \text{ and } dr > 0}$. This means that the front of the train ff is at the junction of two track segments (AC and AD) while the rear of the train rr is wholly within a track segment (AA). In this instance an event $moveff.t.ff.ff'$ occurs representing the front of the train moving from track segment AC to track segment AD. The new values of the state variables are $ff' = AD$, $df' = length(AD)$, $rr' = rr$ and $dr' = dr$. (Note that this corresponds to track circuit AD changing from "no train detected" to "train detected".)

(b) $\underline{dr > df > 0}$. This means that the front and rear of the train are each wholly within a track segment (the rear within AA and the front within AC), but with the front closer to its next track segment than the rear is to its. In this instance the state changes autonomously (i.e., without any causal event) to that in which the front of the train moves to the end of its track segment (i.e., the train moves forward a distance df). The new values of the state variables are $ff' = ff$, $df' = 0$, $rr' = rr$ and $dr' = dr - df$.

(c) $\underline{df \geq 0 \text{ and } dr = 0}$. This means that the rear of the train is at the junction of two track segments (AA and AB). In this instance an event $moverr.t.rr.rr'$ occurs representing the rear of the train t moving from track segment AA to track segment AB. The new values of the state variables are $ff' = ff$, $df' = df$, $rr' = AB$ and $dr' = length(AB)$. (Note that this corresponds to track circuit AA changing from "train detected" to "no train detected".)

In the case $df = dr = 0$, we could have instead chosen to model this as internal non-deterministic choice between events $moverr.t.rr.rr'$ and $moveff.t.ff.ff'$. However, as we are dealing with sets of traces, the encoding as presented is sufficient for safety analysis.

(d) $\underline{df \geq dr > 0}$. This means that the front and rear of the train are each wholly within a track segment (the rear within AA and the front within AC), but with the rear at least as close to its next track segment as the front is to its. In this instance the state changes autonomously to that in which the rear of the train moves to the end of its track segment (i.e., the train moves forward a distance dr). The new values of the state variables are $ff' = ff$, $df' = df - dr$, $rr' = rr$ and $dr' = 0$.

As a note, in a finer-grained model cases (b) and (d) above – where neither end of the train is on an end-point of a track segment – would represent states where time elapses, allowing the train to move along a distance $d < \min(df, dr)$, updating the state variables to be $ff' = ff$, $df' = df - d$, $rr' = rr$ and $dr' = dr - d$.

3.3. Signals and overlaps

Unlike in [3], we do not assume here the presence of Automatic Train Protection (ATP) preventing trains from overrunning red signals. Rather, we use the more realistic assumption that trains may overrun a red light but in such instances will stop on the next track segment.

This being the case, the control table will (be expected to) stipulate that a track section immediately following a signal (an overlap section) will be protected by the signal preceding the one at the start of the section. As we are modelling "open" networks (i.e., with entry and exit tracks), our B model will allow a train to enter an entry track only if the entry track and its overlap track are both clear.

A *moveff* event will be enabled in the first two situations above, that is if (a) $dr > df = 0$ or (b) $dr > df > 0$; whereas a *moverr* event will be enabled in the latter two situations, that is if (c) $df \geq dr = 0$, or (d) $df \geq dr > 0$. The driving rules encoded into our model are then as follows:

(i) in front of a red signal, the train may either stay put, or it may overrun by one track and then stop;
(ii) in front of a green signal, the train may either move or it may stay put.

The behaviour of the train will only be dependent on signals in situation (a), and be modelled in CSP as follows.

```
if $aspect == green$
then
    $moveff!t.ff?ff' \rightarrow TRAIN\_CTRL(t, ff', length(ff'), rr, dr)$
    $\sqcap$
    $stay.t \rightarrow TRAIN\_CTRL(t, ff, df, rr, dr)$
else $(* aspect == red *)$
    $stay.t \rightarrow TRAIN\_CTRL(t, ff, df, rr, dr)$
    $\sqcap$
    $moveff.t.ff.ff' \rightarrow Stop$
```

3.4. Encoding safety

We describe here how the three safety properties are encoded in our B machine. Firstly, a collision is encoded as follows.

```
collision =
SELECT
  $\exists t_1, t_2 \in TRAIN : t_1 \neq t_2 \wedge (ran(pos(t_1)) \cap ran(pos(t_2))) \setminus (EXIT \cup ENTRY) \neq \emptyset$
      THEN skip
 END;
```

Here collision is detected when two different trains t_1 and t_2 occupy the same track segment (different from the *EXIT* and *ENTRY* tracks). This is recognised in the *pos* function which maps trains to the track segments they occupy; the collision condition will be enabled when the ranges of the *pos* functions of the two trains have a nonempty intersection.

Next, run-through is modelled as follows.

```
runthrough =
SELECT $\exists t \in TRAIN \wedge t \in dom(pos) \wedge nullTrack \in ran(pos(t))$
      THEN skip
 END;
```

Here run-through is detected when a train t occupies *nullTrack* which is a special track segment introduced in our CSP model onto which a train is sent when it travels over an incorrectly-set point.

Finally, derailment is modelled as follows.

```
derailment =
SELECT $ran(union(ran(pos))) \cap homePoints[movedPoints] \neq \emptyset$
      THEN skip
 END;
```

Here derailment is detected when the set of track segments currently occupied by trains includes segments which are associated with points that have moved while the trains have been on these segments.

In order to mirror these B events on the CSP level, we add a process that enables these events at all times:

```
$ERR = (collision \rightarrow ERR) \,\square\, (runthrough \rightarrow ERR) \,\square\, (derailment \rightarrow ERR)$
```

The complete CSP‖B models for both case studies can be downloaded from http://www.cs.swan.ac.uk/RAIL/Models/CSPB.

4. Finitisation

In the following, we develop a theory of how to reduce the problem of verifying our CSP‖B models of scheme plans for safety (i.e., freedom from collision, derailment, and run-through) to that of the two-train scenario. Given a scheme plan *SP*, and an unlimited collection *TRAIN* of trains with a function $length : TRAIN \to \mathbb{N}$ that assigns a length to each train, we write CSP ‖ B(*SP*, *TRAIN*) for the instantiation of our generic CSP‖B model with *SP* and *TRAIN*. Note that in general CSP ‖ B(*SP*, *TRAIN*) is an infinite state system due to the inclusion of train identifiers into events and states. Naturally, in railway practice there are only finitely many train lengths in use. We call our theory "finitisation", as it reduces the safety problem over an infinite state system to a safety problem over a finite state system, namely to CSP ‖ B(*SP*, *TRAIN*) where the set *TRAIN* contains two elements only.

Finitisation requires scheme plans to fulfil a number of well-formedness conditions as outlined in Section 4.1. For well-formed scheme plans we establish in Section 4.2 a reduction theorem (Theorem 3) w.r.t. the number of trains involved in a system run. All these conditions are straightforward to check statically. If we are only interested in the movements of a finite set of trains in a given system run – say in the movements of two trains which collide in this system run – then we can define a new system run with "exactly the same movements" for just this selected set of trains. Finitisation works for well-formed scheme plans as it is possible to simulate the influence that one train can have on other trains by suitable route request and route release commands. The validity of this finitisation argument for safety is demonstrated in Section 4.3.

4.1. Well-formedness conditions

In our modelling approach, track plans are encoded in the *Context* and in the *Topology* machines in B. In these machines, tracks are collected in a set *TRACK* with special sets $ENTRY, EXIT \subseteq TRACK$ for the entry and exit tracks. Signals are collected in a set *SIGNAL*; $homeSignal : SIGNAL \to TRACK$ defines the unique track at which a signal is placed; and the connectivity is given by a relation $next \subseteq TRACK \times TRACK$. One can see this structure as a directed graph $(TRACK, next)$ with signals as labels on the nodes. With this notation, we define the concept of a topological route as a path through this graph, which begins after a signal and ends either with the track after the next signal or before an exit track.

Definition 1. A *topological route* is a path $R = \langle t_1, \ldots, t_k \rangle \in TRACK^+$, $k \geq 1$, in the graph $(TRACK, next)$ such that the following holds:

- there is a signal $s \in SIGNAL$ and a track $t \in TRACK$ such that $homeSignal(s) = t$ and $(t, t_1) \in next$, and
- either there is a signal $s \in SIGNAL$ such that $homeSignal(s) = t_{k-1}$ and for all $1 \leq i \leq k-2$ and $s \in SIGNAL$: $homeSignal(s) \neq t_i$;
  or there is a track $t \in EXIT$ such that $(t_k, t) \in next$ and for all $1 \leq i \leq k$ and $s \in SIGNAL$: $homeSignal(s) \neq t_i$.

A *track t belongs to a topological route R*, written as $t \in R$, iff $t = t_i$ for some $1 \leq i \leq k$. *TopoRoute* denotes the *set of all topological routes* of a track plan.

In Fig. 2, the path $\langle AA, AB, AC, AD \rangle$ is a topological route from the first track after signal $S10$ to the first track after signal $S12$; the path $\langle AD, AE, AF \rangle$ is a topological route from signal $S10$ to the track just before the exit track *Exit*.

When designing a scheme plan, the signalling engineer selects and names some of the topological routes and develops control and release tables for them, i.e., there is a set *ROUTE* of route names and an injective map $topo : ROUTE \to TopoRoute$ which assigns a topological route to each route name.

Definition 2. A scheme plan is *well-formed* if the following conditions hold:

1. (Release-Table condition) Locks of a route can only be released by a train movement on that route:
$$\forall r \in ROUTE, p \in POINT, t \in TRACK: \quad (r, p) \in releaseTable(t) \Rightarrow t \in topo(r)$$
2. (Clear-Table condition) The clear table of a route contains at least the tracks of this route:
$$\forall r \in ROUTE: \quad \big\{t \mid t \in topo(r)\big\} \subseteq clearTable(r)$$
3. (Normal/Reverse-Table condition) Every point on a route is in either the normal table or the reverse table of that route (and not both):
$$\begin{aligned}\forall r \in ROUTE: \quad & \big\{p \in POINT \mid homePoint(p) \in topo(r)\big\} \subseteq normalTable(r) \cup reverseTable(r) \\ & \wedge\ normalTable(r) \cap reverseTable(r) = \emptyset\end{aligned}$$

   Here, $homePoint(p)$ depicts the track circuit name of point p.

4. (Route condition) Topologically different routes that share some points are distinguishable by at least one point position of these shared points:

$$\begin{aligned}&\forall r_1, r_2 \in ROUTE: \quad r_1 \neq r_2 \wedge sharedPoints(r_1, r_2) \neq \emptyset \Rightarrow \\ &\quad \big(\exists p \in sharedPoints(r_1, r_2): \\ &\qquad \big(p \in reverseTable(r_1) \wedge p \in normalTable(r_2)\big) \vee \\ &\qquad \big(p \in reverseTable(r_2) \wedge p \in normalTable(r_1)\big)\big)\end{aligned}$$

Here, $sharedPoints(r_1, r_2)$ depicts the points on both routes r_1 and r_2.

The above conditions ensure a minimal consistency between the signalling of routes in the control and release tables on the one hand, and their topological extent as defined by the railway topology on the other hand. As demonstrated by the following example, however, this consistency is not enough to ensure safety.

Example 1. Consider the following changes to the control table of the scheme plan shown in Fig. 2: for route $R10A$ set point $P101$ to be "reverse" rather than "normal", for route $R10B$ set point $P101$ to be "normal" rather than "reverse". In this changed setting all four conditions are fulfilled. The changed scheme plan however is not safe as trains can collide on track BC: Let there be no train in the beginning. Then route $R10A$ can be set, and a train can travel from $Entry$ over AA and AB to BC and stay on track BC. As BC is not in the clear part of route $R10A$, and there are no trains on the track named in the clear part of $R10A$, route $R10A$ can be set again, and another train can travel along the same way. This second train will collide with the first one on track BC.

4.2. A reduction theory

We start the development of our reduction theory with a simple observation on our CSP‖B models. If a signal shows green in a state of a system run, then there exists a uniquely determined route for which in the past a route request must have been granted by the interlocking.

Theorem 1. *Let σ be a system run of* CSP ‖ B$(SP, TRAIN)$ *for a scheme plan SP and a set of trains TRAIN. Then the following holds for all signals $sig \in SIGNAL$: prior to a state in which sig shows green, there is a uniquely determined event $request.r.yes$, $r \in ROUTE$, in σ that caused the signal to become green. We sometimes speak of the uniquely determined route r that has been granted.*

Proof. By definition of the B machine *Interlocking*, a signal is set to green only by the event *request* (when a route is successfully requested). Conversely, a signal is set to red only by the events *moveff* and *release* (when a train passes a signal and when a route is released successfully).

Let S_n, $n \geq 0$, be a state of σ in which *sig* is green. Then, prior to S_n, there must have been a last successful *request* to one of the routes r with $signal(r) = s$ (in S_0, all signals show red). Moreover, after this request no train can have passed *sig* and there cannot have been a successful attempt to release r. Thus, the system run σ up to state S_n has the following form:

$$S_0, e_0, S_1, \ldots, S_{k-1}, request.r.yes, S_k, \underbrace{e_k, \ldots\ldots\ldots\ldots\ldots, e_n,}_{\substack{e_i \neq moveff.id.t_{sig}.n_{sig}, \\ e_i \neq release.r.yes}} S_n$$

where $signalStatus_{S_n}(sig) = green$ (i.e., the signal *sig* in the final state S_n is *green*), $signalHome(sig) = t_{sig}$, $(t_{sig}, n_{sig}) \in next$, and *id* is a train identifier. Furthermore, any event between e_k and e_n inclusively cannot be $request.r'.yes$ for a route $r' \in ROUTE$ with $signal(r') = sig$. This is the case, as one condition in $request.r'.yes$ at state S_i (where $k < i < n$) requires $signalStatus_{S_i}(sig) = red$. Hence, no other route which shares this signal is set from S_k to S_n. □

In the following we show that given a set X of trains which do not cause collisions, derailments or run-throughs (in a precisely defined sense): for every system run σ, there exists a system run σ' involving only the trains not in X in which these trains move as dictated by σ. In particular: if trains collide in σ then they collide in σ'; if a train derails in σ, then it derails in σ'; and if a train has a run-through in σ, then it has a run-through in σ'.

We obtain σ' constructively by defining a replacement function on events. To this end, we first identify those events which are related to the trains in the set X:

Definition 3. Given a set X of train identifiers, we define the *events of X* (as introduced in our model) as

$$\begin{aligned}E(X) = \; & \{enter.b \mid b \in X\} \cup \\ & \{exit.b \mid b \in X\} \cup \\ & \{nextSignal.b \mid b \in X\} \cup \\ & \{moveff.b.cp.np \mid b \in X \wedge cp, np \in ALLTRACK\} \cup \\ & \{moverr.b.cp.np \mid b \in X \wedge cp, np \in ALLTRACK\}\end{aligned}$$

The next step is to define the replacement function $replace_X$ for a given set of trains X. This function is dependent on the current state S as well as the event e being replaced:

$$replace_X(S,e) = \begin{cases} e & e \notin E(X) \\ release.r.yes & e = moveff.b.cp.np \wedge \\ & \exists s \in SIGNAL: \quad homeSignal(s) = cp \wedge \\ & \qquad signalStatus_S(s) = green \wedge \\ & \qquad signal(r) = s \wedge \\ & \qquad currentLocks_S(r) = lockTable(r) \\ idle & \text{otherwise} \end{cases}$$

In the context of a system run, Theorem 1 ensures that this function is well-defined, as it guarantees the uniqueness of the route r to be released in the second clause.

In order to cater for this model transformation, which introduced the new event *idle*, we add the following CSP process *IDLE* in interleaving to our CSP controller:

```
IDLE = (idle → IDLE)
```

This process is only needed for the justification of our model transformations, however, left out in proof practice without losing the correctness of the argument.

Removing the trains in the set X from a system run also effects the states of the B machine. For example, one component of a B machine state S is the map $pos_S : TRAIN \to ALLTRACK^+$, which stores for each train the sequence of tracks it occupies. If we now remove the trains in X, we would hope that for the corresponding state T the following relation holds: $pos_T = pos_S \setminus X \times ALLTRACK^+$. In general, this correspondence between states is not only a projection on the remaining trains. We define:

Definition 4. Let S and T be states of the B machine of CSP $\|$ B(SP, $TRAINS$), let $X \subseteq TRAINS$ be a set of trains. *State T is in X-correspondence to state S*, written $T \leq_X S$, iff the following nine conditions are fulfilled:

$$pos_T = pos_S \setminus X \times ALLTRACK^+ \tag{1}$$

$$nextd_T = nextd_S \tag{2}$$

$$signalStatus_T = signalStatus_S \tag{3}$$

$$normalPoints_T = normalPoints_S \tag{4}$$

$$reversePoints_T = reversePoints_S \tag{5}$$

$$movedPoints_T = movedPoints_S \tag{6}$$

$$\big(currentLocks_T\big[\{r\}\big] = currentLocks_S\big[\{r\}\big] \vee currentLocks_T\big[\{r\}\big] = \emptyset\big) \quad \text{for all } r \in ROUTE \tag{7}$$

$$\begin{aligned}&\forall s \in SIGNAL: \quad \text{if } signalStatus_S(s) = green \text{ then}\\ &\quad \exists! r \in ROUTE: \quad signal(r) = s \wedge currentLocks_S(r) = lockTable(r) \wedge currentLocks_T(r) = lockTable(r)\end{aligned} \tag{8}$$

$$\forall b \in X,\ \forall t \in pos_S(b),\ \forall r \in ROUTE: \quad \text{if } t \in topo(r) \text{ then } currentLocks_T(r) = \emptyset \tag{9}$$

Condition (1) is as expected: the trains in the set X have been removed. Conditions (2) to (6) state that point positions and signal aspects are identical. Condition (7) states that the run without the trains in the set X:

- either has the same locks for a route – reflecting the fact that when a route is set, the locks are the same regardless of the set of trains involved;
- or no locks at all – reflecting the idea that if in S there is a train travelling on a route r, and this train is removed, then we release the route resulting in an empty set of locks.

Condition (8) stipulates that if a signal is green, there exists a unique route associated with the signal which is set. Finally, condition (9) says that the locks of any route that contains a track segment occupied by a train $b \in X$ in state S have been released in state T. It ensure that any route request in S is also possible in T. More specifically:

- if a train b is travelling on a route r and b is removed, then the locks of r must be empty, as we release r at the earliest convenience; and
- no other route makes use of the freed resources.

We want to establish the following simulation property: given a state S and a state T with $T \leq_X S$, and an event e that is enabled in S leading to a state S', the event $replace_X(S, e)$ is enabled in T and leads to a state $T' \leq_X S'$. The following diagram illustrates this situation:

$$\begin{tikzcd} T \arrow[d, "replace_X(S,e)"'] \arrow[r, phantom, "\leq_X"] & S \arrow[d, "e"] \\ T' \arrow[r, phantom, "\leq_X"] & S' \end{tikzcd}$$

We establish this simulation property in the following lemma.

Lemma 1. *Given a scheme plan SP, a set of trains TRAIN, a subset of trains $X \subseteq TRAIN$ and a system run*

$$\sigma = \langle S_0, e_1, S_1, \ldots, e_k, S_k \rangle$$

of CSP ∥ B(*SP*, *TRAIN*) *where trains in X do not cause collision, then there is a well-defined system run*

$$replace_X(\sigma) = \left\langle T_0, replace_X(S_0, e_1), T_1, \ldots, replace_X(S_{k-1}, e_k), T_k \right\rangle$$

of the B machine of CSP ∥ B(*SP*, *TRAIN* $\setminus$ *X*) *in which $T_i \leq_X S_i$ for each i.*

Proof. The proof is by induction on the length of σ. The base case is trivial, and the induction cases are generally unproblematic, with each case (i.e., possible event) demonstrating that $replace_X(S_{i-1}, e_i)$ is enabled in T_{i-1} and leads to a T_i with $T_i \leq S_i$.

The condition that the trains in X do not cause collision is necessary in the proof step regarding movement of trains in X when they pass a signal. In the simulation, we replace the move event *moveff.x.ff.nextff* (where $x \in X$, *ff* is a track in front of a signal) with a *release.r.yes* event (where r is the route of this signal). One of the preconditions of the release event is that there is no train on the track in front of the signal. When now x is removed, this condition may be false should there still be another train which was colliding with train x.

Due to the sheer number of cases to consider, the proof is relegated to Appendix A. □

With this result in place, we consider what conditions guarantee that $events(replace_X(\sigma))$ will be a trace of the CSP controller:

Lemma 2. *Given a scheme plan SP, a set of trains TRAIN, a subset of trains $X \subseteq TRAIN$ and a system run σ of* CSP ∥ B(*SP*, *TRAIN*), *the trace $events(replace_X(\sigma))$ will be a trace of the CSP controller CTRL(SP, TRAIN $\setminus$ X).*

Proof. Let σ be a system run of CSP ∥ B(*SP*, *TRAIN*) and let $\sigma' = replace_X(\sigma)$. As σ is a system run, $events(\sigma)$ is a trace of *CTRL*(*SP*, *TRAIN*). Recall that

$$CTRL(SP, TRAIN) = RW(SP) \mid\mid\mid TRAIN\_CTRL(TRAIN) \mid\mid\mid ERR \mid\mid\mid IDLE$$

$$TRAIN\_CTRL(TRAIN) = \mid\mid\mid_{i \in TRAIN}\ TRAIN\_OFF(i)$$

By looking at the definitions of the processes *RW*(_), *TRAIN_CTRL*(_), *ERR* and *IDLE*, we can note that their alphabets are disjoint; we can thus analyse the situation for the trace $events(\sigma')$ by projection onto these alphabets.

First note that for the original trace we have that $events(\sigma) \restriction E(\{i\}) \in \mathrm{traces}(TRAIN\_OFF(i))$ for all $i \in TRAIN$. Here, $E(\{i\})$ is the set of events associated with train i, see Definition 3, and $\restriction$ is the projection function defined in Section 2. With this result we obtain

- for each train $b \in X$,
  $events(\sigma') \restriction E(\{b\}) = \langle\rangle \subseteq \mathrm{traces}(TRAIN\_OFF(b))$
  since we replace all events related to $b \in X$; and
- for each train $a \notin X$,
  $events(\sigma') \restriction E(\{a\}) = events(\sigma) \restriction E(\{a\}) \in \mathrm{traces}(TRAIN\_OFF(a))$
  since we keep all events related to $a \notin X$.

From the definitions of *RW*(*SP*), *ERROR* and *IDLE*, it follows directly that

- $events(\sigma') \restriction \{|request, release|\} \in \mathrm{traces}(RW(SP))$ and
- $events(\sigma') \restriction \{|collision, derailment, runthrough|\} \in \mathrm{traces}(RW(SP))$.

Therefore, $events(\sigma') \in CTRL(SP, TRAIN \setminus X)$. □

Combining our two lemmas results in the following theorem.

Theorem 2. *Given a scheme plan SP, a set of trains TRAIN, a subset of trains $X \subseteq TRAIN$ and a system run σ of* CSP $\|$ B$(SP, TRAIN)$ *where trains in X do not cause collision, then $replace_X(\sigma)$ is a system run of* CSP $\|$ B$(SP, TRAIN \setminus X)$.

Proof. Let σ be a system run of CSP $\|$ B$(SP, TRAIN)$. By Lemma 1 we know that $replace_X(\sigma)$ is a run of the B machine M of CSP $\|$ B$(SP, TRAIN \setminus X)$, and we especially have $events(replace_X(\sigma)) \in \mathsf{traces}(M)$. By Lemma 2 we know that $replace_X(\sigma) \in \mathsf{traces}CTRL(SP, TRAIN)$. Thus, by definition of the semantics of CSP$\|$B, $replace_X(\sigma)$ is a system run of CSP $\|$ B$(SP, TRAIN \setminus X)$. $\square$

4.3. Verification for safety

Our verification approach for CSP$\|$B is to use model checking with ProB, where we check that in a given model a specific error event does not happen, i.e., it is never enabled.

Safety in the models with long trains is dependent on the train length involved, which motivates the following definition.

Definition 5. Let $ERROR = \{collision, derailment, runthrough\}$ be the set of error events of interest, and $L \subseteq \mathbb{N}$ be a set of possible train lengths.

1. For $n \in \mathbb{N}_{>0}$ and $e \in ERROR$, a scheme plan SP is (n, L) *e-free* iff e is not enabled in any state of any $\sigma \in$ CSP $\|$ B$(SP, TRAIN)$ in which $|TRAIN| = n$ and $\{length(t) \mid t \in TRAIN\} = L$.
2. A scheme plan SP is *L-safe* iff SP is (n, L) e-free for all $n \in \mathbb{N}_{>0}$ and $e \in ERROR$.

Note that our definition of (n, L) e-free requires that $1 \le |L| \le n$.

We can now turn Theorem 2 into a proof method. The following corollary is the basis of the main theoretical result of this paper.

Corollary 1. *Let $L \subseteq \mathbb{N}$ be a set of possible train lengths. If a scheme plan SP is*

1. *$(2, L')$ collision-free for all $L' \subseteq L$, $1 \le |L'| \le 2$, and*
2. *$(1, L')$ derailment-free for all $L' \subseteq L$, $|L'| = 1$, and*
3. *$(1, L')$ run-through-free for all $L' \subseteq L$, $|L'| = 1$,*

then SP is L-safe.

Proof. Assume that SP is not L-safe. This means there exists $n \in \mathbb{N}_{>0}$ and $err \in ERROR$ such that SP is not (n, L) err-free. That is, there is a non-empty collection of runs of CSP $\|$ B$(SP, TRAIN)$ in which some error is enabled in some state of each of these runs. A shortest such run will be of the form

$$\sigma = \langle S_0, e_1, S_1, \ldots, e_k, S_k \rangle$$

in which

$$\text{an error } e \text{ is enabled in } S_k \tag{10}$$

$$\text{no error is enabled in } S_0, \ldots, S_{k-1} \tag{11}$$

Case 1: $\quad e = collision$.

$$\begin{aligned}
(10) &\Rightarrow \exists t_1, t_2 \in TRAIN, t \in TRACK: \quad t \in pos_{S_k}(t_1) \wedge t \in pos_{S_k}(t_2)\\
&\Rightarrow e_k \text{ is a } moveff \text{ of } t_1 \text{ or } t_2 \text{ by } (11)\\
&\Rightarrow replace_{TRAIN \setminus \{t_1, t_2\}}(\sigma) \text{ is a run of CSP } \| \text{ B}\big(SP, \{t_1, t_2\}\big)\\
&\qquad \text{by Theorem 2, as trains in } TRAIN \setminus \{t_1, t_2\} \text{ do not}\\
&\qquad \text{cause collision in } \sigma\\
&\Rightarrow T_k \le_{TRAIN \setminus \{t_1, t_2\}} S_k\\
&\qquad \text{where } T_k \text{ is the last state in } replace_{TRAIN \setminus \{t_1, t_2\}}(\sigma)\\
&\Rightarrow t \in pos_{T_k}(t_1) \wedge t \in pos_{T_k}(t_2)\\
&\Rightarrow collision \text{ is enabled in } T_k\\
&\Rightarrow SP \text{ is not } \big(2, \{length(t_1), length(t_2)\}\big) \text{ collision-free}
\end{aligned}$$

Case 2: $e = derailment$.

$$\begin{aligned}
(10) &\Rightarrow \exists t \in TRAIN, p \in movedPoints_{S_k}: \quad homePoint(p) \in pos_{S_k}(t)\\
&\Rightarrow e_k \text{ is a } request.r.yes \text{ by } (11)\\
&\Rightarrow replace_{TRAIN\setminus\{t\}}(\sigma) \text{ is a run of CSP} \parallel \mathsf{B}\big(SP, \{t\}\big)\\
&\qquad \text{by Theorem 2, as trains in } TRAIN \setminus \{t\} \text{ do not}\\
&\qquad \text{cause collision in } \sigma\\
&\Rightarrow T_k \preceq_{TRAIN\setminus\{t\}} S_k\\
&\qquad \text{where } T_k \text{ is the last state in } replace_{TRAIN\setminus\{t\}}(\sigma)\\
&\Rightarrow p \in movedPoint_{T_k} \wedge homePoint(p) \in pos_{T_k}(t)\\
&\Rightarrow derailment \text{ is enabled in } T_k\\
&\Rightarrow SP \text{ is not } \big(1, \{length(t)\}\big) \text{ derailment-free}
\end{aligned}$$

Case 3: $e = run\text{-}through$.

$$\begin{aligned}
(10) &\Rightarrow \exists t \in TRAIN: \quad nullTrack \in pos_{S_k}(t)\\
&\Rightarrow e_k \text{ is a } moveff \text{ of } t \text{ by } (11)\\
&\Rightarrow replace_{TRAIN\setminus\{t\}}(\sigma) \text{ is a run of CSP} \parallel \mathsf{B}\big(SP, \{t\}\big)\\
&\qquad \text{by Theorem 2, as trains in } TRAIN \setminus \{t\} \text{ do not}\\
&\qquad \text{cause collision in } \sigma\\
&\Rightarrow T_k \preceq_{TRAIN\setminus\{t\}} S_k\\
&\qquad \text{where } T_k \text{ is the last state in } replace_{TRAIN\setminus\{t\}}(\sigma)\\
&\Rightarrow nullTrack \in pos_{T_k}(t)\\
&\Rightarrow run\text{-}through \text{ is enabled in } T_k\\
&\Rightarrow SP \text{ is not } \big(1, \{length(t)\}\big) \text{ run-through-free} \qquad \square
\end{aligned}$$

Corollary 1 works with different numbers of trains: two trains are needed in the case of *collision*, one train is needed otherwise. In order to be able to check safety for all three properties in one go, we prove the following.

Theorem 3. *If a scheme plan SP is* (n, L) *e-free then SP is* (n', L') *e-free for* $n' < n$ *and* $L' \subseteq L$.

Proof. Assume by way of contradiction that SP is not (n', L') e-safe. Then there exists a run $\sigma \in \text{CSP} \parallel \mathsf{B}(SP, TRAIN')$, $|TRAIN'| = n'$, such that e is enabled in some state of σ. Then, σ is also a run of $\text{CSP} \parallel \mathsf{B}(SP, TRAIN)$, $TRAIN' \subseteq TRAIN$, $|TRAIN| = n$ and $L' \subseteq L = \{length(t) \mid t \in TRAIN\}$. $\square$

5. Experimental results

In this section we outline various experimental results carried out on our models. We used the ProB tool to check the validity of the following CTL formula:

$$AG\big(not\big(e(collision) \vee e(runthrough) \vee e(derailment)\big)\big)$$

This formula is false if one of our *ERROR* events is enabled. In the CTL variant of ProB *AG* stands for "on all paths it is globally true", $e(a)$ stands for the enabledness of the event a.

5.1. Demonstration of errors

In order to demonstrate possible errors in a scheme plan, we provide two counterexamples from the verification of the Station case study, presented in Fig. 2, where the control table is deliberately changed to contain errors. In these cases, counterexamples are provided by ProB in terms of traces which contain an event from $\{collision, derailment, run\text{-}through\}$.

Example 2. In the first experiment, we swap the position of point $P101$ for routes $R10A$ (to reverse) and $R10B$ (to normal) – like in Example 1 above. For this, ProB provides the following counterexample:

Plan	Train length	States	≈ Transitions	≈ Size	Time
Station	40 m, 40 m	9093	88 702	186.5 MB	1 m 22 s
	40 m, 200 m	8931	78 626	182.1 MB	1 m 20 s
	200 m, 200 m	8769	78 596	181.7 MB	1 m 19 s
Junction	40 m, 40 m	64 733	896 812	612.3 MB	15 m 13 s
	40 m, 200 m	64 285	897 052	611.1 MB	15 m 10 s
	200 m, 200 m	63 837	883 000	604.6 MB	15 m 04 s

Fig. 8. Verification results of the Station and the Single Junction.

Plan	States	≈ Transitions	≈ Size	Time
Station	6185	63 508	176.8 MB	54.7 s
Junction	51 961	751 225	606.2 MB	11 m 36 s

Fig. 9. Verification results without lengths.

$\langle enter.albert.Entry,\ request.R10A.yes,\ nextSignal.albert.green,$
$moveff.albert.Entry.AA,\ moverr.albert.Entry.AA,\ moveff.albert.AA.AB,$
$moveff.albert.AB.BC,\ moverr.albert.AA.AB,\ moverr.albert.AB.BC,$
$enter.bertie.Entry,\ request.R10A.yes,\ nextSignal.bertie.green,$
$moveff.bertie.Entry.AA,\ moverr.bertie.Entry.AA,\ moveff.bertie.AA.AB,$
$moveff.bertie.AB.BC,\ collision\rangle$

which illustrates a *collision* caused by *albert* and *bertie* at BC.

Example 3. In the second experiment, we swap the position of point $P102$ for routes $R12$ (to reverse) and $R112$ (to normal). For this, ProB provides the following counterexample:

$\langle enter.albert.Entry,\ request.R10A.yes,\ nextSignal.albert.green,$
$moveff.albert.Entry.AA,\ moverr.albert.Entry.AA,\ moveff.albert.AA.AB,$
$moveff.albert.AB.AC,\ moverr.albert.AA.AB,\ moverr.albert.AB.AC,$
$request.R12.yes,\ moveff.albert.AC.AD,\ moverr.albert.AC.AD,$
$moveff.albert.AD.nullTrack,\ run\text{-}through\rangle$

which illustrates a *run-through* caused by *albert*.

5.2. Verification of the case studies

In this section we report on the verification results for safety of the single junction and station case studies. The experiments were carried out using ProB 1.3.6-final [13] to verify the collision, run-through and derailment freedom using CTL model checking over the CSP‖B models. The models are built using our modelling approach as described in Section 3 where train and track lengths are taken into account. Thanks to the finitisation technique developed in Section 4, the CSP‖B model of each case study requires only two trains for the verification of safety. In our example, we assume that train lengths can be either 40 m (i.e., consisting of two coaches, each being 20 m long) or 200 m (i.e., consisting of ten coaches). To this end, for each case study, we performed three experiments which cover all possible combinations of train lengths from $\{40\text{ m}, 200\text{ m}\}$. The experiments were conducted on a PC with a quad-core 3.2 GHz CPU and 8 GB memory. The results are summarised in Fig. 8 where for each experiment of a train length combination we report the number of states in the state space, the number of transitions in the state space, the size of used memory and the total running time.

Fig. 9 shows the verification results for the same case studies without modelling lengths. In these experiments, we consider two trains in the CSP‖B models of the Station and the Single Junction case studies. Since train and track lengths are not included in the CSP‖B models, only one experiment is carried out for each case study. These results show that the sizes of the CSP‖B models increase when we take lengths of trains and tracks into account in our modelling approach.

6. Related work

The railway interlocking problem has long been studied by the Formal Methods community, and our work builds upon prior approaches to the modelling and verification of railways. Prominent studies from the B community include [14,15] whilst [16,17] are classical contributions from process algebra and [18] uses techniques from Algebraic Specification. On a lower abstraction layer, [19–22] verify the safety of interlocking programs with logical approaches.

6.1. Modelling comparison

Our modelling is most related to Winter's approach in CSP [23] and Abrial's modelling in Event-B [24]. In the following we briefly discuss their respective approaches and the manner in which we consider our approach to succeed in combining the successful aspects of these whilst avoiding their perceived deficiencies.

Winter [23] 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 synchronisation 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 [24] 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.

Winter et al. [7] allows a train to occupy two track segments, which is a concession to the assumption made elsewhere (including in our previous studies) that a train can only occupy one track segment. However, we noted in [2] that even this concession is too restrictive to be realistic. It is one of the novelties of our approach here that this assumption is discharged. The other novelty is the discharging of the assumption that only a very few trains may enter the network. This assumption is traditionally used to keep the state space of the analyses under control, with tools being stretched to allow the possibility of ever more trains running through the network. Using our approach, this assumption is no longer required, at least for safety analysis.

Finally worth noting, Haxthausen et al. [25] take a different approach to modelling arbitrary numbers of trains and allowing trains occupying an arbitrary number of track segments. In particular, in their approach sensors are incorporated into the network along with points, signals and track segments; and virtual counters are associated with each sensor which keep count of the number of trains that pass the sensor. Safety violations are then expressed in terms of these counter values.

6.2. 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 [26] and therefore it is important to discuss it in the context of scalability. Ferrari et al. [19] 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. [21] demonstrate positive results on the feasibility of the lower level approach involving program slicing. Also, Cimatti et al. [22] have demonstrated considerable success using NuSMV on industrial scale problems, though they do not address large interlocking systems but rather the integration of a (moderately complex) interlocking system with a train spacing system (ERTMS). 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.

7. Conclusion and future work

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.

Beyond the challenge of faithfully modelling railway systems, we have devised abstraction techniques that yield an effective and efficient verification process based on model checking. In particular, we illustrated this process in terms of various scenarios.

Acknowledgements

The authors would like to thank S. Chadwick and D. Taylor from Invensys Rail for their support and encouraging feedback.

Appendix A. Proof of Lemma 1

Base case:

It requires to shows that $T_0 \leq_X S_0$. After initialisation, we have

$$\begin{aligned}
pos_X &= \emptyset \\
nextd_X &= staticNext \cup dynamicNext[POINT \times \{normal\}] \\
signalStatus_X &= SIGNAL \times \{red\} \\
normalPoints_X &= POINT \\
reversePoints_X &= \emptyset \\
movedPoints_X &= \emptyset \\
currentLocks_X &= \emptyset
\end{aligned}$$

for any $X \in \{T_0, S_0\}$. Hence, it is straightforward that $T_0 \leq_X S_0$.

Induction step:

Assume that $T \leq_X S$ and $S\ e\ S'$. We have to show that $e' = replace_X(e)$ is enabled in T and $T\ e'\ T'$ implies that $T' \leq_X S'$. This is done by considering all possible cases of e (as outlined in the model).

- $\underline{\text{Case } e = moveff.x.cp.np \text{ and } x \notin X}$, then $e' = moveff.x.cp.np$.
  e' is enabled in T:

$$\begin{aligned}
e \text{ is enabled in } S &\Rightarrow x \in dom(pos_S) \wedge \\
&\quad\ cp = first\big(pos_S(x)\big) \\
&\Rightarrow x \in dom(pos_T) \wedge \quad \text{by (1) and } x \notin X \\
&\quad\ cp = first\big(pos_T(x)\big) \quad \text{by (1) and } x \notin X \\
&\Rightarrow e' \text{ is enabled in } T
\end{aligned}$$

  $T' \leq S'$: Since e and e' only change $pos(x)$ $signalStatus(s)$ (if $homeSignal(s) = cp$), and $movedPoints$, we only show (1), (3) and (6):

$$\begin{aligned}
pos_{T'}(x) &= \langle np \rangle \frown pos_T(x) \quad \text{by } moveff.x.cp.np \\
&= \langle np \rangle \frown pos_S(x) \quad \text{by (1) and } x \notin X \\
&= pos_{S'}(x) \quad \text{by } moveff.x.cp.np
\end{aligned}$$

$$\begin{aligned}
signalStatus_{T'}(s) &= red \quad \text{if } cp = homeSignal(s) \\
&= signalStatus_{S'}(s) \quad \text{by } moveff.x.cp.np
\end{aligned}$$

$$\begin{aligned}
movedPoints_{T'} &= \emptyset \quad \text{by } moveff.x.cp.np \\
&= movedPoints_{S'}(s) \quad \text{by } moveff.x.cp.np
\end{aligned}$$

- $\underline{\text{Case } e = moveff.x.cp.np,\ x \in X, \text{ and } cp \notin ran(homeSignal)}$, then $e' = idle$.
  e' is trivially enabled in T.
  $T' \leq S'$: Since e and e' only change $pos(x)$ and $movedPoints$, we only show (1) *and* (6):

$$\begin{aligned}
pos_{T'} &= pos_T \quad \text{by } idle \\
&= pos_S \setminus X \times ALLTRACK^+ \quad \text{by (1)} \\
&= pos_S \setminus \{x\} \times ALLTRACK^+ \cup \big\{x \mapsto \langle np \rangle \frown pos_S(x)\big\} \setminus \\
&\quad\ X \times ALLTRACK^+ \quad \text{by } moveff.x.cp.np \\
&= pos_{S'}(x) \setminus X \times ALLTRACK^+ \quad \text{since } x \in X
\end{aligned}$$

$$\begin{aligned}
movedPoints_{T'} &= \emptyset \quad \text{by } idle \\
&= movedPoints_{S'}(s) \quad \text{by } moveff.x.cp.np
\end{aligned}$$

- $\underline{\text{Case } e = moveff.x.cp.np,\ x \in X, \text{ and } cp = homeSignal(s)}$, for some $s \in SIGNAL$ and $statusSignal_S(s) = red$, then $e' = idle$.
  e' is trivially enabled in T.

$T' \leq S'$: Since e and e' only change $pos(x)$ and $movedPoints$, we only show (1) *and* (6):

$$\begin{aligned} pos_{T'} &= pos_T \quad \text{by } idle \\ &= pos_S \setminus X \times ALLTRACK^+ \quad \text{by (1)} \\ &= pos_S \setminus \{x\} \times ALLTRACK^+ \cup \{x \mapsto \langle np \rangle ^\frown pos_S(x)\} \setminus \\ &\quad X \times ALLTRACK^+ \quad \text{by } moveff.x.cp.np \\ &= pos_{S'}(x) \setminus X \times ALLTRACK^+ \quad \text{since } x \in X \end{aligned}$$

$$\begin{aligned} movedPoints_{T'} &= \emptyset \quad \text{by } idle \\ &= movedPoints_{S'}(s) \quad \text{by } moveff.x.cp.np \end{aligned}$$

- <u>Case $e = moveff.x.cp.np$, $x \in X$, and $cp = homeSignal(s)$</u>, for some $s \in SIGNAL$ such that $signalStatus_S(s) = green$, then $e' = release.r.yes$ where r is uniquely determined by (8).
  e' is enabled in T since:

$$\begin{aligned} signalStatus_T(s) &= signalStatus_S(s) \quad \text{by (3)} \\ &= green \\ currentLocks_T(r) &= lockTable(r) \quad \text{by (8)} \\ homeSignal(s) &\in emtpyTracks_T \\ &\quad\ \text{as (1) and } x \in X \text{ (no collision)} \\ &\quad\ \text{imply only } x \text{ occupies } homeSignal(s) \end{aligned}$$

$T' \leq S'$: Since e and e' only change $pos(x)$ $signalStatus(s)$, and $movedPoints$, we only show (1), (3) and (6):

$$\begin{aligned} pos_{T'} &= pos_T \quad \text{by } release.r.yes \\ &= pos_S \setminus X \times ALLTRACK^+ \quad \text{by (1)} \\ &= pos_S \setminus \{x\} \times ALLTRACK^+ \cup \{x \mapsto \langle np \rangle ^\frown pos_S(x)\} \setminus \\ &\quad X \times ALLTRACK^+ \quad \text{by } moveff.x.cp.np \\ &= pos_{S'}(x) \setminus X \times ALLTRACK^+ \quad \text{since } x \in X \end{aligned}$$

$$signalStatus_{T'}(s) = red = signalStatus_{S'}(s) \quad \text{by } moveff.x.cp.np$$

$$\begin{aligned} movedPoints_{T'} &= \emptyset \quad \text{by } release.r.yes \\ &= movedPoints_{S'}(s) \quad \text{by } moveff.x.cp.np \end{aligned}$$

- <u>Case $e = moverr.x.cp.np$ and $x \notin X$</u>, then $e' = moverr.x.cp.np$.
  e' is enabled in T:

$$\begin{aligned} e \text{ is enabled in } S &\Rightarrow x \in dom(pos_S) \wedge \\ &\quad\ cp = last(pos_S(x)) \\ &\Rightarrow x \in dom(pos_T) \wedge \quad \text{by (1) and } x \notin X \\ &\quad\ cp = last(pos_T(x)) \quad \text{by (1) and } x \notin X \\ &\Rightarrow e' \text{ is enabled in } T \end{aligned}$$

$T' \leq S'$: Since e and e' only change $pos(x)$, $movedPoints$ and $currentLocks$, we only show (1), (6) *and* (7)

$$\begin{aligned} pos_{T'}(x) &= front(pos_T(x)) \quad \text{by } moverr.x.cp.np \\ &= front(pos_S(x)) \quad \text{by (1) and } x \notin X \\ &= pos_{S'}(x) \quad \text{by } moverr.x.cp.np \end{aligned}$$

$$\begin{aligned} movedPoints_{T'} &= \emptyset \quad \text{by } moverr.x.cp.np \\ &= movedPoints_{S'}(s) \quad \text{by } moverr.x.cp.np \end{aligned}$$

$$\begin{aligned} currentLocks_{T'} &= currentLocks_T \setminus releaseTable(np) \\ &\quad\ \text{by } moverr.x.cp.np \\ &= currentLocks_S \setminus releaseTable(np) \text{ or} \\ &\quad\ \emptyset \setminus releaseTable(np) \\ &= currentLocks_{S'} \text{ or } \emptyset \end{aligned}$$

- <u>Case $e = moverr.x.cp.np$ and $x \in X$</u>, then $e' = idle$.
  e' is trivially enabled in T.

$T' \leq S'$: Since e and e' only change $pos(x)$, $movedPoints$ and $currentLocks$, we only show (1), (6) *and* (7):

$$\begin{aligned}
pos_{T'} &= pos_T \quad \text{by } idle \\
&= pos_S \setminus X \times ALLTRACK^+ \quad \text{by (1)} \\
&= pos_S \setminus \{x\} \times ALLTRACK^+ \cup \big\{x \mapsto front\big(pos_S(x)\big)\big\} \setminus \\
&\quad X \times ALLTRACK^+ \quad \text{by } moverr.x.cp.np \\
&= pos_{S'} \setminus X \times ALLTRACK^+ \quad \text{since } x \in X
\end{aligned}$$

$$\begin{aligned}
movedPoints_{T'} &= \emptyset \quad \text{by } idle \\
&= movedPoints_{S'}(s) \quad \text{by } moverr.x.cp.np
\end{aligned}$$

For (7), we have:

$$\begin{aligned}
currentLocks_{T'} &= currentLocks_T \\
&\quad \text{by } idle \\
currentLocks_{S'} &= currentLocks_S \setminus releaseTable(np) \\
&\quad \text{by } idle
\end{aligned}$$

For any $(r, p) \in releaseTable(np)$, we have that $np \in topo(r)$ by assumption (1). Furthermore, $np \in pos_S(x)$, by (9), we have that $currentLocks_T(r) = \emptyset$, hence $currentLocks_{T'}(r) = \emptyset$.
The event of this case moves the rear of a train in X, hence, no signal changes from red to green, then (8) follows immediate; and it is not moved into a new route, then, (9) holds for S' and T'.

- Case $e = request.r.yes$, then $e' = request.r.yes$.
  e' is enabled in T:

$$\begin{aligned}
& e \text{ is enabled in } S \\
\Rightarrow\ & clearTable(r) \subseteq emptyTracks_S \land \\
& signalStatus_S\big(signal(r)\big) = red \land \\
& normalTable(r) \subseteq normalPoints_S \cup unlockedPoints_S \land \\
& reverseTable(r) \subseteq reversePoints_S \cup unlockedPoints_S \\
\Rightarrow\ & clearTable(r) \subseteq emptyTracks_T \land \\
& \quad \text{as (1) implies } emptyTracks_S \subseteq emptyTracks_T \\
& signalStatus_T(signal(r)) = red \ \land \quad \text{by (3)} \\
& normalTable(r) \subseteq normalPoints_T \cup unlockedPoints_T \land \\
& \quad \text{by (4) and (7)} \\
& reverseTable(r) \subseteq reversePoints_T \cup unlockedPoints_T \\
& \quad \text{by (5) and (7)} \\
\Rightarrow\ & e' \text{ is enabled in } T
\end{aligned}$$

$T' \leq S'$: Since e and e' only change $signalStatus(signal(r))$, $normalPoints$, $reversePoints$, $movedPoints$, and $currentLocks(r)$, we only show (3), (4), (5), (6), (7), (8):

$$\begin{aligned}
signalStatus_{T'}(signal(r)) &= green = signalStatus_{S'}(signal(r)) \\
&\quad \text{by } request.r.yes
\end{aligned}$$

$$\begin{aligned}
normalPoints_{T'} &= normalPoints_T \cup normalTable\big[\{r\}\big] \setminus reverseTable\big[\{r\}\big] \\
&\quad \text{by } request.r.yes \\
&= normalPoints_S \cup normalTable\big[\{r\}\big] \setminus reverseTable\big[\{r\}\big] \\
&\quad \text{by (4)} \\
&= normalPoints_{S'} \quad \text{by } request.r.yes
\end{aligned}$$

$$\begin{aligned}
reversePoints_{T'} &= reversePoints_T \cup reverseTable\big[\{r\}\big] \setminus normalTable\big[\{r\}\big] \\
&\quad \text{by } request.r.yes \\
&= reversePoints_S \cup reverseTable\big[\{r\}\big] \setminus normalTable\big[\{r\}\big] \\
&\quad \text{by (4)} \\
&= reversePoints_{S'} \quad \text{by } request.r.yes
\end{aligned}$$

$$\begin{aligned}
movedPoints_{T'} &= (normalPoints_T \setminus normalPoints_{T'}) \cup \\
&\quad (reversePoints_T \setminus reversePoints_{T'}) \\
&\qquad \text{by } request.r.yes \\
&= (normalPoints_S \setminus normalPoints_{S'}) \cup \\
&\qquad \text{by (4) and above} \\
&\quad (reversePoints_S \setminus reversePoints_{S'}) \\
&\qquad \text{by (4) and above} \\
&= movedPoints_{S'} \quad \text{by } request.r.yes
\end{aligned}$$

$$\begin{aligned}
currentLocks_{T'}\big[\{r\}\big] &= currentLocks_T\big[\{r\}\big] \cup lockTable\big[\{r\}\big] \\
&\qquad \text{by } request.r.yes \\
&= lockTable\big[\{r\}\big] \\
&\qquad \text{since } r \text{ locks at most } lockTable\big[\{r\}\big]
\end{aligned}$$

For (8), the existence is immediate by r. The uniqueness follows from assumptions (3) and (4) that it is not possible to have 2 routes sharing signals since they must share points and the locks on these points must be different.
For (9), any route r' such that $t \in topo(r')$ cannot be requested in S and T since assumption (2) and the fact that there is a train on its topology. Hence, $r' \neq r$, i.e., we do not change the locks by r' in S'.

- Case $e = enter.x.t$ and $x \notin X$, then $e' = enter.x.t$.
  e' is enabled in T:

$$\begin{aligned}
e \text{ is enabled in } S &\Rightarrow x \notin dom(pos_S) \wedge \\
&\quad\ t \in ENTRY \wedge \\
&\quad\ nextd_S(t) \in emptyTracks_S \\
&\Rightarrow x \notin dom(pos_T) \wedge \quad \text{by (1)} \\
&\quad\ t \in ENTRY \wedge \\
&\quad\ nextd_T(t) \in emptyTracks_T \quad \text{by (1) and (2)} \\
&\Rightarrow e' \text{ is enabled in } T
\end{aligned}$$

  $T' \leq S'$: Since e and e' only change *pos* and *movedPoints*, we only show (1) and (7):

$$\begin{aligned}
pos_{T'} &= pos_T \cup \big\{x \mapsto \langle t\rangle\big\} \quad \text{by } enter \\
&= pos_S \setminus X \times ALLTRACK^+ \cup \big\{x \mapsto \langle t\rangle\big\} \quad \text{by (1)} \\
&= pos_S \cup \big\{x \mapsto \langle t\rangle\big\} \setminus X \times ALLTRACK^+ \quad \text{as } x \notin X \\
&= pos_{S'} \setminus X \times ALLTRACK^+ \quad \text{by } enter
\end{aligned}$$

$$\begin{aligned}
movedPoints_{T'} &= \emptyset \quad \text{by } enter \\
&= movedPoints_{S'} \quad \text{by } enter
\end{aligned}$$

- Case $e = enter.x.t$ and $x \in X$, then $e' = idle$.
  e' is obviously enabled in T.
  $T' \leq S'$: Since e and e' only change *pos* and *movedPoints*, we only show (1) and (7):

$$\begin{aligned}
pos_{T'} &= pos_T \quad \text{by } idle \\
&= pos_S \setminus X \times ALLTRACK^+ \quad \text{by (1)} \\
&= pos_S \cup \big\{x \mapsto \langle t\rangle\big\} \setminus X \times ALLTRACK^+ \quad \text{as } x \in X \\
&= pos_{S'} \setminus X \times ALLTRACK^+ \quad \text{by } enter
\end{aligned}$$

$$\begin{aligned}
movedPoints_{T'} &= \emptyset \quad \text{by } idle \\
&= movedPoints_{S'} \quad \text{by } enter
\end{aligned}$$

- Case $e = exit.x.t$ and $x \notin X$, then $e' = exit.x.t$.
  e' is enabled in T:

$$\begin{aligned}
e \text{ is enabled in } S &\Rightarrow pos_S(x) = \langle t\rangle \wedge \\
&\quad\ t \in EXIT \\
&\Rightarrow pos_T(x) = \langle t\rangle \wedge \quad \text{by (1) and } x \notin X \\
&\quad\ t \in EXIT \\
&\Rightarrow e' \text{ is enabled in } T
\end{aligned}$$

  $T' \leq S'$: Since e and e' only change *pos* and *movedPoints*, we only show (1) and (7):

$$\begin{aligned}
pos_{T'} &= pos_T \setminus \big\{x \mapsto \langle t\rangle\big\} \quad \text{by } exit \\
&= pos_S \setminus X \times ALLTRACK^+ \setminus \big\{x \mapsto \langle t\rangle\big\} \quad \text{by (1)} \\
&= pos_S \setminus \big\{x \mapsto \langle t\rangle\big\} \setminus X \times ALLTRACK^+ \quad \text{as } x \notin X \\
&= pos_{S'} \setminus X \times ALLTRACK^+ \quad \text{by } exit
\end{aligned}$$

$$\begin{aligned} movedPoints_{T'} &= \emptyset \quad \text{by } exit \\ &= movedPoints_{S'} \quad \text{by } exit \end{aligned}$$

- $\underline{\text{Case } e = exit.x.t \text{ and } x \in X}$, then $e' = idle$.
  e' is obviously enabled in T.
  $T' \leq S'$: Since e and e' only change pos and $movedPoints$, we only show (1) and (7):

$$\begin{aligned} pos_{T'} &= pos_T \quad \text{by } idle \\ &= pos_S \setminus X \times ALLTRACK^+ \quad \text{by (1)} \\ &= pos_S \setminus \{x \mapsto \langle t \rangle\} \setminus X \times ALLTRACK^+ \quad \text{as } x \in X \\ &= pos_{S'} \setminus X \times ALLTRACK^+ \quad \text{by } exit \end{aligned}$$

$$\begin{aligned} movedPoints_{T'} &= \emptyset \quad \text{by } idle \\ &= movedPoints_{S'} \quad \text{by } exit \end{aligned}$$

- $\underline{\text{Case } e = idle}$, then $e' = idle$. The proof is trivial since e and e' only change $movedPoint$ and $movedPoints_{T'} = movedPoints_{S'} = \emptyset$.
- $\underline{\text{Case } e = nextSignal.x \text{ and } x \notin X}$, then $e' = nextSignal.x$.
  e' is enabled in T:

$$\begin{aligned} e \text{ is enabled in } S &\Rightarrow first(pos_S(x)) = ran(homeSignal(s)) \\ &\Rightarrow first(pos_T(x)) = ran(homeSignal(s)) \\ &\qquad \text{by (1) and } x \notin X \\ &\Rightarrow e' \text{ is enabled in } T \end{aligned}$$

  $T' \leq S'$: The proof is trivial since e and e' only change $movedPoint$ and $movedPoints_{T'} = movedPoints_{S'} = \emptyset$.
- $\underline{\text{Case } e = nextSignal.x \text{ and } x \in X}$, then $e' = idle$.
  e' is trivially enabled in T.
  $T' \leq_X S'$: The proof is trivial since e and e' only change $movedPoint$ and $movedPoints_{T'} = movedPoints_{S'} = \emptyset$.
- $\underline{\text{Case } e = release.r.yes}$, then $e' = release.r.yes$.
  e' is enabled in T:

$$\begin{aligned} & e \text{ is enabled in } S \\ \Rightarrow\ & signalStatus_S(signal(r)) = green \wedge \\ & currentLocks_S[\{r\}] = lockTable[\{r\}] \wedge \\ & homeSignal(signal(r)) \in emptyTracks_S \\ \Rightarrow\ & signalStatus_T(signal(r)) = green \wedge \\ & \quad \text{by (3)} \\ & currentLocks_T[\{r\}] = lockTable[\{r\}] \wedge \\ & \quad \text{by (8)} \\ & homeSignal(signal(r)) \in emptyTracks_T \\ & \quad \text{as (1) implies that } emptyTracks_S \subseteq emptyTracks_T \\ \Rightarrow\ & e' \text{ is enabled in } T \end{aligned}$$

  $T' \leq S'$: Since e and e' only change $signalStatus(signal(r))$, $movedPoints$, and $currentLocks(r)$, we only show (3), (6), (7):

$$\begin{aligned} signalStatus_{T'}(signal(r)) &= red \quad \text{by } release.r.yes \\ &= signalStatus_{S'}(signal(r)) \quad \text{by } release.r.yes \end{aligned}$$

$$\begin{aligned} movedPoints_{T'} &= \emptyset \quad \text{by } release.r.yes \\ &= movedPoints_{S'} \quad \text{by } release.r.yes \end{aligned}$$

$$\begin{aligned} currentLocks_{T'}[\{r\}] &= currentLocks_T[\{r\}] \setminus lockTable[\{r\}] \\ &\qquad \text{by } release.r.yes \\ &= \emptyset \quad \text{since } r \text{ locks at most } lockTable[\{r\}] \end{aligned}$$

References

[1] R. Jacquart (Ed.), IFIP 18th World Computer Congress, Topical Sessions, Kluwer, 2004.

[2] F. Moller, H.N. Nguyen, M. Roggenbach, S. Schneider, H. Treharne, Railway modelling in CSP||B: the double junction case study, Electron. Commun. EASST 53 (2012), 15 pages.

[3] F. Moller, H.N. Nguyen, M. Roggenbach, S. Schneider, H. Treharne, Defining and model checking abstractions of complex railway models using CSP||B, in: Proceedings of HVC'12: Eighth Haifa Verification Conference, in: Lecture Notes in Computer Science, vol. 7857, Springer, 2012, pp. 193–208.

[4] F. Moller, H.N. Nguyen, M. Roggenbach, S. Schneider, H. Treharne, Using ProB and CSP||B for railway modelling, in: Proceedings of IFM'12 and ABZ 2012 Posters and Tool Demos Session, 2012, pp. 31–35.

[5] F. Moller, H.N. Nguyen, M. Roggenbach, S. Schneider, H. Treharne, Combining event-based and state-based modelling for railway verification, Tech. rep. CS-12-02, University of Surrey, 2012.
[6] S. Schneider, H. Treharne, CSP theorems for communicating B machines, Form. Asp. Comput. 17 (4) (2005) 390–422.
[7] K. Winter, N. Robinson, Modelling large railway interlockings and model checking small ones, in: Proceedings of the 26th Australasian Computer Science Conference, vol. 16, Australian Computer Society, Inc., 2003, pp. 309–316.
[8] J.-R. Abrial, The B-Book: Assigning Programs to Meanings, Cambridge University Press, 1996.
[9] C.A.R. Hoare, Communicating Sequential Processes, Prentice Hall, 1985.
[10] C.C. Morgan, Of wp and CSP, in: Beauty Is Our Business: A Birthday Salute to Edsger J. Dijkstra, Springer, 1990, pp. 319–326.
[11] O.-S. Nock, Railway Signalling, IRSE Press, 1980.
[12] Y. Isobe, F. Moller, H.N. Nguyen, M. Roggenbach, Safety and line capacity in railways – an approach in Timed CSP, in: IFM'12: Ninth International Conference on Integrated Formal Methods, in: Lecture Notes in Computer Science, vol. 7321, Springer, 2012, pp. 54–68.
[13] ProB 1.3.5 beta15, http://www.stups.uni-duesseldorf.de/ProB, accessed: 23/07/2012.
[14] M. Leuschel, J. Falampin, F. Fritz, D. Plagge, Automated property verification for large scale B models with ProB, Form. Asp. Comput. 23 (6) (2011) 683–709.
[15] D. Sabatier, L. Burdy, A. Requet, J. Guéry, Formal proofs for the NYCT line 7 (flushing) modernization project, in: ABZ, 2012, pp. 369–372.
[16] A. Simpson, J. Woodcock, J. Davies, The mechanical verification of solid-state interlocking geographic data, in: Formal Methods Pacific '97, Springer, 1997, pp. 223–243.
[17] M.J. Morley, Safety in railway signalling data: a behavioural analysis, in: 6th International Workshop on HOLTPA, Springer, 1993, pp. 464–474.
[18] A.E. Haxthausen, J. Peleska, Formal development and verification of a distributed railway control system, IEEE Trans. Softw. Eng. 26 (8) (2000) 687–701.
[19] A. Ferrari, G. Magnani, D. Grasso, A. Fantechi, Model checking interlocking control tables, in: FORMS/FORMAT 2010, 2011, pp. 107–115.
[20] K. Kanso, F. Moller, A. Setzer, Automated verification of signalling principles in railway interlockings, Electron. Notes Theor. Comput. Sci. 250 (2009) 19–31.
[21] P. James, M. Roggenbach, Automatically verifying railway interlockings using SAT-based model checking, Electron. Commun. EASST 35 (2010) 17.
[22] A. Cimatti, R. Corvino, A. Lazzaro, I. Narasamdya, T. Rizzo, M. Roveri, A. Sanseviero, A. Tchaltsev, Formal verification and validation of ERTMS industrial railway train spacing system, in: CAV, 2012, pp. 378–393.
[23] K. Winter, Model checking railway interlocking systems, Aust. Comput. Sci. Commun. 24 (1) (2014).
[24] J.-R. Abrial, Modeling in Event-B, Cambridge University Press, 2010.
[25] A. Haxthausen, J. Peleska, S. Kinder, A formal approach for the construction and verification of railway control systems, Form. Asp. Comput. 23 (2) (2011) 191–219.
[26] A. Fantechi, S. Gnesi, On the adoption of model checking in safety-related software industry, Comput. Saf. Reliab. Secur. (2011) 383–396.