Int J Softw Tools Technol Transfer (2014) 16:685–711
DOI 10.1007/s10009-014-0304-7

Techniques for modelling and verifying railway interlockings

Phillip James · Faron Moller · Hoang Nga Nguyen · Markus Roggenbach · Steve Schneider · Helen Treharne

Published online: 15 March 2014

Abstract We describe a novel framework for modelling railway interlockings which has been developed in conjunction with railway engineers. The modelling language used is CSP||B. Beyond the modelling we present a variety of abstraction techniques which make the analysis of medium- to large-scale networks feasible. The paper notably introduces a covering technique that allows railway scheme plans to be decomposed into a set of smaller scheme plans. The finitisation and topological abstraction techniques are extended from previous work and are given formal foundations. All three techniques are applicable to other modelling frameworks besides CSP||B. Being able to apply abstractions and simplifications on the domain model before performing model checking is the key strength of our approach. We demonstrate the use of the framework on a real-life, medium-size scheme plan.

Keywords Railway verification · CSP · B · Model checking · Safety

1 Introduction

Formal verification of railway control software has been identified as one of the "grand challenges" of computer science [13]. This challenge comes in two parts. The first addresses the question of whether the proposed mathematical models faithfully represent the railway domain; verifications must translate to guarantees in the real world. The second addresses the question of how to employ available technologies effectively; analyses must be doable in practice and not just in theory.

In a series of papers [23–26] we have been developing a new modelling approach for railway interlockings. This work is carried out in conjunction with railway engineers drawn from our industrial partner. By involving the railway engineers from the start, we benefit twofold: they provide realistic case studies; and—more importantly—they guide the modelling approach, ensuring that it is natural to the working engineer and incorporates all relevant concerns. Our approach thus addresses the first part of the grand challenge.

We base our modelling approach on CSP||B [34], 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. Our industrial partners can use our modelling approach, and readily recognise it to be fully faithful to their real-world concerns.

In addressing the second part of the grand challenge, we face the wider challenge for formal methods of overcoming state space explosion. Having rendered a real-world problem into a modelling language, it remains a mystery in general as to how to decompose a verification problem into tractable pieces whose solutions can be composed together to provide a solution to the initial problem. Our approach is to carry out abstractions at the domain level, thus avoiding the lack of general compositional techniques in modelling languages.

We have developed three abstraction techniques which have proven successful in practice, both in isolation and taken together:

P. James · F. Moller · H. N. Nguyen · M. Roggenbach
Swansea University, Wales, UK

S. Schneider · H. Treharne (✉)
University of Surrey, Surrey, UK
e-mail: h.treharne@surrey.ac.uk

1. *Finitisation* reduces the number of trains that need to be considered to prove safety for an unbounded number of trains;
2. *Covering* decomposes the network into a set of sub-networks in a compositional fashion: proving correctness results for the sub-networks suffices to infer the correctness of the whole network; and
3. *Topological abstraction* reduces the number of tracks in the topology of the network, so as to minimise the size and complexity of the network prior to its analysis.

The second abstraction technique is a particular strength of our approach. Winter [38] theorised on the possibility of such compositional proof strategies for the railway domain, but to our knowledge there has since been no practical solution. This is the notable contribution of this paper which has not been presented in our previous work. The other techniques in this paper build upon their presentation in [24]. Firstly, we further reduce the number of trains that need to be considered during analyses. Secondly, we improve upon the topological abstraction technique as a consequence of having more detailed CSP||B models in this paper.

The verification that we focus on in this paper is the safety verification of three safety conditions: *collision freedom*, *run-through freedom* and *no derailment*. Our verification extends beyond checking the correctness of the configuration data of an interlocking. We address behavioural safety since we concern ourselves with train movements in our CSP||B models. Nonetheless, our modelling abstracts from the real-time behaviour of the interlocking and of the network as a train passes through it.

The paper is organised as follows. In Sect. 2, we introduce the traditional engineer's view of railway concepts, including a presentation of a complex real-life example which we shall use as a case study. We also outline three safety conditions that we will concentrate on verifying. In Sect. 3, we outline our approach to verification in general terms independent of any modelling language, as well as then outline a domain-specific modelling language on which we will base our modelling. In Sect. 5, we present our specific modelling language CSP||B, and apply this language to the railway domain in Sect. 6.

Having outlined the modelling framework, the next three sections of the paper outline our abstraction techniques: Sect. 7 presents finitisation, Sect. 8 presents covering and Sect. 9 presents topological abstraction. In Sect. 10, we present experimental results demonstrating the effectiveness of the abstractions. In Sect. 11, we discuss related approaches to the railway verification problem. Finally, in Sect. 12, we recapitulate our achievements and outline directions of future research.

2 Railway systems

Together with railway engineers, we have developed a common view of the information flow in railways. In physical terms, a railway consists 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 regard 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.

For the purposes of modelling, we have made the simplification to only consider two aspect signalling; we do not consider the additional aspects of caution or speed limits. We also make the assumption that track equipment reacts instantly and is free of defects. We furthermore assume that trains are shorter than the track segments in the network. In [15], we address the question of how to extend our modelling framework to deal with lengths of track segments and trains.

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 track equipment and receives information from track sensors on whether a track element is occupied. The interlocking and the trains interact indirectly via the track equipment only. The interlocking serves as the system's clock: in a cycle the status of all the track sensors are read then the interlocking reacts to all of them with one change of state. Routes cannot be in conflict since requests to select and

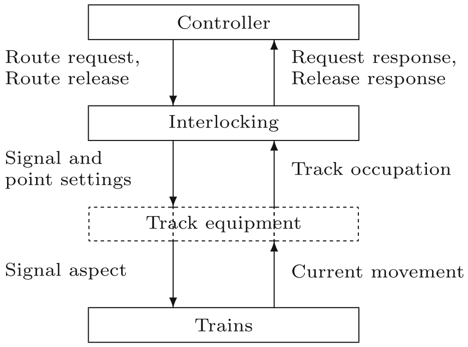


Fig. 1 Information flow

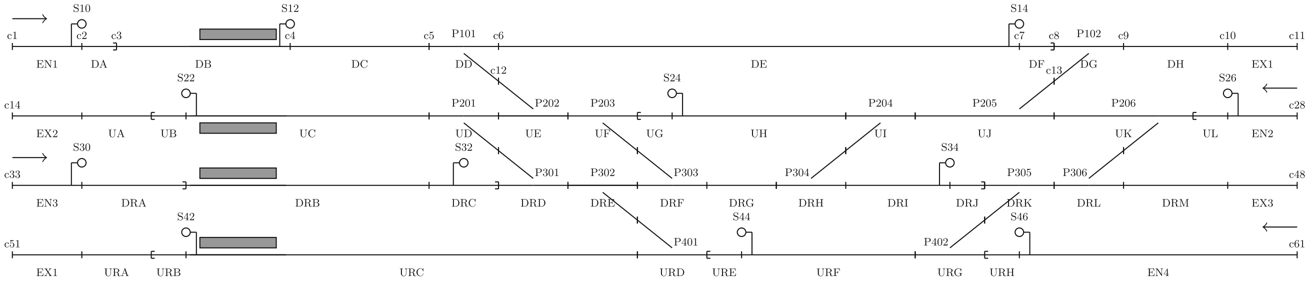


Fig. 2 Track plan based on Langley Station

Route	Normal	Reverse	Clear
R10			DA,DB,DC
R12A	P101,P202		DC,DD,DE,DF
R12B	P204,P304,P305	P101,P202,P203,P303	DC,DD,UE,UF,DRF,DRG,DRH,DRI,DRJ
R14	P102,P205		DF,DG,DH
R26A	P204,P205,P206,P304,P306		UL,UK,UI,UJ,UH,UG
R26B		P206,P306,P305,P402	UL,UK,DRL,DRK,URG,URF,URE
R24	P203,P202,P201,P101,P301		UG,UF,UE,UD,UC,UB
R22			UB,UA
R30			DRA,DRB,DRC
R32A	P301,P302,P303,P304,P203,P204,P401		DRC,DRD,DRE,DRF,DRG,DRH,DRI,DRJ
R32B	P301,P302,P303,P203,P401	P204,P205,P102,P304	DRC,DRD,DRE,DRF,DRG,DRH,UI,UJ,DG,DH
R34	P305,P306,P206		DRJ,DRK,DRL,DRM
R46	P402,P305,		URH,URG,URF,URE
R44A	P401,P302,P304		URE,URD,URC,URB
R44B		P401,P302,P301,P201	URE,URD,DRE,DRD,UD,UC,UB
R42			URB,URA

P101	Occupied
R12A	DE
R12B	UE
R24	UD

P102	Occupied
R14	DH
R32B	DH

P201	Occupied
R24	UC
R44b	UC

P202	Occupied
R12A	DE
R12B	UF
R24	UD

P203	Occupied
R12B	DRF
R24	UE
R32A	DRG
R32B	DRG

P204	Occupied
R12B	DRI
R26A	UH
R26B	DRH
R32A	DRI
R32B	UJ

P205	Occupied
R14	DH
R26A	UI
R32B	DG

P206	Occupied
R26A	UJ
R26B	DRL
R34	DRM

P301	Occupied
R24	UA
R32A	DRE
R32B	DRE
R44B	UD

P302	Occupied
R32A	DRF
R32B	DRF
R44A	URC
R44B	DRD

P303	Occupied
R32A	DRG
R32B	DRG

P304	Occupied
R12B	DRH
R26A	UH
R26B	URG
R32A	DRI
R32B	UI

P305	Occupied
R26B	URG
R26B	URG
R32B	UI
R34	DRL
R46	URF

P306	Occupied
R26A	UJ
R26B	DRK
R34	DRK

P401	Occupied
R32A	DRF
R32B	DRF
R44A	URC
R44B	DRE

P402	Occupied
R26B	URF
R46	URF

Fig. 3 Control table and release tables for Langley Station track plan

release routes are sequentialised. In our modelling we will abstract away from modelling the track equipment explicitly.

In this paper, we analyse a track layout based on Langley Station, a nontrivial station just to the west of London which is used by over 700,000 people each year [31], and considered to be a medium-size station in the UK. Figures 2 and 3 depict the *scheme plan* for the station comprising a track plan, a control table and release tables. The track plan is publicly available from [29]; however, as signalling rules are confidential, our control and release tables are of our own design, though they have been attested by our industrial partners as being realistic.

We explain our modelling approach here with reference to our Langley Station example. In general, we adhere closely to the established principles laid out in [30]. Following the approach of Bjørner [4], we view a *track plan* as being built from tracks, connectors, signals and points. Each track is associated with two connectors (or three if the track contains a point). Two tracks are attached together if they share a connector. Each track is also associated with a direction consisting of a (directed) pair of their associated connectors (or two pairs if the track contains a point). Thus a pair (c_1, c_2) in the direction of a track indicates that trains can travel on that track from c_1 to c_2, c_1 being the connector linking the track to the previous track and c_2 being the connector linking the track to the subsequent track. For example, the Langley Station track plan of Fig. 2 consists of 49 tracks (e.g. the tracks *EN*1 and *DA*), 61 connectors (e.g. the connector c_2

attaching the track *EN*1 and *DA*), 16 signals (e.g. S10 and S12), and 16 points (e.g. P101 and P102). Note that the tracks include entry and exit tracks on which trains can "appear" and "disappear" (e.g. *EN*1, *EX*1). These two kinds of tracks are specially treated during verification.

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 of a signal, there is one row describing the condition under which the signal can show proceed. There are two rows for signal S12: one for route R12A and one for route R12B where, for example, signal S12 can only show proceed when points P101 and P202 are in the normal (straight) position and tracks *DC*, *DD*, *DE*, *DF* are all clear.

The *normal* direction of a point in a track plan is indicated by an uninterrupted line (from connector c_5 to connector c_6), the *reverse* direction with an interrupted line (from connector c_5 to connector c_{12}).

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 R12A is *DF*, and hence *DF* is included in the control table. Trains are assumed to overrun a red signal by maximally one track. In case that such an overrun has happened, trains are assumed to halt.

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 R12A obtains a lock on point P101 and sets it to normal. The lock is released after the train has passed the point. This mechanism allows for the implementation of flank protection. The *release tables* store the relevant track, which is the track after the point.

In this setting, we consider three safety properties:

1. *Collision freedom* excludes two trains occupying the same track;
2. *Run-through freedom* says that whenever a train enters a point, the point is set to cater for this; e.g. when a train travels from track *DF* to track *DG*, point P102 is set so that it connects *DF* and *DG* (and not *UJ* and *DG*);
3. *No derailment* says that whenever a train occupies a point, the point does not 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 Verification workflow

In Fig. 4 we depict the verification workflow employed by our approach. Starting from a scheme plan of a railway system represented in a domain specific language (DSL) [8,21]—bottom left—we transform this scheme plan into a concrete specification SP_C—bottom right. This may be in any of a number of specification languages (e.g. CASL, CSP, Timed CSP and CSP||B) depending on the approach. However, regardless of the formalism, the specification will inevitably be too complex for analysis. To remedy this, some form of abstraction is applied to the scheme plan to produce an abstract scheme plan—top left—which is then transformed into an abstract specification SP_A—top right. With appropriate abstraction correctness results, verification proofs carried out on the abstract specification SP_A imply the relevant correctness of the concrete specification SP_C. For example, we have used this approach with topological abstractions in the context of CASL [14] and CSP||B [24], with a covering abstraction in the context of CSP [22], and with a finitisation abstraction in the context of Timed CSP [12].

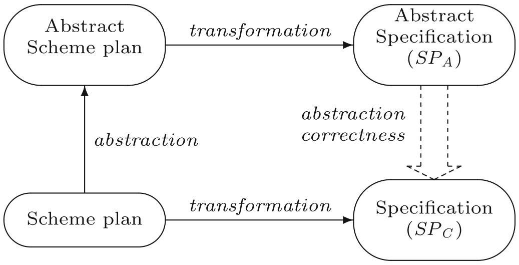


Fig. 4 Verification workflow

4 A railway DSL

Here, we present a general (mathematical) model of railway networks inspired by the work of Bjørner [4]. We implemented this model in our tool OnTrack [17] which also includes an automated transformation of this model into a CSP||B specification.

A railway network is provided by a scheme plan $SP = (Top, CT, RTs)$ which comprises a track plan *Top* defining the topology of the railway network, a control table *CT* and a set *RTs* of release tables. Note that our model is a loose specification of a railway scheme plan. For our purposes, this under-specification has proven to be sufficient.

4.1 Topology

Let Track and Point denote two disjoint, finite sets of tracks and points, respectively. Tracks and points are collectively referred to as units, and we let $\mathsf{Unit} = \mathsf{Track} \uplus \mathsf{Point}$. There is a set Connector whose elements serve as glue between nodes. A track t, having two endpoints, has two distinct connectors, whereas a point p, having three endpoints, has three distinct connectors; we write $connectors(u)$

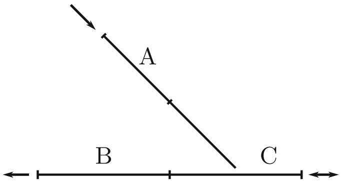


Fig. 5 A point example

to denote the set of all connectors of a unit u. A pair $(c_1, c_2) \in \mathsf{Connector} \times \mathsf{Connector}$ indicates that a train can travel on a unit u from c_1 to c_2, where $c_1, c_2 \in \mathit{connectors}(u)$. In our setting, a track t can be passed in one direction only; in contrast, a point p is associated with two directions where opposing directions and movement between two specific branches are excluded, e.g. in Fig. 2 movement between connector c_6 and c_{12} is not permitted. The two positions that a point can have are called *normal* and *reverse* where $\mathit{directions}(p) = \mathit{normal}(p) \uplus \mathit{reverse}(p)$. The direction of a unit can be read as the "intended use" of the unit, which the signal engineer provides when designing the routes, the control table and release tables. Given a direction $d = (c_1, c_2) \in \mathit{directions}(t)$ of a track or point t, we denote $\mathit{from}(d) = c_1$, $\mathit{to}(d) = c_2$.

A *path* $P = \langle (u_1, d_1), \ldots, (u_k, d_k)\rangle$, $k \geq 1$, in a railway topology, is a non-empty sequence of units and their directions without direct repetitions: $\mathit{to}(d_i) = \mathit{from}(d_{i+1})$ and $u_i \neq u_{i+1}$ for all $1 \leq i < k$. As usual, $\mathit{hd}(P) = u_1$ and $\mathit{last}(P) = u_k$, and $u \in P$ if $u = u_i$ for some $1 \leq i \leq k$. When the connectors are clear, we also write $\langle u_1, \ldots, u_k\rangle$ for P.

Note that the composition of two paths is not necessarily a path as direct repetitions are excluded. A typical example is shown in Fig. 5.

Here, $\langle A, C\rangle$ is a path and $\langle C, B\rangle$ is a path; however, $\langle A, C, C, B\rangle$ is *not* a path. Note however that any non-empty subsequence of a path is a path.

For convenience, we define two functions $\mathit{successor} : \mathsf{Unit} \rightarrow \wp(\mathsf{Unit})$ and $\mathit{predecessor} : \mathsf{Unit} \rightarrow \wp(\mathsf{Unit})$ as follows:

- $\mathit{successor}(u) = \{x \in \mathsf{Unit} \mid \exists c_1, c_2, c_3 \in \mathsf{Connector} : \langle (u, (c_1, c_2)), (x, (c_2, c_3))\rangle$ is a path$\}$, and
- $\mathit{predecessor}(u) = \{x \in \mathsf{Unit} \mid \exists c_1, c_2, c_3 \in \mathsf{Connector} : \langle (x, (c_1, c_2)), (u, (c_2, c_3))\rangle$ is a path$\}$.

Units without predecessors are called entries and units without successors are called exits. In the context of this paper, we consider only track plans where entries and exits are tracks, and denote the set of entry and exit tracks as

- $\mathsf{Entry} = \{t \in \mathsf{Track} \mid \mathit{predecessor}(t) = \emptyset\}$ and
- $\mathsf{Exit} = \{t \in \mathsf{Track} \mid \mathit{successor}(t) = \emptyset\}$.

We assume a set Signal of signals, along with a labelling function $\mathit{signalAt} : \mathsf{Signal} \rightarrow \mathsf{Track}$ indicating tracks at which signals are placed. Each track may be labelled by at most one signal: for each $t \in \mathsf{Track}$, $\mathit{signalAt}(s) = t$ for at most one $s \in \mathsf{Signal}$. Signals are placed at the end of a track to protect the successor track. We require that there is a signal at every entry track. Without such an entry signal, trains could unrestrictedly enter the scheme plan. This would cause collision on the successor of an entry track. Note that the typing of the function *signalAt* ensures that signals are never placed at a point—which follows standard practice in railway engineering.

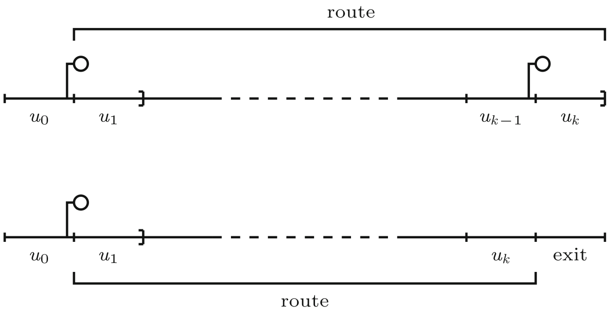


Fig. 6 An illustration of the route definition

As we deal with open railway topologies, we need to give two different definitions of what a route is: the first definition caters for the case in which the route is completely within the railway topology, while the second definition caters for the case in which a route ends at the border of the topology—see Fig. 6. A path $r = \langle u_1, \ldots, u_k\rangle$ is a topological route if one of the following holds:

- there is a unit u_0 such that

  $$\langle u_0, u_1, \ldots, u_k\rangle$$

  is a path in which u_0 and u_{k-1} are labelled with signals, but there are no signals on $u_1, \ldots, u_{k-2}$. In this case, u_k is called the *overlap* of r; or
- there are units u_0 and u_{k+1} such that

  $$\langle u_0, u_1, \ldots, u_k, u_{k+1}\rangle$$

  is a path, u_0 is labelled with a signal, there are no signals on $u_1, \ldots, u_k$ and u_{k+1} is an exit track.

In both cases, we define $\mathit{topoUnits}(r) = \{u_1, \ldots, u_k\}$ and $\mathit{topoSignal}(r) = s$ where $\mathit{signalAt}(s) = u_0$. Finally, we let $\mathsf{TopoRoute}$ denote the set of all topological routes in the railway topology, so that $\mathit{topoUnits} : \mathsf{TopoRoute} \rightarrow \wp(\mathsf{Unit})$ and $\mathit{topoSignal} : \mathsf{TopoRoute} \rightarrow \mathsf{Signal}$.

4.2 Control table

The control table determines the logic for controlling signals and points in the railway network. It specifies conditions when routes can be set which effectively leads to the control of signals' aspects and of points' positions.

Let Route be a set of route names and $topoRoute : \mathsf{Route} \to \mathsf{TopoRoute}$ a function associating topological routes to route names. The function *topoRoute* is not necessarily surjective as there can be topological routes which a signaller cannot control. For example in Fig. 3, the control table does not include a route corresponding to the topological route in Fig. 2 from the signal $S12$ to the exit track $EX1$—from track DB down to track DRG (points $P101$, $P202$, $P203$ and $P303$ all in reverse position) and then again up to track DH (points $P304$, $P204$, $P205$ and $P102$ all in reverse position). We allow for several entries in the control table that are associated with one topological route. The function $signal : \mathsf{Route} \to \mathsf{Signal}$ gives the entry signal of the corresponding topological route, i.e. $signal(r) = topoSignal(topoRoute(r))$. The function $units : \mathsf{Route} \to \wp(\mathsf{Unit})$ gives the set of units of the corresponding topological route, i.e.

$$units(r) = topoUnits(topoRoute(r)).$$

The control table specifies, for each route $r \in \mathsf{Route}$: a set $clear(r)$ of tracks and points to be clear; a set $normal(r)$ of points to be in the normal position; and a set $reverse(r)$ of points to be in the reverse position. Informally, when all units in $clear(r)$ are unoccupied, all points in $normal(r)$ are in the normal position, and all points in $reverse(r)$ are in the reverse position, route r can be set which effectively changes the aspect of $signal(r)$ to "proceed".

Note that there are in general no restrictions on how a control table looks, i.e. signalling engineers are allowed to write down anything. We define the *clear*, *normal* and *reverse* tables to be the columns of a control table.

4.3 Release tables

Each point is associated with a release table which specifies when to remove a lock from this point. Release tables are mappings $release : \mathsf{Point} \to \wp(\mathsf{Route} \times \mathsf{Unit})$. Given an entry $(r, t) \in release(p)$, informally, when a train reaches the unit t, the lock r is released from the point p, i.e. the point can be moved again, provided there is no other lock on it.

4.4 Well-formedness conditions

We postulate some conditions on a scheme plan formulated in our DSL. These 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. These conditions allow for simple static checks.

Definition 1 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:

$$\forall r \in \mathsf{Route}, p \in \mathsf{Point}, t \in \mathsf{Track} : \\ (r, t) \in release(p) \Rightarrow t \in units(r).$$

2. (Clear-Table condition) The clear table of a route contains at least the tracks of this route:

$$\forall r \in \mathsf{Route} : \{t \mid t \in units(r)\} \subseteq clear(r).$$

3. (Normal/Reverse-Table condition) Every point on a route is in either the normal table or the reverse table of that route:

$$\forall r \in \mathsf{Route} : \{p \in \mathsf{Point} \mid p \in units(r)\} \\ \subseteq normal(r) \cup reverse(r).$$

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

$$\forall r_1, r_2 \in \mathsf{Route} : \\ r_1 \neq r_2 \wedge sharedPoints(r_1, r_2) \neq \emptyset \Rightarrow \\ \exists p \in sharedPoints(r_1, r_2) : \\ p \in reverse(r_1) \cap normal(r_2) \vee \\ p \in reverse(r_2) \cap normal(r_1)$$

where $sharedPoints(r_1, r_2) = units(r_1) \cap units(r_2) \cap \mathsf{Point}$.

All scheme plans that we looked at together with our industrial partners were fulfilling these conditions.

5 Background to CSP||B

The CSP||B approach allows us to specify communicating systems using a combination of the B-Method [1] and the process algebra Communicating Sequential Processes (CSP) [11]. The 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 [34] 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 [27]. Therefore, our notion of consistency is that a combined communicating system $Ps \parallel Ms$ is *divergence free*. We do not consider deadlock freedom in this paper as it is concerned with liveness, and the focus of the paper is on safety.

A B MACHINE clause declares a machine and gives it a name. The VARIABLES of a B machine define its state. The INVARIANT of a B machine gives the type of the variables, and more generally it also contains any other constraints on the allowable machine states. There is an INITIALISATION which determines the initial state of the machine. The machine consists of a collection of OPERATIONS that query and modify the state. Operations take one of two forms:

- *Preconditioned operation*—PRE P THEN S END: if this is called when P holds then it will execute S, otherwise it will diverge.
- *Guarded event*—SELECT P THEN S END: this will execute S when P holds, and will *block* when P is false.

Besides this kind of machine, we also define static B machines that provide only sets, constants and properties that do not change during the execution of the system.

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

$$\begin{array}{lll} P & ::= & c?x!y \to P(x) \mid P_1 \,\Box\, P_2 \mid P_1 \sqcap P_2 \mid \\ & & \textbf{if } b \textbf{ then } P_1 \textbf{ else } P_2 \textbf{ end} \mid N(exp) \mid \\ & & P_1 \parallel P_2 \mid P_1 \,{}_A\|_B\, P_2 \mid P_1 \mid\mid\mid P_2 \end{array}$$

The process $c?x!y \to P(x)$ defines a channel communication where x represents all data variables on a channel, and y represents values being passed along a channel. Some of these channels match with operations in a corresponding B machine with the signature $x \longleftarrow c(y)$. Therefore the input y of the B operation c 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 alphabetised parallel operator, $P_{1A}\|_B P_2$, requires synchronisation only in $A \cap B$, allowing independent performance of events outside this set. The interleaving operator, $P_1 \mid\mid\mid P_2$, 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.

For reasoning of 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

$$\sigma = \langle s_0, e_1, s_1, e_2, \ldots, e_n, s_n \rangle$$

where the s_i, $i = 0 \ldots n$, are states of the B machine and the e_i, $1 \leq i \leq n$, are events. Here, we assume that s_0 is a state after initialisation. Given a system run σ, we can extract its trace of events:

$$events(\sigma) = \langle e_1, \ldots, e_n \rangle.$$

6 Modelling railway systems in CSP||B

As outlined in [25], CSP||B caters for the double nature of railways by addressing the 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 centralised 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 full CSP||B model is given in Appendix A.

To tailor the CSP||B model to the ProB [19] tool which we are using for analysis, we put the DSL model of Sect. 3 into a particular form. For example, in the DSL the release table is given by $release : \mathsf{Point} \to \wp(\mathsf{Route} \times \mathsf{Unit})$. However, when considering the movement of trains it is more efficient to capture the information indexed by the track, so the locks released on any particular move are given directly by the position the train has moved to. In the B description we use the name *releaseTable* for explicitness. The relationship between the DSL terminology and the CSP||B terminology is given in Fig. 7. The main difference is the use of Point as the name of the point (e.g. $P101$) rather than the unit associated with it (e.g. DD), and the use of Track to cover both kinds of units. However, this is mainly a matter of convenience and it is straightforward to translate between the two approaches. For the purposes of this paper we consider tracks to be unidirectional.

The trains and controller processes run independently of each other, on the CSP level expressed with an interleaving operator—see Fig. 8 (lines 20 and 21). It is a decision of

DSL	CSP||B
Unit	Track
Point	$\mathrm{ran}(homePt)$
Track	Track – $\mathrm{ran}(homePt)$
name of Point	Point
normal	*normalTable*
reverse	*reverseTable*
clear	*clearTable*
release	*releaseTable*
Point → ℘(Route × Unit)	Track ↔ (Route × Point)

Fig. 7 Relationship between DSL terminology and CSP||B terminology

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

Fig. 8 CSP control processes for controller and trains

the controller which routes are requested to be set or to be released (lines 2–4). Similarly, it is a decision of the train to move through a red light by maximally one track and subsequently stop or to wait for a signal change (lines 13–15). This logic is sometimes referred to as the *driving rules* of a train.

The *Interlocking* machine captures information about the location of trains on tracks using the function *pos* : Train → AllTrack where *pos*(*t*) gives the location of the train *t*. The position of a train consists of exactly one track. It is here we assume that the train's length is smaller than that of a track.

The set AllTrack represents all the tracks and the special *nullTrack* which denotes a nonvalid track used for modelling run-through. The machine also captures the current information about successor tracks through a dynamic function *nextd* : AllTrack → AllTrack which is dependent upon the position of the points. Furthermore, the machine captures information about signal settings using the function *signalStatus*, last moved points using the set *movedPoints*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 six operations of the *Interlocking* machine.

```
1  bb ⟵ 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 on track preceding the route
12        (ie, nothing going through red light) */
13     homeSig(signal(route)) ∈ emptyTracks
14    THEN
15     /* signal of route to red */
16     signalStatus(signal(route)) := red ||
17     /* release locks associated with route */
18     currentLocks := route ⩤ currentLocks ||
19     /* release is successful */
20     bb := yes
21    ELSE
22     bb := no
23    END
24   END
25 END
```

Fig. 9 *Release* operation from *Interlocking*

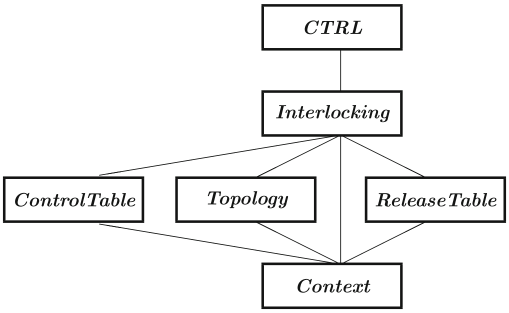


Fig. 10 Architecture

Figure 9 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". Note that a signal of a route may also be set to red when a train occupies the first track section of the route, to avoid several trains to enter the route.

Figure 10 shows the overall architecture of our modelling. The CSP controller and the *Interlocking* machine are independent of any particular scheme plan. They are supported by a *Topology*, a *ControlTable*, a *ReleaseTable* 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. A typical example from the *ControlTable* machine which splits up the modelling of a control table into two relations and one function is given as follows:

$$\begin{aligned} &normalTable \in \mathsf{Route} \leftrightarrow \mathsf{Point} \ \wedge \\ &reverseTable \in \mathsf{Route} \leftrightarrow \mathsf{Point} \ \wedge \\ &clearTable \ \ \in \mathsf{Route} \rightarrow \mathbb{P}(\mathsf{Track}) \end{aligned}$$

A predicate is used to define the relationship between the *Interlocking* machine and the *CTRL* process relates the train parameter *t* and the train position *pos* of the *TRAIN_CTRL* process to the *pos* function within the *Interlocking* machine. This control loop invariant predicate must hold at each recursive call, and hence the system is divergence free.

The *Interlocking* machine uses guarded events to model the safety properties. The guards are enabled in unsafe states which will violate our safety properties. Use of these guarded events does not impact on the divergence freedom requirement of a CSP||B model since they have no affect on the state and do not themselves diverge.

In Sect. 2 we introduced the collision freedom property. In our B machine, we encode an operation which captures the notion of a collision, as follows:

```
collision =
SELECT
   $\exists t_1, t_2 \in \mathsf{Train} : t_1 \neq t_2 \wedge$
      $(\{pos(t_1)\} \cap \{pos(t_2)\}) \setminus (\mathsf{Exit} \cup \mathsf{Entry}) \neq \emptyset$
 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 two trains are at the same position.

Collision freedom can then be established by model checking the validity of the following CTL formula:

$$AG(not(e(collision)))$$

This formula is false if *collision* is enabled. In the CTL variant of ProB *AG* stands for "on all paths it is globally true that", and $e(a)$ stands for "event *a* is enabled".

7 Finitisation

In this section, we develop a theory of how to reduce the problem of verifying of scheme plans for safety (i.e. freedom from collision, derailment and run-through) for any number of trains to that of a two-train scenario. We introduced this idea first for run-through freedom in [24]. Here, we give full proofs on a slightly more involved CSP||B model and generalise it to collision freedom and derailment freedom.

Finitisation requires scheme plans to fulfil a number of well-formedness conditions as outlined in Sect. 4.4. In Sect. 7.1 we establish a reduction theorem (Theorem 3) for such well-formed scheme plans w.r.t. the number of trains involved in a system run. 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 release commands. The validity of this finitisation argument for safety is demonstrated in Sect. 7.2.

Given a scheme plan *SP*, and an unlimited collection Train of trains, we write CSP || B(*SP*, Train) for the instantiation of our generic CSP||B model with *SP* and Train. Note that CSP || B(*SP*, Train) in general is an infinite state system due to the inclusion of train identifiers into events and states. 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 of trains contains two elements only.

7.1 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 Train of trains. Prior to any state in which a signal sig $\in$ Signal shows green, there is a uniquely determined event in σ of the form request.r.yes for some r $\in$ Route which caused that 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* (i.e. when a route is successfully requested). Conversely, a signal is set to red only by the events *move* and *release* (i.e. when a train passes a signal and when a route is successfully released). Analysing a system run where *sig* is green in the last state yields that the route is uniquely determined. □

In the following we show that for every system run σ involving a set $A \uplus B$ of trains there exists a system run σ' which involves trains only from *A*, and where the trains from *A* move identically to σ. In particular: if trains from *A* collide in σ, then they collide in σ'; if a train in *A* derails in σ, then it derails in σ'; and if a train has a run-through in σ, then the same happens in σ'. We obtain σ' constructively from σ by defining a replacement function on events. To this end, we first identify those events which are related to B.

Definition 2 Given a set B of train identifiers, we define the set $E(B)$ of *events of B* as

$$\begin{aligned} E(B) = \ & \{enter.b \mid b \in B\} \cup \\ & \{exit.b \mid b \in B\} \cup \\ & \{nextSignal.b \mid b \in B\} \cup \\ & \{move.b.cp.np \mid b \in B \wedge cp, np \in \mathsf{AllTrack}\} \end{aligned}$$

The next step is to define the replacement function which is dependent on the current state.

$replace_B(S, e) =$

- e, if $e \notin E(B)$;
- $release.r.yes$, if $e = move.b.cp.np$ for some $b \in B$ and $\exists\, s \in \mathsf{Signal}$ such that
  - $homeSig(s) = cp$,
  - $signalStatus_S(s) = green$,
  - $\exists!r \in \mathsf{Route} : signal(r) = s$, and
  - $currentLocks_S(r) = lockTable(r)$;
- $idle$, otherwise.

Note that, in the above definition, when we replace a forward move event $move.b.cp.np$ in front of a green signal by a route release event $release.r.yes$, Theorem 1 guarantees the existence of such a unique route r.

To cater for this model transformation, we enriched our CSP||B model with an event *idle* that does nothing. On the CSP side, this means the addition of a new process $IDLE = idle \rightarrow IDLE$ to the controller; on the B side, this means the addition of a new operation $idle = movedPoint := \emptyset$.. This process is only needed for the justification of our model transformation; it is not required for the verification of safety.

Removing the trains in the set B from a system run also affects the states of the B machine. For example, one component of a B machine state S is the map$pos_S : \mathsf{Train} \rightarrow \mathsf{AllTrack}$ which stores for each train the track it occupies and the direction it moves. Recall from Sect. 6 that $\mathsf{AllTrack}$ contains the special *nullTrack* for modelling run-through. If we remove the trains in B, we would hope that for the corresponding state T the following relation holds:

$$pos_T = pos_S|_{(\mathsf{Train} \setminus B)}.$$

That is, the mapping pos_T should be the same as pos_S, but be defined over the restricted domain $\mathsf{Train} \setminus B$. The correspondence between states may, however, be more than just a projection onto the remaining trains. This consideration motivates the following definition.

Definition 3 Let S and T be states of the B machine of CSP || B(SP, Train) and let $B \subseteq \mathsf{Train}$ be a set of trains. State T is *in B-correspondence to* state S, written $T \leq_B S$, iff the following nine conditions are fulfilled.

f.1: $pos_T = pos_S|_{(\mathsf{Train} \setminus B)}$.
f.2: $nextd_T = nextd_S$.
f.3: $signalStatus_T = signalStatus_S$.
f.4: $normalPoints_T = normalPoints_S$.
f.5: $reversePoints_T = reversePoints_S$.
f.6: $movedPoints_T = movedPoints_S$.
f.7: $\forall\, r \in \mathsf{Route}$.

$currentLocks_T[\{r\}] = currentLocks_S[\{r\}]$ or
$currentLocks_T[\{r\}] = \emptyset$.

(The run without the trains of B either has the same locks for a route or none at all.)

f.8: $\forall\, s \in \mathsf{Signal}$. if $signalStatus_S(s) = green$ then there is a unique $r \in \mathsf{Route}$ such that

$$\begin{aligned} signal(r) &= s, \\ currentLocks_S(r) &= lockTable(r), \text{ and} \\ currentLocks_T(r) &= lockTable(r). \end{aligned}$$

(If a signal is green, then there exists exactly one route associated with that signal which is set.)

f.9: $\forall\, b \in B,\ r \in \mathsf{Route}$. if $pos_S(b) \in units(r)$ then $currentLocks_T(r) = \emptyset$.
(The locks of any route that contains a track segment occupied by a train $b \in B$ in state S have been released in state T.)

With the above correspondence in place, we want to establish the following simulation properties:

(a) For states S and T with $T \leq_B S$, if event e is enabled in S, then $replace_B(S, e)$ is enabled in T;
(b) furthermore, the states S' and T' which result from performing these events are themselves in B correspondence, i.e. $T' \leq_B S'$.

The following diagram illustrates this situation:

$$\begin{tikzcd} S \arrow[d, phantom, "\rotatebox{90}{$\leq_B$}"] & e & S' \arrow[d, phantom, "\rotatebox{90}{$\leq_B$}"] \\ T & replace_B(S, e) & T' \end{tikzcd}$$

We establish these two properties under a condition on the set B. We say that the trains in B never cause a collision in a system run, if in this run the collision event is never enabled with a train $t \in B$ as a witness, i.e. if there is no state in which

$\exists t_1, t_2 \in \mathsf{Train} : t_1 \neq t_2 \wedge (\{pos(t_1)\} \cap \{pos(t_2)\}) \setminus (\mathsf{Exit} \cup \mathsf{Entry}\ ) \neq \emptyset \wedge (t_1 \in B \vee t_2 \in B)$.

Lemma 1 *Given a scheme plan SP and a set* Train *of trains containing* $B \subseteq \mathsf{Train}$*, if* σ *is a system run of* $CSP \,||\, B(SP, \mathsf{Train})$ *in which trains in B do not cause a collision, then* $replace_B(\sigma)$ *is a system run of the B machine of* $CSP\,||\,B(SP, \mathsf{Train} \setminus B)$.

Proof The proof is by induction on the length of σ. The base case is trivial, and the induction cases are generally unproblematic. □

Lemma 1 allows us to extend the function $replace_B$ to system runs $\sigma = \langle S_0, e_1, S_1, \ldots, e_k, S_k\rangle$ as follows.

$$replace_B(\sigma) = \langle T_0, replace_B(S_0, e_1), \ldots, \\ T_{k-1}, replace_B(S_{k-1}, e_k), T_k\rangle$$

Here, $T_0 = S_0$ (the initial state). Lemma 1 guarantees that for all $1 \leq i \leq k$, $replace_B(S_{i-1}, e_i)$ is enabled in T_{i-1} and leads to T_i with $T_i \leq S_i$.

With this result in place, we show that the events of $replace_B(\sigma)$ give a trace of the CSP controller.

Lemma 2 *Given a scheme plan SP and a set* Train *of trains containing* $B \subseteq \mathsf{Train}$*, if* σ *is a system run of* $CSP \,||\, B(SP, \mathsf{Train})$*, then* $events(replace_B(\sigma))$ *is a trace of the CSP controller* $CTRL(SP, \mathsf{Train} \setminus B)$.

Proof Using process algebraic laws, one shows that projections of the trace $events(replace_B(\sigma))$ are traces of the individual processes out of which the controller process $CTRL(SP, \mathsf{Train} \setminus B)$ is built. □

Combining these two lemmas gives the following result.

Theorem 2 *Given a scheme plan SP and a set* Train *of trains containing* $B \subseteq \mathsf{Train}$*, if* σ *is a system run of* $CSP \,||\, B(SP, \mathsf{Train})$ *in which trains in B do not cause a collision, then* $replace_B(\sigma)$ *is a system run of* $CSP\,||\,B(SP, \mathsf{Train} \setminus B)$.

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

7.2 Verification for safety

Safety in the models is dependent on the number of trains which are introduced into the model. This motivates the following definition.

Definition 4 Let

$ERROR = \{collision, derailment, run - through\}$

be the set of error events of interest.

- A scheme plan SP is *n-e-free* (for $n \in \mathbb{N}_{>0}$ and $e \in ERROR$) iff e is never enabled in any state of any $\sigma \in \mathrm{CSP}\,||\,\mathrm{B}(SP, \mathsf{Train})$ with $|\mathsf{Train}| = n$.
- A scheme plan SP is *safe* iff it is n-e-free for all $n \in \mathbb{N}_{>0}$ and $e \in ERROR$.

We can now turn Theorem 2 into a proof method. The following Corollary is the basis of finitisation.

Corollary 1 *A scheme plan SP is safe if it is* 2*-collision free,* 1*-derailment free and* 1*-run* − *through free.*

Proof Assume that SP is not safe, i.e. it is not n-e-safe for some $n \in \mathbb{N}_{>0}$ and $e \in ERROR$. This means that there is a run σ of $\mathrm{CSP}\,||\,\mathrm{B}(SP, \mathsf{Train})$ with $|\mathsf{Train}| = n$ such that e is enabled in some state of σ.

Let $\sigma = \langle S_0, e_1, S_1, \ldots, e_k, S_k\rangle$. Without loss of generality, let us assume that

- (C1) e is enabled in S_k; and
- (C2) $\forall\, e' \in ERROR : e'$is not enable in$S_0, \ldots, S_{k-1}$.

We consider each error type in turn.

Case 1: $e = collision$.

- By (C1), $\exists t_1, t_2 \in \mathsf{Train}, t \in \mathsf{Track}$ such that $t = pos_{S_k}(t_1) \wedge t = pos_{S_k}(t_2)$;
- by (C2), e_k is a *move* of t_1 or t_2;
- trains in $\mathsf{Train} \setminus \{t_1, t_2\}$ do not cause collision in σ;
- by Theorem 2, $replace_{\mathsf{Train}\setminus\{t_1,t_2\}}(\sigma)$ is a run of $\mathrm{CSP}\,||\,\mathrm{B}(SP, \{t_1, t_2\})$;
- $T_k \leq_{\mathsf{Train}\setminus\{t_1,t_2\}} S_k$, where T_k is the last state in $replace_{\mathsf{Train}\setminus\{t_1,t_2\}}(\sigma)$;
- By (f.1), $t = pos_{T_k}(t_1) \wedge t = pos_{T_k}(t_2)$;
- *collision* is enabled in T_k;
- SP is not 2-collision free.

Case 2: $e = derailment$.

- By (C1), $\exists t \in \mathsf{Train}, p \in movedPoints_{S_k}$ such that $homePt(p) = pos_{S_k}(t)$;
- by (C2), e_k is a $request.r.yes$;
- trains in Train do not cause collision in σ;
- by Theorem 2, $replace_{\mathsf{Train}\setminus\{t\}}(\sigma)$ is a run of $\mathrm{CSP}\,||\,\mathrm{B}(SP, \{t\})$;
- $T_k \leq_{\mathsf{Train}\setminus\{t\}} S_k$, where T_k is the last state in $replace_{\mathsf{Train}\setminus\{t\}}(\sigma)$;

- By (f.6) and (f.1), $p \in movedPoint_{T_k} \wedge homePt(p) = pos_{T_k}(t)$;
- derailment is enabled in T_k;
- SP is not 1-derailment free.

Case 3: $e = run - through$.

- By (C1), $\exists\, t \in \mathsf{Train}$ such that $nullTrack = pos_{S_k}(t)$;
- By (C2), e_k is a *move* of t;
- trains in Train do not cause collision in σ;
- by Theorem 2, $replace_{\mathsf{Train}\setminus\{t\}}(\sigma)$ is a run of CSP || B$(SP, \{t\})$;
- $T_k \leq_{\mathsf{Train}\setminus\{t\}} S_k$, where T_k is the last state in $replace_{\mathsf{Train}\setminus\{t\}}(\sigma)$;
- By (f.1), $nullTrack = pos_{T_k}(t)$;
- $run - through$ is enabled in T_k;
- SP is not 1-run-through free.

□

Corollary 1 works with different numbers of trains: two trains are needed in the case of *collision*, one train is needed otherwise. 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-e-free, then SP is k-e-free for any $k < n$.*

Proof If SP is not k-e-safe, then there exists a run $\sigma \in$ CSP || B(SP, Train) with $|\mathsf{Train}| = k$ such that e is enabled in some state of σ. But σ is also a run of CSP || B(SP, Train') where $\mathsf{Train} \subseteq \mathsf{Train}'$, with $\left|\mathsf{Train}'\right| = n$. □

8 Covering

In the following, we develop a theory of covering a scheme plan with a set of smaller sub-scheme plans in such a way that safety of all sub-scheme plans implies safety of the original scheme plan.

The fundamental idea of covering is that any violation of a safety property happens at a specific location. We can say at which (set of) locations L a collision, a run-through or a derailment happens in the track plan. A set of locations L can be influenced in two different ways: (i) a train reaches a location in L or (ii) a train releases a lock of a point which lies on a route towards L. In Sect. 8.1 we provide a construction that, given a set L, computes a set L_∞ which is closed under both influences listed above and includes L. The construction is described using our DSL for the railway domain; see Sect. 4. Thus, it is part of the domain. Consequently, the construction is open for re-use in any modelling formalism.

In Sect. 8.2 we prove in the context of our CSP||B modelling that safety of all sub-scheme plans implies safety of the original scheme plan. First we prove: for any run σ in the CSP||B model of the original scheme plan and for any set L there is a corresponding run σ_L in the CSP||B model of the sub-scheme plan constructed for L. From this result, we prove as corollary: if the CSP||B models of the sub-scheme plans are safe for all sets L to be considered for a specific safety property, then the CSP||B model of the original scheme plan is safe as well. Our proofs in Sect. 8.2 are tightly bound to the language CSP||B; however, in the context of modelling scheme plans in CSP [22], we proved as well that the covering construction of Sect. 8.1 allows compositional verification.

8.1 Domain inherent covering construction

Given a scheme plan $SP = (Top, CT, RTs)$ as described in Sect. 4 and a set $L \subseteq \mathsf{Track} \setminus (ENTRY \cup EXIT)$ of "critical tracks", we describe the construction of a scheme plan $SP_L = (Top_L, CT_L, RTs_L)$. The scheme plan SP_L will be used to investigate safety at tracks in L.

In a first step, we consider all tracks over which a train can travel on the topology towards a track in L. Figure 11 provides an illustration for all notions introduced below.

First, we give a construction that collects the tracks of L together with all tracks over which a train can travel on the topology towards a track in L:

$$Cone(L) = \{u \in \mathsf{Unit} \mid \exists \text{ path } p : hd(p) \in ENTRY,\ last(p) \in L, u \in p\}$$

One can think of each element of L as the apex of a cone and of $Cone(L)$ as the union of these cones—see Fig. 11a.

Then, we define the set of all topological routes that share a unit with L:

$$Routes(L) = \{r \in \mathsf{TopoRoute} \mid \exists\, u \in L : u \in topoUnits(r)\}$$

The *Region* of L consists of those units which are on a route directly leading to L—see Fig. 11b:

$$Region(L) = Cone(L) \cap \left(\bigcup_{r \in Routes(L)} topoUnits(r) \right)$$

We close the region by adding suitable entry and exit units:

$$\begin{aligned} Entries(L) &= (predecessor(Region(L)) \setminus Region(L)) \\ &\quad \cap Cone(L) \\ Exits(L) &= \{u \in successor(Region(L)) \setminus Region(L) \mid \\ &\quad \exists \text{ path p} : \\ &\quad hd(p) \in Entries(L) \wedge last(p) = u\} \end{aligned}$$

where the *successor* and *predecessor* functions are applied point-wise to the set. The *ClosedRegion*—see Fig. 11c—finally is

$$ClosedRegion(L) = Region(L) \cup Entries(L) \cup Exits(L)$$

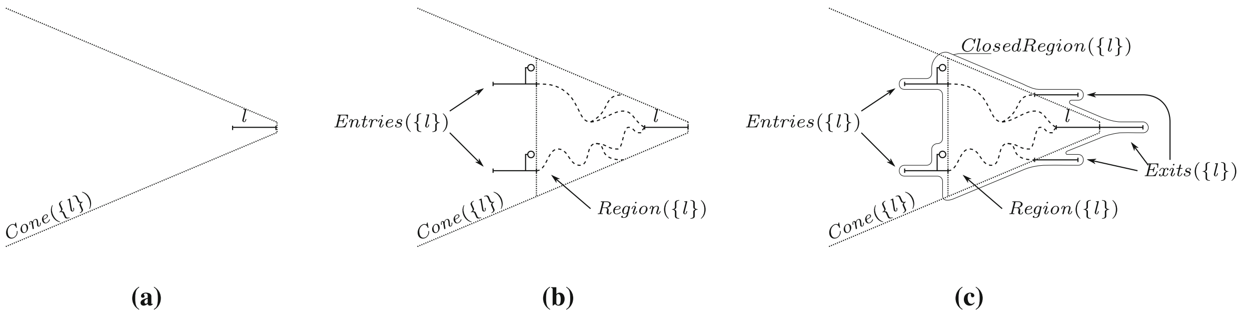


Fig. 11 Influence region (all track directions are *left to right*)

We illustrate this construction by an example:

Example 1 [Closed region of track DF] For track DF of the scheme plan shown in Fig. 2, we compute:

- $Cone(\{DF\}) = \{EN1, DA, DB, DC, DD, DE, DF\}$,
- $Routes(\{DF\}) = \{\langle DC, DD, DE, DF\rangle, \langle DF, DG, DH\rangle\}$,
- $Region(\{DF\}) = \{DC, DD, DE, DF\}$,
- $Entries(\{DF\}) = \{DB\}$,
- $Exits(\{DF\}) = \{DG, UE\}$, and
- $ClosedRegion(\{DF\}) = \{DB, DC, DD, DE, DF, DG, UE\}$.

Note that we include the units of two routes into the *Routes* of DF. This is the case as trains are allowed to overrun a red signal by one track; thus $\langle DC, DD, DE, DF\rangle$ is included. UE is an exit as there is a path from the entry DB to UE.

In the second step, we take the release tables into account for our construction. Here, we want to include all tracks that can release a point in $ClosedRegion(L)$.

Given a route $r \in Routes(L)$, the signal $topoSignal(r)$ can control further routes which not necessarily share a unit with L. To collect these routes, we define

$$Signals(L) = \{s \in \mathsf{Signal} \mid \exists\, r \in Routes(L) : topoSignal(r) = s\}$$

and

$$RouteNames(L) = \{r \in \mathsf{Route} \mid signal(r) \in Signals(L)\}$$

Note that $RouteNames(L)$ consists of names defined in the control table rather than of topological routes.

We are now ready to define the influence zone on a track by closing under topological influence and point releases. To this end, we define the following iteration:

- We set $L_0 = L$.
- For $i \geq 0$, let

  $$\begin{aligned} L_{i+1} = L_i \cup \{t \in \mathsf{Unit} \setminus ClosedRegion(L_i) \mid{} & \exists\, p \in \mathsf{Point} \cap Region(L_i), \\ & \exists\, r \in RouteNames(L_i) : \\ & (r, t) \in release(p)\} \end{aligned}$$

  Here, we increase the set L_i of critical tracks by those tracks in the release tables *RTs* which refer to a point in $Region(L_i)$ and belong to a route which is controlled by a signal in $Signals(L_i)$.

Let L_∞ be the smallest fixed point of the iteration, i.e. the first appearance of $L_i = L_{i+1}$. As the topology consists of finitely many tracks and points, the iteration terminates.

Example 2 (Continuation of Example 1) $P101$ is the only point in $Region(L_0)$. In the release table of $P101$, we find $\{(R12A, DE), (R12B, UE)\} \subseteq release(p101)$ for route names in $RouteNames(L_0) = \{R12A, R12B, R14\}$. Thus, DE and UE are the potential candidates to be added to L_0. As $DE, UE \in ClosedRegion(L_0)$, we have $L_1 = L_0$.

Note that $Entries(L) \subseteq \mathsf{Track}$ thanks to the condition that signals are never located at points. To increase the readability of our proofs in the next section, concerning $Exits(L)$ we add the slightly weaker *exit condition*: for any point $p \in Exits(L)$ it holds that p shares exactly one connector with $Region(L)$.

Given a set L_∞ for which the exit conditions holds, we construct a new scheme plan $SP_L = (Top_L, CT_L, RTs_L)$ as given in Fig. 12. The tracks of SP_L are all the tracks in the closed region of L_∞ together with those points of the closed region which are used in one direction only and thus can be turned into tracks. The points of SP_L are all points within the closed region of L_∞ which have not been turned into tracks. For ease of construction we keep the old set of connectors. The connectivity of the new topology is given by choosing appropriate connectors and directions for the tracks in Track_L and the points in Point_L.

For $t \in \mathsf{Track}_L$, we define:

$$
\begin{aligned}
\mathsf{Track}_L = {} & \{t \in \mathit{ClosedRegion}(L_\infty) \mid t \in \mathsf{Track}\} \\
& \cup \{p \in \mathsf{Point} \mid p \in \mathit{Exits}(L_\infty)\} \\
& \cup \{p \in \mathit{Region}(L_\infty) \mid p \in \mathsf{Point} \land \\
& \qquad \mathit{predecessor}(p) \cup \mathit{successor}(p) \\
& \qquad\qquad \not\subseteq \mathit{ClosedRegion}(L_\infty)\} \\
\mathsf{Point}_L = {} & \{p \in \mathsf{Point} \mid p \in \mathit{ClosedRegion}(L_\infty) \setminus \mathsf{Track}_L\} \\
\mathsf{Unit}_L = {} & \mathsf{Track}_L \cup \mathsf{Point}_L \\
\mathsf{Connector}_L = {} & \mathsf{Connector} \\
\mathsf{Signal}_L = {} & \{s \in \mathsf{Signal} \mid \mathit{signalAt}(s) \in \mathit{ClosedRegion}(L_\infty)\} \\
\mathit{signalAt}_L(s) = {} & \mathit{signalAt}(s) \\
\mathit{RouteNames}_L = {} & \mathit{RouteNames}(L_\infty) \\
\mathit{clear}_L(r) = {} & \mathit{clear}(r) \cap \mathit{Region}(L_\infty) \\
\mathit{normal}_L(r) = {} & \mathit{normal}(r) \cap \mathsf{Point}_L \\
\mathit{reverse}_L(r) = {} & \mathit{reverse}(r) \cap \mathsf{Point}_L \\
\mathit{release}_L(p) = {} & \mathit{release}(p) \cap \\
& (\mathit{RouteNames}_L \times \mathit{ClosedRegion}(L_\infty))
\end{aligned}
$$

Fig. 12 The scheme plan $SP_L = (Top_L, CT_L, RTs_L)$

- If $t \in \mathsf{Track}$, nothing changes, i.e. $c_{1L}(t) = c_1(t)$ and $c_{2L}(t) = c_2(t)$; and $\mathit{directions}_L(t) = \mathit{directions}(t)$.
- If $t \in \mathsf{Point}$ such that $t \in \mathit{Exits}(L_\infty)$ we know that t shares only one connector, say c, with $\mathit{Region}(L_\infty)$. In this case we turn the point into a track. We keep the connector where the point joins $\mathit{Region}(L_\infty)$ and allow travel out of the region, i.e. we set $c_{1L}(t) = c$ and choose for $c_{2L}(t)$ one of $\mathit{connectors}(t) \setminus \{c\}$; $\mathit{directions}(t) = (c_{1L}(t), c_{2L}(t))$.
- If $t \in \mathsf{Point}$ with $t \in \mathit{Region}(L_\infty)$ such that one arm of t ends outside, i.e. $\mathit{predecessor}(t) \cup \mathit{successor}(t) \not\subseteq \mathit{ClosedRegion}(L_\infty)$, we turn the point into a track. We keep those connectors which are on a path towards a unit in L_∞ and allow travel along this path. Let $c \in \mathit{connectors}(t)$ be the connector leading out of the region, i.e. for all $u \in \mathit{ClosedRegion}(L_\infty) \setminus \{t\}$: $c \notin \mathit{connectors}(u)$. Then, choose as $c_{1L}(t)$ one of $\mathit{connectors}(t) \setminus \{c\}$ and define $c_{2L}(t)$ to be the one element in $\mathit{connectors}(t) \setminus \{c, c_{1L}(t)\}$; set $\mathit{directions}_L(t) = \mathit{directions}(t) \cap \{(c_{1L}(t), c_{2L}(t)), (c_{2L}(t), c_{1L}(t))\}$.

For $p \in \mathsf{Point}_L$ nothing changes; i.e. $c_{1L}(p) = c_1(p)$, $c_{2L}(p) = c_2(p)$ and $c_{3L}(p) = c_3(p)$; and $\mathit{directions}_L(p) = \mathit{directions}(p)$.

8.2 Correctness proof of covering in CSP||B

Our encoding method for scheme plans into CSP||B is generic, i.e. given a scheme plan SP, we obtain an encoding CSP || B(SP). Similarly, given a set L of critical units, we obtain an encoding CSP || B(SP_L) of the above constructed scheme plan SP_L. In the following, we show that any run σ on CSP || B(SP) corresponds to a run σ_L on CSP || B(SP_L), where σ_L is obtained from σ by renaming of events and abstraction on the B states.

For ease of readability, we present our correctness proof for convex scheme plans SP_L only. SP_L is convex, if in SP trains cannot travel from a unit $u \in \mathit{Exits}(L)$ to a unit $v \in \mathit{Entries}(L)$. In our proof practice, all scheme plans SP_L have turned out to be convex. The results presented can easily be adapted to non-convex plans by either changing the construction of the closed region or by adding a renaming function on train identifiers that gives a fresh identifier to a train that enters the units of SP_L a second time. Both changes, however, lead to a plethora of notations that obscure the proof idea.

8.2.1 Run construction

Let SP be a scheme plan, let L be a set of critical units, and let σ be a run on CSP || B(SP). On the states of the B machine we define a function π_L to project states of CSP || B(SP) into states of CSP || B(SP_L). Let S be a state of CSP || B(SP), then the projection of S on CSP || B(SP_L) is a state T, written as $\pi_L(S) = T$ where:

cv1: $\mathit{pos}_T = \mathit{pos}_S \cap (\mathsf{Train} \times \mathsf{AllTrack}_L)$
cv2: $\mathit{nextd}_T = \mathit{nextd}_S \cap (\mathsf{Unit}_L \times \mathsf{AllTrack}_L)$
cv3: $\mathit{signalStatus}_T = \mathit{signalStatus}_S \cap (\mathsf{Signal}_L \times \mathsf{Aspect})$
cv4: $\mathit{normalPoints}_T = \mathit{normalPoints}_S \cap \mathsf{Point}_L$
cv5: $\mathit{reversePoints}_T = \mathit{reversePoints}_S \cap \mathsf{Point}_L$
cv6: $\mathit{movedPoints}_T = \mathit{movedPoints}_S \cap \mathsf{Point}_L$
cv7: $\mathit{currentLocks}_T = \mathit{currentLocks}_S \cap (\mathsf{Route}_L \times \mathsf{Point}_L)$

This projection has some simple but important properties:

- $\mathit{emptyTracks}_T = \mathit{emptyTracks}_S \cap \mathsf{AllTrack}_L$ and
- $\mathit{unlockedPoints}_T = \mathit{unlockedPoints}_S \cap \mathsf{Point}_L$.

In the following, we construct a sequence σ_L that we will prove to be a run of CSP || B(SP_L). The sequence σ_L is defined using the function $\mathit{replace}(S, e)$. $\mathit{replace}$ takes a state S of the B machine and an event e as arguments and returns an event. The result of this function is defined according to the table in Fig. 13: we match the structure of e against the patterns given in the first column—e being an event of the CSP||B encoding of our scheme plan—evaluate the condition—stated in our DSL—to obtain the replacement event e'. Roughly speaking, we keep all events that are within the scope of the scheme plan SP_L, and replace all events out of the scope of the scheme plan SP_L with idle.

Given a run $\sigma = \langle S_0, e_1, S_1, \ldots, S_{k-1}, e_k, S_k\rangle$, $k \geq 0$, we extend the above functions π_L and $\mathit{replace}_L$ to sequences:

$$
\begin{aligned}
\mathit{replace}_L(\sigma) = \langle & \pi_L(S_0), \mathit{replace}_L(S_0, e_1), \ldots \\
& \pi_L(S_{k-1}), \mathit{replace}_L(S_{k-1}, e_k), \pi_L(S_k)\rangle
\end{aligned}
$$

Then, we define $\sigma_L = \mathit{replace}_L(\sigma)$. Note that states in σ_L are gained by projection. This is in contrast to our construction

Fig. 13 Definition of the $replace_L$ function on events for covering

e	Condition	e'
$enter.id.p$	$p \in \mathsf{Unit}_L$	e
	$p \notin \mathsf{Unit}_L$	$idle$
$exit.id.p$	$p \in \mathsf{Unit}_L$	$exit.id.p$
	$p \notin \mathsf{Unit}_L$	$idle$
$request.r.yes$	$r \in \mathsf{Route}_L$	e
	$r \notin \mathsf{Route}_L$	$idle$
$request.r.no$	$true$	$idle$
$release.r.yes$	$r \in \mathsf{Route}_L$	e
	$r \notin \mathsf{Route}_L$	$idle$
$release.r.no$	$true$	$idle$
$nextSignal.id.as$	$pos_S(id) \in \mathsf{Track}_L$	$nextSignal.id.as$
	otherwise	$idle$
$move.id.cp.np$	$cp \in \mathsf{Unit}_L, np \in \mathsf{Unit}_L$	$move.id.cp.np$
	$cp \in \mathsf{Exit}_L, np \notin \mathsf{Unit}_L$	$exit.id.cp$
	$cp \notin \mathsf{Unit}_L, np \in \mathsf{Entry}_L$	$enter.id.newp$
	otherwise	$idle$

for finitisation. The difference between the constructions is that in the case of finitisation we have a relation between states, while in the case of covering we project states from the original run.

8.2.2 Proving the run in CSP||B

It remains to show that σ_L is actually a system run on CSP || B(SP_L). To this end we want to mimic train movements on the original scheme plan by entering of a train into the scheme plan SP_L—see the last row of the table in Fig. 13, condition $cp \notin \mathsf{Unit}_L, np \in \mathsf{Entry}_L$. This is only possible for runs where the preconditions of the *enter* operation in the B machine are true, i.e. the following *enter property* holds. For all tracks $t \in \mathsf{Entry}_L$, events $e = move.id.x.t \in \sigma$, $id \in \mathsf{Train}$, $x \in \mathsf{Track}$, states $S \in \sigma$ where S is the state before e in σ, we have:

$(\{t\} \cup successor(t)) \cap dom(ran(pos_S)) = \emptyset,$

i.e. there is no train on t or the successor of t.

Lemma 3 *Given a scheme plan SP, a set L of critical units and a run $\sigma \in CSP||B(SP)$ with the enter property, then*

1. *σ_L is a run of the B machine of $CSP||B(SP_L)$.*
2. *$events(\sigma_L)$ is a trace of the CSP controller of CSP || $B(SP_L)$.*

Proof The proof is by induction on the length of σ and case distinction on the operations. The result w.r.t. the CSP controller uses process algebraic laws to decompose the controller and then explicitly shows that (projections) of the given trace are in the trace sets of the components. □

Corollary 2 *Given a scheme plan SP, a set L of critical units and a run $\sigma \in CSP||B(SP)$ with the enter property, then σ_L is a run of $CSP||B(SP_L)$.*

Proof By Lemma 3 and the definition of the semantics of CSP||B. □

8.2.3 Application to safety

It remains to utilize the above result for compositional reasoning concerning safety:

Corollary 3 *Let SP be a scheme plan, then the following holds:*

1. *If $CSP\,||\,B(SP_L)$ is collision free for all $L = \{u\}$ where $u \in \mathsf{Unit} \setminus (Entries \cup Exits)$, then $CSP\,||\,B(SP)$ is collision free.*
2. *If $CSP\,||\,B(SP_L)$ is run-through free for all $L = \{u\}$ where $u \in \mathsf{Point}$, then $CSP\,||\,B(SP)$ is run-through free.*
3. *If $CSP\,||\,B(SP_L)$ is derailment free for all $L = \{u\}$ where $u \in \mathsf{Point}$, then $CSP\,||\,B(SP)$ is derailment free.*

Proof Ad 1., collision freedom: assume that CSP || B(SP) $\not\models AG(\neg\ e(collision))$. Then, according to Corollary 1, there exists a shortest run σ of CSP || B(SP) involving only two distinct trains $id_1 \in \mathsf{Train}$ and $id_2 \in \mathsf{Train}$ such that the last state of σ, say S, enables the *collision* operation. Let $u = pos_S(id_1)$ be the track or point where the collision occurs. Note that $u \notin \mathsf{Entry} \cup \mathsf{Exit}$ thanks to the precondition of the B operation *collision*. By construction of $SP_{\{u\}}$, u cannot be an entry track of $SP_{\{u\}}$ or the successor of an entry track of $SP_{\{u\}}$. Therefore, σ has the enter property. By Corollary 2, we know that $\sigma_{\{u\}}$ is a run of CSP || B($SP_{\{u\}}$). By construction of $\sigma_{\{u\}}$, its last state is $T = \pi_{\{u\}}(S)$. By definition of $\pi_{\{u\}}$, equation cv1, we have $pos_T(S) = pos_S \setminus \{id \mapsto t \mid t \notin \mathsf{Unit}_{\{u\}}\}$. As $u \in \mathsf{Unit}_{\{u\}}$, $pos_T(id_1) = pos_T(id_2)$, i.e. trains id_1 and id_2 collide in the run $\sigma_{\{u\}}$ of CSP || B($SP_{\{u\}}$).

Ad 2., run-through freedom: assume that in the model CSP || B(SP) $\not\models AG(\neg\ e(runthrough))$. Then, according to

Corollary 1, there exists a shortest run σ of CSP || B(SP) involving only one train such that the last state of σ, say S, enables the $run - through$ operation. For this train with $id \in \mathsf{Train}$ it holds that $pos_S(id) = nullTrack$ in S, i.e. train id has run through a point which was not set for the train's direction. The last move of id in σ is of the form $e = move.id.cp.nullTrack$. Let S' be the state before e in σ and let $cp = pos_{S'}(id)$. Then, $nextds_{S'}(cp)$ is not defined. Thus, $successor(cp) \in \mathsf{Point}$. As any connector can belong to at most two units, $successor(cp)$ is uniquely defined. Let p be this point $successor(cp)$. Let σ' be the prefix of σ up to $S'\ e\ S''$. The run σ' has the enter property because at most one track is occupied in any state of σ. By Corollary 2, we know that $\sigma_{\{u\}}$ is a run of CSP || B($SP_{\{u\}}$). By definition of $\pi_{\{p\}}$, part cv1, we have $pos_{T''} = pos_{S''} \setminus \{id \mapsto d \mid t \notin \mathsf{Unit}_{\{u\}}\})$, therefore $pos_{T''}(id) = nullTrack$, i.e. train id has run through the point p which was not set for the train's direction.

Ad 3., derailment freedom: assume that in the model CSP || B(SP) $\not\models AG(\neg\ e(derailment))$. Then, according to Corollary 1, there exists a shortest run σ of CSP || B(SP) involving only one train such that the last state of σ, say S, enables the $derailment$ operation, i.e. for the train with $id \in \mathsf{Train}$ it holds in S:

$pos_S(id) \in homePoint(movedPoints)$

i.e. train id has derailed at the point $u = pos_S(id)$.

The run σ has the enter property because at most one track is occupied in any state of σ. Thus, by Corollary 2, we know that $\sigma_{\{u\}}$ is a run of CSP || B($SP_{\{u\}}$). By construction of $\sigma_{\{u\}}$, its last state is $T = \pi_{\{u\}}(S)$. By definition of $\pi_{\{u\}}$, part cv1, we have $pos_T(S) = pos_S \setminus \{id \mapsto d \mid t \notin \mathsf{Unit}_{\{u\}}\}$. As $u \in \mathsf{Unit}_{\{u\}}$, $pos_T(id) = pos_S(id)$. By definition of $\pi_{\{u\}}$ part 6, we have $movedPoints_T = movedPoints_S \setminus \{p \mid p \notin \mathsf{Point}_L\}$. As $u \in \mathsf{Point}_{\{u\}}$, we have $u \in movedPoints_T$. This means train id has derailed. □

Remark 1 (Localised safety) We work here with the safety properties as originally defined in Sect. 6. In our proof practice, this approach has been always successful. However, it is possible to define localised safety properties. For instance, one can define the localised safety property "no collision at unit u". The corollary above can be established with such localised safety properties which are weaker than the ones we work with.

9 Topological abstraction

In the following we define a theory for the abstraction of a scheme plan in such a way that safety of the abstraction implies the safety of the concrete scheme plan. This is motivated by [24] where we introduced an abstraction technique which allows the transformation of complex CSP||B models of scheme plans into less involved ones.

In this paper, as described in Sect. 3, the topology of the railway network has been enriched with connectors to be able to capture the dynamic direction of the points; therefore the CSP||B models are also more detailed. This means that we need to define an improved notion for the abstraction of scheme plans which reflects the fact that the topology of the railway network now contains connectors.

In this section the complex CSP||B models are formal representations of concrete scheme plans SP, referred to as SP_C = CSP || B(SP), whereas the less involved CSP||B models are referred to as SP_A. More formally, consider two scheme plans SP_C and SP_A. An *abstraction* (abs_t, abs_c) from SP_C to SP_A consists of

- a total function

  $$abs_t : \mathsf{AllTrack}_C \to \mathsf{AllTrack}_A$$

  satisfying

  $$abs_t[\mathsf{Track}_C] = \mathsf{Track}_A,$$
  $$abs_t(e) = efore \in ENTRY_C \cup EXIT_C, and$$
  $$abs_t(nullTrack) = nullTrack;$$

  and
- a partial function

  $$abs_c : \mathsf{AllConnector}_C \to \mathsf{AllConnector}_A$$

  satisfying

  $$abs_c(C0) = C0and$$
  $$abs_c[\mathsf{Connector}_C] = \mathsf{Connector}_A$$

such that the following 18 properties are satisfied:

a.1: $\mathsf{Entry}\ _A = \mathsf{Entry}\ _C$ (= Entry)
a.2: $\mathsf{Exit}\ _A = \mathsf{Exit}\ _C$ (= Exit)
a.3: $\mathsf{Point}_A = \mathsf{Point}_C$ (= Point)
a.4: $\forall p : \mathsf{Point}\ .\ (homePt_A(p) = abs_t(homePt_C(p)))$
a.5: if $\langle t_1, t_2\rangle$ is a path in SP_C, then $abs_t(t_1) = abs_t(t_2)$ or $\langle abs_t(t_1), abs_t(t_2)\rangle$ is a path in SP_A
a.6: $\forall at : \mathsf{Track}_A\ .\ abs^{-1}[\{at\}]$ is connected.
a.7: if $\langle at_1, at_2\rangle$ is a path in SP_A, then $\exists\, t_1, t_2 : \mathsf{Track}_C$. $t_1 \in abs^{-1}[\{at_1\}]$, $t_2 \in abs^{-1}[\{at_2\}]$ and $\langle t_1, t_2\rangle$ is a path in SP_C.
a.8: $\forall p : \mathsf{Point}\ .\ dynamicDirection_A(p)$
$= abs_c(dynamicDirection_C(p),$
where $abs_c(c_1, c_2) = (abs_c(c_1), abs_c(c_2))$
a.9: $\mathsf{Signal}_A = \mathsf{Signal}_C$
a.10: $\forall s : \mathsf{Signal}\ .\ (homeSig_A(s) = abs_t(homeSig_C(s)))$
(= $homeSig$)

e_C	Condition	e_A
$move.id.currp.newp$	$abs_t(currp) = abs_t(newp)$	$idle$
	$abs_t(currp) \neq abs_t(newp)$	$move.id.abs_t(currp).abs_t(newp)$
$move.id.currp.nullTrack$		$move.id.abs_t(currp).nullTrack$
e	$e \in \{$ $enter, exit, request, release$ $collision, derailment, runthrough\}$	e

Fig. 14 Topological abstraction: relating concrete to abstract events

a.11: if $\langle t_1, t_2\rangle$ is a path in SP_C, and $signalAt_C(s) = t_1$ for some signal s, then $\langle abs_t(t_1), abs_t(t_2)\rangle$ is a path in SP_A
a.12: $\mathsf{Route}_A = \mathsf{Route}_C$ $(= \mathsf{Route})$
a.13: $\forall\, r : \mathsf{Route}$.
$(abs_t^{-1}[clearTable_A(r)] = clearTable_C(r))$
a.14: $\forall\, e : \mathsf{Entry}$.
$(abs_t^{-1}[entryTable_A(e)] = entryTable_C(e))$
a.15: $normalTable_A = normalTable_C$ $(= normalTable)$
a.16: $reverseTable_A = reverseTable_C$ $(= reverseTable)$
a.17: $releaseTable_A =$
$\{(abs_t(t), (r, p)) \mid (t, (r, p)) \in releaseTable_C\})$
a.18: if $\langle t_1, t_2\rangle$ is a path in SP_C, and $t_2 \in dom(releaseTable_C)$ then $\langle abs_t(t_1), abs_t(t_2)\rangle$ is a path in SP_A

Conditions a.1, a.2, a.3, a.9 and a.12 simply state that the entry and exit tracks, points, signals and routes remain unchanged in an abstraction. The only condition that makes use of the abs_c function is a.8, which ensures that the direction of the points are maintained in an abstraction. All the other conditions map tracks, points and signals through the abstraction function abs_t. Conditions a.5, a.6 and a.7 are the interesting ones because these are the ones that constrain how tracks can collapse and how abstract and concrete paths map to each other. Finally, conditions a.13–a.18 ensure that the abstracted topology is correctly reflected in the control and release tables.

Theorem 4 provides the justification that it is enough to model check the abstract scheme plan SP_A to ensure that the required safety properties of the interlocking system hold, and then infer that the same properties hold for the concrete scheme plan SP_C.

Theorem 4 *If there is an abstraction from SP_C to SP_A, then:*

1. *if SP_A is collision free, then SP_C is collision free;*
2. *if SP_A is derailment free, then SP_C is derailment free; and*
3. *if SP_A is run-through free, then SP_C is run-through free.*

Proof (sketch) The conditions a.1–a.18 on the abs_t and abs_c functions are sufficient to ensure that concrete transitions can be matched with abstract ones. In more detail, any move that changes state—clearing a region; passing a signal; releasing a lock—will be matched by an abstract move given in Fig. 14 (or *idle* if the train remains on the same abstract track), and conditions for granting and releasing routes are matched.

The proof proceeds by setting up a linking relation between CSP||B (SP_C) and CSP||B (SP_A) to show that concrete runs are matched by abstract runs. Two states are linked if their signals, points and locks all match, and if the abstract train positions match the concrete train positions under abs_t. Given a matching pair of states, a concrete event transition to a concrete state can be matched by an abstract event transition to a matching abstract state. The proof establishes this by a case analysis on the events.

This means that any concrete run can be matched by an abstract run. Hence, any concrete run containing *collision*, *derailment* or *run−through* can be matched by an abstract run containing the same event. It follows that if no abstact run contains such events, then no concrete run can contain them either. □

If we consider SP_C to be the station based on Langley as shown in Fig. 2, there are no opportunities for abstraction to reduce tracks which satisfy the above conditions. This is not unusual in practice for large scheme plans, since there are limited opportunities to perform abstraction due to the lack of sequences of collapsible sequential tracks, i.e. ones that do not contain signals and are not used in the release tables. However, the benefit of our abstraction technique becomes clear when we apply this technique after we apply the covering technique, introduced in Sect. 8. After applying the covering technique the set of sub-scheme plans derived from SP_C will each contain many opportunities for abstraction. This is because some points in a sub-scheme plan are only considered in one direction, and so are treated as tracks. This gives rise to sequences of tracks which can then be collapsed. For example, Fig. 15 illustrates an example sub-scheme plan for Langley, which focuses on track UC and contains the opportunity to abstract tracks UE/EF and DRD/-DRE. It shows that $abs_t(DRD) = abs_t(DRE) = a\_DRD$, $abs_t(UE) = abs_t(UF) = a\_UE$, and maps t to a_t for all other track names t. abs_c is the corresponding mapping on connectors. Figure 16 illustrates the abstraction for this sub-scheme plan.

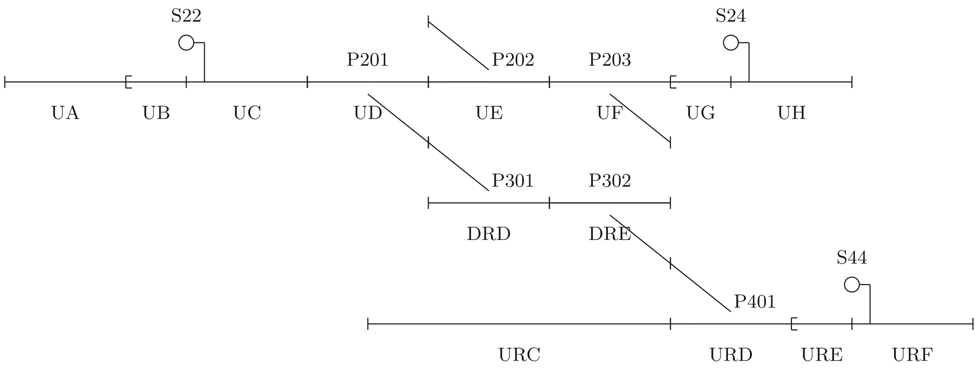


Fig. 15 Langley concrete sub-scheme plan SP_C (resulting from covering for track UC)

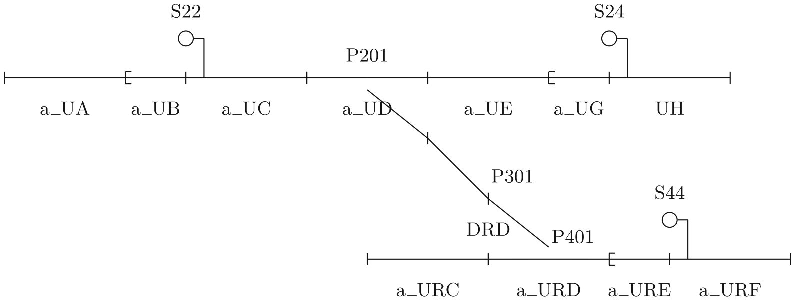


Fig. 16 Langley abstract sub-scheme plan SP_A

10 Experiments

In this section we outline various experimental results carried out on our models. We use the CTL model checker provided by PROB tool [32] (version `1.3.6-final`)—on a standard PC with a quad-core 3.2 GHz CPU and 8 GB memory—to check the validity of the following CTL formula:

$$AG\Big(not\big(\,e(collision) \vee e(run - through) \\ \vee\, e(derailment)\,\big)\Big)$$

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

After summarising our proof method, we report on safety verification results for two case studies: a simple station which we have studied previously, and the complex Langley Station. Though we do not do so here, the production of counter-example traces for a single, unsafe CSP||B model is possible and is discussed in detail in [25].

10.1 Proof method using abstractions

Utilising our three abstraction principles, we apply the following proof method to analyse a scheme plan SP for safety: for all units $u \in Unit$ of a scheme plan SP,

1. we first construct the scheme plan $SP_{\{u\}}$ and encode it as a model CSP || B($SP_{\{u\}}$);
2. we then apply a topological abstraction function *abs* to obtain $abs\big(\text{CSP} \,||\, \text{B}(SP_{\{u\}})\big)$;
3. we then prove that $abs\big(\text{CSP} \,||\, \text{B}(SP_{\{u\}})\big)$ is safe for two trains using the ProB model checker.

In case that the proof in step 3 is successful for all $u \in Unit$, the design SP is guaranteed to itself be safe.

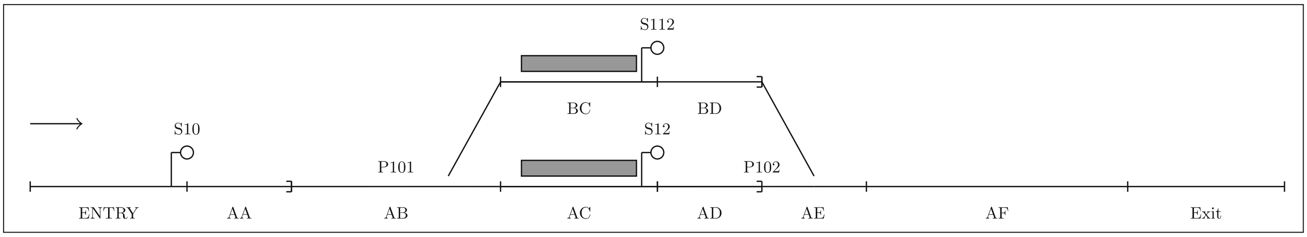
This procedure is sound: by Theorem 4 on topological abstraction we know that CSP || B($SP_{\{u\}}$) is safe for two trains for all $u \in Unit$; by Corollary 1 and Theorem 3 concerning finitisation we have that CSP || B($SP_{\{u\}}$) is safe for any numbers of trains for all $u \in Unit$; and by Corollary 3 on covering we have that CSP || B(SP) is safe for any number of trains. As we argue that our CSP||B modelling is faithful, we conclude that SP is safe.

10.2 Verifying a simple station example

In [24] we studied the simple station case study presented in Fig. 17.

We reconsider this example here to text the effectiveness of our abstraction techniques. However, unlike in [24], here we consider overlaps (i.e. the ability of trains to overrun red lights) which were not permitted in the earlier study due to the assumed use of Automatic Train Protection (ATP).

Overall, an example of this small size and low complexity can be directly verified without applying covering and topological abstraction. The successful verification using finitisation to two trains but without covering and without topological abstraction takes $1m56s$ and produces a state space containing 8394 states and 83279 transitions.



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. 17 Station scheme plan

Fig. 18 Verifying the station example via finitisation and covering

Track	# states	# trans	Track	# states	# trans
AA	73	391	AE	1050	6579
AB	450	2579	AF	1240	7739
AC	448	2499	BC	448	2499
AD	1736	13707	BD	1736	13707

We have also verified the station example using our proof method as outlined in Sect. 10.1. Figure 18 gives an overview of the state space required to verify each sub-scheme plan. The table shows that the number of states and transitions required for each of the 8 sub-plans is much smaller than the number of states and transitions required for the whole scheme plan. The total time to complete the verification of all these sub-plans is $1m11s$, i.e. 39 % faster than verifying the full scheme plan. Furthermore, if we consider the total number of states verified, we can see that in total our new method inspects 7,181 states, which is fewer than the number of states needed for the verification of the full scheme plan.

10.3 Verifying the Langley-based example

Direct verification (with finitisation) of the full scheme plan for Langley is not possible due to the complexity of the scheme plan, which consists of 49 tracks (including 4 entries and 4 exists), 16 points, 12 signals and 16 routes. However, the proof method from Sect. 10.1 enables its successful verification. Figure 19 summarises the number of states and transitions that are to be considered for the verification of each of the 41 sub-scheme plans of the Langley example, though without topological abstraction.

In Fig. 20 we categorise our verification in terms of the numbers of states involved. Over half (51.2 %) of the 41 proofs are trivial and can be conducted within ∼10 s, whilst 15 of the 41 (36.6 %) take around ∼1 min to complete. The remaining five (12.2 %) of the proofs require longer to complete, with sub-plans for *UI* and *UJ* being particularly large and taking up to 2 h each to complete. This is due to *UI* and *UJ* being part of a large number of routes, which give rise to large influence zones.

To consider the effect of topological abstraction, we demonstrate its application to sub-plans of the Langley Station example. Figure 21 gives an illustration of the reduction in terms of sizes of state spaces gained from applying topological abstraction to these sub-scheme plans. Our results shows that a reduction of up to 50 % is possible. In the examples considered, topological abstraction reduces the number of tracks by about the same amount.

Overall, our experiments demonstrate that the proof methodology from Sect. 10.1:

- reduces the verification time significantly for rail networks of small size and low complexity; and
- enables verification for rail networks of large size and high complexity.

11 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 [3,20,33], whilst [28,35] are classical contributions from process algebra and [10] uses techniques from algebraic spec-

Fig. 19 Verification results via finitisation and covering

Track	# states	# trans	Track	# states	# trans
DA	92	339	DRA	92	339
DB	104	385	DRB	104	385
DC	272	2123	DRC	272	2123
DD	330	1883	DRD	2948	30249
DE	328	1819	DRE	2948	30249
DF	1096	8643	DRF	3664	37667
DG	25296	301089	DRG	3824	39287
DH	29162	357085	DRH	6238	64049
UA	122	457	DRI	6770	69449
UB	3329	27479	DRJ	16874	207501
UC	2798	23137	DRK	2696	21917
UD	2638	21837	DRL	2176	17965
UE	2176	17965	DRM	150	565
UF	2176	17965	URA	122	457
UG	1336	10539	URB	1096	8643
UH	440	2479	URC	984	7755
UI	105208	1519773	URD	330	1883
UJ	105208	1519773	URE	92	517
UK	330	1883	URF	3411	27973
UL	99	517	URG	3163	25969
			URH	92	339

State range	# sub-plans	∼verification time
0 – 1500	21	∼ 10 seconds
1500 – 10000	15	∼ 1 minute
10000 – 50000	3	∼ 30 minutes
50000+	2	∼ 2 hours

Fig. 20 Categories of sub-plan verifications with respect to state space sizes

Track	# states	reduction	# trans	reduction
DG	23677	6.4%	288001	4.3%
DRJ	8430	50.0%	103557	50.0%
UI/UJ	96374	8.4%	1394281	8.3%

Fig. 21 Improving verification with topological abstraction

ification. On a lower abstraction layer, [5,7,16,18] verify the safety of interlocking programs with logical approaches.

11.1 Modelling comparison

Our modelling is most related to Winter's approach in CSP [37], Abrial's modelling in Event-B [2] and Haxthausen's modelling using RAISE and the SAL model checker in [9]. Haxthausen also notes the importance of a DSL in [9] and the techniques have been applied to the Stenstrup Station real-world example. The author has successfully developed a toolset supporting the automated, formal modelling and verification of product line of relay interlocking systems based on the use of the SAL model checker. The author shares similar verification goals of verifying the interlocking tables.

In the following we briefly discuss the approaches of Winter and Abrial's 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 [37] 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 decentralised control. Consequently, the model has no interlocking cycle.

Chapter 17 of the book by Abrial [2] 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. [38] 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 [15] that even this concession is too restrictive to be realistic. The novelty of this paper here 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.

11.2 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 [6] and therefore it is important to discuss it in the context of scalability. Ferrari et al. [7] state that model checking large interlocking systems is infeasible with current state-of-the-art model checkers, in particular SPIN and NuSMV. However, Cimatti et al. [5] have demonstrated considerable success using NuSMV on industrial-scale problems. James et al. [16] also 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.

12 Conclusion

In this paper we have discussed an approach to safety verifications within railway interlocking which has been successfully deployed on live problems of substantial complexity proposed for study by our industrial partners. Highlights of our approach are: that it is usable by engineers and not simply a theoretical study; that it makes substantial network analyses tractable through effective abstractions; and that it brings model checking down to the design level—at which engineers work—rather than embedded deep inside a theoretical model, thus allowing for a "push-button" approach to safety verification. Two important lessons learnt through carrying out this work are that language constructs need to be right for the domain in question, and that domain analyses provide the most powerful abstractions. In our strategy for covering, we divide the scheme plan into sub-plans and generate a subplan for each unit. This results in a large number of sub-plans which may significantly overlap with each other. We have not yet considered generating fewer but larger sub-plans, which have a sequence of tracks as a core. This would be an interesting avenue of further work.

There are three additional avenues of further work which come immediately to mind. Firstly, extending the OnTrack tool set [17] to implement the covering technique and the improvements to the other abstraction techniques is obviously desirable to progress towards delivering a truly push-button technology; the approach can then be readily applied to further stations and topologies. Our current work involves implementing the covering technique within the OnTrack tool. Secondly, extending the framework and the abstraction techniques to permit bi-directional travel promises to be straightforward, but remains to be established. Finally, there is a clear desire to adapt the approach to emerging traffic management protocols, specifically ETCS (European Train Control System) [36]. The application of our approach to ETCS Level 2 should be non-problematic, given it continues to incorporate track boundaries, but going beyond Level 2 will require greater effort.

Acknowledgments The authors would like to thank S. Chadwick and D. Taylor from the company Siemens Rail Automation for their support and encouraging feedback. Thanks also to Matthew Trumble for the initial version of the OnTrack tool which will be the basis of future implementation. Thanks also to the reviewers for their detailed feedback.

Appendix A: The CSP||B model

In the following, we give the full CSP||B model of the station example from Fig. 17. It consists of four B machines and one CSP controller.

The Interlocking machine

```
MACHINE Interlocking
SEES
  Context, Topology, ControlTable, ReleaseTable
SETS
  ANSWERS = {yes,no}
VARIABLES
  pos, nextd, signalStatus, normalPoints, reversePoints, movedPoints, currentLocks
INVARIANT
  pos : TRAIN +-> ALLTRACK &
  nextd : ALLTRACK +-> ALLTRACK &
  signalStatus : SIGNAL --> SIGNALSTATUS &
  normalPoints <: POINT &
  reversePoints <: POINT &
  normalPoints /\ reversePoints = {} &
  normalPoints \/ reversePoints = POINT &
  movedPoints <: POINT &
  currentLocks  : ROUTE <-> POINT &
  currentLocks <: lockTable
INITIALISATION
  BEGIN
  pos := {} ||
  signalStatus := SIGNAL * {red} ||
  normalPoints := POINT ||
  reversePoints := {} ||
  movedPoints := {} ||
  currentLocks := {} ||
  nextd := { (t1 |-> t2) | #(c1,c2,c3). ( t1 /= t2 &
         (c1,c2) : direction[{t1}] & (c1,c2) : staticDirection \/ dynamicDirection[POINT*{normal}] &
         (c2,c3) : direction[{t2}] &  (c2,c3) : staticDirection \/ dynamicDirection[POINT*{normal}]  )  }
  END
OPERATIONS
collision =
SELECT #(t1,t2).(t1 : TRAIN & t2 : TRAIN & t1:dom(pos) & t2:dom(pos) & t1 /= t2 &
({pos(t1)} - (EXIT \/ ENTRY)) /\ ({pos(t2)} -  (EXIT \/ ENTRY)) /= {})
THEN skip
END;
derailment =
SELECT ran(pos) /\ homePoint[movedPoints] /= {}
THEN skip
END;
runthrough =
SELECT nullTrack : ran(pos)
THEN skip
END;

bb <-- enter(t,entryPos) =
PRE
  t : TRAIN & entryPos : ENTRY
THEN
  IF (t /: dom(pos) & entryTable(entryPos) /\ ran(pos) = {})
  THEN
    bb := yes ||
    movedPoints := {} ||
    pos(t) := entryPos
  ELSE
    bb := no
  END
END;

exit(t,exitPos) =
PRE t : TRAIN & pos(t) = exitPos & exitPos : EXIT
THEN
  movedPoints := {} ||
  pos := {t} <<| pos
END;
s <-- nextSignal(t) =
PRE t : TRAIN & t : dom(pos) & pos(t) : ran(homeSignal)
THEN
  movedPoints := {} ||
  s := signalStatus(homeSignal~(pos(t)))
END;
```

```
bb <-- request(route) =
PRE route : ROUTE THEN
  LET occTracks,emptyTracks BE occTracks =  ran(pos) & emptyTracks = TRACK - occTracks IN
    /* are the tracks for a route empty */
    IF  ((signalStatus(signal(route)) = red) & (clearTable(route) <: emptyTracks ))
    THEN
      LET unlockedPoints BE unlockedPoints = POINT - ran(currentLocks) IN
        /* all points in right position or unlocked */
        IF ((normalTable[{route}] <: normalPoints \/ unlockedPoints ) &
           (reverseTable[{route}] <: reversePoints \/ unlockedPoints))
        THEN
          LET np, rp BE
            np = (normalPoints \/ normalTable[{route}]) - reverseTable[{route}] &
            rp = (reversePoints \/ reverseTable[{route}]) -  normalTable[{route}]
          IN
            /* move points on the route that need to be moved */
            movedPoints := (normalPoints - np) \/ (reversePoints - rp) ||
            normalPoints := np  ||
            reversePoints := rp ||
            /* for each point p of route, lock p */
            currentLocks := currentLocks\/({route} <| lockTable) ||
            /* set signal of route to greeen */
            signalStatus(signal(route)) := green||
            /* grant the request */
            bb := yes ||
            nextd := {  (t1 |-> t2) | #(c1,c2,c3). ( t1 /= t2 &
                   (c1,c2) : direction[{t1}] & (c1,c2) : staticDirection \/
                        dynamicDirection[np*{normal} \/ rp*{reverse}] &
                   (c2,c3) : direction[{t2}] & (c2,c3) : staticDirection \/
                        dynamicDirection[np*{normal} \/ rp*{reverse}]  )   }
          END /* end let */
        ELSE
          /* refuse request */
          movedPoints := {} ||
          bb:= no
        END /* end if */
      END /* end let */
    ELSE
      /* refuse request */
      movedPoints := {} ||
      bb:= no
    END /* end if */
  END /* let */
END; /* end pre */

newp <-- move(t,currp) =
PRE t : TRAIN & t : dom(pos) & currp = pos(t) THEN
  movedPoints := {} ||
    IF (pos(t) /: dom(nextd)) THEN
      pos(t) := nullTrack ||
      newp := nullTrack
    ELSE
      LET track BE track = nextd(pos(t)) IN
        pos(t) := track ||
        newp := track ||
        IF (pos(t) : ran(homeSignal)) THEN
            signalStatus(homeSignal~(pos(t))) := red
        END ||
        currentLocks := currentLocks - releaseTable[{track}]
      END
    END
END;
```

```
bb <-- release(route) =
PRE route : ROUTE THEN
  movedPoints := {} ||
  LET emptyTracks BE emptyTracks = TRACK - ran(pos) IN
    IF
      /* the signal of the route is green */
      (signalStatus(signal(route)) = green)  &
      /* points locked for the route */
      currentLocks[{route}] = lockTable[{route}] &
      /* no train is in the track preceding the route (i.e. nothing close enough to
        go through the red light ) */
      homeSignal(signal(route)) : emptyTracks
    THEN
      signalStatus(signal(route)) := red ||
      currentLocks := {route} <<| currentLocks ||
      bb := yes
    ELSE
      bb := no
    END
  END /* let */
END
END
```

The Context machine

```
MACHINE Context
SETS
    TRACKSTATUS = {occ,empty};
    ASPECT = {red,green};
    ALLTRACK = {AA, AB, AC, AD, AE, AF, BC, BD, Entry, Exit, nullTrack};
    ALLCONNECTOR = { C0, C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12 };
    SIGNAL = {S10, S12, S112};
    TRAIN = {albert,bertie};
    POINT = {P101,P102};
    POINTPOSITION = {normal,reverse};
    POINTSTATUS = {locked, unlocked};
    ROUTE = {R10A, R10B, R12, R112 }
CONSTANTS
    SIGNALSTATUS, CONNECTOR, TRACK, ENTRY, EXIT
PROPERTIES
    SIGNALSTATUS = ASPECT &
    CONNECTOR = ALLCONNECTOR - {C0} &
    ENTRY = {Entry} &
    EXIT = {Exit} &
    TRACK = ALLTRACK - {nullTrack}
END
```

The Topology machine

```
MACHINE Topology
SEES  Context
CONSTANTS
    signal, homeSignal, homePoint,
    direction, staticDirection, dynamicDirection
PROPERTIES
    signal : ROUTE --> SIGNAL &
    signal = { (R10A |-> S10), (R10B |-> S10),(R12 |-> S12), (R112 |-> S112) } &
    homeSignal : SIGNAL >-> TRACK &
    homeSignal = { S10 |-> Entry, S12 |-> AC, S112 |-> BC } &
    homePoint : POINT --> TRACK &
    homePoint = { (P101 |-> AB), (P102 |-> AE) } &
    direction : TRACK <-> CONNECTOR * CONNECTOR &
    direction = { Entry |-> (C1,C2),
        AA |-> (C2,C3), AB |-> (C3,C4), AB |-> (C3,C10), AC |-> (C4,C5),
        AD |-> (C5,C6),  AE |-> (C6,C7), AE |-> (C12,C7), AF |-> (C7,C8),
        Exit |-> (C8,C9), BC |-> (C10,C11), BD |-> (C11,C12) } &
    staticDirection : CONNECTOR <-> CONNECTOR &
    staticDirection = {(C1,C2),(C2,C3),(C4,C5),(C5,C6),(C7,C8),(C8,C9),(C10,C11),(C11,C12)} &
    dynamicDirection : POINT * POINTPOSITION --> CONNECTOR * CONNECTOR &
    dynamicDirection = { (P101,normal)  |-> (C3,C4),(P101,reverse) |-> (C3,C10),
        (P102,normal)  |-> (C6,C7), (P102,reverse) |-> (C12,C7)} &
    ran(direction) = staticDirection \/ ran(dynamicDirection) &
    staticDirection /\ ran(dynamicDirection) = {}
END
```

The Control Table machine

```
MACHINE ControlTable
SEES Context
CONSTANTS
  entryTable, normalTable, reverseTable, clearTable, lockTable
PROPERTIES
  entryTable: ENTRY --> POW(TRACK) &
  normalTable :  ROUTE <-> POINT &
  reverseTable : ROUTE <-> POINT &
  clearTable :   ROUTE --> POW(TRACK) &
  lockTable :    ROUTE <-> POINT &
  entryTable = { Entry |-> {Entry,AA} } &
  normalTable  = { R10A |-> P101, R12  |-> P102 } &
  reverseTable = { R10B |-> P101, R112 |-> P102 } &
  clearTable = { R10A  |-> {AA,AB,AC,AD}, R10B  |-> {AA,AB,BC,BD},
                 R12   |-> {AD,AE,AF}, R112  |-> {BD,AE,AF} }  &
  lockTable = { R10A |-> P101, R12  |-> P102, R10B |-> P101, R112 |-> P102 } &
  lockTable = normalTable \/ reverseTable
END
```

The Release Table machine

```
MACHINE ReleaseTable
SEES Context
CONSTANTS
    releaseTable
PROPERTIES
    releaseTable :  TRACK <-> (ROUTE*POINT) &
    releaseTable = { AC |-> (R10A,P101),BC |-> (R10B,P101),AF |-> (R12, P102),AF |-> (R112,P102) }
END
```

The CSP controller

```
datatype TRAIN = albert | bertie
datatype SIGNAL = red | green
datatype POS =  AA | AB | AC | AD | AE | AF | BC | BD |
                Entry | Exit | nullTrack
ALLTRACK = POS
TRACK = diff(ALLTRACK,{nullTrack})
ENTRY = {Entry}
EXIT = {Exit}
SIGNALHOMES = {Entry, BC, AC }
datatype ROUTE = R10A | R10B | R12 | R112
datatype ANSWERS = yes | no

channel enter: TRAIN.ENTRY.ANSWERS
channel exit: TRAIN.EXIT
channel nextSignal : TRAIN.SIGNAL
channel move : TRAIN.ALLTRACK.ALLTRACK
channel request : ROUTE.ANSWERS
channel release : ROUTE.ANSWERS
RW_CTRL =
  ([] r : ROUTE @ request!r?ans -> RW_CTRL)
  []
  ([] r : ROUTE @ release!r?ans -> RW_CTRL)
TRAIN_OFF(t) =
  [] entryPos : ENTRY  @
    enter!t!entryPos?ans ->
    (if (ans == yes) then
      TRAIN_CTRL(t,entryPos)
     else
      TRAIN_OFF(t))
TRAIN_CTRL(t,pos) =
(member(pos,EXIT) & exit.t.pos -> STOP)
[]
(not(member(pos,EXIT)) and
 not(member(pos,SIGNALHOMES)) &
   move.t.pos?newp -> TRAIN_CTRL(t,newp)
)
[]
(not(member(pos,EXIT)) and
 member(pos,SIGNALHOMES) &
   nextSignal!t?aspect ->
   (if (aspect==green) then
       move.t.pos?newp -> TRAIN_CTRL(t,newp)
    else
       ((move.t.pos?newp -> STOP) |~| TRAIN_CTRL(t,pos))
   )
)
ALL_TRAINS = ||| t : TRAIN @ TRAIN_OFF(t)
channel collision, runthrough, derailment
ERR = (collision -> ERR) [] (runthrough -> ERR) [] (derailment -> ERR)
CTRL = RW_CTRL ||| ALL_TRAINS ||| ERR
MAIN = CTRL
```

References

1. Abrial, J.-R.: The B-Book: Assigning Programs to Meanings. CUP, Cambridge (1996)
2. Abrial, J.-R.: Modeling in Event-B, chapter 17-Train System. CUP, Cambridge (2010)
3. Antoni, M.: Practical formal validation method for interlocking or automated systems. In: 3rd International Workshop on Dependable Control of Discrete Systems (DCDS), 2011, pp. ix–x (2011)
4. Bjørner, D.: Dynamics of railway nets: on an interface between automatic control and software engineering. Elsevier. In: CTS (2003)
5. Cimatti, A., Corvino, R., Lazzaro, A., Narasamdya, I., Rizzo, T., Roveri, M., Sanseviero, A., Tchaltsev, A.: Formal verification and validation of ERTMS industrial railway train spacing system. In: CAV, volume 7358 of LNCS, pp. 378–393. Springer (2012)
6. Fantechi, A., Gnesi, S.: On the adoption of model checking in safety-related software industry. In: Computer Safety, Reliability, and Security, pp. 383–396 (2011)
7. Ferrari, A., Magnani, G., Grasso, D., Fantechi, A.: Model checking interlocking control tables. FORMS/FORMAT 2010, 107–115 (2011)
8. Fowler, M.: Domain Specific Languages. Addison-Wesley, Reading (2010)
9. Haxthausen, A.E.: Automated generation of safety requirements from railway interlocking tables. In: ISoLA (2), volume 7610 of LNCS, pp. 261–275. Springer (2012)
10. Haxthausen, A.E., Peleska, J.: Formal development and verification of a distributed railway control system. IEEE Trans. Softw. Eng. **26**(8), 687–701 (2000)
11. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Englewood Cliffs (1985)
12. Isobe, Y., Moller, F., Nguyen, H.N., Roggenbach, M.: Safety and line capacity in railways-an approach in Timed CSP. In: IFM, pp. 54–68 (2012)
13. Jacquart, R. (ed.): IFIP 18th World Computer Congress, Topical Sessions, chapter TRain: The Railway Domain-A Grand Challenge. Kluwer, Dordrecht (2004)
14. James, P., Beckmann, A., Roggenbach, M.: Using domain specific languages to support verification in the railway domain. In: Proceedings of HVC'12: Eighth Haifa Verification Conference, LNCS. Springer (to appear)
15. James, P., Moller, F., Nguyen, H.N., Roggenbach, M., Schneider, S., Treharne, H. : On Modelling and Verifying Railway Interlockings: Tracking Train Lengths. Technical Report CS-13-03, University of Surrey, Department of Computing (2013)
16. James, P., Roggenbach, M.: Automatically verifying railway interlockings using SAT-based modelchecking. ECEASST, 35 (2010)
17. James, P., Trumble, M., Treharne, H., Roggenbach, M., Schneider, S.: OnTrack: an open tooling environment for railway verification. In: Proceedings of NFM'13: Fifth NASA Formal Methods Symposium (2013)
18. Kanso, K., Moller, F., Setzer, A.: Automated verification of signalling principles in railway interlockings. Electron. Notes Theor. Comput. Sci. **250**, 19–31 (2009)
19. Leuschel, M., Butler, M.: ProB: an automated analysis toolset for the B method. Int. J. Softw. Tools Technol. Transf. **10**(2), 185–203 (2008)
20. Leuschel, M., Falampin, J., Fritz, F., Plagge, D.: Automated property verification for large scale B models with ProB. Formal Asp. Comput. **23**(6), 683–709 (2011)
21. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv., **37**(4) (2005)
22. Moller, F., Nguyen, H.N., Roggenbach, M.: Covering for CSP. Swansea University,Technical report (2013)
23. Moller, F., Nguyen, H.N., Roggenbach, M., Schneider, S., Treharne, H.: Combining Event-Based and State-Based Modelling for Railway Verification. Technical Report CS-12-02, University of Surrey (2012)
24. Moller, F., Nguyen, H.N., Roggenbach, M., Schneider, S., Treharne, H.: Defining and model checking abstractions of complex railway models using CSP||B. In: Proceedings of HVC'12: Eighth Haifa Verification Conference, p. 16 (2012) (to appear in Springer Lecture Notes in Computer Science)
25. Moller, F., Nguyen, H.N., Roggenbach, M., Schneider, S., Treharne, H.: Railway modelling in CSP||B: The double junction case study. Electron. Commun. EASST, **53**, 15 (2012)
26. Moller, F., Nguyen, H.N., Roggenbach, M., Schneider, S., Treharne, H.: Using ProB and CSP||B for railway modelling. In: Proceedings of IFM'12 and ABZ 2012 Posters and Tool demos session, pp. 31–35 (2012)
27. Morgan. C.C.: Of wp and CSP. In: Beauty is Our Business: A Birthday Salute to Edsger J. Dijkstra, pp. 319–326. Springer (1990)
28. Morley, M.J.: Safety in railway signalling data: a behavioural analysis. In: HOLTPA, pp. 464–474. Springer (1993)
29. National Electronic Sectional Appendix. http://www.networkrail.co.uk/aspx/10563.aspx. Accessed: 01/05/2013
30. Nock, O.-S.: Railway Signalling. IRSE (1980)
31. Office of Rail Regulations. Estimates of station usage 2011/12 report. (2013). http://www.rail-reg.gov.uk/server/show/nav.1529
32. The ProB animator and model checker (ProB 1.3.6-final). (2013). http://www.stups.uni-duesseldorf.de/ProB. Accessed: 01/05/2013
33. Sabatier, D., Burdy, L., Requet, A., Guéry, J.: Formal proofs for the NYCT line 7 (flushing) modernization project. In: ABZ, pp. 369–372 (2012)
34. Schneider, S., Treharne, H.: CSP theorems for communicating B machines. Formal Asp. Comput. **17**(4), 390–422 (2005)
35. Simpson, A., Woodcock, J., Davies, J.: The mechanical verification of solid-state interlocking geographic data. In: Formal Methods Pacific 97. Springer (1997)
36. UIC: The International Union of Railways. ETCS reference documents. (2013). http://www.uic.org. Accessed: 01/05/2013
37. Winter, K.: Model checking railway interlocking systems. Aust. Comput. Sci. Commun. **24**(1) (2002)
38. Winter, K., Robinson, N.: Modelling large railway interlockings and model checking small ones. In: Proceedings of the 26th Australasian Computer Science Conference-Volume 16, pp. 309–316. Australian Computer Society Inc, (2003)