Defining and Model Checking Abstractions of Complex Railway Models using CSP||B

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

[1] Swansea University, Wales, UK
[2] University of Surrey, England, UK

Abstract. The safety analysis of interlocking railway systems involves verifying collision and derailment freeness. In this paper we propose a structured way of refining track plans, in order to expand track segments so that they form collections of track segments. We show how the abstract model can be model checked to ensure the safety properties, which must also hold in the corresponding concrete track plan, so that we will never need to model check the concrete track plan directly. We also identify the minimal number of trains that needs to be considered as part of the model checking, and we demonstrate the practicality of the approach on various scenarios.

1 Introduction

Formal verification of railway control software has been identified as one of the "Grand Challenges" of Computer Science [9]. As is typical with Formal Methods, this challenge comes in two parts: the first addresses the question of whether the mathematical models considered are legitimate representations of the physical systems of concern. The modelling of the systems, as well as of proof obligations, needs to be *faithful*. The second part is the question of how to utilize available technologies, for example model checking or theorem proving. Whichever verification process is adopted, it needs to be both *effective* and *efficient*.

In [11, 10] we propose a new modelling approach for railway interlockings. We use CSP||B [13], which combines event-based with state-based modelling. This reflects the double nature of railway systems, which involves events such as train movements and, in the interlocking, state based reasoning. In this sense, CSP||B offers the means for the natural modelling approach we strive for: the formal models are close to the domain models. To the domain expert, this provides traceability and ease of understanding. This addresses the first of the above stated challenges: *faithful* modelling.

In this paper, we address the question of how to *effectively* and *efficiently* verify various safety properties within our CSP||B models. To this end we develop a set of abstraction techniques for railway verification that allow the transformation of complex CSP||B models into less involved ones, prove that they are correct, and demonstrate that they allow one to verify a variety of railway systems via model checking. The first set of abstractions reduces the number of trains that need to be considered in order to prove safety for an unbounded number of trains. Their correctness proof involves slicing of event traces. The second set of abstractions simplifies the underlying track topology. Here, the correctness proof utilizes event abstraction specific to our application domain similar to the ones suggested by Winter in [15].

Outline We first introduce our modelling language CSP||B. In Section 3 we summarise our generic railway modelling approach using CSP||B, as described in [11, 10]. In Section 4, we present our first set of abstraction techniques based on event traces. Then in Section 5 we present our data abstraction techniques. The application of the abstraction results is presented via a set of example scenarios in Section 6. In Section 7 we put our contribution in the context of related approaches.

2 Background to CSP||B

The CSP||B approach allows us to specify communicating systems using a combination of the B-Method [4] and the process algebra CSP (Communicating Sequential Processes) [7]. The overall specification of a combined communicating system comprises two separate specifications: one given by a number of CSP process descriptions and the other by a collection of B machines. Our aim when using B and CSP is to factor out as much of the "data-rich" aspects of a system as possible into B machines. The B machines in our CSP||B approach are classical B machines, which are components containing state and operations on that state. The CSP||B theory [13] 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 [12]. Therefore, our notion of consistency is that a combined communicating system $Ps \parallel Ms$ is *divergence-free* and also *deadlock-free*.

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. 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:

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

The process $e?x!y \rightarrow P(x)$ defines a channel communication where x represents all data variables on a channel, and y represents values being passed along a channel. Channel e is referred to as a *machine channel* as there is a corresponding operation in the controlled B machine with the signature $x \longleftarrow e(y)$. Therefore the input of the operation y corresponds to the output from the CSP, and the output x of the operation to the CSP input. Here we have simplified the communication to have one output and one input but in general there can be any number of inputs and outputs. The other CSP operators have the usual CSP semantics.

For reasoning of CSP||B models we require the following notation.

- Since a B machine is interpretable as a CSP process, the various CSP refinements also apply to CSP||B. In this paper we focus on trace refinement where $P \sqsubseteq_T Q$ if $traces(Q) \subseteq traces(P)$. This refinement preserves safety properties, such as collision freedom or derailment freedom as we shall discuss in Section 3.
- Furthermore, we apply CSP renaming $f(P)$ and CSP hiding $P \setminus A$ to CSP processes, B machines and to CSP||B models, which all semantically represent sets of traces. Given a set of traces T, $f(T)$ represents the set of all traces $tr \in T$ where the events are replaced point-wise by the function f; $T \setminus A$ to represent the set of all traces $tr \in T$ where the events from the set A are removed from tr.
- A system run σ (of a CSP||B model) of length $n \geq 0$ is a finite sequence

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

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

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

3 Modelling and safety verification of railway systems using CSP||B

Together with railway engineers we developed a common view on the information flow in railways. In physical terms a railway consists of, at least, four different components. These components are shown in Figure 1. The *Controller* selects and releases routes for trains. The *Interlocking* serves as a safety mechanism with regards to the Controller and, in addition, controls and monitors the Track equipment. The *Track equipment* consists of elements such as signals, points, and track circuits (logical names for tracks and points from the track plan as discussed above; in the railway domain, tracks and track circuits are often confused): 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 make the assumption that track equipment reacts instantly and is free of defects. The information flow shown in Figure 1 is as follows: the controller sends a request message to the interlocking to which the interlocking responds; the interlocking sends signalling information to the trains; and the trains inform the interlocking about their movements. The interlocking serves as the system's clock: messages can be exchanged once per cycle.

In this paper, we study various track plans, one of which is a station illustrated in Figure 2(b). It depicts the *scheme plan* for the station, which comprises a track plan, a control table, and release tables. (We will discuss Figure 2(a) in Section 6).

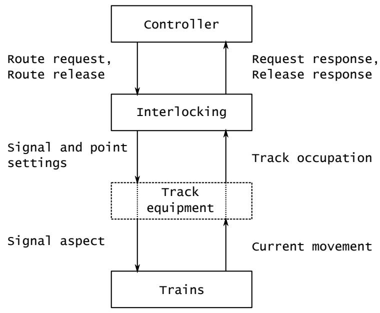


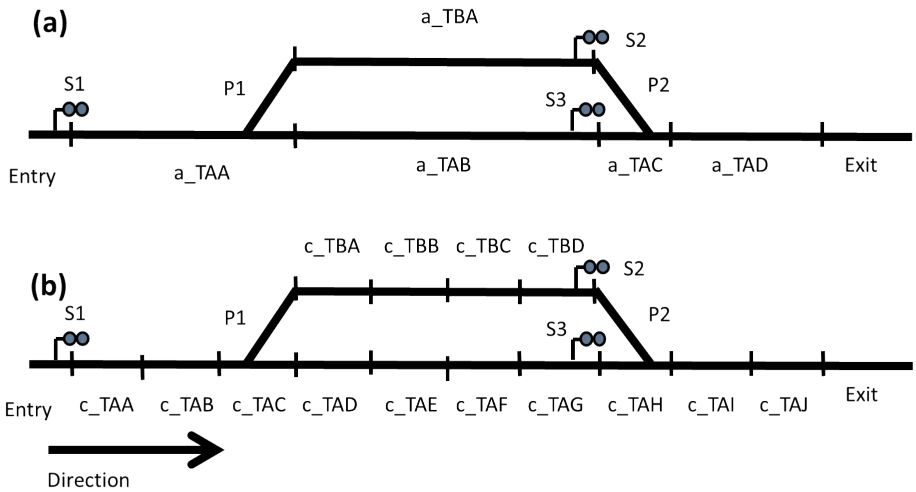
Fig. 1. Information flow.

The *track plan* provides the topological information of the station which consists of 16 tracks (e.g., the track c_TAA), three signals (e.g., $S1$), and two points (e.g., P1). Note that the tracks include entry and exit tracks on which trains can "appear" and "disappear". These two kinds of tracks are specially treated during verification.

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 signal, there is one row describing the condition under which the signal can show proceed. There are two rows for signal $S1$: one for the main line (Route A1) and one for the side line (Route B1). A route comprises tracks and points between two signals. For example, signal S1 for the main line can only show proceed when point P1 is in normal (straight) position and tracks c_TAA, c_TAB, c_TAC, c_TAD, c_TAE, c_TAF, c_TAG are all clear. Here we assume that trains are equipped with an Automatic Train Protection system which prevents trains from moving over a red light and therefore, overlaps are not needed, e.g., the overlap for Route A1 would be c_TAH. For further discussion on this see [8].

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 A1 obtains a lock on point P1, and sets it to normal. The lock is released after the train has passed the point. *Release tables* store the relevant track.

In this setting, we consider two safety properties: *collision-freedom* excludes two trains occupying the same track; and *no-derailment* says that whenever a train enters a point, the point is set to cater for this; e.g., when a train travels from track c_TAG to track c_TAH, point P2 is set so that it connects c_TAG and c_TAH (and not c_TBD and c_TAH). The correct design for the con-



Control table

Route	Normal	Reverse	Clear
A1	P1		c_TAA, c_TAB, c_TAC, c_TAD, c_TAE, c_TAF, c_TAG
B1		P1	c_TAA, c_TAB, c_TAC, c_TBA, c_TBB, c_TBC, c_TBD
A2	P2		c_TAH, c_TAI, c_TAJ
B2		P2	c_TAH, c_TAI, c_TAJ

Release tables

P1	Occupied
A1	c_TAD
B1	c_TBA

P2	Occupied
A2	c_TAI
B2	c_TAI

Fig. 2. One Station - Abstract (a) and Concrete (b) Track Plan (Scenario 2 from Fig. 4)

trol table and release tables is safety-critical: mistakes can lead to collision or derailment.

3.1 CSP||B modelling of railways

In previous work [10, 11] we have demonstrated that CSP||B caters for railways. It is possible to read the actual models together with railway engineers in order to validate them. This review demonstrates that the models can be clearly understood by railway engineers. Here, we refrain from elaborating on the modelling approach and refer the interested reader to [11] for the details. However, the concepts from the models central for verification (in Section 4 and Section 5), namely static and dynamic state representation and also train movements, are discussed below.

The static state information of a CSP||B model is defined in context machines, i.e., machines that contain set and function definitions. For example, the names of all the track circuits is defined in a set called *ALLTRACKS*. The topology of the track plan is captured using a collection of relations that capture how the elements of the track plan are related. For example, $next : TRACK \leftrightarrow TRACK$ is a relation between tracks and possible successor tracks. Therefore, (c_TAC, a_TAD) and (c_TAC, c_TBA) are elements of the *next* relation within the one-station example in Figure 2.

The *Interlocking* machine models the dynamics of the system. Its state evolves over time. It consists of the following variables: *pos* representing the position of all trains, *nextd* representing the current position of all points (and thus the dynamic relation between tracks and their successors), *signalStatus* representing the aspect of each signal, *normalPoints* representing the points which are in normal position, *reversePoints* representing the points which are in reverse position, and *currentLocks* representing the current semaphores on points.

In the CSP||B models, a train a can perform one of the following events: $move.a.currp.newp$ represents a moving from track *currp* to track *newp*, $nextSignal.a.aspect$ represents a seeing the particular *aspect* (red or green) at the next signal, $enter.a.p$ represents placing a on an entry track p, and $exit.a.p$ represents a leaving the system. Trains that have left the system can be placed again on an entry track; we call this behaviour recurring trains. Note that in the situation where *currp* and *newp* are separated by a signal the event $move.a.currp.newp$ is possible only if this signal shows green.

4 Identifying appropriate trace abstractions in order to constrain the model checking

The state space of our railway models grew fast with the number of trains involved:

number of trains	1	2	3	4	5	6
number of states	31	161	813	3859	17315	74205

The above table demonstrates this dependency for the linear track plan shown in the middle of in Figure 3(b) which has two routes, where each route consists of two tracks.

In this section we provide two methods of how to reduce the number of trains when proving collision freedom and derailment freedom.

4.1 Minimum number of trains for verifying collision

The following theorem turns the question if a railway scheme plan is collision free into a finite state problem by reducing the – in principle – unbounded number of trains to be considered into a finite number:

Theorem 1. *Let S be a railway scheme plan with r routes. S is collision free iff all systems runs with $r + 1$ recurring trains are collision free.*

Proof. We prove the "if" direction only, as the other direction trivially holds.

We first note that if there are two trains on a route then a collision can occur (as these two trains are not separated by a signal). Therefore, as long as there is no collision there will be at most r trains on S. Assuming we have $r+1$ trains there will always be one train available to move onto an entry track. Thus, $r+1$ recurring trains are sufficient. □

4.2 Minimum number of trains for verifying derailment

Regarding derailment, we obtain an even stronger result. The reduction argument, however, holds only for "reasonable" scheme-plans where the various tables are free of trivial mistakes with respect to the railway topology. Concretely, we say that 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 this route (e.g., in Figure 2, there is the lock c_TAD on P1 for route A1; c_TAD appears in the clear column of the control table for the route A1).
2. **Clear-Table condition.** The clear table of a route contains at least the tracks of this route (e.g., in Figure 2 route A1 topologically goes from signal S1 to signal S3 and all tracks from c_TAA to c_TAG are in the clear column of the control table for the route A1).
3. **Normal/Reverse-Table condition.** The normal table or the reverse table of a route contain at least the points on this route (e.g., in Figure 2 route A1 topologically goes from signal S1 to signal S3, it includes the only point P1, and P1 is in the normal column of the control table for the route A1).
4. **Route condition.** Topologically different routes are distinguishable by point positions in the control table (e.g., in Figure 2 route A1 and route B1 are topologically different, point P1 is in the normal column of the control table for route A1, point P1 is in the reverse column of the control table for route B1).
5. **Lock-Table condition.** Routes with different lock tables are distinguishable by point positions in the control table (e.g., in Figure 2 route A1 and route B1 have different lock table entries, namely, c_TAD and c_TBA respectively, in the control table the position of P1 distinguishes them as seen above).

The scheme plan of Figure 2 is well-formed.

Note that there is exactly one condition per table (release table, clear table, normal/reverse table, lock table) plus one condition which links routes as defined topologically with the route definition in the tables. All five conditions are static and can easily be decided for a given scheme-plan. It is worthwhile to point out that well-formedness does not imply the property "no-derailment":

Observation 1 *There exist well-formed scheme-plans with derailment.*

For example, altering the scheme plan of Figure 2 by exchanging the position of point P2 for route A2 and route B2 leads to derailment as explained in Section 3. This exchange, however, preserves well-formedness.

Our current modelling refrains from explicitly representing the routes which are set. Therefore, we need to establish the following theorem:

Theorem 2. *For all system runs of a well-formed scheme-plans holds: When a signal s shows green, then there exists a route r with $signal(r) = s$ which is set.*

We now define a reduction method that will allow us to establish Theorem 3 below. The starting point is a system run σ which involves $k \geq 1$ trains. Let $Trains = \{a_1, \ldots, a_k\}$ and an additional train b, where b does not derail in σ. From this run σ we construct a new system run σ' which

- does not speak about b, which,
- however, preserves the movement of all trains $a \in Trains$.

The events in σ concerning b are:

$$\begin{aligned} E(b) := \{e \in s \ |\, & e = move.b.currp.newp \\ & e = nextSignal.b.aspect \\ & e = enter.b.p \\ & e = exit.b.p\} \end{aligned}$$

In a system run σ, the train b can influence the trains $a \in Trains$ in two ways: (i) b might prevent a train $a \in Trains$ from moving (because a signal in front of a shows red because b uses a resource); (ii) b might allow a train in $Trains$ to move (a move from b releases a lock, so that the signal in front of a can change to green). When "taking away" b from σ our only concern is (ii): we wish to preserve moves. This insight leads to the definition of the following replacement function $replace_b$ concerning events (where ϵ stands for the empty work, i.e., for deletion of the event):

1. $replace_b(e) = e$ if $e \notin E(b)$
2. $replace_b(move.b.currp.newp) = release.r.bb$ if there exists a signal s with $currp = homeSignal(s)$. As `move` is only enabled if signal s shows green, Theorem 2 guarantees that there exists a route r which is set. Well-formedness of the scheme-plan guarantees uniqueness.
3. $replace_b(e) = \epsilon$ if e is any of $move.b.currp.newp$, where $currp \neq homeSignal(s)$ for any signal s, or $nextSignal.b.aspect$, or $enter.b.p$, or $exit.b.p$.

$replace_b$ keeps all events not related to b (1.), releases all locks related to b at the earliest possible opportunity (2.), and deletes all other events related to b.

We want to show that, given a system run $\sigma = \langle s_0, e_0, s_1, e_1, \ldots, e_{n-1}, s_n \rangle$, there exists a system run σ' such that

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

To this end, we need to relate the states in σ and σ':

In case of "deletion" there is no state change in σ':

$$\begin{array}{ccc} S & \mathit{move.b.currp.newp} & S' \\ \geq_b & & \geq_b \\ T & \epsilon & T \end{array}$$

In case of "replacement", states can change in σ and σ',e.g.,

$$\begin{array}{ccc} S & \mathit{move.b.currp.newp} & S' \\ \geq_b & & \geq_b \\ T & \mathit{release.R.bb} & T' \end{array}$$

If S relates to T, all trains $a \in \mathit{Trains}$ shall in T be in the same position as in S, where T naturally, does not speak about the train b. Furthermore, the state T shall offer trains $a \in \mathit{Trains}$ the same possibilities to move as they would have in S. To capture these ideas formally, we define that $S \geq_b T$ if

1. Compared to S, T just deletes the information regarding b.
   $T(\mathit{pos}) = S(\mathit{pos}) \setminus \{b \mapsto \mathit{track} \mid \mathit{track} \in \mathit{TRACK}\}$
2. Track equipment is in the same state.

$$\begin{array}{ll} S(\mathit{nextd}) = T(\mathit{nextd}) & S(\mathit{signalStatus}) = T(\mathit{signalStatus}) \\ S(\mathit{normalPoints}) = T(\mathit{normalPoints}) & S(\mathit{reversePoints}) = T(\mathit{reversePoints}) \end{array}$$

3. A route r causes locks in T only if it does so in S:
   $S(\mathit{currentLocks}[\{r\}]) = \emptyset \Rightarrow T(\mathit{currentLocks}[\{r\}]) = \emptyset$ and
   $T(\mathit{currentLocks}[\{r\}]) \neq \emptyset \Rightarrow S(\mathit{currentLocks}[\{r\}]) = T(\mathit{currentLocks}[\{r\}])$

Using this relation $\geq_b$, we can establish the following theorem:

Theorem 3. *For any collision free system run on a well-formed scheme plan involving $k \geq 1$ trains $\mathit{Trains} = \{a_1, \ldots, a_k\}$ and a train b which does not derail in this run, there exists a system run involving only the trains $\{a_1, \ldots, a_k\}$ with identical movements.*

Proof. (Sketch) By induction on the length of a system run σ. The base case is given by $S_0 \geq_b S_0$ where S_0 is the initial state which has no trains in and locks of points. In the induction step we show: (i) if an event e is enabled in S then $\mathit{replace}_b(e)$ is enabled in the corresponding state; (ii) $\geq_b$ is preserved under the execution under an event e and its corresponding event $\mathit{replace}_b(e)$. Both arguments rely on the fact that σ is a system run, i.e., is a control flow allowed by the CSP processes. □

The condition "collision free" on the system run σ is required, as we "simulate" the movement of the train b by a route release request. Routes can only be released if there is no train on the track t directly in front of the corresponding signal. In the corresponding run σ', b will not be on track t, as b has been removed. There might, however, be another train a. We exclude this by the condition "collision freedom": if there was a train a on the same track t as train b, there would be a collision in σ.

Corollary 1. *For collision free and well-formed scheme plans holds: if they are derailment free for one train, then they are derailment free for any number of trains.*

5 Towards a data abstraction framework of CSP||B railway models

In this section we demonstrate a refinement between two track plans. We want a structured way of doing this so that if collision-freeness and derailment-freeness is shown using the techniques from Section 4 on an abstract track plan then it is preserved automatically for a concrete track plan. This will allow model-checking on the abstract version, which will be more efficient, and allow the results to carry over to the concrete version.

Let us consider two examples to aid our understanding. To appreciate the impact on model checking, Figure 3 shows one abstract track plan (a) that corresponds to two different concrete track plans, one with two tracks per abstract route (b) and one with four tracks per abstract route (c). As in Section 4, this state space also grows fast, however, here in the number of tracks per route (illustrated using 3 trains):

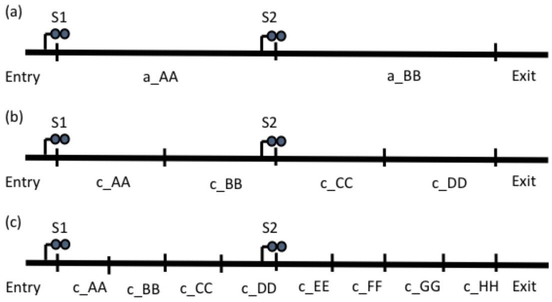


Fig. 3. Linear Scenario Abstraction

number of tracks per abstract track	2	4	8	16
number of states	806	1472	3483	9615

To further visualise the abstraction of track circuits, Figure 2(b) shows an illustration of a concrete track plan to which Figure 2(a) shows an abstract track plan. In the following, we discuss how to formalise such an abstraction and suitable conditions.

As discussed in Section 3.1, a track plan is essentially given by the set $ALLTRACK$ of its track circuits and a relation $next$ between them. We use the prefix $a\_$ for abstract, and $c\_$ for concrete when considering two track plans and the relationship between them. Thus, $a\_ALLTRACK$ is the abstract set of track circuits (of tracks and points). Similarly, $c\_ALLTRACK$ is the concrete set of track circuits. We assume that these are disjoint, apart from the special element $nullTrack$. The relations a_next and c_next define how track circuits are connected. Each concrete track circuit is associated with one abstract track circuit, defined by the following total surjective function:

$$abs : c\_ALLTRACK \twoheadrightarrow a\_ALLTRACK$$

such that $abs(nullTrack) = nullTrack$.

The definition of *abs* for the one-station example is as follows in terms of relational image:

$$\begin{aligned} abs[\{c\_TAA, c\_TAB, c\_TAC\}] &= \{a\_TAA\} \\ abs[\{c\_TBA, c\_TBB, c\_TBC, c\_TBD\}] &= \{a\_TBA\} \\ abs[\{c\_TAD, c\_TAE, c\_TAF, c\_TAG\}] &= \{a\_TAB\} \\ abs[\{c\_TAH\}] &= \{a\_TAC\} \\ abs[\{c\_TAI, c\_TAJ\}] &= \{a\_TAD\} \end{aligned}$$

There are a number of necessary conditions on the abstraction function *abs*. These include prominently:

- Points are preserved under abstraction, i.e., a track circuit belonging to a point in the concrete topology is mapped to a point in the abstract topology.
- Routes are preserved under abstraction, e.g., $abs[\{c\_TAD, c\_TAE, c\_TAF, c\_TAG, c\_TAH\}]$ cannot be $\{a\_TBA\}$ since the set of concrete track circuits is not within one route.
- Any concrete *c_next* pair of track circuits should either both be related to the same abstract track circuit, or should reflect the relation between and abstract *a_next* pair, i.e.,

$$\begin{aligned} &\forall\, c\_t1, c\_t2 \bullet (c\_t1 \mapsto c\_t2) \in c\_next) \Rightarrow \\ &\quad abs(c\_t1) = abs(c\_t2) \lor (abs(c\_t1) \mapsto abs(c\_t2)) \in a\_next \end{aligned}$$

  For example, a move within the same abstract track circuit is given by $(c\_TAB \mapsto c\_TAC) \in c\_next \Rightarrow abs(c\_TAB) = abs(c\_TAC)$.

Beside the *abs* function, there are further functions needed in order to describe the full abstraction between track plans. They allow to formulate further conditions upon the relations defined in a track plan also on the tables, e.g.,

$$a\_clearTable \mathbin{⨾} abs^{-1} = c\_clearTable$$

Our modelling approach works generically for all scheme plans. Thus, given a concrete and an abstract one, we have two formal models to compare. This comparison is performed using B refinement and CSP trace refinement. In the following, we focus on the B refinement.

We establish the refinement relationship between the *Interlocking* B machines by relating states with a linking invariant. To this end, we prove that each operation preserves the linking invariant. The linking invariant consists of three parts: the relationship between the positions of the trains $a\_pos = c\_pos \mathbin{⨾} abs$, the relationship between the current positions of the points (which follows directly due to the static relationships), and the relationship between the track equipment which remains unchanged.

We illustrate the proof by comparing abstract and concrete versions of the *move* operation. For example, the concrete $move.c\_TAC.c\_TAD$ corresponds to the abstract $move.a.a\_TAA.a\_TAB$; here, both have an effect on the B state. In contrast to this, the concrete $move.a.c\_TAB.c\_TAC$ corresponds to the abstract $move.a.a\_TAA.a\_TAA$; the latter has no effect on the B state. Therefore, we can consider the abstract event $move.a.a\_TAA.a\_TAA$ as the B operation $skip$. In a B refinement, a new concrete event can refine $skip$. This can be expressed in the following two lemmas:

Lemma 1 (Renamed move). *If* $(abs(c\_t1) \mapsto abs(c\_t2)) \in a\_next$ *then*

$$abs(c\_t1), abs(c\_t2) \longleftarrow a\_move(t) \sqsubseteq c\_t1, c\_t2 \longleftarrow c\_move(t)$$

Lemma 2 (New move). *If* $abs(c\_t1) = abs(c\_t2)$ *then*

$$c\_t1, c\_t2 \longleftarrow skip(t) \sqsubseteq c\_t1, c\_t2 \longleftarrow c\_move(t)$$

As a consequence of the above lemmas (and similar lemmas for all other operations) the relationship between the abstract machine M_A and the concrete one M_C is given by $M_A \sqsubseteq_T f(M_C \setminus N)$, where f and N are defined by:

$$f(move.a.currp.newp) = move.a.abs(currp).abs(newp)$$

$$N = \{move.a.currp.newp \mid abs(currp) = abs(newp)\}$$

for all trains a in the abstract and the concrete model.

Hence we can now consider the combination of the B machines M_A and CSP processes P_A to obtain:

Theorem 4. *Let abs be an abstraction function from a concrete topology to an abstract topology. Let $P_A \parallel M_A$ be the CSP||B model wrt the abstract topology, let $P_C \parallel M_C$ be the CSP||B model wrt the concrete topology, such that both models are defined over the same set of trains. Let*

1. $M_A \sqsubseteq_T f(M_C \setminus N)$ *and*
2. $P_A \sqsubseteq_T f(P_C \setminus N)$.

Then collision (derailment) freedom of $P_A \parallel M_A$ implies collision (derailment) freedom of $P_C \parallel M_C$.

Proof. We compute:

$$\begin{aligned} P_A \parallel M_A &\sqsubseteq_T f(P_C \setminus N) \parallel f(M_C \setminus N) \quad \text{(by conditions 1 and 2)} \\ &\sqsubseteq_T f(P_C \setminus N \parallel M_C \setminus N) \quad \text{(by distributivity of renaming)} \\ &\sqsubseteq_T f((P_C \parallel M_C) \setminus N) \quad \text{(by distributivity of hiding)} \end{aligned}$$

With regards to collision freedom, we obtain:

$$\begin{aligned} P_A \parallel M_A \text{ is collision free} &\Rightarrow f((P_C \parallel M_C) \setminus N) \text{ is collision free} \\ &\qquad \text{(by trace refinement)} \\ &\Leftrightarrow P_C \parallel M_C \setminus N \text{is collision free} \\ &\qquad \text{(as } f(collision) = collision) \\ &\Leftrightarrow P_C \parallel M_C \text{ is collision free} \quad \text{(as } collision \notin N) \end{aligned}$$

Similarly for derailment freedom. □

Note that Theorem 4 decomposes the proof obligation into a B proof and a CSP proof respectively. In order to establish condition 1, we sketched above a general construction based upon techniques related to B refinement. Condition 2 can be verified using the model checker FDR on CSP processes only.

6 Example scenarios of CSP||B railway models

In order to demonstrate the effectiveness of our techniques outlined in Section 4 and Section 5 we conducted experiments on four scenarios. The experiments were carried out using ProB 1.3.5 beta 15 [3] to verify the collision and derailment freedom of the abstract and concrete track plans using CTL model checking over the CSP||B models. If the verification is successful then we conclude that the model is right and has the right properties. The CSP||B models were also required to be divergence- and deadlock-free. Figure 4 summarises that all our scenarios are collision- and derailment-free.

In scenarios 1, 2 and 3 we have performed experiments based on the number of trains required to verify derailment freedom and the number of trains required to verify collision freedom being one more than the number of routes. To give an indication of the size of the models: scenario 1(a) has 6 tracks, 0 points, 2 signals and 2 routes; scenario 2 has 16 tracks, 2 points, 3 signals and 4 routes; scenario 3 has 15 tracks, 1 point, 5 signals and 6 routes, and finally scenario 4 has 22 tracks, 2 points, 9 signals and 10 routes. Notice that there is a significant reduction in the number of states being explored in all the abstract scenarios, the concrete track plan of scenario 3(b) failed to complete within a reasonable time. Note that in the case of the double junction we only performed the verification using 2 trains. Our conjecture is that verifying collision-freedom is enough with two trains and that the worst case scenario of verifying collision freedom using 11 trains is not required. The double junction scenario is one which we have referred to in our previous work [11], it provides an interesting example of abstraction since the abstraction surrounding one of the points is a biased one, i.e., the normal position of one of the points remains unchanged in the abstraction, whereas the reverse position of the point is an abstraction of its track circuit and another track circuit.

7 Related work

Several industrial studies have been done on using model checking to verify railway applications, e.g., for example SNCF [2], and it is clear that their formal is industrially important. A comparison of the use of different model checkers in the analysis of control tables has been conducted by Ferrari *et al.* [5]. Winter in a recent paper [14] considers different optimising strategies for model checking using NuSMV and demonstrates the efficiency of their approach on very large models. These analyses are at a lower level of abstraction than our models as the models are defined in terms of boolean equations and do not focus on providing behavioural models.

Scenario	Model Description	Number of Trains	Abstract States Checked	Concrete States Checked
1(a)(i)	Linear with 2 concrete tracks per abstract track	1	27	31
1(a)(ii)	Linear with 2 concrete tracks per abstract track	3	596	806
1(b)(i)	Linear with 4 concrete tracks per abstract track	1	27	39
1(b)(ii)	Linear with 4 concrete tracks per abstract track	3	596	1472
2(a)	Station	1	70	203
2(b)	Station	5	151508	968700
3(a)	Single Junction	1	600	756
3(b)	Single Junction	7	326405	Not completed
4	Double Junction	2	173846	379404

Fig. 4. Variations of Four Example Scenarios checked

Others have applied theorem proving in the verification of railway interlocking systems, for example, the Advance FP7 project [1] is developing Event-B models of such systems and verifying comparable safety properties. Indeed it would be interesting for us to investigate further the relationship between the combination of generic proofs and model checking. In this paper, we have demonstrated that the data abstraction on the B part of the CSP||B models is generic but more work will be needed on this when we enrich the models to contain trains which extend over more than one track circuit and can move in more than one direction.

The research most closely related to ours is Winter [15]. The way in which the ASM models are defined closely resonate with ours since they have the same concept of routes, which contain tracks and points, between two signals, and contain a static and a behavioural definition. Their models are more advanced than ours since we currently restrict ourselves to have signals in one direction and we do not include shunting. The simplifications to the Winter models includes combining multiple track circuits into one provided they are always grouped together in the control table; this again resonates with the data abstraction we defined in Section 5, but we formalise the abstraction more explicitly.

8 Conclusion

We have successfully complemented our faithful modelling approach of railway interlockings as presented in [11, 10] by defining abstraction techniques that yield effective and efficient verification process based on model checking. We illustrated this process in terms of various scenarios. The correctness arguments in Sections 4 provides a new proof technique for event- and state-based reasoning. Section 5 demonstrates an interesting data abstraction using decomposition.

Heitmeyer in [6] discusses the importance of complete abstractions. Our abstractions are sound. It is future work to investigate if completeness can be established. In Section 6 we identified that the reduction of Theorem 1 is not sufficient for complex scheme plans. Here we hope to prove our conjecture that two trains are sufficient to verify collision freedom. Our current models lack certain details as discussed in Section 7. Adding these features will allow us to study more fine grained data abstractions. Following recent discussions with Winter, we also agree that another obvious optimisation to consider is the decomposition of track schemes.

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

References

1. Advance FP7 project. `http://www.advance-ict.eu/`. Accessed: 23/07/2012.
2. Practical formal validation method for interlocking or automated systems. `http://www.dcds11.uni-saarland.de/plenaries/practical-formal-validation-method-for-interlocking-or-automated-systems.htmll`. Accessed: 23/07/2012.
3. Prob 1.3.5 beta15. `http://www.stups.uni-duesseldorf.de/ProB`. Accessed: 23/07/2012.
4. J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. CUP, 1996.
5. A. Ferrari, G. Magnani, D. Grasso, and A. Fantechi. Model checking interlocking control tables. In *FORMS/FORMAT*, pages 107–115, 2010.
6. C. L. Heitmeyer, J. Kirby, B. G. Labaw, M. Archer, and R. Bharadwaj. Using abstraction and model checking to detect safety violations in requirements specifications. *IEEE Trans. Software Eng.*, 24(11):927–948, 1998.
7. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
8. Y. Isobe, F. Moller, H. N. Nguyen, and M. Roggenbach. Safety and line capacity in railways - an approach in timed csp. In *IFM*, pages 54–68, 2012.
9. R. Jacquart, editor. *IFIP 18th World Computer Congress, Topical Sessions*, chapter TRain: The Railway Domain - A Grand Challenge. Kluwer, 2004.
10. F. Moller, H. Nguyen, M. Roggenbach, S. Schneider, and H. Treharne. Combining event-based and state-based modelling for railway verification. Technical Report CS-12-02, University of Surrey, 2012.
11. F. Moller, H. Nguyen, M. Roggenbach, S. Schneider, and H. Treharne. Railway modelling in CSP||B: the double junction case study. In *Proceedings of AVOCS*, 2012 (to appear).
12. C. C. Morgan. Of wp and CSP. In W. Feijen, A. J. M. van Gesteren, D. Gries, and J. Misra, editors, *Beauty is our Business: a birthday salute to Edsger J. Dijkstra*. Springer-Verlag, 1990.
13. S. Schneider and H. Treharne. CSP theorems for communicating B machines. *Formal Asp. Comput.*, 17(4):390–422, 2005.
14. K. Winter. Optimising ordering strategies for symbolic model checking of railway interlockings. In *Proceedings of ISOLA*, 2012 (to appear).
15. K. Winter and N. J. Robinson. Modelling large railway interlockings and model checking small ones. In *ACSC*, pages 309–316, 2003.