Science of Computer Programming 96 (2014) 315–336

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

# On modelling and verifying railway interlockings: Tracking train lengths

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

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

ARTICLE INFO

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

*Keywords:*
Railway verification
CSP‖B
Modelling and analysis

ABSTRACT

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

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

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



## 1. Introduction

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

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

* Corresponding author.

http://dx.doi.org/10.1016/j.scico.2014.04.005
0167-6423/

We base our approach on CSP‖B [6], which combines event-based with state-based modelling. This reflects the double nature of railway systems, which involves events such as train movements and – in the interlocking – state based reasoning. In this sense, CSP‖B offers the means for the natural modelling approach we strive for. The formal models are by design close to the domain models. To the domain expert, this provides traceability and ease of understanding. This addresses the first of the above stated challenges: *faithful* modelling. The validity of this claim was demonstrated in particular in [2] where a non-trivial case study – a complex double junction – was provided which was understandable and usable by our industrial partners.

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

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

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

## 2. Background to CSP‖B

The CSP‖B approach allows us to specify communicating systems using a combination of the B Method [8] and the process algebra CSP (Communicating Sequential Processes) [9]. The overall specification of a combined communicating system comprises two separate specifications: one given by a number of CSP process descriptions and the other by a collection of B machines. Our aim when using B and CSP is to factor out as much of the "data-rich" aspects of a system as possible into B machines. The B machines in our CSP‖B approach are classical B machines, which are components containing state and operations on that state. The CSP‖B theory [6] allows us to combine a number of CSP processes *Ps* in parallel with machines *Ms* to produce *Ps* ‖ *Ms* which is the parallel combination of all the controllers and all the underlying machines. Such a parallel composition is meaningful because a B machine is itself interpretable as a CSP process whose event-traces are the possible execution sequences of its operations. The invoking of an operation of a B machine outside its precondition within such a trace is defined as divergence [10]. Therefore, our notion of consistency is that a combined communicating system *Ps* ‖ *Ms* is *divergence-free* and also *deadlock-free* [6].

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

Operations of a B machine are given in one of the following formats:

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

The declaration $oo \longleftarrow op(ii)$ for preconditioned operation introduces the operation: it has name *op*, a (possibly empty) output list of variables *oo*, and a (possibly empty) input list of variables *ii*. The precondition of the operation is predicate $P$. This must give the type of any input variables, and can also give conditions on when the operation can be invoked. If it is invoked outside its precondition then divergence results. Finally, the body of the operation is $S$. This is a *generalised substitution*, which can consist of one or more assignment statements (in parallel) to update the state or assign to the output variables. Conditional statements and nondeterministic choice statements are also permitted in the body of the operation. The guarded event simply has a name *op*. If its condition fails, then its execution is blocked rather than leading to a divergence.

In combined communicating systems we also define B machines that do not have operations and only contain sets, constants and invariants. These are included in order to provide contextual information to a system.