An Experiment in Formalising and Analysing Railyard Configurations

Lars-åke Fredlund and Fredrik Orava
Swedish Institute of Computer Science
Kista, Sweden
email: {fred,fredrik}@sics.se

Abstract

We present a method by which the abstract behaviour of railyards can be specified, and analysed for safety. Our analysis is based on well established railway signalling concepts such as train routes *and* flank protection *and attempts to verify safety properties which state that if the railyard is configured in a correct way, no unwanted situations such as train collisions or derailings will occur. We specify the railyards in process algebra and the safety properties in the modal μ-calculus.*

1 Introduction

The main purpose of a railway signaling system is to guarantee safety, that is, to prevent train collisions and derailings while allowing normal train movements. An often used strategy to establish this goal is to allocate *routes* — safe paths between signals in the railyard — to trains and to allow train movements only along these. The signaling practice used to establish train routes is usually implemented in an *interlocking system.* The main function of such a system is to act as a filter to which the operator (the train dispatcher) can direct requests to, for example, set and release train routes. The filter evaluates (or analyses) the requests and if it finds them safe they are carried out on the physical yard. In this paper we address the problem of deciding whether or not given routes in railyard configurations are to be considered safe.

We describe accurately the basic capabilities of rail components and analyse the result of conjoining components into railyards; we specify in detail the workings of a track segment, a signal and a point. Our formalisation of a track segment, for example, has two endpoints that can be connected to other railyard components, and at most one train can occupy a segment safely. We specify the railyard components in process algebra, structured so that each type of component makes up a separate process. The component types provide the shapes of building blocks; their interfaces are defined in terms of which endpoints they possess, and they can be connected together by identifying the appropriate endpoints. In LOTOS [15] this is achieved by sharing the same gate identifiers and connecting them together using the parallel operator.

In the work reported here the railyard components, as well as railyard obtained by connecting components, are specified in the π-calculus [12] and LOTOS; to evaluate the specifications we verify a number of safety properties on railyard components and configurations. One such class of safety properties states that whenever a railyard configuration is set up for a particular train route, and trains respect signals, then, at any time, no two trains will occupy the same component. The analysis is performed using the tools MWB [16] and CÆSAR [4].

This paper has a different focus than previous publications in the area of analysing safety-critical properties of railway systems. Groote, van Vlijmen and Koorn [7] model an interlocking system for a particular railyard in μCRL [8] and verify safety-critical properties by transforming both the program and the correctness criteria to propositional logic. Morley [13] develops a method, based on theorem proving in Higher Order Logic, to establish correctness of the signaling rules embedded in the geographic database of the Solid State Interlocking (SSI) developed by British Rail. The approach is general and can be instantiated to particular installations of the SSI. In [2] Bruns analyses, using CCS, safety-critical properties of a communication sub-system used to connect distant parts of a SSI. Hansen [9] uses VDM [1] to model concepts such as track segments, points and signals. For particular configurations safety criteria are defined and validated through simulation. In [10] King reports on the formalisation in Z [14] of the signaling rules in the SSI.

Our work differs mainly in the level of abstraction. The publications cited above are all concerned with signaling rules in the sense of relationships between aspects of signals and directions of points; the main question addressed is whether or not such a relation is safe, or if a given configuration of a railyard conforms to a safe relation. The work reported in [7, 13, 2] is oriented towards implementations of specific installations. Safety requirements are typically expressed as relationships between values of variables in the interlocking system studied.

Our work is more on a conceptual level; we specify the abstract working of railyard components in terms of events such as *a train enters a component* and *a train does not respect a red signal*. Furthermore, we do not address at all the problem of *establishing* train routes, the question we consider is: *given a railyard configuration C and a route R in C determine if R is safe*, that is, if trains respect signals, is it possible for two trains to collide?

In [9] and [10] the signaling rules are treated on a more abstract level; however, formal analysis is not an issue in these works.

The outline of the rest of the paper is as follows: In Section 2 we describe informally the behaviour of the railway components we consider together with a formal specification in the π-calculus. Specifications of components are conjoined into a small railyard for which we discuss safety properties. A short introduction to the π-calculus is included in the section. The informally expressed safety properties are expressed formally in Section 3; some properties of the individual components are discussed. An informal introduction to the formalism used to express the properties is given. Verification of the formalised properties using a number of automatic tools is discussed and some experiences gained in the exercise are reported. A brief comparison with an alternative specification in LOTOS is given. Some concluding remarks and ideas for further work are discussed in Section 4.

2 Specification of railyard components and configurations

Informal descriptions of each type of railyard component considered are presented, followed by formal specifications in the π-calculus of their behaviour. A brief introduction to the π-calculus is also included. A small railyard is specified by conjoining specifications of the constituent components. Safety properties of the yard are discussed informally.

2.1 General principles

We specify each type of component as a separate process with ports corresponding to both the physical endpoints of the object modelled as well as handles via which the internal state of the component can be investigated. For example, we model a point as a process with four ports: i, l, r and pl; the three first ones model the input, left and right tongues of the point respectively. The pl port is a handle via which the state of the model can be accessed. We use this handle in the analysis to test in which direction the point is set. Via the ports i, l and r trains can enter and exit the point. An unwanted situation, for example when more than one train occupy the point, is modelled by a special process ($PANIC$) which repeatedly reports on the port *panic*.

One common property of our model of points and segments is that they can never refuse a train to enter into the component. If a train enters an already occupied component, the component will enter the panic mode and start to report on the *panic* port. A signal on the other hand can at most contain one train; a train occupying a signal must first exit before the next train can enter. Thus it is impossible for collisions to occur in signals. One way of interpreting this is by appealing to physical distribution: points and segments have physical distribution and thus collisions can occur in these objects. Signals on the other hand has no distribution; they are objects standing between objects with distributions. Collisions do not occur in signals, they occur in the surrounding objects.

Signals in our model are asymmetrical objects, they have one front and one back side. The front of a signal displays the *aspect*, that is, the color of the signal which in our specification can be red ("stop") or green ("go ahead"). Trains passing facing signals should respect the displayed aspect. Since the back side does not display any aspect, trains should ignore opposite facing signals.

By conjoining processes modelling points, segments and signals, arbitrary railyards can be modelled. Components are connected by identifying ports that model endpoints of adjacent objects in the yard.

2.2 The π-calculus

In this section we give the syntax and informal semantics of the particular variant of π-calculus that we use in this paper. Let $x, y, z, u, v, \ldots$ range over the set of *names*, $\mathcal{N}$ and assume a set of *agent identifiers* ranged over by $A, B, \ldots$. We let $P, Q, \ldots$ range over the *agents*, which are of the following kinds: $\mathbf{0}$ is an agent which can do nothing. The output prefix, $\overline{x}\langle y\rangle \,.\, P$, is an agent whose first action will be to output the name y on port x; thereafter it behaves as P. The input prefix, $x\,(y)\,.\,P$, is an agent whose first action is to receive a name on port x; thereafter it behaves as P but with the newly received name in place of y; this name is just a place holder for the new name to be received. $P + Q$ is an agent which behaves like either P or Q. The parallel composition of P and Q, written $P|Q$, is an agent that can do anything P or Q can do, and moreover communications between P and Q can occur if one agent outputs a name and the other inputs a name on the same port. $(\boldsymbol{\nu}\, y)\, P$ is an agent which acts like P but the name y is *restricted*, i.e. it cannot be used for communications with the environment of the agent. The matching $[x = y]P$ is an agent which behaves like P if x and y are the same name; otherwise it does nothing. $A(y_1, \dots, y_n)$ is an agent if A is an identifier; for any such identifier there is a *defining equation* written $A(x_1, \dots, x_n) \stackrel{\text{def}}{=} P$, where the names $x_1, \dots, x_n$ are distinct and are the only names which may occur free in P. The agent $A(y_1, \dots, y_n)$ behaves like P where y_i is substituted for x_i for all $i = 1, \dots, n$. The x_i:s may be considered formal parameters of A, while the y_i:s are actual parameters in $A(y_1, \dots, y_n)$. Agent identifiers provide recursion since the defining equation for A may contain A itself.

The formal operational semantics of agents is defined and explained in the papers [12]; hopefully this paper can be understood without it.

Additionally, the following abbreviations are used: an agent $\pi\,.\,\mathbf{0}$ can be written π and an output prefix $\overline{x}\langle\rangle$ can be written $\overline{x}$. Analogous abbreviations are introduced for input prefixes. Repeated restrictions can be concatenated, that is, $(\boldsymbol{\nu}\, x_1, \dots, x_n)\, P$ is equivalent to $(\boldsymbol{\nu}\, x_1) \dots (\boldsymbol{\nu}\, x_n)\, P$.

In order to encode finite sets of data values we designate a finite set of names to represent them. Such names are referred to as *constants*; these are just names which will never be bound by an input or restriction operator. Syntactically constants are written in `typewriter` font. As a convention, whenever a particular constant is implied by context we will omit the constant in formal and actual parameters to agent identifiers.

2.3 Formal specifications of the components

2.3.1 Points

We specify a point as a process, $Pt(i, l, r, pl, dir)$ (see Figure 1), with four ports used for communication: i, l, r modelling the input, left and right branching tongues of the point, and pl which is used to investigate the current direction of the point. The fifth parameter of the process records the direction of the point which can be left (`lft`) or right (`rht`) as seen from the input tongue.

In the initial state the point is unoccupied and can accept a train on any of the ports i, l or r. If a train arrives via the port i, representing the normal input tongue, the point will change state to $Pt'(i, l, r, pl, dir, o, tr)$ modelling a point occupied by the train. The parameter o represents the tongue via which the train will leave and depends naturally on its direction; the parameter tr models the identity of the train. Trains entering the point in reverse direction are modelled by input via the ports l and r; the resulting state is also in this case $Pt'(\cdots)$ with the output parameter set to i, the normal input tongue. Observe that a train is only allowed to enter in reverse direction via the left (right) branching tongue if the point is set to left (right). Entering in reverse direction via one of the branches while the point is set to the other leads to a panic situation modelled by the process:

$$PANIC \stackrel{\text{def}}{=} \overline{panic}\,.\,PANIC$$

Finally, it is always possible to poll the direction of the point via the pl port.

An occupied point, $Pt'(i, l, r, pl, dir, o, tr)$, can always deliver the occupying train via the port recorded in the o-parameter and then become an empty point. If a second train should enter via any of the ports i, l or r, a collision occurs which we model as *PANIC*. The direction of a point can always be polled via port pl.

$$\begin{aligned}
&Pt(i, l, r, pl, dir) \stackrel{\text{def}}{=} \\
&\quad i\,\langle tr\rangle\,.\,\Big(\quad [dir = \texttt{lft}]Pt'(i, l, r, pl, dir, l, tr) \\
&\qquad\qquad + [dir = \texttt{rht}]Pt'(i, l, r, pl, dir, r, tr)\Big) \\
&\quad + l\,\langle tr\rangle\,.\,\Big(\quad [dir = \texttt{lft}]Pt'(i, l, r, pl, dir, i, tr) \\
&\qquad\qquad + [dir = \texttt{rht}]PANIC\Big) \\
&\quad + r\,\langle tr\rangle\,.\,\Big(\quad [dir = \texttt{lft}]PANIC \\
&\qquad\qquad + [dir = \texttt{rht}]Pt'(i, l, r, pl, dir, i, tr)\Big) \\
&\quad + \overline{pl}\langle dir\rangle\,.\,Pt(i, l, r, pl, dir) \\
&Pt'(i, l, r, pl, dir, o, tr) \stackrel{\text{def}}{=} \\
&\quad \overline{o}\langle tr\rangle\,.\,Pt(i, l, r, pl, dir) \\
&\quad + i\,\langle tr'\rangle\,.\,PANIC \\
&\quad + l\,\langle tr'\rangle\,.\,PANIC \\
&\quad + r\,\langle tr'\rangle\,.\,PANIC \\
&\quad + \overline{pl}\langle dir\rangle\,.\,Pt'(i, l, r, pl, dir, o, tr)
\end{aligned}$$

Figure 1: Specification of a point object.

2.3.2 Segments

The operation of a segment (see Figure 2) is quite simple. An empty segment is described by $Seg(p_1, p_2)$, where p_1 and p_2 model its two endpoints. Initially it is prepared to accept a train via any of its endpoints resulting in a transformation of the specification into one describing an occupied segment $Seg'(p_1, p_2, o, tr)$ where o is the endpoint via which

$$
\begin{aligned}
&\mathit{Seg}(p_1,p_2) \stackrel{\text{def}}{=}\\
&\quad p_1\,\langle tr\rangle\,.\,\mathit{Seg}'(p_1,p_2,p_2,tr)\\
&+ p_2\,\langle tr\rangle\,.\,\mathit{Seg}'(p_1,p_2,p_1,tr)\\
&\mathit{Seg}'(p_1,p_2,o,tr) \stackrel{\text{def}}{=}\\
&\quad \overline{o}\langle tr\rangle\,.\,\mathit{Seg}(p_1,p_2,pl)\\
&+ p_1\,\langle tr'\rangle\,.\,\mathit{PANIC}\\
&+ p_2\,\langle tr'\rangle\,.\,\mathit{PANIC}
\end{aligned}
$$

Figure 2: Specification of a segment object.

$$
\begin{aligned}
&\mathit{Sig}(i,o,pl,sh,vl,clr) \stackrel{\text{def}}{=}\\
&\quad i\,\langle tr\rangle\,.\,\Big(\ [clr=\texttt{gr}]\mathit{Sig}'(i,o,pl,sh,vl,\texttt{re},o,tr)\\
&\qquad\qquad + [clr=\texttt{re}]\mathit{Sig}'(i,o,pl,sh,\texttt{tt},clr,o,tr)\Big)\\
&+ o\,\langle tr\rangle\,.\,\mathit{Sig}'(i,o,pl,sh,vl,clr,i,tr)\\
&+ \overline{pl}\langle clr\rangle\,.\,\mathit{Sig}(i,o,pl,sh,vl,clr)\\
&+ [vl=\texttt{tt}]\overline{sh}\ .\,\mathit{Sig}(i,o,pl,sh,vlated,clr)\\
&\mathit{Sig}'(i,o,pl,sh,vl,clr,p,tr) \stackrel{\text{def}}{=}\\
&\quad \overline{p}\langle tr\rangle\,.\,\mathit{Sig}(i,o,pl,sh,vl,clr)\\
&+ \overline{pl}\langle clr\rangle\,.\,\mathit{Sig}'(i,o,pl,sh,vl,clr,p,tr)\\
&+ [vl=\texttt{tt}]\overline{sh}\ .\,\mathit{Sig}'(i,o,pl,sh,vl,clr,p,tr)
\end{aligned}
$$

Figure 3: Specification of a signal object.

the train (*tr*) later will exit. An occupied segment will emit *panic* if an additional train enters.

2.3.3 Signals

Signals are asymmetrical having one front side displaying the aspect of the signal and one back side. A signal "guards" the piece of railyard behind it from trains approaching the front. A train *respects* a signal if it does not pass the signal when the red aspect is displayed. A train *violates* a signal if it does pass the signal from the front when the red aspect is displayed. The aspect of a signal is irrelevant for trains approaching it from behind.

We model the behaviour of signals as processes $\mathit{Sig}(i,o,pl,sh,vl,clr)$ (see Figure 3) where i and o are the front and back entrance respectively. The port pl is used to poll the color (`re` or `gr`) recorded in the parameter clr; vl and sh are used when signals are violated (see below).

As all other components signals can be occupied or not. Trains can enter an unoccupied signal (modelled by $\mathit{Sig}(\cdots)$) which then becomes occupied ($\mathit{Sig}'(\cdots)$). Entering a green signal from the front causes the signal switch to red. If a train enters from the front while the aspect is red, the state variable vl is set to `tt` recording the fact that the signal has been violated; a violated signal can always report via the port sh ("shout"). Finally, the color can be polled via pl.

$$
\begin{aligned}
&\mathit{Yard}(in,out,out_2,out_3,pl_0,pl_1,pl_2,pl_3,pl_4,pl_5,\\
&\qquad pl_6,pl_7,sh,panic) \stackrel{\text{def}}{=}\\
&\ (\boldsymbol{\nu}\,p_1,p_2,p_3,p_4,p_5,p_6,p_7,p_8,p_9)\\
&\quad\Big(\ \ \mathit{Sig}(in,p_1,pl_0,sh,\texttt{ff},\texttt{gr})\\
&\quad\ \mid\ \mathit{Pt}(p_1,p_2,p_3,pl_1,\texttt{lft})\\
&\quad\ \mid\ \mathit{Pt}(p_2,p_4,p_5,pl_2,\texttt{lft},)\\
&\quad\ \mid\ \mathit{Seg}(p_4,p_6)\\
&\quad\ \mid\ \mathit{Sig}(out,p_6,pl_3,sh,\texttt{ff},\texttt{re})\\
&\quad\ \mid\ \mathit{Pt}(p_7,p_9,p_3,pl_5,\texttt{lft})\\
&\quad\ \mid\ \mathit{Sig}(out_3,p_9,pl_7,sh,\texttt{ff},\texttt{re})\\
&\quad\ \mid\ \mathit{Pt}(p_8,p_7,p_5,pl_6,\texttt{lft})\\
&\quad\ \mid\ \mathit{Sig}(out_2,p_8,pl_4,sh,\texttt{ff},\texttt{gr})\Big)
\end{aligned}
$$

Figure 4: Specification of a railyard.

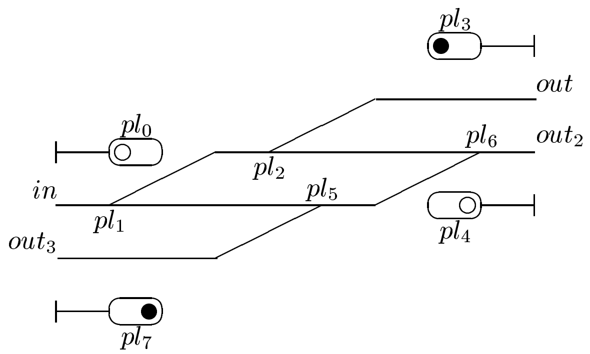

Figure 5: The structure of the specified railyard.

An occupied signal ($\mathit{Sig}'(i,o,pl,sh,vl,clr,p,tr)$) can shout if violated and be polled for the color it displays; furthermore, it can deliver the occupying train (tr) via the port recorded in the parameter p.

2.4 Specification of a railyard

A railyard is defined by connecting processes modelling points, segments and signals. Desirable safety and liveness properties are informally discussed, the formalisation and verification of some of these properties are discussed in Section 3.

In Figure 4 a small railyard consisting of four points, four signals and one segment is specified by connecting processes modelling the components and restrict internal interconnection points. The structure of the specified system is displayed in Figure 5. The "state" of the railyard, that is, the direction of the points and the aspects displayed by the signals can be polled from the ports $pl_0 \ldots pl_7$.

The motivation for the work presented here is to experiment with verification of safety properties of railyard configurations. To illustrate our method we define a number of train routes through the railyard specified above together with requirements on the part of the yard outside the routes sufficient to guarantee safety. Below we describe our way of defining train routes and safety requirements.

A train route defines settings of the constituent points and reversed signals. For example, a route from *in* to *out* in Figure 5 above is given by the settings of the points at pl_1 and pl_2 together with the setting of the signal at pl_3. Thus, in our method, a train route defines a set of global states, or configurations, of the formal specification of the yard, namely the set of global states in which the points and signals included in the route have their required settings.

There are several requirements that one would like to establish for a train route to be set and to be considered safe: (i) there should be a path from the start signal to the end signal of the route, (ii) the end signal of the route should display a red aspect in the reversed direction relative to the route, and (iii) it should not be possible to enter (or cross) the route other than via the input and output signals.

Requirement (i) states that it should be possible to move a train along the route. Requirement (ii) states that moving a train in reverse direction along the route should immediately violate a signal. Requirement (iii) is called *flank protection* and is obtained by setting points outside the route in directions physically prohibiting trains to enter the route from outside of it. Note that the only way a train can enter the route (other than via the input and output signals) is via unused tongues of its points.

In the configuration displayed in Figure 5 a train route from *in* to *out* requires the following:

1 The point at pl_1 should be set at left.

2 The point at pl_2 should be set at left.

3 The signal at pl_3 should be red.

Note that we do not require the signal at pl_0 to be green for the train route to be set, rather our verification amounts to establish that this signal can safely be set to green if the above is true and the flank protection (defined below) is fulfilled. The flank protection of the route is defined by:

4 The point at pl_5 should be set at left.

5 The point at pl_6 should be set at left.

Note that the signals at pl_4 and pl_7 are immaterial for the flank protection.

The safety property we would like to establish for railyard configurations as the one above can informally be stated as: *if the train route from in to out is set and flank protection is established, then a train can safely move from in to out.* We consider a railyard to be safe if two trains never can occupy the same component of the yard. However, this property is not in general true of railyards; it crucially depends on trains to respect signals, that is, not to pass signals which display red aspects. To establish the safety of a configuration we instead verify that if a specified train route and flank protection are established in the yard, then as long as no train violates any signal no panic situation will occur. This property is of course valid only if the railyard initially is empty, that is, we implicitly assume that no component of the railyard under consideration contains any trains.

We close this section by noting that there is of course possible to have more than one train route set in the same yard at the same time. In the yard discussed above a route from out_2 to out_3 is established by the additional requirement:

6 The signal at pl_7 should be red.

Note finally that the two routes constitute flank protection for each other.

3 Formalising and verifying correctness requirements

The informal correctness requirements described in earlier sections are here expressed formally in a variant of the modal μ-calculus [11], and are verified by means of model-checking procedures.

3.1 Logic

The full definition of the logic will not be given here, rather we will briefly present the intuitive meaning of the constructs in Figure 6. A formal definition of the logic is given in [3], albeit with a slightly different syntax.

$$
\begin{array}{rcl}
F & ::= & \mathtt{tt} \\
& | & \mathtt{ff} \\
& | & F_1 \ \wedge \ F_2 \mid F_1 \ \vee \ F_2 \mid F_1 \ \Rightarrow \ F_2 \\
& | & \langle\alpha\rangle\, F \\
& | & [\alpha]\, F \\
& | & \mathit{lfp}\; X\; .\; F \\
& | & \mathit{gfp}\; X\; .\; F \\
& | & X
\end{array}
$$

Figure 6: Syntax of formulae

Informally then, `tt` and `ff` represent truth and falsity, respectively. The possibility modality, $\langle\alpha\rangle F$, is satisfied by a process that can perform an action α

and then satisfy F[1]. Similarly, the necessity modality $[\alpha]F$ is satisfied by a process that satisfies F after performing α. The '$-$' symbol is used as a wildcard, and may occur in the place of an action or name. The greatest fixpoint of an equation is denoted by $\mathit{gfp}\ X\ .\ F$, where the formula F may contain a reference to the fixpoint identifier X. Analogously, the least fixpoint identifier is denoted with lfp .

Variables representing actions or names are written in bold ('$\boldsymbol{x}$') whereas constant names are written in italics ('x').

3.2 Correctness requirements

In the following the example railyard defined in Section 2.4 will be used to illustrate our method for analysing whether train routing through a railyard is safe. While we focus attention on a particular railyard configuration, the method is general, and therefore applicable to other railyards as well.

The most basic information required to analyse a railyard is its static configuration, i.e., of what track objects it is built up and how they are interconnected. The state of track objects, i.e., the aspect of signals and direction of points, is specified in a train route with flank protection, that precisely captures the necessary conditions for safe train operations in the railyard. That a given train route with flank protection indeed guarantees safe operations is the main proof obligation of the method.

A train route with flank protection does not necessarily completely specify the state of every track object in the railyard, e.g., the direction of a point may be left unspecified in which case train operations must be shown to be safe regardless of its direction.

The requirements on the specification can be separated into the following categories.

3.2.1 Requirements on components

Here requirements on individual types of components are stated, e.g.,

> A signal switches to "red" after a train has passed through the signal.

Such requirements can easily be specified in the logic. However, the formalisation of such properties are generally not easier to understand than the specification of the component itself. As an example the coding of the above requirement for a signal s is shown below:

$$\mathit{gfp}\ X\ .\ \Big([-]X \ \wedge\ (\langle\overline{pl_s}\langle gr\rangle\rangle\mathtt{tt} \ \Rightarrow \\ [\overline{s}\langle\boldsymbol{tr}\rangle]\langle\overline{pl_s}\langle re\rangle\rangle\mathtt{tt})\Big) \qquad (1)$$

[1] We assume early semantics for input actions.

The constant names re and gr indicate the colours red or green. Note that $\boldsymbol{tr}$ is a variable, that matches any train that can pass through the signal s. Informally the formula states that for all reachable states ($\mathit{gfp}\ X\ .\ \big([-]X \ \wedge\ F\big)$) the formula F holds, where F expresses that if the signal is green ($\langle\overline{pl_s}\langle gr\rangle\rangle\mathtt{tt}$), and a train passes through the signal ($[\overline{s}\langle\boldsymbol{tr}\rangle]$), then the colour of the signal will be red ($\langle\overline{pl_s}\langle re\rangle\rangle\mathtt{tt}$)) in the next state. In conjunction with a property stating that the colour of a signal is either green or red the above formula expresses the desired correctness property.

Two additional properties are implicit below:

> The passing of a train through a red signal is indicated by a *sh* action.

and

> If two (or more) trains occupy the same track segment (or point) then a *panic* action becomes possible.

3.2.2 Well-formedness

Some requirements express the "well-formedness" of the specification, e.g., for behaviours that no deadlocks or livelocks exist, and in terms of structure that well-known design principles have been followed, for instance that every entrance to the railyard is guarded by a signal.

As an example we show the coding of the deadlock freedom property in the logic:

$$\mathit{gfp}\ X\ .\ \big(\langle-\rangle\mathtt{tt} \ \wedge\ [-]X\big) \qquad (2)$$

The translation of the specification in Section 2.4 into LOTOS is deadlock-free, and there are no livelocks either.

3.2.3 Train Routes

The most interesting requirements express global system properties, focusing in particular on the concepts of train routes and flank protection and how they guarantee safe train operations.

The claim that a train route has been set up can easily be specified by referring to the components for the train route, e.g., for the route from *in* to *out* in our example:

$$\mathit{Route} \overset{\mathrm{def}}{=} \langle\overline{pl_1}\langle \mathit{lft}\rangle\rangle\mathtt{tt} \wedge \langle\overline{pl_2}\langle \mathit{lft}\rangle\rangle\mathtt{tt} \wedge \langle\overline{pl_3}\langle re\rangle\rangle\mathtt{tt} \quad (3)$$

That is, points 1 and 2 switch to the left and signal 3 shows red.

Flank protection is defined analogously:

$$\mathit{Flank} \overset{\mathrm{def}}{=} \langle\overline{pl_5}\langle \mathit{lft}\rangle\rangle\mathtt{tt} \ \wedge\ \langle\overline{pl_6}\langle \mathit{lft}\rangle\rangle\mathtt{tt} \qquad (4)$$

Since our specification only addresses static configurations, we expect the route and flank protection to be set initially, and to remain set during the execution of the specification:

$$RouteFlank \stackrel{\text{def}}{=} \\ gfp\ X\ .\ \Big(Route\ \wedge\ Flank\ \wedge\ [-]X \Big) \qquad (5)$$

The main safety property to check given a railyard configuration together with a proposed train route and flank protection is that if no train ever violates a signal (passes a red signal) then no two trains will ever occupy the same track segment:

$$RouteFlank\ \Rightarrow \\ gfp\ X\ .\ \Big(\langle \overline{sh} \rangle \mathtt{tt}\ \vee\ ([\overline{panic}]\mathtt{ff}\ \wedge\ [-]X) \Big) \quad (6)$$

That is, if the route and flank protection has been set then *panic* remains impossible until a *sh* can occur. This property is satisfied by the configuration depicted in Figure 4.

Note that the formulae *Route* and *FlankProtection* do no put down any requirements on the signal at out_2. Thus a further proof obligation is to show that the configuration satisfies the safety property even if its colour is green, i.e., when more than one train is allowed to enter the railyard.

Furthermore, no requirement has been put on the entrance signal to the train route (*in*). If that signal initially shows red, then trivially the train route is safe since no train can pass along the route.

To determine whether a train route is useful, as opposed to merely safe, we formulate a property stating that if the entrance signal to the route is green, a train enters the train route, and the railyard is in a safe state (no signals have been violated) then there is a possibility that the same train eventually leaves the train route at the exit signal without any signals being violated:

$$RouteFlank\ \wedge\ \langle \overline{pl_0} \langle gr \rangle \rangle \mathtt{tt}\ \wedge\ [\overline{sh}]\mathtt{ff}\ \Rightarrow \\ \langle in\, \langle \boldsymbol{tr} \rangle \rangle\ lfp\ X\ . \\ \Big([\overline{sh}]\mathtt{ff}\ \wedge\ (\langle \overline{out} \langle \boldsymbol{tr} \rangle \rangle \mathtt{tt}\ \vee\ \langle - \rangle X) \Big) \quad (7)$$

Properties such as these can often be verified by means of widely used (software) engineering methods such as simulation and/or testing, rather than by model checking.

3.3 Tool support

This section contains a brief discussion on the actual verifications, i.e., the application of the automatic verification tools MWB [16] and CÆSAR [4] on the railyard example is briefly discussed.

3.3.1 The Mobility Workbench

The π-calculus specification from Section 2.4 was verified with respect to the correctness properties from Section 3.2 using the Mobility Workbench.

It proved easy to translate the logic formulae into the syntax of the mobility workbench, although they grew considerably in size. The growth was mainly due to the absence of the "possible-any" construct ($\langle - \rangle F$) from the logic, which had to be replaced by a case analysis on whether an input, output or τ action was possible.

Variants of the properties (1)-(7) were all verified on the example railyard, with the exception of the deadlock freedom property (2). The model checking procedure used by the Mobility Workbench is a local algorithm, i.e., it visits only the states necessary to ascertain whether a formula holds. Moreover, the model checking algorithm performs a depth-first traversal of the state space, remembering only the states encountered on the current path from the initial state. Thus the algorithm demands surprisingly small amounts of memory, in essence trading memory for a longer execution time. This strategy proved successful for checking most formulae, since only part of the state space of the specification needed to be explored. In checking Formula 6, for instance, all branches starting with a *sh* action need not be explored further. For verifying the deadlock freedom property the algorithm has to visit all states, and is likely to visit each state many times. Since a comparable specification in LOTOS had approximately 700.000 states (state space generated by CÆSAR) the checking of the deadlock freedom property can be expected to take a long time[2].

3.3.2 LOTOS and the CÆSAR toolset

The π-calculus specification was subsequently translated into LOTOS, and the CÆSAR toolset was used to verify the correctness of the specification. The reason for this additional experiment was primarily to compare the tools.

The coding of the railyard example in LOTOS proved surprisingly awkward. The first approximation that had to be made (to enable verification using CÆSAR) was to limit the number of identities of trains that could pass through the railyard to a single one.

Compared to the π-calculus (and CCS) the communication patterns of LOTOS processes are rigidly controlled by the parallel operator. For each two processes put in parallel, it must be explicitly defined on which set of names (gates) they may synchronise.

[2] We aborted the verification attempt after it had run for a week on a fast SUN workstation.

This proved to be an added complication in the translation of the railyard example to LOTOS. Thirdly, in contrast to π-calculus (and CCS) communication in LOTOS is symmetrical, i.e., there is no explicit sender or receiver. Since the coding of two adjacent track objects in the π-calculus both offered to receive a train on their shared name, a straight coding of the track objects in LOTOS would result in spontaneous communications between adjacent track objects. To remove these unwanted communications a new directional parameter was introduced in all communications between track objects. For example, the coding of a track segment in LOTOS offers to synchronise on both its endpoints, but with different directional parameters:

```
process Segment[p1,p2,panic](ori:PAIN)
        :noexit:=
 p1 ?tr:TRAIN !ori;
   TrInSeg[p1,p2,p2,panic](tr,ori,ori)
[]
 p2 ?tr:TRAIN !rev(ori);
   TrInSeg[p1,p2,p1,panic](tr,ori,rev(ori))
endproc
```

The `ori` parameter is "left-to-right" for a track component oriented in the normal fashion, and "right-to-left" otherwise.

The resulting LOTOS specification was verified using the "evaluator" tool from the CÆSAR toolset. The evaluator tool can check whether a LOTOS specification (or state graph) satisfies a μ-calculus formula using a local model checking algorithm [6], thus potentially allowing verification of large systems.

The variant of the μ-calculus accepted by the evaluator tool is more restrictive than the logic accepted by the Mobility Workbench in that variables are not admitted in formulae. For the purpose of checking the correctness properties of the railyard example this restriction has not been difficult to handle. However, we expect that the formulation of correctness properties for, say, a communication protocol will prove to be more cumbersome due to the lack of variables.

To reduce the state space of the LOTOS specification the signal object was modified to send out only a single *sh* indication when a train violated the signal (passed a red signal). On the other hand the complexity of verifying the LOTOS specification was increased by modifying the specification to not issue *panic* or *sh* indication for track objects that are not in the train route.

Variants of the properties (1)-(7) were all verified on the example railyard, including the deadlock freedom property (2). The evaluator tool was employed in two different fashions: (i) applied directly on the LOTOS specification, and (ii) applied on the state graph resulting from minimising the LOTOS specification using the Aldébaran tool [5].

Method (i) has the obvious advantage that the state space of the specification need not necessarily be generated, with potentially large reductions in memory consumption and execution time. On the other hand, checking several properties for which a large portion of the state space needs to be examined can take a long time. As a result we experimented with both of the above methods, and they seemed to complement each other nicely. The use of the evaluator tool is not trouble-free, however, since the time elapsed to verify a formula can depend heavily on the structure of the formula being verified. A simple reordering of the subterms of a formula can result in an analysis that is several orders of magnitude faster (or slower).

3.3.3 Diagnostics

A common problem when using either the Mobility Workbench or the evaluator in the CÆSAR toolset is how to make sure that a specification satisfy a formula for the right reasons. That is, checking that the formula indeed express desirable properties and that the reason a specification satisfies the formula is not completely trivial and uninteresting.

The Mobility Workbench offers little help here, apart from displaying the number of "inferences" required to check a formula. The evaluator tool has an (currently undocumented) option of generating a reduced state graph derived from the specification, such that the formula when applied on the reduced graph yields the same result as when applied on the original specification. In practise this can sometimes be of help to understand why a formula does not hold, but more often, the reduced state graph is huge, and can easily take longer to generate than the actual checking of the formula requires. In conclusion, much further work is needed in the area of diagnostics to make the tools more practically useful.

4 Concluding remarks and directions for further work

In this paper we have outlined a method to specify and analyse the abstract behaviour of railyards. Railyards are described by specifying the behaviour of generic components of yards, for example points, track segments and signals, in terms of physical events such as "train enters point" and "train passes signal" as well as more abstract events such as "an unwanted situation has been detected in point" and "train does not respect red signal". The analysis of railyard configurations is based on the well established signalling concepts train route and flank protection. The main proof obligation of our method is to verify that if a train route with flank protection is established and trains respect the signals of the route, then a train can move safely along it. It should be noted here that we do not impose any restrictions on aspects of signals not part of the train route under consideration, nor positions of points not included in route or flank protection.

Our motivation for embarking on this line of research was an ongoing discussion with a railway signalling company in Sweden. The work started as a vehicle for us to understand railway signalling concepts and applications, and furthermore to demonstrate the kind of formal design methods we are familiar with.

We envisage the methods to be used as a "debugging tool" for proposed train routes in several different ways. If fixed train routes are to be defined, together with different suggestions for flank protection, these can be verified for safety once and for all. The verified configurations can subsequently be used safely. A more dynamic usage of the method is to verify train routes with flank protection suggested by some computerised interlocking system. It should be noted that our method is quite general; it does not assume any particular interlocking system to be used, nor any specific signalling protocol, to guarantee safety on the railyard.

The examples in this paper are of course heavily simplified, we have for example not at all considered relations between aspects of subsequent signals imposed by most signalling protocols (for example, a green signal should not directly be followed by a red one, there should be one intermediate signal notifying the driver that the next signal cannot be assumed to display green aspect); however, we conjecture that the method generalises also to these kinds of properties.

As the number of components in the specification increases, the specification will naturally grow in size. A natural question to ask is whether specifications of large railyards are amenable to automatic verification. To answer this question it is important to investigate methods by which this increase in state space can be tamed; one promising direction for further work is to explore compositional verification methods. A preliminary idea in this direction is to define a "minimal" specification of a train route such that the specification is behavioural invariant under concatenation, that is, the concatenation of two minimal train routes is required to be equal (in some behavioural meaning) to a minimal train route. The verification of a train route then boils down to proving the proposed path through the railyard equal to a minimal train route. Due to the invariance under concatenation, this can be done in a compositional manner by adding sufficiently small train routes so as to keep the state space small.

References

[1] D. Bjørner and C.B. Jones. *Formal Specification and Software Development.* Prentice-Hall International, 1982.

[2] G. Bruns. A case study in safety-critical design. In *Proc. 4^{th} International Workshop on Computer Aided Verification*, volume 663 of *Lecture Notes in Computer Science.* Springer Verlag, 1992.

[3] M. Dam. Model checking mobile processes. In *Proc. CONCUR'93 4^{th} International Conference on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 22–36. Springer Verlag, 1993. Full version as SICS research report RR:94-01. Accepted for publication in *Information and Computation.*

[4] J.-C. Fernandez, H. Garavel, L. Mounier, A. Rasse, C. Rodriguez, and J. Sifakis. A toolbox for the verification of LOTOS programs. In L. Clarke, editor, *Proc. 14^{th} International Conference on Software Engineering*, pages 246–259. ACM, 1992.

[5] J.-C. Fernandez and L. Mounier. A tool set for deciding behavioral equivalences. In J.C.M. Baeten and J.F. Groote, editors, *Proc. CONCUR'91 2^{nd} International Conference on Concurrency Theory*, volume 527 of *Lecture Notes in Computer Science.* Springer Verlag, 1991.

[6] J.-C. Fernandez and L. Mounier. A local checking algorithm for boolean equation systems. Research Report 95-07, VERIMAG, 1995.

[7] J.F. Groote, J.W.C. Koorn, and S.F.M. van Vlijmen. The safety guaranteeing system at station Hoorn-Kersenboogerd. In *Proc. 10^{th} IEEE Conference on Computer Assurence (COMPASS)*, Gaitherburg, 1995.

[8] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicationg Processes (ACP94)*, Workshops in Computing. Springer Verlag, 1995.

[9] K.M. Hansen. Validation of a railway interlocking model. In *Proc. Formal Methods Europe*, volume 873 of *Lecture Notes in Computer Science.* Springer Verlag, 1994.

[10] T. King. Formalising British Rail's signalling rules. In *Proc. Formal Methods Europe*, volume 873 of *Lecture Notes in Computer Science*. Springer Verlag, 1994.

[11] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[12] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I and II. *Information and Computation*, 100(1):1–77, 1992.

[13] M.J. Morley. Safety in railway signaling data: A behavioural analysis. In *Proc. 6^{th} annual workshop on higher order logic, theorem proving and its applications*, volume 780 of *Lecture Notes in Computer Science*, Vancouver, 1993. Springer Verlag.

[14] J.M. Spivey. *The Z Notation — A Reference Manual.* Prentice-Hall International, 1992.

[15] P.H.J. van Eijk, C.A. Vissers, and M. Diaz, editors. *The Formal Description Technique LOTOS.* North-Holland, 1989.

[16] B. Victor and F. Moller. The Mobility Workbench: A tool for the π-calculus. In *Proc.* 6^{th} *International Workshop on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer Verlag, 1994.