# Safety and Line Capacity in Railways – An Approach in Timed CSP

Yoshinao Isobe[1], Faron Moller[2],
Hoang Nga Nguyen[2], and Markus Roggenbach[2,⋆]

[1] AIST, Japan
[2] Swansea University, UK

**Abstract.** Railways need to be safe and, at the same time, should offer high capacity. While the notion of safety is well understood in the railway domain, the meaning of capacity is understood only on an intuitive and informal level. In this study, we show how to define and analyse capacity in a rigorous way. Our modelling approach builds on an established modelling technique in the process algebra CSP for safety alone, provides an integrated view on safety as well as capacity, and offers proof support in terms of (untimed) model checking.

## 1 Introduction

Overcoming the constraints on railway capacity caused by nodes (stations and junctions) on the rail network is one of the most pressing challenges to the rail industry. In 2007, the UK governmental White Paper "Delivering a Sustainable Railway" [9] stated: *"Rail's biggest contribution to tackling global warming comes from increasing its capacity."* High capacity, however, is but one design aim within the railway domain. Railways are safety-critical systems. Their malfunction could lead to death or serious injury to people, loss or severe damage to equipment, or environmental harm. This work, carried out in cooperation with our industrial partner Invensys Rail, aims to develop an integrated view of rail networks within which capacity can be investigated and enhanced without compromising safety.

The process algebra CSP [11,18] has successfully been applied to modelling, analysing and verifying railways for safety aspects, see e.g. [20,21]. Solely concerned with safety, these approaches have ignored any aspect of time. However, the capacity of a rail network node is highly dependent on time: moving a point or moving a train through a node takes time, and sighting and braking distance are both functions of time. Thus, rather than using CSP, we apply Timed CSP [19,17] in order to achieve an integrated view on safety and capacity. While, e.g., [20,21] model safety within CSP, to the best of our knowledge we are the first to consider railway capacity within Timed CSP or any other similar formalism. One of the benefits of using (Timed) CSP is its naturalness; it takes little

⋆ This work was supported by RSSB/EPSRC under the grant EP/I010807/1.

J. Derrick et al. (Eds.): IFM 2012, LNCS 7321, pp. 54–68, 2012.

effort to explain our formal models to our industrial partners who have assisted us throughout the process in ensuring that our models remain faithful to their engineering designs.

Of the various capacity notions within the railway domain, we deal here with so-called theoretical line capacity. "Theoretical capacity" denotes the capacity (i.e., throughput) that in principal can be scheduled (as opposed to the capacity actually used). By "line capacity" we refer to a situation in which all trains are of the same characteristic (e.g., all trains have the same braking behaviour and the same maximal speed) and take the same path through a network. It remains future work to capture the more complex notion of "network capacity" (the number of trains that can operate on a rail network in a given time period).

The literature on railway capacity classifies the various approaches into analytical, optimisation and simulation methods. Analytical methods, e.g., [3,7], model the railway infrastructure by means of mathematical expressions where a preliminary solution can be easily determined. This measures theoretical capacity and helps to identify bottlenecks. In contrast to this, optimisation methods, e.g., [4,16], utilise theoretical capacity by providing optimal time tables. Finally, simulation methods, see [6] for a survey, imitate the operation of the real world railway network over time. They present the dynamic behaviour of the network as moving from state to state with respect to well-defined rules. Our approach is closest to simulation methods. We differ from them as we take all possible system runs into account and therefore obtain a more concise result concerning capacity. Taking into account the whole behaviour of the system allows us also to consider safety. Overall, this leads to an integrated method.

Concerning safety, we build on the work of [21]. In general, other approaches outside of the Csp world, e.g., [10,13] verify the safety of interlocking programs with logical approaches and SAT-based model checking as the underlying proof technique.

Our paper is organized as follows. In Section 2, we discuss basic railway concepts in terms of a realistic double junction example provided as a real-world challenge by our industrial partner Invensys Rail, and use this example to motivate the question of capacity. In Section 3, we review the approach advocated by Winter [21] to modelling safety in the railway domain using Csp. In Section 4, the language Timed Csp and the idea of timed traces is introduced. In Section 5, we describe how to extend Winter's approach in order to capture the timing of events on a railway. Given such a timed behaviour, we ask in Section 6 what capacity it has by defining capacity as a function on timed traces which we then encode as a Timed Csp refinement. In Section 7 we apply these results to our original example, before concluding with an outline of future work in Section 8.

This paper is a significantly improved variant of [12], which was presented in an informal workshop setting without proper proceedings. We would like to thank Simon Chadwick and Dominic Taylor from Invensys Rail for their encouraging support and continuous invaluable feedback.

## 2 Railway Terminology and the Double Junction Example

We explain typical railway concepts in terms of the track plan shown in Figure 1. Engineers from Invensys Rail proposed this plan of a realistic *double junction* for our study as it exhibits typical challenges for safety and capacity. Their plan consisted of the elements in normal font; we added the components in boldface in order to facilitated the analysis and verification of railway protocols working over this junction. The double junction connects a *side line* with a *main line*. Concerning safety, its challenges include avoiding train collisions and preventing derailments. Concerning capacity, one is interested in optimising single paths through the junction as well as in reducing the time that one path blocks another.

The track plan depicted in Figure 1 consists of various elements. There are a number of individual *tracks*, which in the plan are named with two characters, e.g., AA; there are two *points*, namely point 101 and point 102; and finally there is the diamond crossing BW. A point may be in one of two positions: *normal* or *reverse*. If point 101 is in *normal position*, a train can pass from track AB to track AD; if it is in *reverse position*, a train can pass from track AB to track BW. The diamond crossing BW can be passed in two ways: it connects the tracks BV and BX, and it also connects the tracks AC and CM. The double junction is designed in such a way that trains can travel through it along four paths. There are two paths on the *main line*, paths $\overrightarrow{AB}$ and $\overrightarrow{DC}$. Path $\overrightarrow{AE}$ leaves the main line and enters the *side line* on track CM. Finally, path $\overrightarrow{FC}$ leaves the side line after track DR and enters the main line on track BY. On the main line, trains can travel at a speed of 120mph, whereas on the side line (i.e., on tracks CM, CL, DR, and DP) there is a speed restriction of 70mph. There is a further speed restriction of 40mph on the points 101 and 102 when they are in reverse position. These speed restrictions, which are as provided as a realistic scenario by Invensys Rail, are not shown on the track plan. There are six signals in the original plan, labelled 2, 3, 4, 5, 16 and 17, which show either *proceed* or *stop*. Train drivers are only allowed to enter a track, say AB, when its protecting signal, in our example signal 3, shows proceed; otherwise, the train driver has to stop and to wait until the signal changes to proceed.

Railway signals are controlled by an interlocking system which aims to guarantee safety. In general, train movements are considered safe if there is no risk of collision (through allowing two trains on the same track at the same time) nor any risk of derailment (through allowing a point to move while there is a train passing over the point, or by allowing a train to pass too fast over a point). In our study, we concentrate on capacity and consider only the collision-freedom aspect of safety.

An interlocking system gathers inputs from the physical railway, such as train locations with respect to tracks, and sends out commands to control signal aspects and point positions. To this end, it implements so-called *control tables* which dictate its behaviour. The control table for our double junction example appears in the lower left corner of Figure 1 and restricts the behaviour of the railway according to current UK regulations. For each signal, there is one row

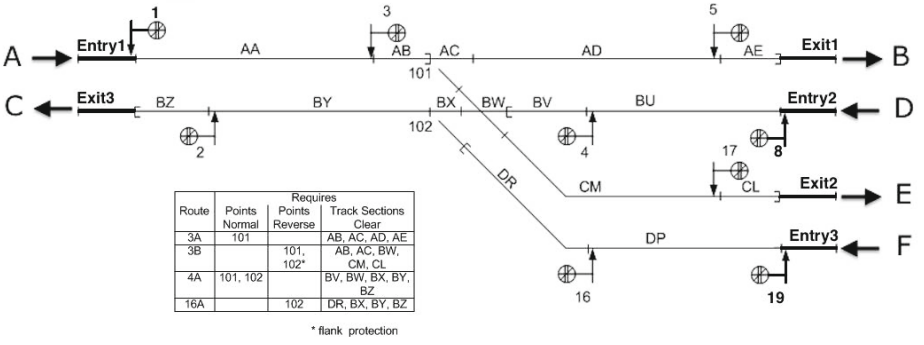


**Fig. 1.** The track plan of the *double junction*

describing the condition under which the signal can show proceed. There are two rows for signal 3: one for when the train stays on the main line (Route 3A) and one for when the train moves to the side line (Route 3B). Signal 3 for the main line can only show proceed when point 101 is in normal position and tracks AB, AC, AD and AE all are clear. The track AE is called an *overlap*. This rule ensures that the driver will always be able to stop the train before entering the track following AE, even if signal 5 is seen too late (e.g., just as the train is passing it). Besides the condition shown in the control table, signal 3 for the side line can only show proceed if an approaching train on track AA is slow enough. This is controlled by measuring the time that the train occupies track AA (approach control). This forces the train to slow down to 40mph even before it reaches signal 3. We refer to this version of control as Scenario 1.

Scenario 2 makes the assumption that all trains are equipped with an Automatic Train Protection (ATP) system [14]. ATP ensures that trains brake when needed and reduce their speed as required. Thanks to ATP, trains are guaranteed to stop at or before signal 5. Therefore the overlap AE can be removed from the clear part of the control table of signal 3. ATP guarantees that trains slow down in time to 40mph when passing a point. Thus, approach control is not needed. For Scenario 2, we remove all overlaps from the control table and work without approach control. Under current UK regulations, Scenario 2 is not allowed.

In the railway domain, capacity is regarded as an elusive concept which is not easy to define and measure. In general, it can be described as below:

> "Capacity determines the maximum number of trains that would be able to operate on a given railway infrastructure, during a specific time interval, given the operational conditions." [5].

Returning to our double junction example, the general view in the railway industry – shared by our industrial partner Invensys Rail – is the following. Removing overlaps such as track AE from control tables and removing approach control increases capacity. Our scientific questions are: can safety still be guaranteed? And how can the expected effect be measured? Based on the answers to these

questions, the political question would be: does the resulting capacity increase justify changes to regulations?

## 3 Modelling Railways for Safety in CSP

The process algebra Csp [11,18] is an established formalism for describing concurrent systems. While there is still ongoing research on foundations, Csp has many applications, e.g., in train controllers, in avionics, and in security protocols. We describe here only the basic constructs of Csp that we shall exploit.

Csp describes reactive systems in terms of abstract, discrete events such as "train 12 enters track 1". The events of a system are collected together in an *alphabet of communications* $\Sigma$. All such communications are atomic. In Csp terminology, a reactive system is referred to as a *process*. The most basic process is *Stop*, which represents the system that does not do anything. Another basic process is *Skip*, which represents the system that performs the termination event ✓ (pronounced as tick) and then behaves like *Stop*. Given an event $a$ and a process $P$, the Csp process $(a \rightarrow P)$ represents the system that engages in the event $a$ and then behaves like $P$. Csp provides two operators which allow a choice between processes: the process $(P \sqcap Q)$ is the *internal* choice operation which represents a system which will behave as either $P$ or $Q$ with the choice made nondeterministically by the system; while the process $(P \;\Box\; Q)$ is the *external* choice operation in which the choice between behaving as either $P$ or $Q$ is made by the environment. Csp provides various operators to combine two processes $P$ and $Q$ in parallel, but the only such operator of interest to us is $(P \,|[\, X \,]|\, Q)$ which requires the processes $P$ and $Q$ to cooperate on the events in the set $X$. Finally, the process $(P \setminus X)$ behaves like $P$ but makes the events of $X$ invisible to the environment. Semantically, Csp describes a process $P$ by the set of all its traces $\mathcal{T}[\![P]\!]$, i.e., all finite sequences of events that the process can perform. A process *SPEC* is refined by a process *IMP*, written as $SPEC \sqsubseteq_T IMP$, iff $\mathcal{T}[\![IMP]\!] \subseteq \mathcal{T}[\![SPEC]\!]$. This refinement preserves safety: if a forbidden sequence of events $s$ is excluded from *SPEC*, then $s$ cannot be a trace of *IMP* if $SPEC \sqsubseteq_T IMP$. In practice, one usually works with $\text{Csp}_M$, a machine-readable version of Csp which also includes concepts of functional programming which handle data.

Winter [21] describes how to model railway systems in Csp for proving safety. We illustrate Winter's approach by modelling the double junction shown in Figure 1. Here we include the bold elements: without signal 1 trains could collide on track AA. The first step is to formalise the track plan as a graph:

```
datatype TrackIDs  = AA | AB | AC | AD | AE | ...
datatype SignalIDs = S2 | S3A | S3B | S4 | S5 | S16 | S17 | ...
datatype PointIDs  = P101 | P102
next(AA) = {AB}  next(AB) = {AC}  next(AC) = {AD,BW} ...
```

Then, trains are modelled as processes. Here, it is necessary to change Winter's definitions, as tracks can be shorter than trains (track AE is 50m long, Invensys suggested to work with a train length of 200m). We associate every track $t$ (and

train *id*) with its length, denoted as *tracklength*($t$) (and *trainlength*(*id*)) and define a process, namely *RearBehaves*, which refrains rear moves if the front track is shorter than the train length. Then, a train process is characterized by its identifier *id*, by the position of its *front* and by a list of *rearmoves* as follows:

```
TrainBehave(id,front,rearmoves) =
 if (front==Exit and null(rearmoves)) then TrainBehave(id,entry(id),<>)
 else ([] n1 : next(front) @ moveff.front.n1 ->
  RearBehaves(trackLength(n1),id,n1,
              rearmoves^<(moverr.front.n1,trainLength(id))>))
```

Next, control tables are modelled. We give here only the basic idea as presented in [21]. The signalling in the double junction requires a slightly more involved approach. When the front of a train enters the protected area, the signal state becomes *Red* indicating "halt". Similarly, when the rear of a train leaves the protected area, the signal state becomes *Green* indicating "proceed".

```
SignalBehave(id.aspect) =
   (aspect == Green & [] n : next(signalhome(id)) @
       moveff.signalhome(id).n -> SignalBehave(id.Red))
[] (aspect == Red & [] n : next(signalend(id)) @
      moverr.signalend(id).n -> SignalBehave(id.Green))
```

Finally, the whole train system comprises trains and signals which interact through a set of synchronized events:

```
TrainSystem = Trains [| union(
      Union({{ moveff.signalhome(id).n |
               n<- next(signalhome(id)) } | id <- SignalIDs }),
      Union({{ moverr.signalend(id).n |
               n<- next(signalend(id)) } | id <- SignalIDs }))
    |] Signals
```

We formalise the property `NoCollision` following Winter [21]. The difference is that we exclude the entry and exit tracks from the safety analysis:

```
P(F,R) =
([] on:union(F,R) @
(not(member(next(on),union(F,R))) or member(next(on),Exits) &
 (moveff.on.next(on) ->
   P(union(diff(F,{on}),union({next(on)},Entries)),R)))
  []
  (moverr.on.next(on) ->
  P(F,union(F,union(diff(R,{on}),union({next(on)},Entries))))))
SafeMove = P(Entries,Entries)
NoCollision = SafeMove ||| CHAOS(diff(Events,{|moveff,moverr|}))
```

A railway is safe iff it can perform only safe moves. This can equivalently be formulated in Csp as the refinement statement over the traces of the respective processes: $NoCollision \sqsubseteq_T TrainSystem$.

## 4 Timed CSP and Timed Traces

Timed Csp [19] conservatively extends the process algebra Csp with timing primitives, modelling the passage of time with reference to a single, conceptually global clock. Syntactically, the core extension of Csp to Timed Csp is modest. There are only three new operators, including timeout after $d$ time units: $(P \triangleright^d Q)$. Based on these, Timed Csp adds many operators as syntactic sugar. Most prominent are $(Wait\ d) = (Stop \triangleright^d Skip)$ – the process, which waits for $d$ time units before it terminates – and a delayed event prefix $(a \xrightarrow{d} P) = (a \rightarrow (Stop \triangleright^d P))$ which performs $a$ and then behaves as $P$ after a delay of $d$ time units.

Semantically, processes in Timed Csp perform *timed events* $(r, e) \in \mathbb{R}_{\geq 0} \times \Sigma$ : $r$ is the time at which event $e$ occurs. Events are instantaneous, i.e., they do not take any time. The execution of a system leads to a *timed trace*. We write $\langle\rangle$ for the empty trace and $t = \langle(r_1, e_1), \ldots, (r_n, e_n)\rangle$ for a finite observation with $\forall\, j > i \geq 1 : r_i \leq r_j$ and $\forall\, n > i \geq 1 : e_i \neq \checkmark$. Given a non-empty timed trace $t$, the time stamp of its first visible event is given by $begintime(t) = r_1$; that of its last visible event is given by $endtime(t) = r_n$; and its duration is given by $duration(t) = endtime(t) - begintime(t)$. We define $duration(\langle\rangle) = 0$. $\#t$ denotes the number of timed events occurring within a timed trace $t$. Given a set of events $A$, $t \upharpoonright A$ denotes the projection of $t$ onto $A$, i.e., the subsequence of timed events from $t$ which consists only of events from $A$. Using these notations, $t \downarrow A = \#(t \upharpoonright A)$ is the number of timed events from $A$ in $t$. Given two timed traces $t_1$ and $t_2$, $t_1 \frown t_2$ denotes their concatenation; if the time stamps do not match, this concatenation is undefined.

We denote the set of all timed traces by $TT$, and write $\mathcal{T}_{\mathbb{R}}[\![P]\!] \subseteq TT$ for the set of all timed traces of a Timed-CSP process $P$. Given two Timed Csp processes $IMP$ and $SPEC$, we say that $SPEC$ refines to $IMP$, denoted by $SPEC \sqsubseteq_{TT} IMP$, iff $\mathcal{T}_{\mathbb{R}}[\![SPEC]\!] \supseteq \mathcal{T}_{\mathbb{R}}[\![IMP]\!]$.

For the case that $SPEC$ is independent of time, i.e., $SPEC$ does not include any timed operator, and $IMP$ is an integer-wait Timed Csp process, Roscoe [18] provides the following proof technique: $SPEC \sqsubseteq_{TT} IMP$ over Timed Csp is equivalent to $time(SPEC) \sqsubseteq_T time(IMP)$ over (untimed) Csp. The latter proof obligation can be discharged using standard CSP tools such as the model checker FDR [1]. The function $time$ adds a special event called $tock$ to the alphabet of the processes in order to indicate the passage of time, e.g., $time(Wait\ 0) = SKIP$ and $time(Wait\ (n+1)) = tock \rightarrow time(Wait\ n)$. For the external choice operator, we use Schneider's construction [19]: $time((?x : A \rightarrow P) \,\Box\, (?y : B \rightarrow Q)) = time(?x : A \rightarrow P) \,\Box\, time(?y : B \rightarrow Q) \,\Box\, tock \rightarrow time((?x : A \rightarrow P) \,\Box\, (?y : B \rightarrow Q))$ which provides a correct translation if $A \cap B = \emptyset$; this can be seen, e.g., by comparison with the semantics of $\Box_{tock}$ (see [19]). The automatic verification of real-time systems with FDR is competitive with other approaches [15].

We use the timed refusal trace semantics of Timed Csp as defined in [17], which guarantees a semantics for recursion even if the processes involved fail to be timed-guarded (see [19]). The timed traces of a process $P$ can be extracted straightforwardly from the timed refusal traces of $P$ [17].

## 5 Modelling Timed Behaviours of Railway System

In the following, we make two assumptions concerning time in railways. Firstly, we assume signalling to be instantaneous. In the real world, the cycle time of an interlocking is in the region of two seconds. This time is (nearly) negligible compared to the time a train needs to move from one track to another, so for the current study we disregard this delay rather than require the interlocking to await confirmation that the signal has been changed. Slightly more critical is our second assumption; for this study we assume that trains accelerate and brake immediately. The consequence is that we overestimate capacity (as trains move faster than in reality). The second assumption is clearly an over simplification to be remedied in future work.

In order to model time, we enrich Winter's model [21] of a track plan. To this end, we record the time it takes to travel a distance $l$ at a speed limit $s$ in a table $delay(l, s)$. The second change to the untimed model is that trains get one more parameter: besides their identity *id* and the position *front*, *rearmoves*, they also have the speed allowed on track *front*. The following Timed Csp code summarizes the essential part of these changes; all other processes remain as described in Section 3.

```
TrainBehave(id,front,rearmoves,curspeed) =
  if (front==Exit and null(rearmoves))
  then TrainBehave(id,entry(id),<>,0)
  else ([] n1 : next(front) @
    moveff.front.n1?speed ->
    if (tracklength(n1)>trainlength(id))
      then Wait(delay(trainlength(id),speed))
      else Wait(delay(tracklength(n1),speed));
    RearBehaves(tracklength(n1),id,n1,
                rearmoves^<(moverr.front.n1,trainlength(id))>),speed)
```

## 6 Modelling Railway Capacity

We now develop a semantic concept of capacity based on timed traces, and characterise a railway's capacity via time-wise refinement in Timed Csp.

### 6.1 Capacity Semantically

In this section, we present a formal definition of railway capacity which is compliant with the quotation given in Section 2 and compatible with existing analytical methods, for example [3]. Informally speaking, we want to count the number of trains appearing and operating within the railway. This number depends on two parameters: namely, (i) when we start counting and (ii) how long we observe. Thus, we speak of an *observation window* characterised by a starting time and a duration. There are two kinds of trains that we can observe in such a window: those trains which are already present at the starting time of the window, and those trains, which appear in the window while it is open.

Initially, we assume that there are no trains in the railway. As trains enter, travel through and leave a railway, their movements are recorded in a timed trace. Relative to a given track plan, we define *Entering* and *Leaving* as the sets of timed events which indicate the entering and leaving of trains, respectively. In our example, $moveff.Entry1.AA$ is an element of *Entering*, the set *Leaving* includes, e.g., the element $moverr.AE.Exit$.

Let $s$ be a timed trace of a railway model. The number of trains in the railway after $s$ is given by the number of trains entering the railway reduced by the number of trains leaving the railway:

$$storage(s) \;=\; s \downarrow Entering - s \downarrow Leaving$$

The number of trains entering the railway during $s$ is given by

$$increase(s) \;=\; s \downarrow Entering$$

Relative to the duration $\delta$ of an observation window, we define the capacity of a Train System $TS$ by

$$capacity(TS, \delta) \;=\; \max \left\{ \begin{array}{l} storage(s_1) + increase(s_2) \;: \\ s_1 \frown s_2 \in \mathcal{T}_{\mathbb{R}}[\![TS]\!] \text{ and } duration(s_2) \leq \delta \end{array} \right\}.$$

Each decomposition $s_1 \frown s_2 \in \mathcal{T}_{\mathbb{R}}[\![TS]\!]$ of a timed trace yields a value to be considered for capacity. We determine how many trains are in the system after the set-up phase $s_1$ and how many trains enter the system during the observation window $s_2$, and maximise the sum $storage(s_1) + increase(s_2)$ over all timed traces $s_1 \frown s_2 \in \mathcal{T}_{\mathbb{R}}[\![TS]\!]$ in which $duration(s_2) \leq \delta$. The following result shows that this definition of capacity nicely fits with refinement:

**Theorem 1 (Capacity and Refinement).** *If* $TS_2 \sqsubseteq_{TT} TS_1$ *then* $\forall\, \delta \geq 0 :$ $capacity(TS_1, \delta) \leq capacity(TS_1, \delta)$.

For our purposes the decomposition of a timed trace into a set-up phase and an observation window gives a good insight into a railway system (see especially the paragraph on simulation later in Section 7). However, we note in passing that a notion of capacity for a Train System $TS$ which is independent of observation duration, giving a long-term rate of "trains per unit time", could be defined by

$$\lim_{\delta \to \infty} \frac{capacity(TS, \delta)}{\delta}.$$

### 6.2 Capturing Storage and Increase in Timed Csp

In this section, we provide a construction in Timed Csp which turns capacity into an observable event. To this end, we run the process *TrainSystem* in a two-layered environment. The first layer consists of an observer process, while the second layer controls the whole set-up. The observer process synchronises

with *TrainSystem* over events indicating the entering and leaving of trains with respect to the railway. The controller process synchronises with the observer process on the duration of the observation window and the observed capacity. The observer process works in two phases: the process *Storage* (see below) realises the function *storage* (see above); after a *startObs* signal from the control layer, control goes over to the second phase in which the process *Increase* (see below) realises the function *increase* (see above).

The process *Storage* counts the entering trains and reduces this number by one for every leaving train:

```
Storage(n) = ([] n1 : next(Entry) @ moveff.Entry.n1?_  -> Storage(n+1))
          [] ([] n1 : pre(Exit)   @ moverr.n1.Exit  -> Storage(n-1))
          [] startObs?delta -> Increase(n,0,delta)
```

In addition, the process listens on the channel *startObs*. When it receives a value *delta*, it passes control to the process $Increase(n, 0, delta)$. Here, $n$ is the number of trains which are on the railway already, 0 is the duration since the observation started, and *delta* is the size of the observation windows.

The process *Increase* counts the entering trains as long as the observation window is open. When the window is closed, it informs the controller on the channel *infocap* on the observed capacity. After this, it goes to an idle state *EndCapObserver*. The process *Increase* is defined as follows:

```
Increase(n,d,delta) =
d<=delta & ([] n1 : next(Entry) @  moveff.Entry.n1?_ @ u ->
                if d+u<=delta then Increase(n+1,d+u,delta)
                              else Infocap(n))
         [] ([] n1 : pre(Exit) @  moverr.n1.Exit @ u ->
                if d+u<=delta then Increase(n,d+u,delta)
                              else Infocap(n))
Infocap(n) =  infocap.n -> EndCapObserver
         [] ([] n1 : next(Entry) @  moveff.Entry.n1?_ -> Infocap(n))
         [] ([] n1 : pre(Exit) @  moverr.n1.Exit ->  Infocap(n))
EndCapObserver = ([] n1:next(Entry)@moveff.Entry.n1?_ -> EndCapObserver)
              [] ([] n1:pre(Exit)@moverr.n1.Exit ->  EndCapObserver)
```

The process *Controller* decides when the observation window starts, and later receives the value of the observed capacity through the channel *infocap*. This process is defined as follows:

```
Controller(delta) = startObs.delta -> infocap?n -> Stop
```

The overall set-up is given by the process *TrainSystemWithCapacity*:

```
TrainSystemWithCapacity =
    (TrainSystem
     [|union(
       { moveff.Entry.n._ | n <- next(Entry) },
       { moverr.n.Exit | n <- pre(Exit) })|]
     Storage(0)) [| {|startObs, infocap|} |] Controller(delta)
```

We define that a process $Q$ does not block a process $P$ with alphabet $\Sigma_P$ over a synchronization set $X$ if $(\mathcal{T}_{\mathbb{R}}[\![P]\!] = \mathcal{T}_{\mathbb{R}}[\![P \,|[\, X \,]|\, Q]\!]) \upharpoonright \Sigma_P$. We establish the following result for the coupling of *TrainSystem* and *Storage* in the definition of the process *TrainSystemWithCapacity*:

**Theorem 2.** *Storage*$(0)$ *does not block TrainSystem.*

*Proof (Sketch).* In *Storage*$(0)$, every event in $\{moveff.Entry.n.\_ \mid n \in next(Entry)\} \cup \{moverr.n.Exit \mid n \in pre(Exit)\}$ is always ready to engage.

This insight provides the following result.

**Theorem 3.** *The following are equivalent:*

- $capacity(TrainSystem, \delta) = n$.
- $n' \leq n$ *iff there exists a timed trace* $t \in \mathcal{T}_{\mathbb{R}}[\![TrainSystemWithCapacity]\!]$ *such that* $(r, infocap.n') \in t$ *for some* $r \in \mathbb{R}$.

*Proof (Sketch).* The following two correspondences hold between the timed traces of the process *TrainSystem* and the process *TrainSystemWithCapacity*:

- If $t_1 = s_1 \frown s_2 \in \mathcal{T}_{\mathbb{R}}[\![TrainSystem]\!]$ then $t_1' = s_1 \frown \langle (begintime(s_2), startObs.\delta) \rangle \frown s_2 \frown \langle (endtime(s_2), infocap.n) \rangle \in \mathcal{T}_{\mathbb{R}}[\![TrainSystemWithCapacity]\!]$.
- If $t_2 = s_1 \frown \langle (r_1, startObs.\delta) \rangle \frown s_2 \frown s_3 \frown \langle (r, infocap.n) \rangle \in \mathcal{T}_{\mathbb{R}}[\![TrainSystemWithCapacity]\!]$ then $t_2' = s_1 \frown s_2 \in \mathcal{T}_{\mathbb{R}}[\![TrainSystem]\!]$.

### 6.3 Capacity via Refinement

We formulate a process which allows at most $n$ trains operating within an observation window of duration *delta*. Here, we use only events of the interface between the observer process and the controller process:

```
CapacityFrom(n,delta)=|~|n':{0..n}@ startObs.delta -> infocap.n' -> Stop
```

With regards to this process, we have the following result:

**Theorem 4.** *Given a length* $\delta$ *of observation,* $capacity(TrainSystem, \delta) = n$ *iff*

- *for all* $k \geq n$ *holds:* $CapacityFrom(k, \delta) \sqsubseteq_{TT} TrainSystemWithCapacity \setminus MoveEvents$, *and*
- *for all* $0 \leq l < n$ *holds:* $CapacityFrom(l, \delta) \not\sqsubseteq_{TT} TrainSystemWithCapacity \setminus MoveEvents$,

*where* $MoveEvents = \{moveff.x.y.\_,\ moverr.x.y \mid x, y \in Tracks\}$

*Proof (Sketch).* By Theorem 3 and the definition of $CapacityFrom(n, \delta)$.

## 7 Studying Safety and Capacity in the Context of the Double Junction

In this section, we study *safety and capacity in one model* formulated in Timed Csp. To this end, we consider each of the paths $\overrightarrow{AB}$, $\overrightarrow{DC}$, $\overrightarrow{AE}$ and $\overrightarrow{FC}$ shown in Figure 1 in isolation. It remains future work to study the double junction as a whole. For each of the four paths, we encode Scenarios 1 and 2 from Section 2. It turns out that both scenarios are safe and that capacity increases when signalling is changed from Scenario 1 to Scenario 2.

For the model of each path, we determine the minimal amounts of time a train travels from one end to the other of each track. Here, we use data suggested by our industrial partner Invensys Rail about the lengths of trains and tracks. We take the length of trains to be 200m, the length of tracks where there is either a point or a diamond crossing to be 50m, the length of overlap tracks to be 200m, the length of other tracks to be 1500m, and the track lengths on path $\overrightarrow{AE}$ to be summarised in the following table:

| Track | AA | AB | AC | BW | CM | CL |
|---|---|---|---|---|---|---|
| Length | 1500m | 200m | 50m | 50m | 1500m | 200m |

The minimal amounts of time to travel such distances in different speed limits can be easily calculated. For example, it takes at least $4s$ for a train to travel on $AA$ at a speed of 120mph. These constants are incorporated into the Timed Csp models as presented in Section 5. These result in processes $\mathit{TrainSystem}_{p,s}$, where $p$ ranges over $\{\overrightarrow{AB}, \overrightarrow{AE}, \overrightarrow{DC}, \overrightarrow{FC}\}$ and $s \in \{1, 2\}$.

These models are collision-free iff $\mathit{NoCollision} \sqsubseteq_{TT} \mathit{TrainSystem}_{p,s}$. Since the processes $\mathit{TrainSystem}_{p,s}$ contain only integer-wait operators and *NoCollision* does not include any timed operator, we utilise FDR to prove the refinements $\mathit{time}(\mathit{NoCollision}) \sqsubseteq_{T} \mathit{time}(\mathit{TrainSystem}_{p,s})$. FDR shows that all these refinements hold. Thus, all paths are safe in both scenarios.

In order to deal with capacity, we simulate both scenarios with our Timed Csp Simulator tool [8]. This is possible as the processes $\mathit{TrainSystem}_{p,s}$ involve only rational numbers for time. To this end, we apply the *automatic simulation* available in the Timed Csp Simulator, which randomly chooses between events and prioritises events over the evolution of time. Simulating $\mathit{TrainSystem}_{p,s}$ in the Timed Csp Simulator yields one of its timed traces.

We determine capacity in a three step process. First, we make estimates on the length of the set-up phase and on the minimal length $\delta$ of an observation window. We choose these numbers in such a way that we can expect a difference in the capacities of Scenarios 1 and 2. Next, we validate this estimation. Both these steps are based on simulation with the Timed Csp Simulator. Finally, given a good estimate for the length $\delta$, Theorem 4 allows us to determine $\mathit{capacity}(\mathit{TrainSystem}_{p,s}, \delta)$ for each $\mathit{TrainSystem}_{p,s}$. Here, we discharge the involved proof obligations with FDR. This is possible as the process $\mathit{CapacityFrom}(n)$ does not have any timed operator and there are only integer-waits in the process $\mathit{TrainSystemWithCapacity}$.

*Step 1:* Simulation with the Timed Csp Simulator suggests that it takes a fixed time $\mu$ from one train entering $TrainSystem_{p,s}$ until this train leaves. Furthermore, it suggests that it takes a fixed time $d$ from one train entering $TrainSystem_{p,s}$ to the next train entering $TrainSystem_{p,s}$. Let $\mu_1$, $d_1$ and $\mu_2$, $d_2$ be the estimates from the simulation of Scenarios 1 and 2, respectively. For the length of the observation window we select $\delta = d_1 x$ for some $x$ such that $d_1 x = d_2(x+1)$. For total run-times we choose $\mu_1 + \delta$ ($\mu_2 + \delta$) for Scenario 1 (Scenario 2). This "guarantees" (based on the simulation data) that the two scenarios show different capacities. For the path $\overrightarrow{AE}$, we obtain $\mu_1 = 113s$, $\mu_2 = 105s$, $d_1 = 85s$ and $d_2 = 70s$. Thus, we choose $\delta = 397s$ and simulate Scenario 1 for $\mu_1 + \delta = 510s$ and Scenario 2 for $\mu_2 + \delta = 502s$.

*Step 2:* Automatic simulation of Scenario 1 for $\mu_1 + \delta$ and of Scenario 2 for $\mu_2 + \delta$ yields two timed traces. The capacity observed on these timed traces gives lower bounds for the capacity of $TrainSystem_{p,s}$. For path $\overrightarrow{AE}$, we obtain a capacity of 7 in Scenario 1 and a capacity of 8 in Scenario 2.

*Step 3:* Finally, we verify with FDR that these numbers are indeed the capacity. For path $\overrightarrow{AE}$, we obtain:

| $\overrightarrow{AE}$ | 6 | 7 | 8 | Running time in FDR |
|---|---|---|---|---|
| Scenario 1 | x $(1,12 \times 10^6)$ | ✓ $(1,14 \times 10^6)$ | - | 25s |
| Scenario 2 | - | x $(1,67 \times 10^6)$ | ✓ $(1,73 \times 10^6)$ | 33s |

Each row in the table provides the result for one scenario. Column $n$ expresses if the refinement $CapacityFrom(n) \sqsubseteq_{TT} TrainSystemWithCapacity \setminus MoveEvents$ holds. "✓" stands for successful verification, "x" indicates that the refinement does not hold, "-" says that this check was not carried out. We associate the round number of states that are checked by FDR in each refinement. The last column shows how long FDR spends for running all checks needed for determining capacity in each scenario[1]. The results for the other paths are obtained in the same way as for $\overrightarrow{AE}$. We summarise these in the following table:

| Path | Window length | Capacity in Scenario 1 | Capacity in Scenario 2 |
|---|---|---|---|
| $\overrightarrow{AB}$ | 379s | 12 | 13 |
| $\overrightarrow{DC}$ | 399s | 12 | 13 |
| $\overrightarrow{FC}$ | 328s | 7 | 8 |

Interpreting our results within the railway domain, we can state: under optimal conditions, we expect one more train approximately every 6.5 minutes in Scenario 2 compared with Scenario 1 without compromising safety. It takes about 2 minutes of set-up time to observe this difference.

For windows of length larger than 6.5 minutes, $TrainSystem_{p,2}$ has at least the capacity of $TrainSystem_{p,1}$. This holds by Theorem 1. We observe that $TrainSystem_{p,1}$ and $TrainSystem_{p,2}$ are identical but for the value $x$ in the

[1] On a machine with a 2GHz 64 bit processor with 4GByte memory running Mac OS.

*Wait* $x$ processes involved. Here, $x' \leq x$ for the corresponding *Wait* processes, where $x$ is the value in $\mathit{TrainSystem}_{p,1}$ and $x'$ is the value in $\mathit{TrainSystem}_{p,2}$. Thus, any delay $y$ between two timed events of a timed trace of $\mathit{TrainSystem}_{p,1}$ can be reproduced in $\mathit{TrainSystem}_{p,2}$ as for the corresponding *Wait* $x'$, we have $x' \leq x \leq y$.

Lessons learnt from using tools for CSP and Timed CSP for our performance analysis in the railway domain include:

1. Obviously, one would like to study safety for the whole junction and not, as we do above, for single paths in isolation. However, it turns out that handling the complete junction (even in the untimed case) is beyond the proof support given by FDR, at least for our current modelling approach.
2. Reducing proof obligations over Timed CSP to proof obligations over (untimed) CSP works well. Tool support for the translation (which we carry out manually) would be welcome.
3. Concerning capacity, it might be possible to determine it by "optimal" simulation in the Timed CSP Simulator. To this end, one would have to argue which simulation strategy leads to a timed trace showing the capacity of the railway system.
4. Proper time-wise refinement, as in $\mathit{TrainSystem}_{p,2} \sqsubseteq_{TT} \mathit{TrainSystem}_{p,1}$, still lacks convincing tool support. On our examples, the PAT system [2] was running out of memory for the refinement checks.

## 8 Summary and Future Work

In close cooperation with railway industry, we have provided a formal definition of line capacity based on the timed traces that one can observe in a natural, timed model of railway systems. This definition can equivalently be characterized as a refinement statement in Timed CSP. By adapting the safety formulation of Winter [21], we are able to study both safety and capacity in one formal model in Timed CSP. As the refinements for safety and capacity only require the checking of qualitative properties, both refinement statements can be discharged by translation into untimed CSP. This approach has the advantage that one can re-use established tool support for CSP alone.

To illustrate our approach, we have applied it to a standard double junction from the (UK) railway domain. For this junction, we can answer fundamental questions from railway industry: changing control tables in the way suggested by railway engineers yields a capacity increase without compromising safety. This increase can be quantified: under optimal conditions, after the change there can be one more train every six minutes in our example. Having shown the increases that can be gained via changing signalling rules through the trusted use of ATP, this encourages changes to be proposed to the current UK railway regulation.

The double junction example demonstrates the limitations of the current proof support in terms of the model checker FDR. For complex examples, e.g., with more tracks, 3-aspect or 4-aspect signalling, long or different-length delays, the translational approach is inefficient. Dedicated proof support, e.g., in the form of a Timed CSP-Prover (currently under construction) is necessary.

It remains future work to include further timing aspects into our modelling, such as the cycle time of signalling and point movements or braking and accelerating curves of trains. Finally, we intend to develop our definition further, so that it also captures the more complex notion of network capacity.

**Acknowledgement.** The authors would like to thank Erwin R. Catesbeiana (Jr) for pointing out that immobility is the enemy of capacity.

## References

1. FDR2, `http://www.fsel.com/software.html`
2. PAT, `http://www.comp.nus.edu.sg/~pat/`
3. UIC Leaflet 405 OR. Links between Railway Infrastructure Capacity and the Quality of Operations. International Union of Railways (1996)
4. UIC Leaflet 406. Capacity. International Union of Railways (2004)
5. Abril, M., Barber, F., Ingolotti, L., Salido, M., Tormos, P., Lova, A.: An assessment of railway capacity. Transportation Research Part E: Logistics and Transportation Review 44(5), 774–806 (2008)
6. Barber, F., Abril, M., Salido, M., Ingolotti, L., Tormos, P., Lova, A.: Survey of automated systems for railway management. Technical Report. TU Valencia (2007)
7. Burdett, R.L., Kozan, E.: Techniques for absolute capacity determination in railways. Transportation Research Part B: Methodological 40(8), 616–632 (2006)
8. Dragon, M., Gimblett, A., Roggenbach, M.: A Simulator for Timed CSP. In: AVoCS 2011. Technical Report. Newcastle University (2011)
9. Department of Transport. Delivering a Sustainable Railway. White Paper CM 7176 (2007)
10. Fokkink, W., Hollingshead, P.: Verification of interlockings: from control tables to ladder logic diagrams. In: Proceedings of FMICS 1998, pp. 171–185 (1998)
11. Hoare, T.: Communicating Sequential Processes. Prentice Hall (1985)
12. Isobe, Y., Nguyen, H.N., Roggenbach, M.: Towards safe capacity in the railway domain – an experiment in Timed-CSP. In: DSW 2011 (2011)
13. James, P., Roggenbach, M.: Automatically Verifying Railway Interlockings using SAT-based Model Checking. In: AVoCS 2010. EASST (2011)
14. Kerr, D., Rowbotham, T.: Introduction To Railway Signalling. Institution of Railway Signal Engineers (2001)
15. Khattri, M., Ouaknine, J., Roscoe, A.: Automated translation of timed automata to Tock-CSP. In: AVoCS 2010. Technical Report. Düsseldorf University (2010)
16. Landex, A., Kaas, A., Schittenhelm, B., Schneider-Tilli, J.: Practical use of the UIC 406 capacity leaflet by including timetable tools in the investigations. In: Proceedings of the 10th International Conference on Computers in Railways (2006)
17. Ouaknine, J., Worrell, J.: Timed CSP = closed timed $\varepsilon$-automata. Nordic Journal of Computing 10, 1–35 (2003)
18. Roscoe, B.: Understanding Concurrent Systems. Springer (2010)
19. Schneider, S.: Concurrent and Real-time systems. Wiley (2000)
20. Simpson, A., Woodcock, J., Davies, J.: The mechanical verification of solid-state interlocking geographic data. In: Formal Methods Pacific 1997. Springer (1997)
21. Winter, K.: Model checking railway interlocking systems. Australian Computer Science Communications 24(1) (2002)