SE752 Assignment 1
The Design of an Integrated Automatic Train Control System

Ernest Wong
ewon746@aucklanduni.ac.nz

September 2020

Contents

1 Introduction **3**

2 Stakeholder Requirements **3**

3 Responsibilities of the Train Control System **4**

4 System Structure and Modelling Approaches **5**

5 Rail Network Modelling and Interlocks **8**
5.1 Train Tracks 8
5.2 Drivable Routes 9
5.3 Background on Block Sections 10
5.3.1 Signal Modelling 11
5.3.2 Route Setting 11
5.3.3 Interlockings 11
5.4 Modelling Block Sections 11
5.4.1 Construction Constraints for Block Sections 12
5.4.2 Interlocking Rules for Block Sections 14
5.4.3 “Active” Block Sections 14
5.5 Inactive Block Section Management 15
5.5.1 Installing and Removing Tracks 15
5.5.2 Registering New Switches 16
5.5.3 Registering New Crossings 17
5.5.4 Removing Switches and Crossings 18
5.5.5 Registering and Unregistering Routes 18
5.6 Active Block Section Management 19
5.6.1 Setting a Route 20
5.6.2 Cancelling a Route and Cancelling All Routes 20

6 Train Control System – The Big Picture **21**

7 System Properties **22**
7.1 Safety Properties 22
7.2 Operational Properties 22

1 Introduction

Modern-day train transport is attractive because it's safe, fast, and congestion-free. This is only possible, however, due to the introduction of train control automation. Since trains are heavy, taking a long time to stop, and travel too fast to spot incoming danger to have enough time to react, train drivers have to rely on control systems that exist beyond the train itself to ensure that trains do not collide.

Over many years, train control systems continued to evolve taking on more and more responsibilities. One of the earliest control systems feature a watch (known as railroad chronometers), and that system relied the drivers to time their trains correctly to avoid colliding with other trains. Nowadays, digital train control systems has taken over the operation of track switching equipment, signalling equipment, timetabling, and even the driving of trains with the introduction to driverless train technology.

In this report, we will be presenting a high-level, verifiable design of a train control system in the form of a Z-model. Note that there is a focus on the signalling aspects of the system in this report. We will first begin by taking a brief analysis of the stakeholders involved with this hypothetical system, followed by a more detailed enumeration of the use-cases of this system. Before the design is presented, some modelling decisions and approaches are presented to give the reader a sense of what to expect. Finally, we conclude by discussing some properties of the system that we can verify.

It should be noted that this high-level design should be refined to a more concrete specification that uses mathematical formulations that more closely aligns with programming constructs and data structures. After creating mappings from this lower-level Z-model to the high-level design specified in this report, it is possible to verify that both models are equivalent making it less likely that some implementation detail will invalidate the design.

2 Stakeholder Requirements

To obtain a good coverage and a good balance of system requirements, we have identified several key stakeholders and users of this system who are involved from rail network planning operations to day-to-day operations.

1. The **train driver** will primarily be following the signals provided by our system to operate the train. They need to trust that the signals provided by the system are safe.

2. The **signaller**, traffic manager and train dispatchers working in a train control centre will be using this system to set the right signals to help the train get to the right destination safely and quickly. Oftentimes, the signaller will also need to help regulate the traffic, planning detours in the wake of incidents or congestion. Having to react quickly to unplanned situations means that

it should be easy to visualise the system state, and that it should be easy to undertake corrective actions.

3. The **maintenance crew** for the network will be using this system to monitor train traffic for safety, monitoring the network behaviour for diagnostics, and closing off sections to allow for maintenance.

4. **Railway engineers** will be using this system to plan the next infrastructure upgrades, rail network extensions, and taking new sections online into the rest of the network once the rail section is ready.

3 Responsibilities of the Train Control System

From the stakeholders, we can summarise what users expect from the system and what is in-scope for the system to manage in terms of the following four responsibilities:

1. **Rail Network Modelling.** The system should be able to store, retrieve, visualise, and use information regarding the track topology, railroad switches, signalling equipment states, platforms, stations, and train positions. New equipment and tracks can be installed and removed. Signallers can see the rail network state through a large "mimic panel" – a large display with the rail schematics and rail state overlaid on it.

2. **Interlocking.** The system should prevent unsafe combinations of switch states and signal states from happening. This is done by modelling block sections (areas where only one train is allowed) and crossover assemblies (combinations of switches that need to be operated together). Each block section can have a route set, and the interlocking decodes the route setting into signal and switch state changes. More information regarding this interlocking responsibility will be discussed in subsequent sections.

3. **Automatic Train Protection (ATP).** The system should prevent trains from passing a red signal (also known as Signal Passed At Danger, SPAD) by automatically applying brakes before it is too late.

4. **Automatic Train Operation (ATO).** The system should automatically drive and brake trains.

The following aspects were also initially planned, but these have now been removed from the system's scope for now.

1. **Automatic Route Setting (ARS).** The system should facilitate the automatic setting of routes for each block section and populate the berths with the train's future headcode when given a timetable for all trains.

2. **Timetable Management.** The system should allow timetables to be planned for each train, and be able to derive the practical timetables that ARS should follow given train delays.

3. **Train Management.** The system should facilitate the management of train operation, train doors, driver displays such as train timetables, in-cab signals, and in-train displays such as message boards and real-time map with location indication.

4. **Station Management.** The system should facilitate the management of train stations, including its platform doors, platform timetables, and station timetables.

4 System Structure and Modelling Approaches

This section is to serve as a guide to understanding the Z-model's structure, as well as to provide a quick showcase of some decisions and modelling idioms used in this design specification.

This train control system is complex enough to warrant some need to modularise it's structure. Its many states and constraints are covered by the 85 paragraphs in Z/EVES. A specification pattern known as "promotion" or "framing" is employed in this system, where the system are split into smaller subsystems, and the system operations that modify these subsystems rely on using the subsystem's operations and a system-wide "promotion" schema that "promotes" it into a system-wide operation.

There are 2 major subsystems that have their own subsystem operations: *ActiveBlocks* and *InactiveBlocks*. These represent block sections in the railway network that are either in-use ("active") for signalling, or deactivated for maintenance. The system schema *TrainControlSystem* delegates block operations to those respective subsystems using the $Update_{A}ctiveBlockSection$ and $Update_{I}nactiveBlockSection$ schemas, which some users of Z-notation refer to as the promotion schemas. The available operations for each subsystem, and for the overall system, are documented in Tables 1, 2, and 3.

Some state schemas require a lot of constraints, such as for an active *BlockSection*. To document these constraints with readability in mind, these constraints are divided into separate schemas according to their real-life meanings.

To further simplify the predicates, new relations and functions are axiomatically defined wherever appropriate so that the predicates for the subsequent schemas are more readable. For instance, the mathematical formulas needed to search the right route when given the user's input coordinates have been abstracted away into the *RouteMatches* relation. This allows us to retrieve the right route by using

$$\exists r : \mathrm{dom}(\mathrm{dom}\, signals \lhd routeMatches \rhd \{(entry?, exit?)\})$$

Table 1: Operations for the *InactiveBlockSection* subsystem. These are not exposed to the user, but are relied upon by the system.

Operation	Description
InactiveBlockSection_InstallTrack	• Adds a new track to the block section.
InactiveBlockSection_RemoveTrack	• Removes a track given by the coordinates from the block section.
InactiveBlockSection_RegisterSwitch	• Identifies a collection of intersecting tracks as part of a railway switch.
InactiveBlockSection_RegisterCrossing	• Identifies a collection of intersecting tracks as part of a crossing.
InactiveBlockSection_UnregisterJunction	• Removes any associated switch or crossing that intersects a certain coordinate.
InactiveBlockSection_RegisterRoute	• Marks an entry-exit coordinate pair as available to be routed through this section.
InactiveBlockSection_UnregisterRoute	• Removes any route that was previously registered with this entry-exit coordinate pair.

Table 2: Operations for the *ActiveBlockSection* subsystem. These are not exposed to the user, but are relied upon by the system.

Operation	Description
ActiveBlockSection_SetRoute	• Reconfigure the track equipment to allow for trains to pass through this section via this route, defined by an entry-exit coordinate pair.
ActiveBlockSection_CancelRoute	• Cancels a previously set route using an entry-exit coordinate pair.
ActiveBlockSection_CancelAllRoutes	• Cancels all routes currently set in the block section.

Table 3: Operations for the *TrainControlSystem*.

Operation	Description
InstallNewBlockSection	• Adds a new, empty, inactive block section. The block ID is returned.
InstallNewLevelCrossing	• Adds a new road-track crossing with an automated barrier, specified by the block IDs that this crossing passes through.
InstallTrack	• Adds a new track to a given inactive block, given that this track does not already exist on the network.
RemoveTrack	• Removes a track from a given inactive block.
RegisterSwitch	• Adds a switch to a given inactive block.
RegisterCrossing	• Adds a crossing to a given inactive block.
UnregisterJunction	• Removes a switch or crossing from a given inactive block.
SetRoute	• Sets a route to a given active block.
CancelRoute	• Unsets a route from a given active block.
CancelAllRoutesForBlock	• Clears all routes from a given active block.
ActivateBlockSection	• Activates an inactive block secoin if it is ready.
DeactivateBlockSection	• If no train is present on the block, it cancels all routes from the block and deactivates the block.
BuildNewStation	• Adds a new stations by specifying the blocks that designate each platform.
RegisterNewLine	• Adds a new train line by specifying the sequence of stations that the line visits.
IntroduceNewTrain	• Add a new train to the network to a given track that is long enough.
RefreshAutoTrainOperation	• Updates the speed of all trains according to whether there is a blocked signal in front of the train or not.
DisplayMimicPanel	• Outputs the positions and states of tracks, signals, stations, and level crossings so it could be displayed on a large mimic panel in control centres.
ReportTrainLocation	• This is a system interface that allows track equipment to report where the train's location currently is.

without having to extract the entry and exit locations from the head and the tail of the route's location path sequence.

Now our model structure and approach has been described, we will now proceed to present the system's specification using a bottom-up approach, starting from the fine-details of train tracks and rail topology, and finishing with the overall system and its properties.

5 Rail Network Modelling and Interlocks

In this section, we will describe the requirements for the subsystem that manages the rail network infrastructure and signalling. This portion of the subsystem allows users to model their rail network and to safely control the signals and switches that guide the trains to their destinations. We will be going through some background on track topology, followed by an introduction to a modular concept called "block sections". Finally, we present how our subsystem can be used to manage these block sections, and therefore, control the rail network.

5.1 Train Tracks

The rail network is an undirected graph of *train tracks* that connect different places together. Tracks have no intrinsic directionality, so edges in this graph are best represented as an unordered pair such as a set of length 2.

$$
\begin{array}{|l}
\hline
Edge[V] \\
endpoints : \mathbb{F}\, V \\
\hline
\#endpoints = 2 \\
\hline
\end{array}
$$

Additional information about the track, such as whether it is compatible with electric trains, are stored alongside in the *Track* schema.

$$Location == \mathbb{Z} \times \mathbb{Z}$$

$$TrainModel ::= AM \mid ADK \mid ADC \mid DC \mid DX \mid DA$$

$$
\begin{array}{|l}
\hline
TrainTrack \\
Edge[Location] \\
compatibility : \mathbb{F}\, TrainModel \\
length : \mathbb{N}_1 \\
maxSpeed : \mathbb{N} \\
\hline
\end{array}
$$

For the sake of clarity, we define the following functions to help express the design of subsequent modules of the system. The partial function *getTrack* finds the unique track that is identified by the two locations they connect.

$$getTrack : \mathbb{F}\, TrainTrack \times \mathbb{F}\, Location \nrightarrow TrainTrack$$

$$\begin{aligned} &getTrack \\ &\quad = (\lambda\, ts : \mathbb{F}\, TrainTrack;\ e : \mathbb{F}\, Location \\ &\qquad \mid \#e = 2 \land \#\{t : ts \mid t.endpoints = e\} = 1 \\ &\qquad \bullet\ (\mu\, t : TrainTrack \mid t.endpoints = e)) \end{aligned}$$

Meanwhile, the *getTrackEndpoints* function finds the set of all track's endpoints given a set of tracks.

$$getTrackEndpoints : \mathbb{F}\, TrainTrack \to \mathbb{F}(\mathbb{F}\, Location)$$

$$\begin{aligned} &getTrackEndpoints \\ &\quad = (\lambda\, ts : \mathbb{F}\, TrainTrack \bullet \{e : \mathbb{F}\, Location \mid \exists\, t : ts \bullet e = t.endpoints\}) \end{aligned}$$

5.2 Drivable Routes

The system also needs to model the paths in which a train can travel through to assist with the signalling and routing responsibilities of the control system. We denote these paths as *routes*. Trains are heavy vehicles that are difficult to stop and reverse directions, so routes should not require the train from reversing.

Moreover, not all train tracks are connected equally. A railroad switch contains one track that diverges into several tracks (usually two). The one track on the converging side can continue to any of the tracks on the diverging side, and tracks on the diverging end of the switch can continue to the track on the converging side. However, tracks on the diverging side do not continue to each other, so trains cannot travel from one diverging track to another without reversing from the converging track.

Therefore, we introduce two graphs to model the railway: one graph models the spatial continuity of the tracks by using tracks as edges of the graph, and the other graph models the track continuity by using the tracks as vertices of this graph. A route is therefore a sequence of tracks that forms a path on both graphs.

We denote the edges between tracks as *TrackContinuation*, and these edges of the track continuity graph connect two tracks through a common spatial location:

TrackContinuation

$$\begin{aligned} &Edge[TrainTrack] \\ &location : Location \\ &\hline \\ &location \in \bigcap(getTrackEndpoints\ endpoints) \\ &\#(\bigcap(getTrackEndpoints\ endpoints)) = 1 \end{aligned}$$

$$getTrackContinuation : \mathbb{F}\, TrainTrack \nrightarrow TrackContinuation$$

$$\begin{aligned} &getTrackContinuation \\ &\quad = (\lambda\, ts : \mathbb{F}\, TrainTrack \mid \#ts = 2 \land \#(\bigcap(getTrackEndpoints\ ts)) = 1 \\ &\qquad \bullet\ (\mu\, c : TrackContinuation \mid c.endpoints = ts)) \end{aligned}$$

From this, we model the two paths corresponding two the two graphs, and link those two paths together via the train tracks.

$$
\begin{array}{|l}
\textit{Path}[V] \\
\hline
\textit{edges} : \operatorname{iseq} \textit{Edge}[V] \\
\textit{vertices} : \operatorname{iseq} V \\
\hline
\#\textit{edges} + 1 = \#\textit{vertices} \\
\forall i, j : \operatorname{dom} \textit{vertices} \\
\quad \bullet\ i + 1 = j \Rightarrow (\textit{edges}\ i).\textit{endpoints} = \{\textit{vertices}\ i, \textit{vertices}\ j\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{Route} \\
\hline
\textit{Edge}[\textit{Location}] \\
\textit{trackContinuations} : \mathbb{F}\, \textit{TrackContinuation} \\
\textit{trackContinuationPath} : \textit{Path}[\textit{TrainTrack}] \\
\textit{locationPath} : \textit{Path}[\textit{Location}] \\
\hline
\forall c : \textit{TrackContinuation} \\
\quad \bullet\ c \in \textit{trackContinuations} \\
\qquad \Leftrightarrow (\exists e : \operatorname{ran} \textit{trackContinuationPath.edges} \\
\qquad\quad \bullet\ e.\textit{endpoints} = c.\textit{endpoints}) \\
\#\textit{trackContinuationPath.vertices} = \#\textit{locationPath.edges} \\
\forall i : \operatorname{dom} \textit{trackContinuationPath.vertices} \\
\quad \bullet\ (\textit{trackContinuationPath.vertices}\ i).\textit{endpoints} \\
\qquad = (\textit{locationPath.edges}\ i).\textit{endpoints} \\
\textit{endpoints} = \{l : \textit{Location} \\
\quad \mid (\exists t : \operatorname{ran} \textit{trackContinuationPath.vertices} \bullet l \in t.\textit{endpoints}) \\
\qquad \wedge (\forall c : \textit{trackContinuations} \bullet l \neq c.\textit{location})\} \\
\hline
\end{array}
$$

5.3 Background on Block Sections

Traditional train signalling methods use something known as *block sections*. These are signalled regions of tracks where at most one train is permitted to occupy at any time. We extend this concept to model any modular unit within a signalled rail network. In our system, a "block section" refers to a small connected component of tracks that are equipped with signals, railroad switches and crossings.

Multiple trains can occupy the same block section, but only if they are on different routes that are compatible with each other. This modular unit of the rail network cannot be subdivided further because some equipment that spans across two lines need to be worked together, such as the pair of switches in a crossover.

To help capture this notion of having a connected component of tracks, the concept of a path of routes (*RoutePath*) is introduced. With this concept, there must be a sequence of routes that can connect from any track in the block section to every other track in the block section.

$$
\begin{array}{|l}
\textit{RoutePath} \\
\hline
\textit{Path}[\textit{Location}] \\
\textit{routes} : \mathbb{F}\,\textit{Route} \\
\hline
\forall\, \textit{route} : \textit{Route} \\
\quad \bullet\ \textit{route} \in \textit{routes} \\
\quad\quad \Leftrightarrow (\exists\, \textit{edge} : \operatorname{ran} \textit{edges} \bullet \textit{edge.endpoints} = \textit{route.endpoints}) \\
\hline
\end{array}
$$

5.3.1 Signal Modelling

To simplify our model, our system employs a basic two-aspect signalling system to tell the train driver whether to proceed into the block or not. Here, "two-aspect" simply refers to the two different possible signal states.

$$\textit{Signal} ::= \textit{SIGNAL\_CLEARED} \mid \textit{SIGNAL\_BLOCKED}$$

5.3.2 Route Setting

In modern days, signallers in train control centres no longer control the individual signals and switch equipment manually. Rather, they work on a higher level of abstraction known as route setting. Using something known as the entry-exit method (often abbreviated as "NX"), signallers specify the locations of the signals to which the signaller want to allow a train to enter and exit through. Our block section concept encapsulates this route setting methodology in which the user can only set and cancel routes. By setting and cancelling routes, the appropriate signals are cleared, and any track switches that needs to switch to a different track are switched to allow the route to complete.

5.3.3 Interlockings

Closely associated with route setting is the concept of interlocking. As the control of the signals and track equipment are crucial to the safety of the train passengers, safety devices were implemented to prevent an unsafe combination of signals and track equipment from being configured. In the past, these were mechanical systems that prevent switch and signal levers from being pulled incorrectly, which then evolved into electromechanical relay systems. Nowadays, train control systems employ software-based and solid-state-based interlocking systems. We model this interlocking system as a set of constraints applied to which routes can be set and how it reflects in the track equipment.

5.4 Modelling Block Sections

We now begin defining the requirements of a block section. We do this in several steps: First we define the shape of the state variables in *InactiveBlockSection*,

which represents a block section without any constraints. Next, we define different constraints for an active *BlockSection* in separate schemas for clarity, before conjoining them into the *BlockSection* schema. This is useful because it helps document the purpose for each constraint, as well as providing an unconstrained version of the block section that can serve as a temporary building block for when the user registers a new block section into the system.

The block section is a collection of tracks, signals, route settings, switches, and crossings. We assume that each registered route has a corresponding signal.

InactiveBlockSection

$$
\begin{aligned}
&tracks : \mathbb{F}\, TrainTrack \\
&signals : Route \pfun Signal \\
&routeSettings : \mathbb{F}\, Route \\
&switches : \mathbb{F}\, TrackContinuation \pfun TrackContinuation \\
&crossings : \mathbb{F}(\mathbb{F}\, TrackContinuation)
\end{aligned}
$$

5.4.1 Construction Constraints for Block Sections

Next, we introduce a series of constraints that make the block section valid in terms of its construction and topology. One condition, as mentioned previously, is that the tracks should form one connected component. This is because if some tracks are not connected to the rest, then there is no reason for them to be signalled together in the same block section.

BlockSection_TracksShouldBeConnected

InactiveBlockSection

$$
\begin{aligned}
&\forall t1, t2 : tracks \\
&\quad \bullet \exists p : RoutePath \\
&\qquad \bullet p.routes \subseteq \mathrm{dom}\, signals \\
&\qquad\quad \land \{head\ p.vertices, last\ p.vertices\} \\
&\qquad\qquad \subseteq t1.endpoints \cup t2.endpoints
\end{aligned}
$$

We then describe what it means to be a valid route. The routes available to the signaller to set are prescribed by the domain of the *signals* function, and those routes should start and end at the boundaries of the block section (meaning that incomplete routes that end inside the block section are not allowed). Moreover, the route should have track-continuity. This means that if the route encounters any junction with more than two tracks intersecting the same location, then the path that the route takes must be covered by either a railroad switch or a crossing that is installed in the block section.

BlockSection_AvailableRoutesAreMaximalAndShouldBeEquipped

InactiveBlockSection

$$
\begin{array}{l}
\forall r : \mathrm{dom}\, signals \\
\quad \bullet\ \mathrm{ran}\, r.trackContinuationPath.vertices \subseteq tracks \\
\quad \land\ (\forall l : r.endpoints \bullet \#\{t : tracks \mid l \in t.endpoints\} = 1) \\
\quad \land\ (\forall c : r.trackContinuations \\
\qquad \bullet\ (\#\{t : tracks \mid c.location \in t.endpoints\} = 2 \\
\qquad\quad \lor\ \#\{t : tracks \mid c.location \in t.endpoints\} > 2 \\
\qquad\qquad \land\ ((\exists s : \mathrm{dom}\, switches \bullet c \in s) \\
\qquad\qquad\quad \lor\ (\exists crossing : crossings \bullet c \in crossing))))
\end{array}
$$

All junctions (where more than two tracks intersect a common location) in a block section must also be described by either a crossing or a switch, or else the continuity of the junction is ambiguous. Moreover, each junction should be specified by exactly one switch or exactly one crossing, and so the switch size and crossing sizes must match the equivalent junction size.

BlockSection_JunctionsAreEquippedCompletely

InactiveBlockSection

$$
\begin{array}{l}
\forall l : Location \bullet \exists n : \mathbb{N} \\
\quad \bullet\ n = \#\{t : TrainTrack \mid t \in tracks \land l \in t.endpoints\} \land n > 2 \\
\quad \Rightarrow\ (\exists s : \mathrm{dom}\, switches \bullet (\exists c : s \bullet l = c.location) \land \#s + 1 = n) \\
\qquad \lor\ (\exists crossing : crossings \\
\qquad\quad \bullet\ (\exists c : crossing \bullet l = c.location) \land \#crossing * 2 = n)
\end{array}
$$

We model railroad switches as a set of *trackContinuities* that specify how one track from the converging side of the switch can continue on to several different tracks on the diverging side of the switch. In this way, switches are star graphs with the converging track with the converging track as internal node, and the diverging tracks as the leaves of the star.

BlockSection_SwitchesAreValidStarGraphs

InactiveBlockSection

$$
\begin{array}{l}
\forall continuations : \mathrm{dom}\, switches \\
\quad \bullet\ switches\ continuations \in continuations \\
\quad \land\ (\forall c : continuations \bullet c.endpoints \subseteq tracks) \\
\quad \land\ (\exists t : tracks;\ l : Location \\
\qquad \bullet\ (\forall c : continuations \bullet (c.location = l \land t \in c.endpoints))) \\
\quad \land\ \#continuations > 1
\end{array}
$$

Crossings, on the other hand, are pairs of straight train tracks. We model this as a set of *trackContinuation*s that only share a single common centre location.

BlockSection_CrossingsAreValidAndDisjoint

InactiveBlockSection

$$
\begin{array}{l}
\forall\, continuations : crossings \\
\quad \bullet\ (\forall\, c : continuations \bullet c.endpoints \subseteq tracks) \\
\quad \land\ (\forall\, c1, c2 : continuations \\
\qquad \bullet\ (c1 \neq c2 \Rightarrow c1.endpoints \cap c2.endpoints = \varnothing)) \\
\quad \land\ (\exists\, l : Location \bullet (\forall\, c : continuations \bullet c.location = l)) \\
\quad \land\ \#continuations > 1
\end{array}
$$

5.4.2 Interlocking Rules for Block Sections

We now proceed into the constraints that ensure all track equipment have a safe combination of states at any point in time. We do this in two steps: First, if multiple routes are set, these routes need to be compatible with each other, meaning that they should never intersect. Secondly, all equipment must reflect the route settings. In particular, signals must remain blocked until its route has been set, and switches that are needed for the route has been set correctly.

Interlocking_RouteSettingsAreCompatible

InactiveBlockSection

$$
\begin{array}{l}
routeSettings \subseteq \mathrm{dom}\, signals \\
\forall\, r1, r2 : routeSettings \\
\quad \bullet\ r1 \neq r2 \\
\qquad \Rightarrow r1.locationPath.vertices \cap r2.locationPath.vertices = \varnothing
\end{array}
$$

Interlocking_EquipmentMatchesRouteSettings

InactiveBlockSection

$$
\begin{array}{l}
\forall\, r : \mathrm{dom}\, signals \\
\quad \bullet\ signals\ r = SIGNAL\_CLEARED \Leftrightarrow r \in routeSettings \\
\forall\, r : routeSettings;\ s : \mathrm{dom}\, switches \\
\quad \bullet\ \forall\, c : r.trackContinuations \bullet c \in s \Rightarrow switches\ s = c
\end{array}
$$

5.4.3 "Active" Block Sections

Pulling all the constraints together, we form the requirements of an "active" block section. Here, "active" is used to distinguish it from "inactive" blocks that are not constrained and are not under use in the rail network.

BlockSection

InactiveBlockSection
BlockSection_TracksShouldBeConnected
BlockSection_AvailableRoutesAreMaximalAndShouldBeEquipped
BlockSection_JunctionsAreEquippedCompletely
BlockSection_SwitchesAreValidStarGraphs
BlockSection_CrossingsAreValidAndDisjoint
Interlocking_RouteSettingsAreCompatible
Interlocking_EquipmentMatchesRouteSettings

5.5 Inactive Block Section Management

We will now present to you the interface and operations through which block sections can be managed. For now, we will be focusing on the management of a single inactive block section in isolation.

The primary use-case for inactive blocks is to modify the block's construction when it is not in use by the rest of the train network. Active blocks need to be "taken offline" first, so to speak, before they can be modified. Similarly, new blocks start off as inactive, and after verifying the block's design is correct, it can be "take online" to be used by the rest of the train network.

This subsystem allows tracks, switches, crossings, and routes to be added and removed. Most of these use a "point-and-click" interface to specify these block components. For instance, installing a track is done by specifying the two locations the track is to be installed on, and later, these tracks are referred to using these two location.

5.5.1 Installing and Removing Tracks

InactiveBlockSection_InstallTrack

$\Delta InactiveBlockSection$
$from? : Location$
$to? : Location$
$length? : \mathbb{N}_1$
$compatibleTrainModels? : \mathbb{F}\, TrainModel$
$maxSpeed? : \mathbb{N}$

$\neg\, (\exists t : tracks \bullet t.endpoints = \{to?, from?\})$
$\exists t : TrainTrack$
$\quad \bullet\ t.endpoints = \{from?, to?\}$
$\quad \land\ t.length = length?$
$\quad \land\ t.compatibility = compatibleTrainModels?$
$\quad \land\ t.maxSpeed = maxSpeed?$
$\quad \land\ tracks' = tracks \cup \{t\}$
$signals' = signals$
$routeSettings' = routeSettings$
$switches' = switches$
$crossings' = crossings$

InactiveBlockSection_RemoveTrack

$\Delta InactiveBlockSection$
$from? : Location$
$to? : Location$

$\exists t : tracks \bullet t.endpoints = \{to?, from?\} \land tracks' = tracks \setminus \{t\}$
$signals' = signals$
$routeSettings' = routeSettings$
$switches' = switches$
$crossings' = crossings$

5.5.2 Registering New Switches

$getEquipedJunctionLocations : InactiveBlockSection \to \mathbb{F}\, Location$

$getEquipedJunctionLocations$
$\quad = (\lambda\, b : InactiveBlockSection \bullet \{l : Location$
$\quad\quad \mid (\exists s : \mathrm{ran}\, b.switches \bullet s.location = l)$
$\quad\quad\quad \lor\ (\exists cs : b.crossings \bullet (\exists c : cs \bullet c.location = l))\})$

$$
\begin{array}{|l}
getDivergingTracks \\
\quad : \mathbb{F}\, TrainTrack \times Location \times \mathbb{F}\, Location \pfun \mathbb{F}\, TrainTrack \\
\hline
getDivergingTracks \\
\quad = (\lambda\, ts : \mathbb{F}\, TrainTrack;\ c : Location;\ ds : \mathbb{F}\, Location \\
\qquad \mid \forall d : ds \bullet \#\{t : ts \mid t.endpoints = \{c, d\}\} = 1 \\
\qquad \bullet \{t : ts \mid \exists d : ds \bullet t = getTrack(ts, \{c, d\})\})
\end{array}
$$

$$
\begin{array}{|l}
getSwitch : TrainTrack \times \mathbb{F}\, TrainTrack \to \mathbb{F}\, TrackContinuation \\
\hline
getSwitch \\
\quad = (\lambda\, convergingTrack : TrainTrack;\ divergingTracks : \mathbb{F}\, TrainTrack \\
\qquad \bullet \{c : TrackContinuation \\
\qquad\quad \mid \exists d : divergingTracks \\
\qquad\qquad \bullet\ c = getTrackContinuation\{convergingTrack, d\}\})
\end{array}
$$

$$
\begin{array}{|l}
\textit{InactiveBlockSection\_RegisterSwitch} \\
\hline
\Delta InactiveBlockSection \\
centre? : Location \\
convergentEnd? : Location \\
divergentEnds? : \mathbb{F}\, Location \\
initialPosition? : Location \\
\hline
initialPosition? \in divergentEnds? \\
\#(divergentEnds? \cup \{centre?, convergentEnd?\}) \\
\quad = 2 + \#divergentEnds? \\
\forall end : divergentEnds? \cup \{convergentEnd?\} \bullet \\
\quad \exists t : tracks \bullet t.endpoints = \{centre?, end\} \\
centre? \notin getEquipedJunctionLocations\ \theta InactiveBlockSection \\
\exists to\_add : \mathbb{F}\, TrackContinuation;\ initc : TrackContinuation \bullet \\
\quad to\_add = getSwitch \\
\qquad ((getTrack(tracks, \{centre?, convergentEnd?\})), \\
\qquad (getDivergingTracks(tracks, centre?, divergentEnds?))) \\
\quad \land\ initc = (\mu\, c : to\_add \\
\qquad \mid initialPosition? \in \bigcup(getTrackEndpoints\ c.endpoints)) \\
\quad \land\ switches' = switches \oplus \{(to\_add \mapsto initc)\} \\
tracks' = tracks \\
signals' = signals \\
routeSettings' = routeSettings \\
crossings' = crossings
\end{array}
$$

5.5.3 Registering New Crossings

InactiveBlockSection_RegisterCrossing

$\Delta InactiveBlockSection$
$centre? : Location$
$ends? : \mathbb{F}(Location \times Location)$

$centre? \notin getEquipedJunctionLocations\ \theta InactiveBlockSection$
$\forall e : ends?$
$\quad \bullet \exists t1, t2 : tracks$
$\quad\quad \bullet t1 \neq t2$
$\quad\quad\quad \wedge t1.endpoints = \{first\ e, centre?\}$
$\quad\quad\quad \wedge t2.endpoints = \{second\ e, centre?\}$
$crossings' = crossings$
$\quad \cup \{\{c : TrackContinuation \mid \exists e : ends?$
$\quad\quad \bullet c = getTrackContinuation$
$\quad\quad\quad \{(getTrack(tracks, \{(first\ e), centre?\})),$
$\quad\quad\quad (getTrack(tracks, \{(second\ e), centre?\}))\}\}\}$
$tracks' = tracks$
$signals' = signals$
$routeSettings' = routeSettings$
$switches' = switches$

5.5.4 Removing Switches and Crossings

InactiveBlockSection_UnregisterJunction

$\Delta InactiveBlockSection$
$centre? : Location$

$centre? \in getEquipedJunctionLocations\ \theta InactiveBlockSection$
$switches' = switches \rhd \{c : TrackContinuation \mid c.location = centre?\}$
$tracks' = tracks$
$signals' = signals$
$routeSettings' = routeSettings$
$crossings'$
$\quad = crossings \setminus \{cs : \mathbb{F}\, TrackContinuation \mid \exists c : cs \bullet c.location = centre?\}$

5.5.5 Registering and Unregistering Routes

For ease of use, registering and unregistering a route also uses the entry-exit (NX) method. Here, note that we are assuming for the sake of simplicity that each block section only contains one route that connects the entry-exit pair.

InactiveBlockSection_RegisterRoute

$\Delta InactiveBlockSection$
$entry? : Location$
$exit? : Location$

$\#\{t : tracks \mid entry? \in t.endpoints\} = 1$
$\#\{t : tracks \mid exit? \in t.endpoints\} = 1$
$(entry?, exit?) \notin \text{ran}(\text{dom}\, signals \lhd routeMatches)$
$\exists r : \text{dom}(routeMatches \rhd \{(entry?, exit?)\});\ signals' : Route \pfun Signal$
$\quad \bullet\ signals' = signals \oplus \{(r \mapsto SIGNAL\_BLOCKED)\}$
$\quad\quad \land\ BlockSection\_AvailableRoutesAreMaximalAndShouldBeEquipped'$
$tracks' = tracks$
$routeSettings' = routeSettings$
$switches' = switches$
$crossings' = crossings$

InactiveBlockSection_UnregisterRoute

$\Delta InactiveBlockSection$
$entry? : Location$
$exit? : Location$

$\#\{t : tracks \mid entry? \in t.endpoints\} = 1$
$\#\{t : tracks \mid exit? \in t.endpoints\} = 1$
$\exists r : \text{dom}(\text{dom}\, signals \lhd routeMatches \rhd \{(entry?, exit?)\})$
$\quad \bullet\ signals' = \{r\} \rhd signals$
$tracks' = tracks$
$routeSettings' = routeSettings$
$switches' = switches$
$crossings' = crossings$

5.6 Active Block Section Management

Once a block is "taken online", the signaller can set and cancel routes on it using the entry-exit (NX) method. For readability, we define the following relation *routeMatches* to help find the route that the user has specified.

$routeMatches : Route \leftrightarrow Location \times Location$

$\forall r : Route;\ entry : Location;\ exit : Location$
$\quad \bullet\ r \mapsto (entry, exit) \in routeMatches$
$\quad\quad \Leftrightarrow head\ r.locationPath.vertices = entry$
$\quad\quad\quad \land\ last\ r.locationPath.vertices = exit$

5.6.1 Setting a Route

Setting a route involves finding a matching route in the block section, registering it with the section's book-keeping, and updating the switches and signals so that trains can enter into the specified entry coordinates and exit from the specified exit coordinates.

BlockSection_SetRoute

$$
\begin{array}{l}
\Delta BlockSection \\
entry? : Location \\
exit? : Location \\
\hline
\mathrm{dom}\, switches' = \mathrm{dom}\, switches \\
\exists r : \mathrm{dom}(\mathrm{dom}\, signals \lhd routeMatches \rhd \{(entry?, exit?)\}) \\
\quad \bullet\ r \notin routeSettings \\
\quad \land\ r \in \mathrm{dom}\, signals \\
\quad \land\ (\forall r2 : routeSettings \\
\qquad \bullet\ r.locationPath.vertices \cap r2.locationPath.vertices = \varnothing) \\
\quad \land\ signals' = signals \oplus \{(r \mapsto SIGNAL\_CLEARED)\} \\
\quad \land\ routeSettings' = routeSettings \cup \{r\} \\
\quad \land\ (\forall s : \mathrm{dom}\, switches \\
\qquad \bullet\ (\forall c : r.trackContinuations \\
\qquad\quad \bullet\ ((c \notin s \Rightarrow switches'\ s = switchess) \\
\qquad\qquad \land\ (c \in s \Rightarrow switches'\ s = c)))) \\
tracks' = tracks \\
crossings' = crossings
\end{array}
$$

5.6.2 Cancelling a Route and Cancelling All Routes

Likewise for cancelling a route, the matching route is found and removed from book-keeping. While the signals need to be updated, the switches do not.

BlockSection_CancelRoute

$$
\begin{array}{l}
\Delta BlockSection \\
entry? : Location \\
exit? : Location \\
\hline
\exists r : \mathrm{dom}(\mathrm{dom}\, signals \lhd routeMatches \rhd \{(entry?, exit?)\}) \\
\quad \bullet\ r \in routeSettings \\
\quad \land\ signals' = signals \oplus \{(r \mapsto SIGNAL\_BLOCKED)\} \\
\quad \land\ routeSettings' = routeSettings \setminus \{r\} \\
tracks' = tracks \\
switches' = switches \\
crossings' = crossings
\end{array}
$$

BlockSection_CancelAllRoutes

$\Delta BlockSection$

$\mathrm{dom}\, signals' = \mathrm{dom}\, signals$

$\forall r : \mathrm{dom}\, signals \bullet signals' r = SIGNAL\_BLOCKED$

$routeSettings' = \varnothing$

$tracks' = tracks$

$switches' = switches$

$crossings' = crossings$

6 Train Control System – The Big Picture

With the block section subsystems described, we are finally able to describe the overall train control system and how it interacts with the block sections. The system keeps the active and inactive blocks in separate "dictionaries" (functions) that are indexed by a unique integer ID.

We will now discuss the meanings and interpretations of the mathematical structures for each state variable. Level crossings are indexed by the set of block IDS which the level crossing interact with. Stations are indexed by their names and map to platforms that describe the block the platform reside on. Train lines represent a sequence of station names that trains on that line will pass through, and is a commuter-friendly abstraction over individual train tracks and blocks. Finally, trains are indexed by their "headcode" (also known as train describer) which the user specifies. Many of the other schemas in this system are not included in this report for brevity.

TrainControlSystem

$activeBlocks : \mathbb{N} \rightarrowtail BlockSection$

$inactiveBlocks : \mathbb{N} \rightarrowtail InactiveBlockSection$

$levelCrossings : \mathbb{F}\, \mathbb{N} \nrightarrow BarrierState$

$stations : String \rightarrowtail \mathrm{iseq}\, Platform$

$lines : String \rightarrowtail \mathrm{iseq}\, String$

$trains : HeadCode \rightarrowtail Train$

$\mathrm{ran}\, lines = \mathrm{iseq}(\mathrm{dom}\, stations)$

$\bigcup(\mathrm{dom}\, levelCrossings) \subseteq \mathrm{dom}\, activeBlocks \cup \mathrm{dom}\, inactiveBlocks$

$\forall s : \mathrm{ran}\, stations \bullet \forall platform : \mathrm{ran}\, s$

$\quad \bullet\ platform.block \in \mathrm{ran}\, activeBlocks \cup \mathrm{ran}\, inactiveBlocks$

$\forall l : \mathrm{ran}\, lines \bullet \mathrm{ran}\, l \subseteq \mathrm{dom}\, stations$

$\forall train : \mathrm{ran}\, trains$

$\quad \bullet\ \mathrm{ran}\, train.occupiedTracks$

$\quad\quad \subseteq \{t : TrainTrack \mid \exists b : \mathrm{ran}\, activeBlocks \bullet t \in b.tracks\}$

The initial state for this system is when there are no blocks, crossings, stations,

lines, or trains registered with the system.

Init

TrainControlSystem

$$activeBlocks = \varnothing$$
$$inactiveBlocks = \varnothing$$
$$levelCrossings = \varnothing$$
$$stations = \varnothing$$
$$lines = \varnothing$$
$$trains = \varnothing$$

7 System Properties

With the system described, we would like to confirm that it satisfies a few properties that would make the system safe and useful. While the constraints imposed onto the schemas in the Z-model are usually more detailed and specific, these properties aims to be as simple as possible for two reasons: Firstly, the real-life meaning of the property should be easy to understand and interpret unambiguously. Secondly, we would like these properties to generalise over the entire system statespace as much as possible.

7.1 Safety Properties

One crucial property of a train control system is that trains should never collide. This is described by ensuring that each track should at most occupy one train at any point in time.

theorem TrainsNeverCollide

$$\forall TrainControlSystem \bullet \forall block : \mathrm{ran}(activeBlocks \cup inactiveBlocks) \bullet \forall track : block.tracks \bullet \#(trackOccupancy(track, (\mathrm{ran}\, trains))) \leq 1$$

7.2 Operational Properties

Most safety properties are described directly as constraints to the statespace of the system schemas. This includes the interlocking rules in combination with the constraints of the dual graph model of routing. Therefore, we need to check that our system design is not over-constrained.

We describe a simple theorem that, if proven, ensures that a safe block section can be created within this system.

theorem SafeBlockSectionDesignExists

$$\exists InactiveBlockSection \bullet BlockSection$$

Another important requirement is that our system should never get "stuck" at a particular state. In the event of incidents, signallers need to respond and recover the system quickly. To achieve this, it is necessary that they always have access to their means of control: setting and cancelling routes. The following theorem, if proven, will verify whether the a block section can always be controlled and that the block section can never get stuck at a particular state.

theorem BlockSectionsAreAlwaysControllable
$\forall BlockSection$
$\bullet\ \exists BlockSection'$
$\bullet\ BlockSection\_SetRoute \lor BlockSection\_CancelRoute$