XVII. Train System (July 2008)

1 Informal Introduction

The purpose of this chapter is to show the specification and construction of a complete computerized system. The example we are interested in is called a *train system*. By this, we mean a system that is practically managed by a *train agent*, whose role is to control the various trains crossing part of a certain *track network* situated under his supervision. The computerized system we want to construct is supposed to help the train agent in doing this task.

Before entering in the informal description of this system (followed by its formal construction), it might be useful to explain the reason why we think it is important to present such a case study in great details. There are at least four reasons which are the following:

1. This example presents an interesting case of quite complex data structures (the track network) whose mathematical properties have to be defined with great care: we want to show that this is possible.
2. This example also shows a very interesting case where the reliability of the final product is absolutely fundamental: several trains have to be able to safely cross the network under the complete automatic guidance of the software product we want to construct. For this reason, it will be important to study the bad incidents that could happen and which we want to either completely avoid or safely manage. In this chapter however, we are more concerned by *fault prevention* than *fault tolerance*. We shall come back to this in the conclusion.
3. The software must take account of the external environment which is to be carefully controlled. As a consequence, the formal modelling we propose here will contain not only a model of the future software we want to construct but also a detailed model of its environment. Our ultimate goal is to have the software working in perfect synchronization with the external equipment, namely the track circuits, the points, the signals, and also the train drivers. We want to *prove* that trains obeying the signals, set by the software controller, and then (blindly) circulating on the tracks whose points have been positioned, again by the software controller, that these trains will do so in a completely safe manner.
4. Together with this study, the reader will be able to understand the kind of methodology we recommend. It should be described, we hope, in sufficiently general terms so that he or she will be able to use this approach in similar examples.

We now proceed with the informal description of this train system together with its informal (but very precise) definitions and requirements. We first define a typical track network which we shall use as a running example throughout the chapter. We then study the two main components of tracks, namely points and crossings. The important concepts of blocks, routes, and signals are then presented together with their main properties. The central notions of route and block reservations are proposed. Safety conditions are then studied.This is followed by the complementary train moving conditions allowing several trains to be present in the network at the same time. We propose a number of assumptions about the way trains behave. Finally we present possible failures that could happen and the way such problems are solved.

The formal development (model construction) is preceded by the *refinement strategy* we shall adopt in order to proceed in a gentle and structured manner. This is followed by the formal model construction.

1.1 Methodological Conventions for the Informal Presentation

In the following sections, we give an informal description of this train system, and, together with this description, we state what its main *definitions and requirements* are. Such definitions and requirements will be inserted as separate labelled boxes in the middle of an explanatory text. These boxes must all together clearly define what is to be taken into account by people doing the formal development. The various definitions and requirements will be labelled according to the following taxonomy:

ENV	Environment
FUN	Functional
SAF	Safety

MVT	Movement
TRN	Train
FLR	Failure

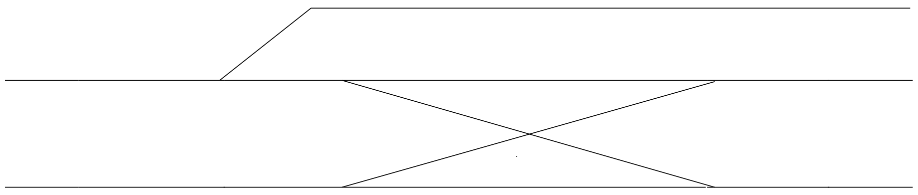
- "Environment" definitions and requirements are concerned with the structure of the track network and its components.
- "Functional" definitions and requirements are dealing with the main functions of the system.
- "Safety" definitions and requirements define the properties ensuring that no classical accidents could happen.
- "Movement" definitions and requirements ensure that a large number of train may cross the network at the same time.
- "Train" definitions and requirements define the implicit assumptions about the behavior of trains.
- "Failure" definitions and requirements finally define the various failures against which the system is able to react without incidents.

Here is our first very general requirement:

The goal of the train system is to safely control trains moving on a track network	FUN-1

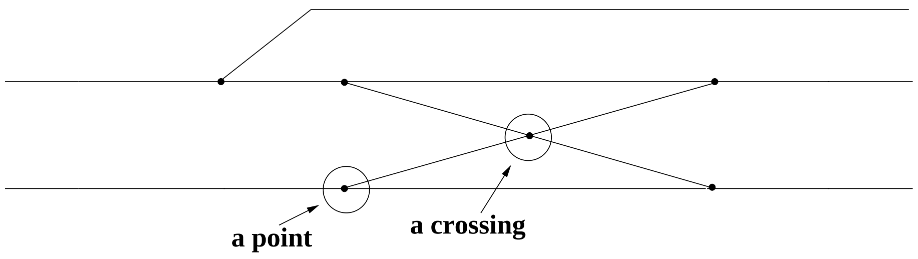
1.2 Network Associated with a Controlling Agent

Here is a typical track network that a train agent is able to control. In what follows, we are going to use that network as a running example:



1.3 Special Components of a Network: Points and Crossings

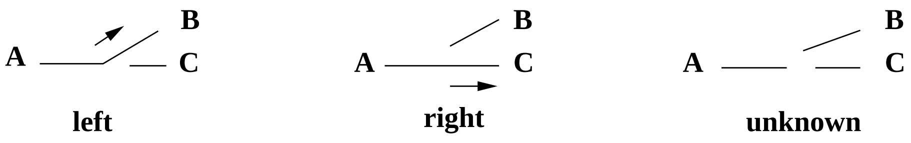
Such a network contains a number of *special components*: these are the *points* and the *crossings* as illustrated in the following figure (five points and one crossing).



A point is a device allowing a track to split in two distinct directions. A crossing, as its name indicates, is a device that makes two different tracks crossing each other. In what follows we briefly describe points and crossings.

A track network may contain some special components: points and crossings	ENV-1

Point. A point special component can be in three different positions: left, right, or unknown. This is indicated in the following figure.



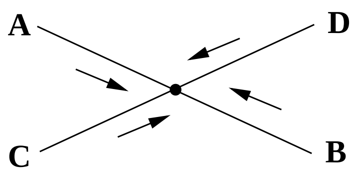
Note that the orientation from A to C is sometimes called the *direct track* whereas the one from A to B is called the *diverted track*. In what follows however, we shall continue to call them right and left respectively are there is no ambiguity in doing so.

In the first two cases above, the arrow in the figure shows the convention we shall use to indicate the orientation of the point. Note that these arrows do not indicate the direction followed by a train. For example, in the first case, it is said that a train coming from **A** will turn left, a train coming from **B** will turn right, and a train coming from **C** *will probably have some troubles*! Also note that a train encountering a point oriented in an unknown direction (third case) might have some trouble too, even more if a point suddenly changes position while a train is on it (we shall come to this in section 1.8).

The last case is the one that holds when the point is moving from left to right or vice-versa. This is because this movement is supposed to take some time: it is performed by means of a motor which is part of the point. When the point has reached its final position (left or right) it is locked, whereas when it is moving it is unlocked. Note however that in the coming development we shall not take this into account. In other words, we shall suppose, as a simplification, that a *point moves instantaneously and that it is thus always locked*. In other words, the unknown case is not treated, we then just require in this development that a point may have only two positions: left or right.

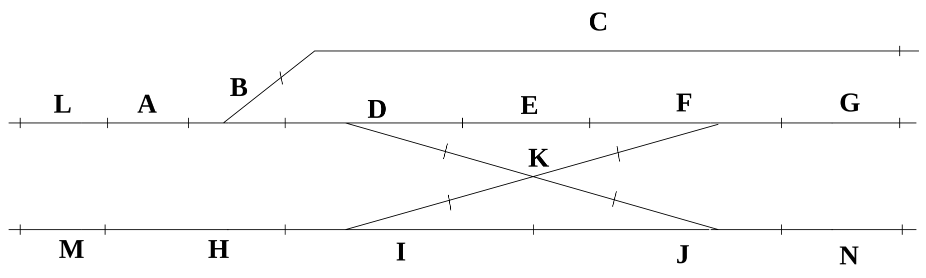
A point may have two positions: left or right	ENV-2

Crossing. A crossing special component is completely static: it has no state as points have. The way a crossing behaves is illustrated in the following figure: trains can go from **A** to **B** and vice-versa, and from **C** to **D** and vice-versa.



1.4 The Concept of Block

The controlled network is statically divided into a fixed number of *named blocks* as indicated in the following figure where we have 14 blocks named by single letters from A to N:



A track network is made of a number of fixed blocks	ENV-3

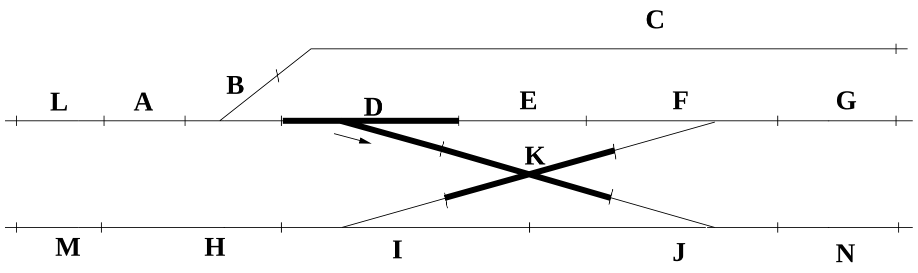
Each block may contain at most one special component (points or crossings).

A special component (points or crossings) is always attached to a given block. And a block contains at most one special component	ENV-4

For example in our case, block C does not contain any special component, whereas block D contains one point, and block K contains a crossing. Each block is equipped with a, so-called, *track circuit* which is able to detect the presence of a train on it. A block can thus be in two distinct states: unoccupied (no train on it) or occupied (a train is on it).

A block may be occupied or unoccupied by a train	ENV-5

In the following figure, you can see that a train is occupying the two adjacent blocks D and K (this is indicated in the figure by the fact that the blocks in question are emphasized).



Notice that when a train is detected in a block we do not know a priori the precise position of the train in it, nor do we know whether the train is stopped or moving. Moreover, in the last case, we do not know in which direction the train is moving. But all such informations are not important for us: as will be seen in this development, it is only sufficient for our purpose to know that a block is occupied or not.

1.5 The Concept of Route

The blocks defined in the previous section are always structured in a number of statically *pre-defined routes*. Each route represents a possible path that a train may follow within the network controlled by the train agent. In other words, the routes define the various ways a train can traverse the network. A route is composed of a number of adjacent blocks forming an ordered sequence.

A network has a fixed number of routes. Each route is characterized by a sequence of adjacent blocks	ENV-6

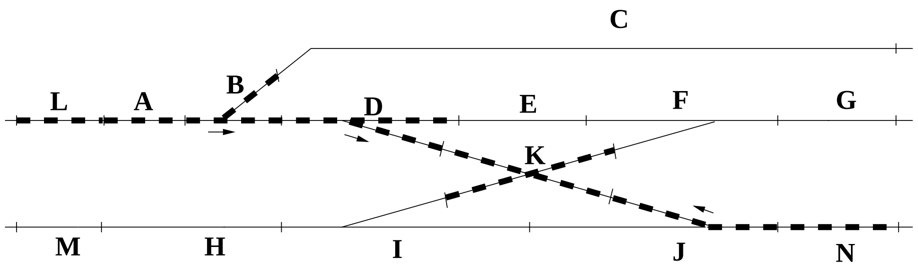
A train following a route is supposed to occupy in turn each block of that route. Note that a train may occupy several adjacent blocks at the same time (even a short train). Also note that a given block can be part of several routes. All this is shown below in the following table where 10 pre-defined routes are proposed:

$R1$	$L\ A\ B\ C$	$R6$	$C\ B\ A\ L$
$R2$	$L\ A\ B\ D\ E\ F\ G$	$R7$	$G\ F\ E\ D\ B\ A\ L$
$R3$	$L\ A\ B\ D\ K\ J\ N$	$R8$	$N\ J\ K\ D\ B\ A\ L$
$R4$	$M\ H\ I\ K\ F\ G$	$R9$	$G\ F\ K\ I\ H\ M$
$R5$	$M\ H\ I\ J\ N$	$R10$	$N\ J\ I\ H\ M$

Besides being characterized by the sequence of blocks composing it, a route is also statically characterized by the positions of the points which are parts of the corresponding blocks. For example, route $R3$ ($L\ A\ B\ D\ K\ J\ N$) is characterized as follows:

– the point in block B is positioned to right,
– the point in block D is positioned to right,
– the point in block J is positioned to right.

This is illustrated in the following figure where route $R3$ $(L\ \ A\ \ B\ \ D\ \ K\ \ J\ \ N)$ is emphasized. The little arrows situated next to the points of blocks B, D, and J indicate their position:



A route is also characterized by the positions of the points which are situated in blocks composing it	ENV-7

Routes have two additional properties. The first concern the first block of a route:

The first block of a route cannot be part of another route unless it is also the first or last block of that route	ENV-8

And the second one concerns the last block of a route :

The last block of a route cannot be part of another route unless it is also the first or last block of that route	ENV-9

At the end of the next section, we shall explain why the constraints we have presented just now are important. Finally, a route has some obvious continuity property:

A route connects its first block to its last one in a continuous manner	ENV-10

and it has no cycle:

A route contains no cycles	ENV-11

1.6 The Concept of Signal

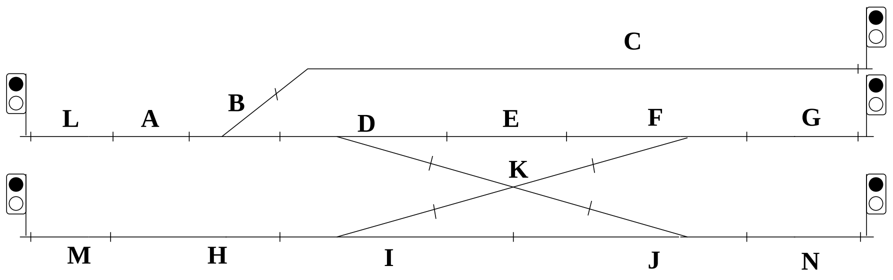
Each route is protected by a *signal*, which can be red or green. This signal is situated just before the first block of each route. It must be clearly visible from the train drivers.

Each route is protected by a signal situated just before its first block	ENV-12

When a signal is red, then, by convention, the corresponding route cannot be used by an incoming train. Of course, the train driver must obey this very fundamental requirement.

A signal can be red or green. Trains are supposed to stop at red signals	ENV-13

In the next figure, you can see the signal protecting each route:



Notice that a given signal can protect several routes. For example, the signal situated on the left of block L protects route $R1$ $(L \ A \ B \ C)$, $R2$ $(L \ A \ B \ D \ E \ F \ G)$, and $R3$ $(L \ A \ B \ D \ K \ J \ N)$: this is because each of these routes starts with the same block, namely block L.

Routes having the same first block share the same signal	ENV-14

In the previous figure and in the coming ones, we use the convention that a signal which is situated to the left of its pole protects the routes situated on its right and vice-versa. For example, the signal situated on the right hand side of block C protects route R6, namely $(C \ B \ A \ L)$.

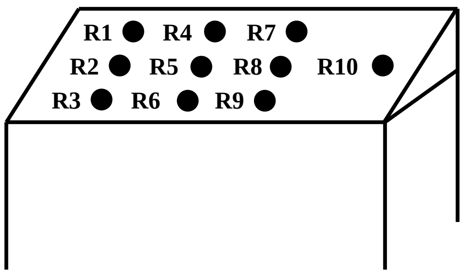
A last important property of a signal protecting the first block of a route is that, when green, it turns back automatically to red as soon as a train enters into the protected block.

A green signal turns back to red automatically as soon as the first block is made occupied	ENV-15

The reason for the constraints defined at the end of section 1.5 must now be clear: we want a signal, which is always situated just before the first block of a route, to clearly identify the protection of that route. If a route, say r1, starts in the middle of another one r2, then the signal protecting r1 will cause some trouble for train situated in route r2. As very often the reverse of a route is also used as a route, the previous constraint applies for the last block of a route: it cannot be common to another route except if it is also the last block of that route.

1.7 Route and Block Reservations

The train agent is provided with a panel offering a number of commands corresponding to the different routes he can assign to trains traversing his "territory".



When a train is approaching the network, the train agent is told that this train will cross the network by using a certain route. The train agent then presses the corresponding command in order to *reserve* that route. Note that other trains might already be crossing the network while the train agent is pressing that command. As a consequence, the situation faced by the train agent is potentially dangerous: we shall come back to this very important fact in section 1.8. This is the reason why the forthcoming reservation process is entirely controlled by the software we want to construct.

A route can be reserved for a train. The software is in charge of controlling the reservation process	FUN-2

The reservation process of a route r is made of three phases:

1. the individual reservation of the various blocks composing route r is performed,
2. the positioning of the relevant points of route r is accomplished,
3. the turning to green of the signal protecting route r is done.

When the first phase is not possible (see next section), the reservation fails and the two other phases are then cancelled. In this case, the reservation has to be re-tried later by the train agent. Let us now describe these phases in more details.

Phase 1: Block Reservation. The block reservation performed during the first phase induces another state for a block (besides being occupied or unoccupied by a train, as seen in section 1.3): a block can be reserved or free.

A block can be reserved or free	FUN-3

Note that an occupied block must clearly be already reserved.

An occupied block is always reserved	FUN-4

At the end of this first successful phase, the route is said to be *reserved*, but it is not ready yet to accept a train.

Reserving a route consists in reserving the individual blocks it is made of. Once this is done, the route is said to be *reserved*	FUN-5

Phase 2: Point Positioning. When the reservation of all blocks of a route r is successful, the reservation process proceeds with the second phase, namely the positioning of the corresponding points in the direction corresponding to the route r. When all points of r are properly positioned, the route is said to be *formed*.

Once it is reserved, a route has to be *formed* by properly positioning its points	FUN-6

Note that a formed route remains reserved.

A formed route is always a reserved route	FUN-7

Phase 3: Turning Signal to Green. Once a route r is formed, the third and last phase of the reservation can be done: the signal controlling route r is turned green: a train can be accepted in it. A train driver, looking at the green signal, then leads the train within the reserved and formed route. We already know from requirement ENV-15 that the signal will then turn red immediately.

Once it is formed, a route is made available for the incoming train by turning its signal to green	FUN-8

1.8 Safety Conditions

As several trains can cross the network at the same time and as the train agent (or rather the software he uses) is sometimes re-positioning a point when forming a route, there are clearly some serious risks of bad incidents. This is the reason why we must clearly identify such risks and see how we can safely avoid them. This is, in fact, the main purpose of the software we would like to build in order to help the train agent in a systematic fashion. There are three main risks which are the following:

1. Two (or more) trains traversing the network at the same time hit each other in various ways.
2. A point may change position under a train.
3. The point of a route may change position in front of a train using that route. In other words, the train has not yet occupied the block of this point but it will do so in the near future since that block is situated on that route.

Case 1 is obviously very bad since the hit trains may derail. Case 2 would have the consequence to cut the train into two parts and, most probably, the train will derail too. Case 3 may have two distinct consequences: either to move the train outside its current route so that it can now hit another train (case 1), or to have the train derailing in case the point now disconnect the current route. We are thus going to set up a number of safety conditions in order to prevent such risks from happening. The first risk (train hitting) is avoided by ensuring two safety conditions:

1. a given block *can only be reserved for at most one route at a time*,

A block can be reserved for *at most one* route	SAF-1

2. the signal of a route is green *only* when the various blocks of that route are all reserved for it and are unoccupied, and when all points of that route are set in the proper direction.

The signal of a route can *only be green* when all blocks of that route are reserved for it and are unoccupied, and when all points of this route are properly positioned	SAF-2

As a consequence (and also thanks to requirement FUN-4 stating that an occupied block is always a reserved block), several trains never occupy the same block at the same time, *provided, of course, that train drivers do not overpass a red signal.* We shall come back to this important point in section 1.11.

The second and third risks (points changing direction under certain circumstances) are avoided by ensuring that a point can only be maneuvered when the corresponding block is that of a route which is *reserved (all its blocks being reserved) but not yet formed.*

A point can *only be re-positioned* if it belongs to a block which is in a reserved but not yet formed route	SAF-3

The last safety requirement ensures that no blocks of a reserved, but not yet formed, route are occupied by a train.

No blocks of a reserved, but not yet formed, route are occupied	SAF-4

A consequence of this last safety requirement is that the re-positioning of a point, done according to requirement SAF_3, is always safe.

1.9 Moving Conditions

In spite of the safety conditions (which could be preserved by not allowing any train to cross the network!) we want to allow a large number of trains to be present in the network at the same time without danger. For this, we allow each block of a reserved route to be freed as soon as the train does not occupy it any more.

Once a block of a formed route is made unoccupied, it is also freed	MVT-1

As a result, the only reserved blocks of a formed route are those blocks which are occupied by the train or those blocks of the route which are not yet occupied by the train.

A route remains formed as long as there are some reserved blocks in it	MVT-2

When no block of a formed route is reserved any more for that route, it means that the train has left the route, which can thus be made free.

A formed route can be made free (not formed and not reserved any more) when no blocks are reserved for it any more	MVT-3

1.10 Train Assumptions

Note that it is very important that a block once freed for a route (after being occupied and subsequently unoccupied) *cannot be made occupied again for this route* unless the route is first made free and then formed again. The reason for this is that the freed block in question can be assigned to another route. For achieving this, we must assume that trains obey two properties. First, a train cannot split in two or more parts while in the network.

A train cannot split while in the network	TRN-1

And second, a train cannot move backwards while in the network

A train cannot move backwards while in the network	TRN-2

This is so because in both cases a freed block can be made occupied again. Note that clearly trains do split and move backwards (for example, in Zurich main station): it simply means that the blocks where they do so are not within the network controlled by a train system.

Another important implicit assumption about trains is that they cannot enter "in the middle" of a route (it cannot land on a route!)

A train cannot enter in the middle of a route. It has to do so through its first block.	TRN-3

Likewise, a train cannot disappear in the middle of a route (it cannot take off!)

A train cannot leave a route without first occupying then freeing all its blocks	TRN-4

1.11 Failures

In this section, we study a number of abnormal cases which could happen. The fact that their probabilities are very low is not a reason to preclude these cases.

The first and most important case of failures is obviously the one where, for some reasons, the driver of a train does not obey the red signal guarding a route. In section 1.6 we said in requirement ENV_14 that "trains are supposed to stop at red signals". Now, is it always the case?

The solution to this problem is local to the train. This case is detected within the faulty train by a device called the Automatic Train Protection. As soon as this device detects that the train passes a red signal, it automatically activates the emergency brakes of the train. The distance between the signal and the first block of the route it protects is calculated so that we can be sure that the train will stop before entering that first block. Note that this protection is not certain as the Automatic Train Protection could be broken while the train does not stop at a red signal!

Trains are equipped with the Automatic Train Protection system, which guarantees that they cannot enter a route guarded by a red signal	FLR-1

In section 1.10, we claimed in requirement TRN_1 that "a train cannot split while in the network". Is it possible that it happens nevertheless by accident? The solution to this problem is again local to the train. Each train is now equipped with special bindings so that it forms a continuous body that cannot be mechanically broken. Here again, the solution is not certain but professionals claim that the risk is extremely low.

Trains are equipped with special bindings, which guarantee that they cannot be mechanically broken.	FLR-2

Another case raised in section 1.10 is requirement TRN_2 claiming that "a train cannot move backwards while in the network". Here again, the Automatic Train Protection system is used. It detects immediately any backward move and in that case it activates automatically the emergency brakes. But one has to be sure that the train nevertheless does not occupy again a block that it has freed recently. This is guaranteed by the fact that the transmission of the occupancy of a block by the track circuit is slightly delayed. As a consequence, when the train has physically left a block, this fact is not immediately transmitted to the controller, it is only done when the back of the train has moved to a certain distance. If the train slightly moves backwards then it does not occupy again the block since it did not left it (as "seen" from the software controller).

The Automatic Protection System and a slight delay observed by the track circuit guarantee that a train moving backward cannot occupy again a block which has been physically freed.	FLR-3

In section 1.10, we said in requirement TRN-3 that "a train cannot enter in the middle of a route". This is certainly the case for trains. The problem is that the software controller does not "see" trains. It only detects that a block is occupied or free by means of track circuits connections. As a consequence, it is possible that, for some reasons, a block is detected to be occupied by its track circuit because a piece of metal is put on the rail. The software controller can detect such a faulty occupancy. In that case the train agent can take some emergency action. But this is not always the case however. This risk is therefore accepted but not treated here.

The risk of a faulty detection of a block occupancy is not treated	FLR-4

In section 1.10, we said in requirement TRN-4 that "a train cannot leave a route without first occupying then freeing all its block". This is not always the case however in the very rare circumstance where a short train (say a single engine) derails and then falls down: it suddenly quits the block where it is situated! This case can certainly be detected by the software controller and some emergency action can be taken by the train agent. We do not treat this case here however.

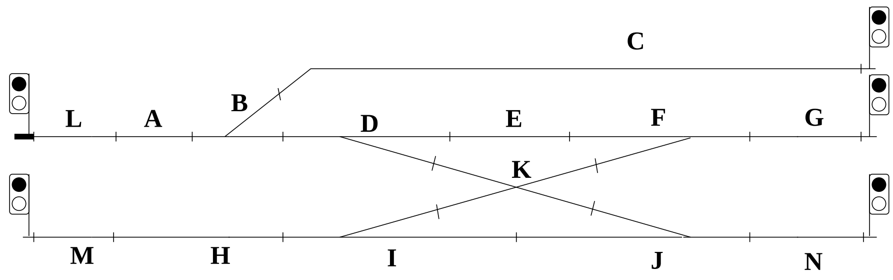
The case where a short train derails and leaves its block is not treated here	FLR-5

Note that the last two cases of failure raise a difficult problem which is the one of restarting the system after an emergency. It seems that the only solution consists in carefully inspecting the network to decide whether a normal situation has been reached again.

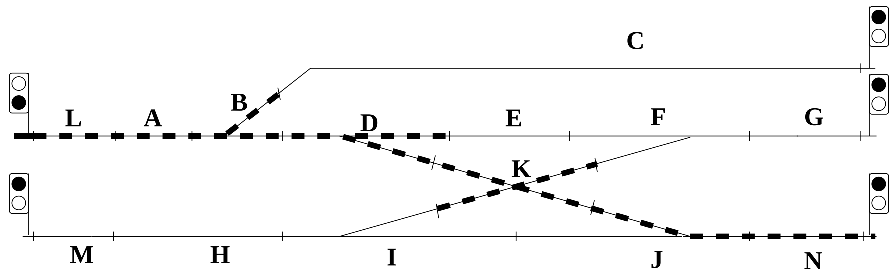
1.12 Examples

In this section, we illustrate the previous concepts with examples of trains safely crossing the network.

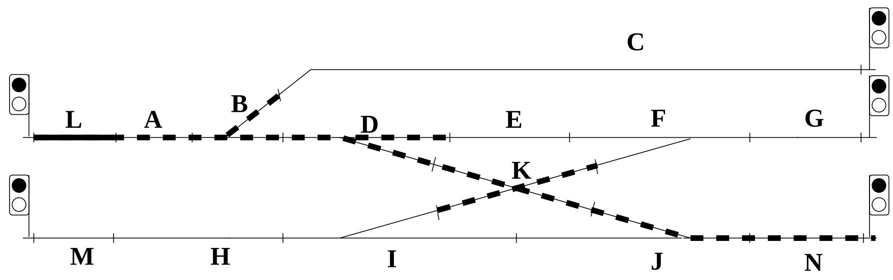
To begin with, we can see a train T1 approaching block L. This is indicated by a little thick line on the left of block L. The train cannot continue since the signal is red.



The train agent is told to form route R3, that is: $L\ A\ B\ D\ K\ J\ N$. This consists in checking that the various blocks of this route are not reserved for another route and that the points are oriented in the proper directions. Once this is done, the corresponding signal is turned green: route R3 is indeed formed. In the following figure[1], you can see the situation just after the "formation" of route R3:



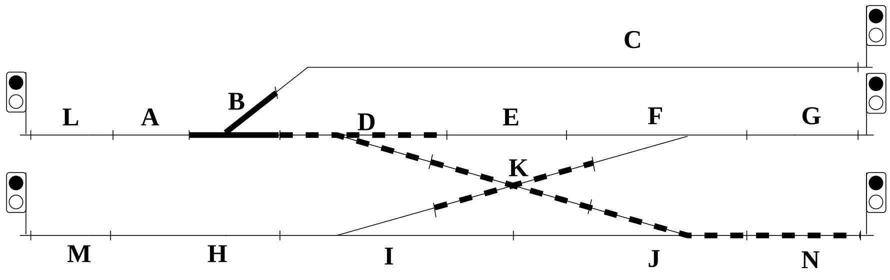
Train T1 now enters route R3. It occupies block L[2]. The signal protecting route $R4$ has been turned to red again so that no train can enter this route any more.



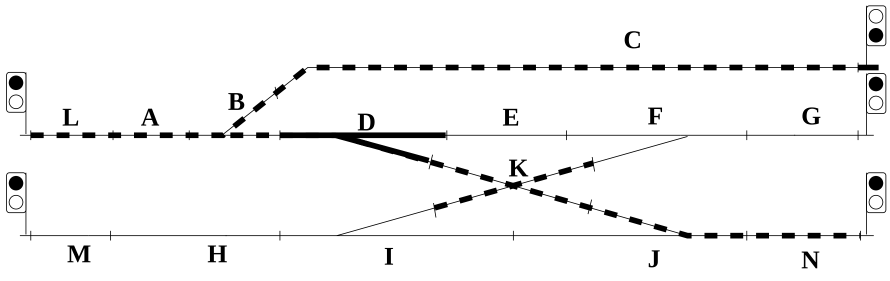
[1] On subsequent, figures, we use the following convention: a reserved, but not yet occupied, block is represented by a thick dashed line (we have already seen in section 1.3 that a reserved and occupied block is represented by a thick plain line)

[2] In this running example, we suppose that trains never occupy more than one block at a time. This is a simplification that does not correspond to the reality: a train can clearly occupy two adjacent blocks when a car is covering the transition between them. We use this simplification here just to make the examples shorter.

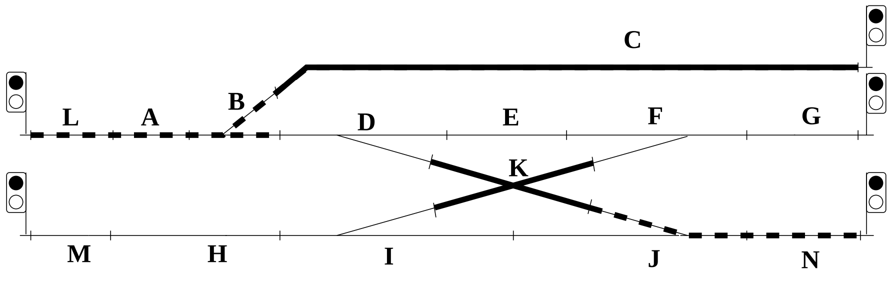
Train T1 moves further to block A (this is skipped) and then to block B. Block L and A are freed as soon as the train does not occupy them any more.



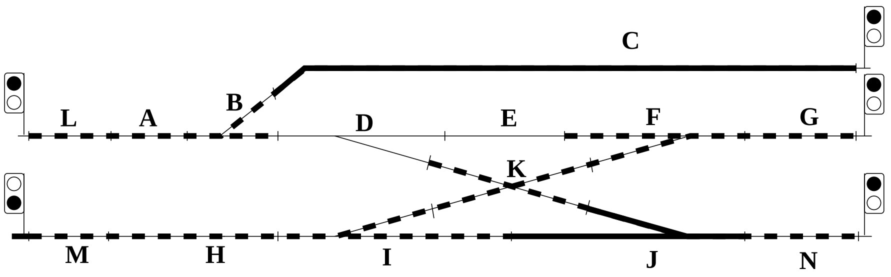
Train T1 moves further to block D. Block B is freed. A second train, T2, arrives and approaches block C. The train agent is told to form route R6, that is: $C \quad B \quad A \quad L$, for that new train. This is possible because there is no conflict with already reserved blocks. The corresponding points are set properly and the signal protecting route R6 is turned green.



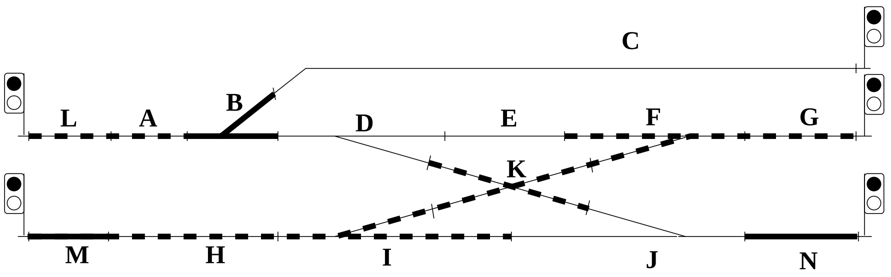
Train T1 moves to block K. Train T2 enters route R6. It occupies block C. The signal protecting route R6 is turned red.:



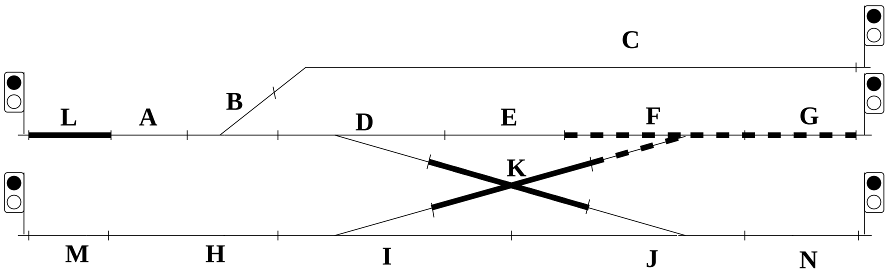
Train T1 now occupies block J and frees block K, and train T2 still occupies block C. A third train, T3, approaches block M. The train agent is told to form route R4, that is: $M \quad H \quad I \quad K \quad F \quad G$, for that new train. This is possible because there is no conflict with already reserved blocks. The corresponding points are set properly and the signal protecting route R4 is turned green.



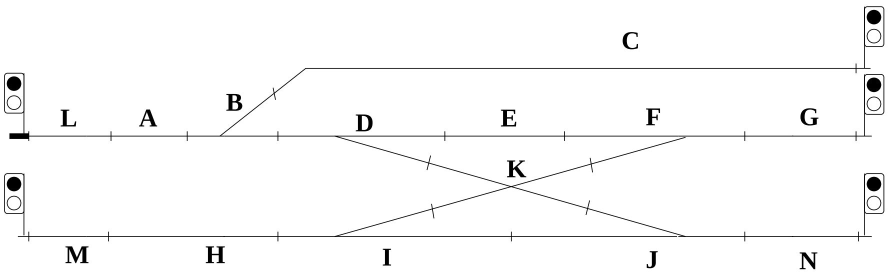
Train T3 enters route R4 by occupying block M. The signal protecting route R4 turns back to red. The other trains are moving in their respective routes.



Train T1 now leaves route R3. Trains T2 and T3 continue to move.



Eventually trains T2 and T3 leave their routes.



2 Refinement Strategy

The summary of the various informal requirements we have seen in previous sections is as indicated below. We have all together 39 requirements. Of course, a real train system might have far more requirements than this: it must be clear that what we are presenting here is only a very simplified version of such a train system.

ENV	Environment	15
FUN	Functional	8
SAF	Safety	4

MVT	Movement	3
TRN	Train	4
FLR	Failure	5

The role of the formal phase which we start now is to build models able to take account of these requirements. As it is out of the question to incorporate all of them at ounce, we are going to proceed by successive approximations, which are called *refinements*. In this section, we define thus the refinement strategy we are going to follow. It is very important indeed to define the order in which we are going to extract the various requirements which have been exhibited in the previous phase.

1. In the initial model, the blocks and routes concepts are formalized. Blocks are defined from a logical point of view however
2. In the first refinement we introduce the physical blocks and thus start to formalize part of the environment. We establish the connection between the logical blocks and the physical ones. This is done in an abstract way however as we do not introduce yet the points.
3. In the second refinement, we introduce the notion of readiness for a route. This corresponds to an abstract view of the green signals.
4. In the third refinement, we introduce the physical signals. We data-refine (implement) the readiness of a route by means of green signals.
5. In the fourth refinement, we introduce the points.
6. Some other refinements are needed in order to finalize details. Such refinements are not treated in this chapter however.

3 Initial Model

3.1 The State

The state is made of a number of carrier sets, constants and variables, which we study in the following sections.

Carrier Sets. The initial model is concerned with blocks and routes. We thus take account of requirement ENV-3 of section 1.4, and of requirement ENV-6 of section 1.5. We do not take account of points or signals for the moment, this will be done in further refinements. We have thus only two carrier sets, B and R, standing for blocks and routes. In what follows, we shall use the convention that carrier sets are named using single upper case letters.

sets: B
R

Constants. The organization of the track network, which is made of a number of routes, is formalized by means of two constant: $rtbl$ (pronounce "routes of blocks") relating routes to blocks and nxt (pronounce "next") relating blocks to blocks for each route.

constants: $rtbl$
nxt

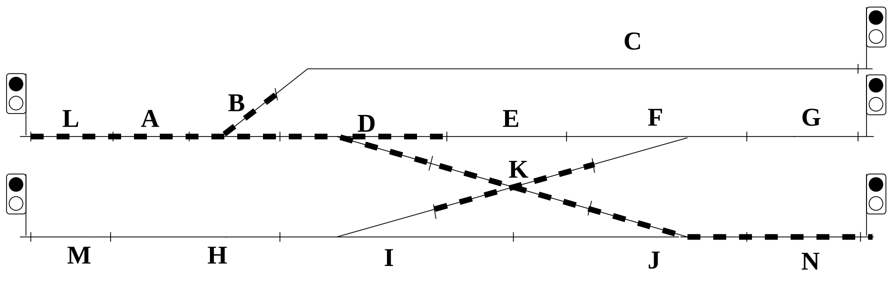
The constant $rtbl$ is a total (all routes are concerned) and surjective (all blocks are concerned) binary relation from B to R (**axm0_1**). This is so because a route may have many blocks and a block can belong to several routes:

axm0_1: $rtbl \in B \leftrightarrow\!\!\!\!\twoheadrightarrow R$

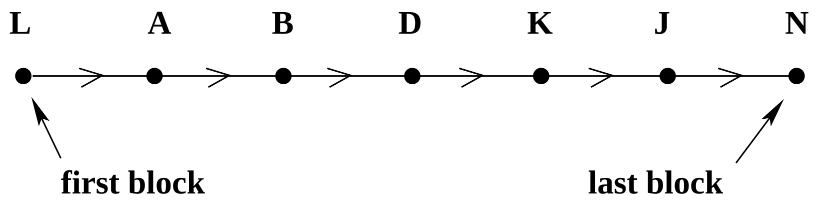
The constant nxt denotes the succession of each blocks associated with a route (ENV-6). This succession forms an injective function from blocks to blocks (that is, a function whose inverse is also a function):

axm0_2: $nxt \in R \rightarrow (B \rightarrowtail\!\!\!\!\!\!\shortmid\;\; B)$

For example, a route like route R3 comprising the following blocks $L \;\; A \;\; B \;\; D \;\; K \;\; J \;\; N$ in that order



is represented as follows by the injective function $nxt(R3)$. As can be seen, the function $nxt(R3)$ establishes a continuous connection between the first block L and last block N of route R3:



As the first and last block of a route will play a certain role in further properties, we have to introduce them explictely in our state by means of some new constants. We thus extend our set of constants by introducing the first and last block of each route r: fst and lst

constants: $\cdots$
fst
lst

These first and last elements of a route enjoy the following obvious properties: they are defined for each route (**axm0_3** and **axm0_4**) and they are genuine blocks of the route (**axm0_5** and **axm0_6**). Moreover the first and last block of a route are distinct (**axm0_7**) :

axm0_3: $fst \in R \to B$

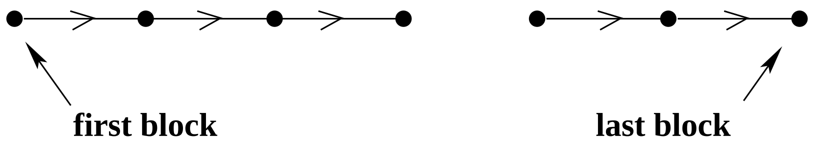
axm0_4: $lst \in R \to B$

axm0_5: $fst^{-1} \subseteq rtbl$

axm0_6: $lst^{-1} \subseteq rtbl$

axm0_7: $\forall r \cdot r \in R \Rightarrow fst(r) \neq lst(r)$

As illustrated in the previous figure, we want the connection represented by the function $nxt(r)$ for each route r, to be continuous as required by requirement ENV-10 of section 1.5. In other words, we want to exclude cases like this, which do not make sense for a route:

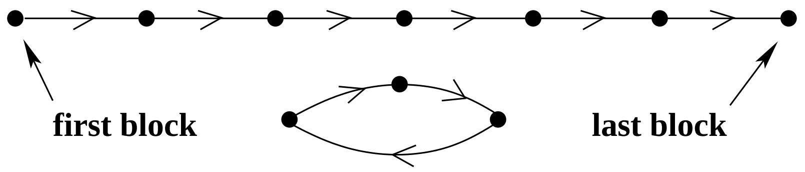


Moreover, we want to express that the blocks of a route r which are present in the domain and range of the injection $nxt(r)$ are exactly the blocks of route r, namely $rtbl^{-1}[\{r\}]$. In order to express all this, we just say that the injection $nxt(r)$ is indeed a bijection from $rtbl^{-1}[\{r\}] \setminus \{lst(r)\}$ to $rtbl^{-1}[\{r\}] \setminus \{fst(r)\}$:

axm0_8: $\forall r \cdot r \in R \;\Rightarrow\; nxt(r) \;\in\; s \setminus \{lst(r)\} \rightarrowtail\!\!\!\!\twoheadrightarrow s \setminus \{fst(r)\}$

where s is $rtbl^{-1}[\{r\}]$

But this is not sufficient, as the following pathological case can happen:



We have then to express that there is no such cycles in the connection. This corresponds to requirement ENV-11 of section 1.5. This can be done by stating that the only subset S of B which is included in its image under $nxt(r)$, that is $nxt(r)[S]$, is the empty set (a genuine cycle is indeed equal to its image under $nxt(r)$):

axm0_9: $\forall r \cdot r \in R \Rightarrow (\forall S \cdot S \subseteq nxt(r)[S] \Rightarrow S = \varnothing)$

A final property of the routes is that they cannot depart or arrive in the *middle* of another one. However several routes can depart from the same block or arrive at the same block. All this corresponds to requirements ENV-8 and ENV-9 of section 1.5. It is expressed by the following two properties:

axm0_10: $\forall r, s \cdot r \in R$
$s \in R$
$r \neq s$
$\Rightarrow$
$fst(r) \notin rtbl^{-1}[\{s\}] \setminus \{fst(s), lst(s)\}$

axm0_11: $\forall r, s \cdot r \in R$
$s \in R$
$r \neq s$
$\Rightarrow$
$lst(r) \notin rtbl^{-1}[\{s\}] \setminus \{fst(s), lst(s)\}$

Note that the previous properties do not preclude two routes to have the same first or last blocks.

Variables. In this initial model, we have four variables named $resrt$, $resbl$, $rsrtbl$, and OCC (see below, the box entitled **variables**). In what follows, we shall use the convention that *physical variables* are named using upper case letters only. By "physical variable", we mean a variable representing part of the external equipment (here OCC denotes the set of physical blocks which are occupied by trains). The other variables (named using lower case letters only) are called the *logical variables*: they represent variables which will be part of the future software controller. The invariants corresponding to all these variables can be seen on the right table below (these invariants are explained below):

variables: $resrt$
$resbl$
$rsrtbl$
OCC

inv0_1: $resrt \subseteq R$

inv0_2: $resbl \subseteq B$

inv0_3: $rsrtbl \in resbl \rightarrow resrt$

inv0_4: $rsrtbl \subseteq rtbl$

inv0_5: $OCC \subseteq resbl$

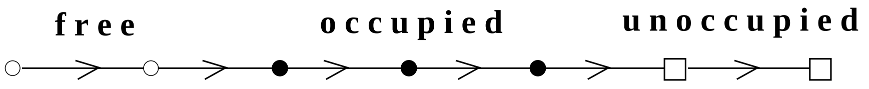
Variable $resrt$ (pronounce "reserved routes") denotes the reserved routes (**inv0_1**): this is coherent with requirement FUN-2 of section 1.7, which says that a route can be reserved for a train.

The second variable, $resbl$ (pronounce "reserved blocks"), denotes the set of reserved blocks (**inv0_2**): this is coherent with requirement FUN-3 of section 1.7, which say that a block can be reserved for a route.

Our third variable, $rsrtbl$ (pronounce "reserved routes of reserved blocks"), relates reserved blocks to reserved routes: it is a *total function* from reserved blocks to reserved routes (**inv0_3**): this is coherent with requirement SAF-1 stating that a block cannot be reserved for more than one route. Of course, this connection is compatible with the static relationship $rtbl$ between blocks and routes (**inv0_4**): a reserved block for a route is a block of that route.

Finally, variable OCC denotes the set of occupied blocks: this is coherent with requirement ENV-5 stating that a block might be occupied by a train. Such occupied blocks are obviously reserved for some route (**inv0_5**): this is coherent with requirement FUN-4 of section 1.7 stating that an occupied block is always reserved.

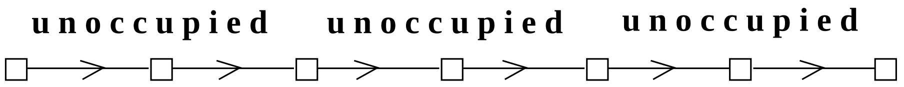
We have to define now more invariants corresponding to the way a train can occupy a reserved route. The general situation is illustrated on the following figure:



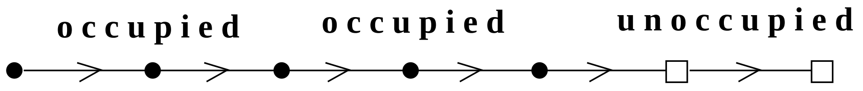
The blocks of a reserved route are divided in three areas:

1. In the first area (on the left, where the blocks are represented by white circles), the blocks are freed by that route because the train does not occupy them any more. They can readily be reused (and maybe they are already) for another reserved route.
2. In the second area (in the center, where the blocks are represented by black circles), the blocks are all reserved and occupied by a train.
3. In the third area (on the right, where the blocks are represented by white squares), the block are all reserved but not occupied yet by a train.

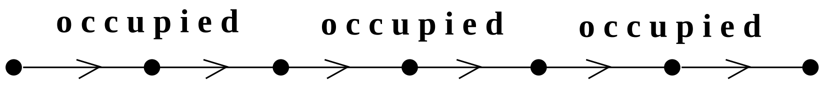
There are other situations corresponding to some special cases of the general situation depicted in the previous figure. In the first special case, areas 1 and 2 are empty: the route is reserved but the train has not yet entered the route:



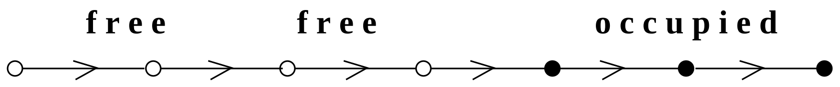
The second special case is the one where area 1 is empty, but not areas 2 and 3. In fact, the train is entering the route as illustrated in the following figure:



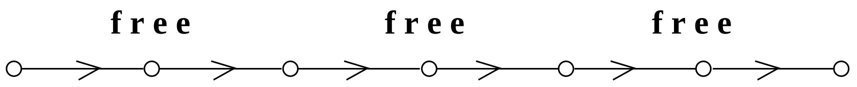
A third special case is one where a (long) train occupies all blocks in a route:



The fourth special case it the one where the train is leaving the route:



The last special case is the one where all blocks in the reserved route have been freed by that route. The route itself is then ready to be freed



More formally, let us call M the set of *free blocks* in a reserved route (those behind the train), N the set of *occupied blocks* in a reserved route, and finally P the set of reserved but *unoccupied blocks* of a reserved route (those situated in front of a train). Sets M, N, and P are formally defined as follows for a given reserved route r:

$$\begin{aligned} M &= rtbl^{-1}[\{r\}] \setminus rsrtbl^{-1}[\{r\}] \\ N &= rsrtbl^{-1}[\{r\}] \;\cap\; OCC \\ P &= rsrtbl^{-1}[\{r\}] \setminus OCC \end{aligned}$$

Note that M, N, and P partition $rtbl^{-1}[\{r\}]$. According to the previous presentation, the only transitions that are allowed are the following:

$$M \to M \qquad M \to N \qquad N \to N \qquad N \to P \qquad P \to P$$

This can be represented by the following conditions

$$nxt(r)[M] \;\subseteq\; M \cup N \qquad nxt(r)[N] \;\subseteq\; N \cup P \qquad nxt(r)[P] \;\subseteq\; P$$

Such conditions are equivalent to the following ones (since $nxt(r)[rtbl^{-1}[\{r\}]]$ is included in $rtbl^{-1}[\{r\}]$ according to **axm0_8**):

$$nxt(r)[M] \cap P = \varnothing \qquad nxt(r)[N \cup P] \;\subseteq\; N \cup P \qquad nxt(r)[P] \;\subseteq\; P$$

All this is eventually formalized in the following invariants:

inv0_6: $\forall r \cdot r \in R \;\Rightarrow\; nxt(r)[rtbl^{-1}[\{r\}] \setminus s] \;\cap\; (s \setminus OCC) \;=\; \varnothing$

inv0_7: $\forall r \cdot r \in R \;\Rightarrow\; nxt(r)[s] \;\subseteq\; s$

inv0_8: $\forall r \cdot r \in R \;\Rightarrow\; nxt(r)[s \setminus OCC] \;\subseteq\; s \setminus OCC$

where s is $rsrtbl^{-1}[\{r\}]$

These invariants are coherent with the train requirements TRN-1 to TRN-4 defined in section 1.10.

3.2 The Events

The four variables $resrt$, $resbl$, $rsrtbl$, and OCC are initialized to the empty set: initially, no trains are in the network and no routes or blocks are reserved. Besides the initialization event (which we do not present here), we have five normal events. Events define transitions which can be observed. In what follows, we shall use the convention that physical events corresponding to transitions occurring in the environment are named using upper case letters only. Here are the events of the initial model:

- route_reservation,
- route_freeing,
- FRONT_MOVE_1,
- FRONT_MOVE_2,
- BACK_MOVE.

Event route_reservation corresponds to the reservation of a route r. It is done on an unreserved route (i.e. $r \in R \setminus resrt$) whose blocks are not already reserved for a route (i.e. $rtbl^{-1}[\{r\}] \cap resbl = \varnothing$). Route r is then reserved together with its blocks. This is coherent with requirement FUN-5 which says that a route can be reserved as soon as all its blocks are themselves reserved.

```
route_reservation
  any $r$ where
    $r \in R \setminus resrt$
    $rtbl^{-1}[\{r\}] \cap resbl = \varnothing$
  then
    $resrt := resrt \cup \{r\}$
    $rsrtbl := rsrtbl \cup rtbl \rhd \{r\}$
    $resbl := resbl \cup rtbl^{-1}[\{r\}]$
  end
```

```
route_freeing
  any $r$ where
    $r \in resrt \setminus \mathrm{ran}(rsrtbl)$
  then
    $resrt := resrt \setminus \{r\}$
  end
```

Event route_freeing makes a reserved route free when it does not contain reserved blocks any more. This is coherent with requirement MVT-3 which says that a route can be made free when no blocks are reserved for it any more.

Event FRONT_MOVE_1 corresponds to a train entering a reserved route r. The first block of r must be reserved and unoccupied. Moreover, the reserved route corresponding to the first block of r must be r itself. The first block is made occupied:

```
FRONT_MOVE_1
  any $r$ where
    $r \in resrt$
    $fst(r) \in resbl \setminus OCC$
    $rsrtbl(fst(r)) = r$
  then
    $OCC := OCC \cup \{fst(r)\}$
  end
```

```
FRONT_MOVE_2
  any $b, c$ where
    $b \in OCC$
    $c \in B \setminus OCC$
    $b \mapsto c \in nxt(rsrtbl(b))$
  then
    $OCC := OCC \cup \{c\}$
  end
```

Event FRONT_MOVE_2 corresponds to the occupancy of a block which happens to be different from the first block of a reserved route. Given a block b which is occupied and preceded (in the same route) by a block, say c, which is not occupied, then c is made occupied.

Finally, event BACK_MOVE corresponds to the move of the rear part of the train. This happens for a block b which is occupied and is the last block of a train. This is detected when block b has a follower in the route r reserved for b and that follower, if reserved, is not reserved for r (this corresponds to the big implicative guard). Moreover, when b has a predecessor, that predecessor must be occupied so that the train does not disappear before reaching the end of route r (this corresponds to the last guard). The action corresponding to that event makes b unoccupied and unreserved. This is coherent with requirement MVT-1 which says that "once a block of a formed route is made unoccupied, it is also freed":

```
BACK_MOVE
  any $b, n$ where
    $b \in OCC$
    $n = nxt(rsrtbl(b))$
    $\begin{pmatrix} b \in \mathrm{ran}(n) \wedge \\ n^{-1}(b) \in \mathrm{dom}(rsrtbl) \\ \Rightarrow \\ rsrtbl(n^{-1}(b)) \neq rsrtbl(b) \end{pmatrix}$
    $b \in \mathrm{dom}(n) \Rightarrow n(b) \in OCC$
  then
    $OCC := OCC \setminus \{b\}$
    $rsrtbl := \{b\} \lhd rsrtbl$
    $resbl := resbl \setminus \{b\}$
  end
```

Important Remark. It might seem strange at first glance (and even incorrect) to have physical events such as FRONT_MOVE_1, FRONT_MOVE_2, and BACK_MOVE using non-physical variables in their guards. Clearly, a physical event can be enabled under certain conditions depending on physical variables only: a physical event cannot magically "see" the non-physical variables. The reason for having non-physical variables in the guards here is that we are still in an abstract version where such abnormalities are possible. Of course, in the final refined version of physical events we have to check that it is not the case any more.

4 First Refinement

In this first refinement, we introduce the *physical tracks*. So that the movements of the train will correspond entirely on the physical situation of the track. Note however that we do not introduce yet the points and the signals.

4.1 The State

We do not introduce new carrier sets or new constants in this refinement.

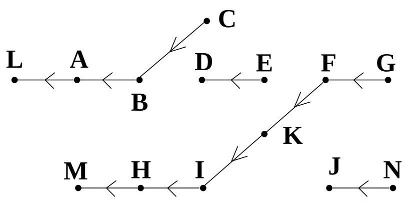
Variables. In this refinement, we have three new variables named TRK (pronounce "track"), frm (pronounce "formed routes"), and LBT (pronounce "last blocks of trains"). Notice that the variables introduced in the initial models, namely $resrt$, $resbl$, $rsrtbl$, and OCC, are kept in this refinement.

```
variables:  $\cdots,$
            $TRK$
            $frm$
            $LBT$
```

Variable TRK is a partial injection (**inv1_1**) from blocks to blocks defining the *physical succession of blocks*. It also contains the direction taken by trains following the tracks. Note that this last information is not "physical" (you cannot "see" it on the track), it corresponds however to the physical movements of trains on the physical tracks. Next is the invariant defining variable TRK as an injective function.

inv1_1: $TRK \in B \rightarrowtail\!\!\!\!\!\!+\;\; B$

Here is an illustration of the variable TRK in a certain situation:



As can be seen, route $R9$ ($G\ \ F\ \ K\ \ I\ \ H\ \ H\ \ M$) is now established on the physical track. In section 4.2, we shall see how the event, which is positioning the points will modify this situation. Note that the crossing in block K is "broken" and that the physical track "remembers" the direction followed by trains circulating on it: of course, this is not what happen in the real tracks, but this is a convenient abstraction.

Finally, all pairs belonging to TRK also belong to $nxt(r)$ for some route r (**inv_2**):

inv1_2: $\forall x, y \cdot x \mapsto y \in TRK \;\Rightarrow\; (\exists r \cdot r \in R \;\wedge\; x \mapsto y \in nxt(r))$

Variable frm represents the set of formed routes: it is a subset of the reserved routes (**inv1_3**) This is coherent with requirement FUN-7 which says that "a formed route is always a reserved route". We have a number of invariants involving the formed routes. The reserved routes of occupied blocks are formed routes (**inv1_4**). A route r which is reserved but not yet formed is such that its reserved blocks are exactly the constant reserved blocks associated with r (**inv1_5**). The two previous invariants are coherent with requirements SAF-4 which says that "no blocks of a reserved but not yet formed route are occupied"

inv1_3: $frm \subseteq resrt$

inv1_4: $rsrtbl[OCC] \subseteq frm$

inv1_5: $\forall r \cdot r \in resrt \setminus frm \Rightarrow rtbl \rhd \{r\} = rsrtbl \rhd \{r\}$

Now comes the most important invariant (**inv1_6**): it relates the logical succession of blocks in a route (represented by the function $nxt(r)$ for each route r) to the physical tracks on the terrain (represented by the variable TRK). It says that for each formed route r, the logical succession of blocks (where the train is supposed to be and where it has to go when proceeding through route r) *agrees with the physical tracks on the terrain*. In other words, when a route r is formed, then the portion of the physical blocks where the train is or where it will be in the future when proceeding along this route, this portion of the physical blocks corresponds to what is expected in the logical blocks as recorded by the controller.

$$\textbf{inv1\_6:}\quad \forall r \cdot r \in frm \Rightarrow rsrtbl^{-1}[\{r\}] \lhd nxt(r) = rsrtbl^{-1}[\{r\}] \lhd TRK$$

Finally, variable LBT denotes the set of blocks occupied by the back of each train: this is also a "physical" variable like variable TRK. The first invariant (**inv1_7**) concerning this variable, quite naturally says that the last block of a train is indeed occupied by a train:

$$\textbf{inv1\_7:}\quad LBT \subseteq OCC$$

And now we state (**inv1_8**) that the last block b of a train, if it has a follower a on its route, then a, if reserved, is not reserved for the route of b.

$$\begin{array}{ll}\textbf{inv1\_8:} & \forall a, b \cdot b \in LBT \\ & b \in \operatorname{ran}(nxt(rsrtbl(b))) \\ & a = nxt(rsrtbl(b))^{-1}(b) \\ & a \in \operatorname{dom}(rsrtbl) \\ & \Rightarrow \\ & rsrtbl(a) \neq rsrtbl(b)\end{array}$$

Thanks to the introduction of the physical variables TRK and LBT we shall be able to define the movements of the train based only on what the train find on the terrain, namely the physical blocks. Notice that a train "knows" that the last part of it occupies a block belonging to LBT.

4.2 The Events

Event route_reservation is not modified in this refinement.Other events are modified as shown below. We also introduce two more events:

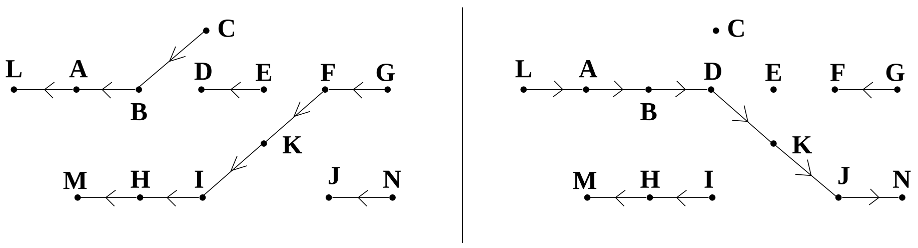
- point_positioning,
- route_formation

Event point_positioning is still very abstract in this refinement. It conveys however the essence of the communication between the future software and the outside equipment: the physical TRK is modified according to the logical route $nxt(r)$. This event is coherent with requirement SAF-3 which says that "a point can *only be re-positioned* if it belongs to a block which is in a reserved but not yet formed route". In further refinements, this modification of the physical track will correspond to the controller action modifying the point positions:

```
point_positioning
  any $r$ where
    $r \in resrt \setminus frm$
  then
    $TRK := (\operatorname{dom}(nxt(r)) \lhd\!\!- TRK \,-\!\!\rhd \operatorname{ran}(nxt(r))) \cup nxt(r)$
  end
```

As can be seen, this logical event has an effect on the physical variable TRK. This is due to the fact that this event is effectively changing (at ounce for the moment) the physical position of the points of route r.

Next is an illustration of the physical situations just before and just after an occurrence of event point_positioning. As can be seen, after this occurrence we have three properties: (1) route $R3$ $(L\ A\ B\ K\ J\ N)$ is established on the physical track, (2) the points have been modified accordingly, and (3) the crossing situated in block K has been "reorganized":



Event route_formation explains when a route r can be "formed", namely when the physical and logical track agree, that is after event point_positioning has acted on route r.

```
route_formation
  any  $r$  where
    $r \in resrt \setminus frm$
    $rsrtbl^{-1}[\{r\}] \lhd nxt(r) \;=\; rsrtbl^{-1}[\{r\}] \lhd TRK$
  then
    $frm := frm \cup \{r\}$
  end
```

It can be seen that this event refers to the physical variable TRK in its guard.This is due to the fact that this event is enabled when the controller detects (here at ounce for the moment) that all points of route r are correctly positioned.

Event route_freeing is slightly extended by making the freed route not formed any more. This is coherent with requirement MVT_2, which says that "a route remains formed as long as there are some reserved blocks in it" and MVT-3, which says that "a formed route can be made free (not formed and not reserved any more) when no blocks are reserved for it any more".

```
route_freeing
  any  $r$  where
    $r \in resrt \setminus \mathrm{ran}(rsrtbl)$
  then
    $resrt := resrt \setminus \{r\}$
    $frm := frm \setminus \{r\}$
  end
```

Event FRONT_MOVE_1 is only slightly modified for the moment as we have not introduced the signals yet: this will be done in further refinements. The present modification consists in extending the set LBT by adding to it the singleton $\{fst(r)\}$. As a matter of fact, when a train is entering a route, the last block of the train for that route is certainly the first block of the route until that block is freed when the back of the train will move in event BACK_MOVE.

```
FRONT_MOVE_1
  any $r$ where
    $r \in frm$
    $fst(r) \in resbl \setminus OCC$
    $rsrtbl(fst(r)) = r$
  then
    $OCC := OCC \cup \{fst(r)\}$
    $LBT := LBT \cup \{fst(r)\}$
  end
```

```
FRONT_MOVE_2
  any $b$ where
    $b \in OCC$
    $b \in \mathrm{dom}(TRK)$
    $TRK(b) \notin OCC$
  then
    $OCC := OCC \cup \{TRK(b)\}$
  end
```

Event FRONT_MOVE_2 is now following the physical situation on the real track. We shall have to prove that it refines its abstraction however. As can be seen, all guards are now defined in terms of physical variables.

Event BACK_MOVE is split into two events. Event BACK_MOVE_1 corresponds to the last block of the train leaving the route. Event BACK_MOVE_2 corresponds to the last block of the train progressing in the route.

```
BACK_MOVE_1
  any $b$ where
    $b \in LBT$
    $b \notin \mathrm{dom}(TRK)$
  then
    $OCC := OCC \setminus \{b\}$
    $rsrtbl := \{b\} \lhd rsrtbl$
    $resbl := resbl \setminus \{b\}$
    $LBT := LBT \setminus \{b\}$
  end
```

```
BACK_MOVE_2
  any $b$ where
    $b \in LBT$
    $b \in \mathrm{dom}(TRK)$
    $TRK(b) \in OCC$
  then
    $OCC := OCC \setminus \{b\}$
    $rsrtbl := \{b\} \lhd rsrtbl$
    $resbl := resbl \setminus \{b\}$
    $LBT := (LBT \setminus \{b\}) \cup \{TRK(b)\}$
  end
```

Remark 1. As can be seen, the guards of physical events FRONT_MOVE_2, BACK_MOVE_1, and BACK_MOVE_2 are all now involving physical variables only (remember our "important remark" at the end of section 3.2). It is still not the case for event FRONT_MOVE_1 however. Only wait until refinement 3 in section 6 where we shall see that event FRONT_MOVE_1 will be enabled as a consequence of a green signal, which clearly is a physical condition.

Remark 2. We notice that physical events BACK_MOVE_1 and BACK_MOVE_2 both make reference to some non-physical variables in their action part ($rsrtbl$ and $resbl$). We wonder whether this is allowed. It would seem obvious that a physical event cannot modify a controller variables. The reason to have some non-physical variables still present in the action parts of these events is because these events *have still to be decomposed* into two events: the "pure" physical event and a corresponding event in the controller. The reason can clearly be seen here: when the train does a physical back move, the controller has to react by freeing the corresponding logical block. The connection between the physical move and the (separate) logical reaction in the controller will be done later (in some refinement step to be done, but not presented in this chapter) by having the physical track circuit *sending a message to the controller* when it is physically made unoccupied. Upon receiving this message, the controller can then react.

Remark 3. Notice that both events FRONT_MOVE_1 and FRONT_MOVE_2 do not make any reference in their action part to some non-physical variables. It means that such events have no influence on the controller. This is quite understandable, when the front of the train proceeds, we have nothing to do in the controller whereas when the back of the train proceeds we have something to do (block freeing).

5 Second Refinement

In this refinement, we introduce the notion of *readiness* for a route. A route is ready when it is able to accept a new train. In the next refinement, we shall introduce the signals. As we shall see, the ready routes will have a green signal.

5.1 The State

We do not introduce new carrier sets or new constants.

Variables. In this refinement, we introduce the new variable rdy which denotes the set of ready routes.

variables: $\cdots,$
rdy

Here are the basic properties of a ready route. A ready route is one which is formed (**inv2_1**), has all its blocks reserved for it (**inv2_2**), and has all its blocks unoccupied (**inv2_3**):

inv2_1: $rdy \subseteq frm$

inv2_2: $\forall r \cdot r \in rdy \Rightarrow rtbl \rhd \{r\} \subseteq rsrtbl \rhd \{r\}$

inv2_3: $\forall r \cdot r \in rdy \Rightarrow \mathrm{dom}(rtbl \rhd \{r\}) \cap OCC = \varnothing$

5.2 The Events

Events point_positioning, route_reservation, route_freeing, FRONT_MOVE_2, BACK_MOVE_1, and BACK_MOVE_2 are not modified in this refinement, they are thus not copied below. Event route_formation is extended by making the corresponding route ready besides being formed (this action was performed in the previous refinement).

```
route_formation
  any $r$ where
    $r \in resrt \setminus frm$
    $rsrtbl^{-1}[\{r\}] \lhd nxt(r) = rsrtbl^{-1}[\{r\}] \lhd TRK$
  then
    $frm := frm \cup \{r\}$
    $rdy := rdy \cup \{r\}$
  end
```

The guards of event FRONT_MOVE_1 are simplified (and made stronger) by stating that the route r is a ready route (this event will be further simplified in the next refinement where we introduce the signals). We put the abstract version of this event next to the refined one to show the differences between the two guards:

```
(abstract-)FRONT_MOVE_1
  any $r$ where
    $r \in frm$
    $fst(r) \in resbl \setminus OCC$
    $rsrtbl(fst(r)) = r$
  then
    $OCC := OCC \cup \{fst(r)\}$
    $LBT := LBT \cup \{fst(r)\}$
  end
```

```
(concrete-)FRONT_MOVE_1
  any $r$ where
    $r \in rdy$
    $rsrtbl(fst(r)) = r$
  then
    $OCC := OCC \cup \{fst(r)\}$
    $LBT := LBT \cup \{fst(r)\}$
    $rdy := rdy \setminus \{r\}$
  end
```

6 Third Refinement

In this refinement, we define the signals. The role of a signal is to express, when green, that a route is ready.

6.1 The State

Carrier Sets. We introduce the new carrier set S defining the signals.

sets: B, R, S

Constants. In this refinement, we define one constant named SIG (pronounce "signal of first block"). This constant yields the unique signal associated with the first block of a route (**axm3_1**). This corresponds to requirements ENV-12 and ENV-14 of section 1.6. It is a bijection since every signal is uniquely associated with the corresponding first block of a route and vice-versa. Notice that routes sharing the same first block share the same signal.

constants: $\cdots$
SIG

axm3_1: $SIG \in \mathrm{ran}(fst) \rightarrowtail\!\!\!\!\twoheadrightarrow S$

Variables. In this refinement, we introduce the variable GRN denoting the set of green signals (**inv3_1**). This variable data-refines variable rdy which disappears. The connection between the two is established by saying that signals of the first blocks of ready routes are exactly the green signals (**inv3_2**). We have thus established a correspondence between the abstract notion of ready routes and the physical notion of green signals.

variables: $\cdots$
GRN

inv3_1: $GRN \subseteq S$

inv3_2: $SIG[fst[rdy]] = GRN$

6.2 The Events

The only two events that are modified in this refinement are events route_formation and FRONT_MOVE_1. Event route_formation is refined by turning to green the signal associated with the first block of the newly formed route. This is coherent with requirement FUN-8 which says that "once it is formed, a route is made available for the incoming train by turning its signal to green". This event is also coherent with requirement SAF-2, which says that "the signal of a route can *only be green* when all blocks of that route are reserved for it and are unoccupied". This is due to invariant **inv3_2** equating the blocks with green signal with ready routes, and invariants **inv2_2** and **inv2_3** telling that ready routes have all their blocks reserved and unoccupied.

```
route_formation
  any $r$ where
    $r \in resrt \setminus frm$
    $rsrtbl^{-1}[\{r\}] \lhd nxt(r) = rsrtbl^{-1}[\{r\}] \lhd TRK$
  then
    $frm := frm \cup \{r\}$
    $GRN := GRN \cup \{SIG(fst(r))\}$
  end
```

This logical event acts on the physical variable GRN. It corresponds to the controller sending a command to turn the physical signal of the first block of route r to green.

Event FRONT_MOVE_1 now reacts to a green signal rather than to a ready route as in the previous refinement. We take at last account of requirement ENV-13.

```
FRONT_MOVE_1
  any $b$ where
    $b \in \mathrm{dom}(SIG)$
    $SIG(b) \in GRN$
  then
    $OCC := OCC \cup \{b\}$
    $LBT := LBT \cup \{b\}$
    $GRN := GRN \setminus \{SIG(b)\}$
  end
```

As can be seen, the physical movement of trains follows the indication of green signals. Note that a green signal is *automatically turned red* when the train enters the corresponding block: this is coherent with requirement ENV-15.

7 Fourth Refinement

7.1 The State

In this refinement, we introduce the points from an abstract point of view for the moment. They are denoted by the set of blocks which contain points. We know from requirement ENV-4 that a block may contain at most one special component: point or crossing.

Constants. We introduce three constants in this refinement: $blpt$, lft, and rht. Constant $blpt$ (pronounce "blocks with points") denotes the set of blocks containing points (**axm4_1**). Each block b containing a point is connected to another block situated on the left of b and another block situated on its right. This is represented by two total functions lft and rht from $blpt$ to B (**axm4_2** and **axm4_3**). Notice that the two function lft and rht are disjoint (**axm4_4**) because a block cannot be situated simultaneously to the left and to the right of a point:

constants: $\cdots$
$blpt$,
lft,
rht

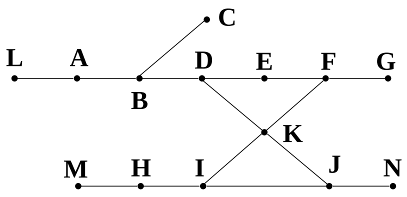
axm4_1: $blpt \subseteq B$

axm4_2: $lft \in blpt \to B$

axm4_3: $rht \in blpt \to B$

axm4_4: $lft \cap rht = \varnothing$

Let us recall our usual example network:



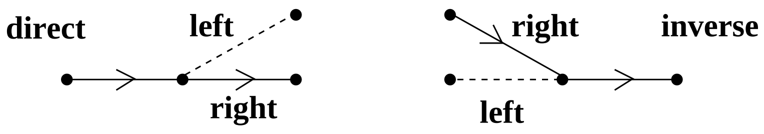
Next are the set $blpt$ and both functions lft and rht corresponding to this example:

$$blpt = \{B, D, F, I, J\}$$

$$lft = \{B \mapsto C,\ D \mapsto E,\ F \mapsto K,\ I \mapsto K,\ J \mapsto I\}$$

$$rht = \{B \mapsto D,\ D \mapsto K,\ F \mapsto E,\ I \mapsto J,\ J \mapsto K\}$$

Each point situated in a route is either in the "direct" or "inverse" direction of this route. This is illustrated in the following figure where you can see fragments of two routes: on the left, we have a point oriented "direct-right", and on the right we have a point oriented "inverse-right".



More precisely, a point is represented in a route by either the left or the right connection, and also on the direct direction of the route or the inverse one. For example, in route $R2$ ($L\ A\ B\ D\ E\ F\ G$), there are three points: in B, in D, and in F. The one in B is direct and represented by the pair $B \mapsto D$ which is a member of rht, the one in D is direct and represented by the pair $D \mapsto E$ which is a member of lft, and finally the one in F is inverse and represented by the pair $F \mapsto E$ which is a member of rht. The connection of each point-block to the next one in a route must be functional (since the point is either in the right or in the left position). This can be formalized as follows:

axm4_5: $\forall r \cdot r \in R \Rightarrow (lft \cup rht) \cap (nxt(r) \cup nxt(r)^{-1}) \in blpt \nrightarrow B$

Notice that the position of each point relative to a given route r is the following: $(lft \cup rht) \cap (nxt(r) \cup nxt(r)^{-1})$.

We also have to add a technical property saying that there is no point in the first or last block of a route (**axm4_6** and **axm4_7**):

axm4_6: $blpt \cap \mathrm{ran}(fst) = \varnothing$

axm4_7: $blpt \cap \mathrm{ran}(lst) = \varnothing$

Variable. We have no new variable in this refinement, only a new invariant expressing that the point positioning is, as expected, functional in the real track. This is expressed by invariant **inv4_1** below

inv4_1: $(lft \cup rht) \cap (TRK \cup TRK^{-1}) \in blpt \nrightarrow B$

Notice that the function is partial only: this is due to the crossing. It is not difficult to prove that this invariant is maintained by event point_positioning, which is recalled below:

```
point_positioning
  any $r$ where
    $r \in resrt \setminus frm$
  then
    $TRK := (\mathrm{dom}(nxt(r)) \lhd TRK \rhd \mathrm{ran}(nxt(r))) \cup nxt(r)$
  end
```

A few additional refinements are clearly needed in order to complete this modelling development. It should contain the decomposition of events route_reservation, route_formation, and point_positioning in more atomic events so as to construct corresponding loops.

8 Conclusion

As was said in the introduction, this chapter contains more material on fault prevention than on fault tolerance. This is essentially due to the problem at hand were faults have to be avoided by all means. But faults can happen as was explained in section 1.11, so it is interesting to see how this could have been taken into account in the modelling process.

It would not have been difficult to incorporate the Automatic Train Protection System (alluded above in section 1.11) within the formal models because we have a global approach taking account of the environment. This would take care of requirements FLR-1 (drivers passing a red signal) and FLR-3 (trains moving backwards) which are protected by the Automatic Train Protection System.

As much as I understand form experts, the other failures are not treated simply because people consider that their probability is extremely low. However, such failures could sometimes be detected in the case of FLR-4 (wrong block occupancy) and that of FLR-5 (train leaving a block). In these cases, the controller has to stop the system by not allowing any signal to be turned green and by not doing any point positioning. This default phase is to last until the environment is inspected and the system is reset. It would be also very easy to model this.

What we have presented here is very close to similar studies done in [1] and [3]. The approach of [1] itself follows from original approaches done in the past by applying the "Action System" methodology [2]. The important lesson learned from Action System is the idea of reasoning at a global level by introducing not only the intended software into the picture but also its *physical environment*.

In the present study, we insisted on the preliminary informal phase consisting in presenting the structured "Definitions and Requirements" of the system we want to build. We think that it is extremely important from a methodological point of view as it is quite frequently a very weak point in similar industrial applications. It seems that we have also made a more complete mathematical treatment of the track network model.

References

1. M. Butler. *A System-based Approach to the Formal Development of Embedded Controllers for a Railway.*. Design Automation for Embedded Systems, 2002.
2. M. Butler, E. Sekerinski, and K. Sere. *An Action System Approach to the Steam Boiler Problem.* In Formal Methods for Industrial Applications. LNCS Volume 1165. Springer, 1996.
3. A.E. Haxthausen and J. Peleska. *Formal Development and Verification of a Distributed Railway Control System.* FM'99, LNCS Volume 1709. Springer, 1999.