Improving formal analysis of computerised rail traffic control systems using domain models

Karin Ahlman
Abstract

Improving formal analysis of computerised rail traffic control systems using domain models

Karin Ahlman

During the formal analysis of a computerized railway control system, it may be difficult to understand if a found counterexample to a requirement is a scenario which can happen in the real world or not. By putting sensible constraints on the inputs to the system, i.e. by defining a domain model for the system, some impossible scenarios are excluded from the formal analysis, which means that the formal analysis is simplified. This thesis presents a domain model for railway control systems, expressing constraints on how trains can behave in a railway network. The railway network is abstracted into a simple graph structure and the model is described in a temporal predicate logic using operators for the initial (I) and the next (X) value. The model is carefully defined in order not to introduce any unrealistic behavior.
# Contents

1 Sammanfattning på svenska 4

2 Introduction 5
   2.1 The need for reliable control systems in the railway domain 5
   2.2 Formal verification 5
   2.3 Domain models 6
   2.4 Goal 7

3 The underlying system 8
   3.1 The railway net - a graph 8
   3.1.1 Orientation 8
   3.2 Computerized railway control systems 9
      3.2.1 Detection - track circuits 9
      3.2.2 Route setting 9
   3.3 Cycle time 9

4 Tools and methods 10
   4.1 The language 10
   4.2 The model 10
      4.2.1 Describing the state - combinatorial part 11
      4.2.2 Describing the transitions - temporal part 11
      4.2.3 Division into static and dynamic predicates 12

5 Model 1: Occupied blocks 13
   5.1 The railway graph 13
   5.2 Static properties of the railway graph 14
      5.2.1 Block-types 14
   5.3 Dynamic properties of the railway graph 15
      5.3.1 Constraints 16
   5.4 Train movements in the railway graph 16
      5.4.1 Movements of trains 16
      5.4.2 Constraints on \(\text{Occ}(b)\) 17

6 Model 2: Trains and positions 18
   6.1 The position graph 18
   6.2 Dynamic properties of the position graph 19

7 Union of the two models 20
   7.1 Resolve conflicts 20
8 Analysis
  8.1 Configurations ........................................... 22
  8.2 Formal verification ....................................... 23
  8.3 Analysis of the first model .............................. 23
  8.4 Analysis of the union .................................... 24

9 Related work .................................................. 25
  9.1 Towards a TRain Book - for the RAilway DomaiN .......... 25
  9.2 Comparison .................................................. 25

10 Conclusions ................................................ 27
  10.1 Summary .................................................... 27
  10.2 Reflections ................................................ 27
  10.3 Future work ............................................... 28

A Formulas for model 1 ........................................ 29
  A.1 Domain ...................................................... 29
  A.2 Static predicates .......................................... 29
  A.3 Dynamic predicates ....................................... 30
  A.4 Constraints for static predicates .......................... 30
  A.5 Constraints for dynamic predicates ....................... 31

B Formulas for model 2 ........................................ 32
  B.1 Domain ...................................................... 32
  B.2 Static predicates .......................................... 32
  B.3 Dynamic predicates ....................................... 32
  B.4 Constraints for static predicates .......................... 32
  B.5 Constraints for dynamic predicates ....................... 32

C Formulas for the union of the two models .................. 33
  C.1 Static predicates .......................................... 33
  C.2 Constraints ................................................. 33
Chapter 1

Sammanfattning på svenska


Genom att formulera en mängd av begränsande regler som beskriver hur tägen kan röra sig skulle man kunna förenkla den formella verifieringen genom att orealistiska motexemplet utesluts. En sådan mängd av regler betecknas i detta arbete som en tägmodell.

Det här ejobben presenterat en tägmodell i predikatlogik. Modellen består av regler och villkor vars syfte är att beskriva alla möjliga rörelser av tåg i ett järnvägsystem. Meningen med modellen är att den ska kunna användas i formell analys av datorstyrd kontrollsystem för att filtrera ut orealistiska motexemplet till säkerhetskraev.

Modellen är skriven i predikatlogik med de två temporala operatorerna $I$ och $X$. $I$ används för att ge det initiale värdet av en variabel och $X$ används för att ge en variabels värde i nästa tidsteg.


Modellen har tagits fram med hjälp av diskussioner med handledare samt studier av en tägmodell konstruerad av Björner med flera (9).

---

1Det är också möjligt att satsbevisaren värken kan bevisa eller motbevisa kravet.
Chapter 2

Introduction

2.1 The need for reliable control systems in the railway domain

The railway systems of today are large and complex.

Each train is a big and heavy vehicle, operating at high speed with no possibility of steering away from any obstacle that may block its movement on the track. The trains move in concurrency with each other, sharing the same railway net and under the constraints made by time tables. The inherent danger is obvious.

Consequently there is a need for reliable control systems and during the years of railway history safety has improved step by step [1]. Today there are computerized control systems, both on board the trains and on the side of the track. There are different types of railway control systems. Some of these systems can be classified into one of the following categories [2]

- Interlocking systems (IXL) - The system that monitors objects in the railway yard (like switch points and track circuits) in order to ensure the safe movement of trains between and inside railway stations.

- Traffic management systems (TMS) - The system that provides movement authorities to the trains.

- Automatic train control (ATC) systems - A system located on board of the train which helps the driver to control and manage the train, such as controlling the speed in order to maximize operational ability. Some ATC systems even allow driver-less operation.

These different parts work together to form a real-time distributed safety-critical control system. Each part may be validated individually but in addition the different interfaces between these parts then need to be validated in order to ensure that the complete system works correctly.

It goes without saying that the proper working of any such control system in a safety-critical domain, like the railway domain, needs to be ensured in a satisfactory manner.

2.2 Formal verification

Traditional verification techniques include testing and simulation [3]. The problem with using these kinds of approaches is that it is in general impossible to cover all types of behavior of a system. In order to test or simulate all kinds of behavior one must explore the entire state space of the system, which is in most cases a highly time-consuming and probably entirely infeasible task. Therefore testing and simulation is a great way to explore the normal behavior of a system but is not enough for ensuring the complete safety of a system. As Dijkstra famously put it [5] "Program testing can be used to show the presence of bugs, but never to show their absence!". Formal verification, on the other hand, attempts to not only find bugs but also to prove the absence of bugs.
Proving something correct can naturally be divided into two steps; defining what one means by correct behavior of the system and then formally proving that the system fulfills it [4]:

- Write a formal specification of the system describing the expected behavior of the system. The specification needs to be unambiguous and clear.
- Use formal analysis techniques to prove that the system meets the requirements of its specification.

There are many different approaches to formal verification, but the different types of analysis techniques can be broadly categorized into deductive verification and decision procedures [6].

- deductive verification: The correctness of the program is defined as a mathematical statement, which is then proved using a theorem prover [7].
- decision procedure: An algorithm that, given a decision problem, terminates with a correct yes/no answer. [8]

One commonly used verification approach is model checking which is a decision procedure that can be automated. Using this approach, one models the system as a finite state machine (FSM). The specification is then defined as a set of propositions, called proof obligations, which may or may not be satisfied by the FSM. A proof obligation which is satisfied by the system is valid in the system. If the proof obligation is not valid, i.e., it is falsified, the algorithm may return with a counterexample, which is a run of the FSM where the proof obligation is falsified.

If a safety requirement is not satisfied and a counterexample of the requirement is found, then there are two possibilities. Either the counterexample shows the existence of an actual flaw in the system, which then needs to be corrected. The other possibility is that the counterexample is not real in the sense that the example cannot happen in reality. A counterexample of the second kind is called a false negative.

A domain model, describing the underlying system and the movements of trains, can help with filtering away such false negatives.

### 2.3 Domain models

A domain model for a software system is a description of the domain in which the system is intended to be used and can be seen as an extension of the specification of the software. If a software entity is seen as a function with inputs and outputs, the domain model gives restrictions to the inputs of the function. Decreasing the number of possible input combinations to the system will also decrease the state space of scenarios that may occur in the system during the formal analysis. This results in an easier system to verify.

Care must be taken when defining the domain model. If the constraints of the model are too strict, the model will exclude scenarios which may happen in reality. A proof obligation which is proven valid when the system does not satisfy the requirement, is called a false positive. A false positive is even worse than a false negative as it gives a false sense of security and the underlying bug may not be found. Therefore it is important that the train model does not exclude any realistic behavior.

This thesis defines a domain model for railway control systems, expressing constraints on how trains can behave in a railway network. The model is divided into two parts, where the second part is an extension of the first part. The first model is simple in the sense that even though it describes all realistic movements it also allows a lot of unrealistic behavior. The second model restricts the first model somewhat, so that some of this behavior is excluded.

The impressively detailed railway domain model by Bjørner and others [9] has been used as inspiration for the model presented in this thesis.
2.4 Goal

This thesis describes a train model. This means that an abstraction of the railway net is presented along with constraints defined on the abstraction to restrict the movement of trains. A good model is restricted as much as possible without excluding any realistic movements.

Acknowledgments

I would like to thank my supervisor Olav Bandmann at Prover Technology and my reviewer Lars-Henrik Eriksson at Uppsala University for helping and encouraging me throughout this project. Also I want to say that I am grateful to all employees at Prover Technology for all their kindness and helpfulness.
Chapter 3

The underlying system

The model will describe the movements of trains in a railway system. As with all systems in nature, the structure is complex and we cannot hope to model it exactly. Having one description we can always find another, more detailed, one. Moreover, given a system to describe there will be multiple (if not infinite) ways of choosing which part to describe in the system. The model will not describe them all. Accordingly there is a need to state what parts of the system we want to describe. This chapter informs about which assumptions has been made about the underlying railway system. As such it includes basic railway terminology taken from [1].

3.1 The railway net - a graph

Our simplified railway net is composed of straight tracks and switch points.

A switch (also called a set of points or a turnout) is used in a railway system to direct a train from one track onto another. The switch has three states; two of these are fixed positions. We will refer to these as the left position and the right position of the switch. If the switch is set in its left position it means that a train, entering into the switch from the joining part of the switch, will take the left track in the train’s movement direction. Similarly, the same train will take the right track if the switch is set in its right position.

The last state of the switch is when the switch is said to be out of control. This happens when the switch is not confirmed set in any position. In normal operation the switch is out of control when it is in movement between the left and the right position. A broken switch, which cannot attain either left or right position, is also said to be out of control.

This simple composition of the railway net into straight tracks and the simple kind of switch points described above means that the the railway net can be abstracted using a graph structure with nodes and edges, where each node has either one, two or three edges. This abstraction is presented in section 5.1.

3.1.1 Orientation

Naturally there are two directions of travel on a specific part of the railway track. In this thesis, the two directions will be distinguished by calling them odd and even.

Definition 1 (Orientable). A railway net is said to be orientable if the direction of travel for a train cannot change while the train is inside the system.

Example 1. The following small railway net is not orientable.

\[ \begin{array}{c}
  & & \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad
\end{array} \]

A train entering on the left hand side will pass through the loop and exit with a direction of travel opposite to its original direction.
Let us denote the direction of the railway net by arrows pointing in the even direction. If the railway net is orientable it means that there exists an assignment of directions to each edge in the graph such that any two arrowheads do not meet, as that would entail a change of direction. Similarly any two arrowtails do not meet in such a graph. Hence the orientation of a switch point is then either

or

A switch point is said to be facing for a train if the train enters the point from the joining leg of the switch point. The switch point to the left above is therefore facing in the even direction whereas the second point is facing in the odd direction.

Similarly, a switch point is said to be trailing for a train if the train enters the point from either the left or the right leg of the switch point. The switch point to the left above is therefore trailing in the odd direction whereas the second point is trailing in the even direction.

In this thesis we will assume that the railway net is orientable. This means that we can classify the switch points in the railway net into one of the two kinds described above.

3.2 Computerized railway control systems

Any combination of railway control systems tries to prevent collisions and derailments of trains.

In order to prevent collisions it is important to check that the track is clear of any obstacles before letting a train pass. This means that it is necessary to have some kind of train detection.

If a switch is not properly locked into the correct position for an approaching train, the train might, in the worst case, derail. A common choice in complex track layouts is to use interlocking systems to set the route ready for an approaching train. This is called route setting.

3.2.1 Detection - track circuits

A railway control system needs to keep track of the trains. There are many techniques for doing so, but mainly these techniques can be divided into fixed-size block detection or moving-block detection.

In fixed-size block detection, the railway track is divided into smaller sections, called blocks, and whenever some train part is somewhere in that block, the complete block will be detected as occupied.

In this thesis it is assumed that the train detection is performed by fixed-sized block detection. The train model should be able to talk about the train detection of the rail yard blocks.

3.2.2 Route setting

In the railway net, there are different possible paths for the trains to take, depending on from where the train enters into the rail yard and in what positions the switches in the rail yard are locked. Each path is referred to as a route. When a train is supposed to take a specific route it is the task of the interlocking to set the route, meaning that each point in the route needs to be locked in the correct positions for the train to pass.

The train model needs to be able to talk about the different positions of the points in the rail yard.

3.3 Cycle time

The cycle time of a system is the time it takes for the process of the system to update all internal variables and outputs in the system according to new inputs. Some models assume a fixed cycle time. In the model presented in this thesis, the cycle time is not assumed to be fixed.
Chapter 4

Tools and methods

This chapter introduces the syntax and semantics of the language in which the model is written.

4.1 The language

The model will be defined by a set of formulas, the axioms of the model. The formulas will be written in predicate logic extended with the two temporal operators $I$ and $X$. See [10] or a similar book for an introduction to predicate logic and temporal logic. The operators $I$ and $X$ will be explained later in this chapter.

Each word of the language has the following Backus Naur form (BNF).

$$\phi ::= P(t_1, t_2, ..., t_n) \mid \neg \phi \mid \phi \land \phi \mid \phi \lor \phi \mid \phi \Rightarrow \phi \mid \phi \Leftrightarrow \phi \mid \forall x \phi \mid \exists x \phi \mid I(\phi) \mid X(\phi)$$

where $x$ is a variable, $t_1, t_2, ..., t_n$ are variables and constant symbols and $P$ is a predicate symbol of arity $n \geq 1$.

We will use the same convention for the order of operators as given in [10];

$\neg, \forall x, \exists x$ are evaluated first

$\land$ and $\lor$ are evaluated next

$\Rightarrow$ and $\Leftrightarrow$ are evaluated last.

For each interesting property of the underlying railway system, a predicate will be formed. The domain of a predicate is the set of argument values for which the predicate is defined.

The domain for the predicates will be constant symbols each representing its constant, which in this case is something in the underlying railway system, such as trains or railway blocks. The term object will be used instead of constant and constant symbol throughout this thesis.

The symbol $::=$ will be used to define predicates. So when $P(x) ::= \phi$, for a statement $\phi$, it means that $P(x) \Leftrightarrow \phi$.

4.2 The model

Movement of an object can be described as a function from points in time to points in space. Movements of trains on a railway track can be described by the same means.

Example 2. The red box represents a train moving on the track represented by a straight line.

\begin{center}
\begin{tabular}{ccc}
\text{State 1} & \text{State 2} & \text{State 3} \\
\end{tabular}
\end{center}

The pictures show the state of the system in three consecutive time steps.
Thus, if we discretize the time into time steps then the model can be defined by correctly answering two questions:

1. What are the possible states of the system?
2. For any series of states, what are the possible choices of states in the next time step?

Note that the model will not be deterministic. In each step of time there is a finite number of possible states for the next time step. The model includes constraints on the number of possible states in such a way that all correct movements are described while excluding as many incorrect movements as possible.

4.2.1 Describing the state - combinatorial part

After choosing a suitable abstraction of the system, a state of the system can be described using a set of predicates.

**Example 3.** As an example of an abstraction we could divide the track from example 2 into the smaller parts \(b_1, b_2, b_3\).

\[
\begin{array}{ccc}
& b_1 & b_2 & b_3 \\
\text{State 1} & & & \\
\end{array}
\]

\[
\begin{array}{ccc}
& b_1 & b_2 & b_3 \\
\text{State 2} & & & \\
\end{array}
\]

\[
\begin{array}{ccc}
& b_1 & b_2 & b_3 \\
\text{State 3} & & & \\
\end{array}
\]

The system can now be described by the value of a predicate \(\text{Occupied}(b_i)\) for each block \(b_1, b_2, b_3\).

\[
\text{Occupied}(b_i) \leftrightarrow \text{the train is on the track part } b_i
\]

Using this predicate, State 1 can be defined as:

\[
\text{Occupied}(b_1) \land \neg \text{Occupied}(b_2) \land \neg \text{Occupied}(b_3)
\]

The description given in example 3 is explicit in the sense that each of the three states has an exact definition. Less constrained descriptions can be formed. See the following example.

**Example 4.** As we can see in example 3, the length of the train is such that it fits inside each block-length. Hence it can never stretch out to occupy all three blocks. Knowing this and assuming the model will only include this one train, we could add the following constraint.

\[
\neg (\text{Occupied}(b_1) \land \text{Occupied}(b_2) \land \text{Occupied}(b_3))
\]

This constraint means that there will never be a time step where all three predicates \(\text{Occupied}(b_1)\), \(\text{Occupied}(b_2)\) and \(\text{Occupied}(b_3)\) are all true.

By adding more predicates and objects we will get a larger language, making it easier to describe the model in more detail. By adding more constraints on the state of the model we make the model smaller and more accurate.

4.2.2 Describing the transitions - temporal part

As we explained in the beginning of this section, an example of correct train movements inside a railway system will be described as a linearly ordered set of states. Furthermore, in section 4.2.1, we described the state by a set of predicates evaluated for the set of objects in the model, which means that the state in fact is a set of Boolean variables. Each such Boolean variable will have a truth value in each time step and so can be described as a linearly ordered set of Boolean values. This linearly ordered set of values will be referred to as a stream.

Thus, the model will be viewed as a a set of streams of Boolean values, one stream for each \(P(y)\) for each choice of predicate \(P\) and object \(y\).
Given a state of the system we need to define what the next state can look like. This can be done using the two temporal operators $I$ and $X$. Let $P$ be a predicate and let $z$ be an object. Then $P(z)$ is the value of the predicate $P$ evaluated for the object $z$ in an arbitrary time step, $X(P(z))$ denotes the value of $P(z)$ in the next time step and $I(P(z))$ denotes the value of $P(z)$ in the initial time step.

**Example 5.** Let $A$ and $B$ be predicates with arity 1, defined on the variable $x$. Let $A(x)$ denote the statement "$B(x)$ has been true in some earlier time step" then $A(x)$ could be defined as follows.

\[
I(A(x)) := false \\
X(A(x)) := A(x) \lor B(x)
\]

Then $A(x)$ is false in the initial time step and will be false in each consecutive time step until $B(x)$ was true in the previous time step. Then $A(x)$ is true and stays true in each consecutive time step.

In example 5 the operators $I$ and $X$ are used to define the stream $A(x)$, but they could also be used to define constraints.

**Example 6.** Take a look at the states of example 3 another time. Assume that the train can only travel one block in each time step. Then the train cannot occupy $b_1$ in one time step and occupy $b_3$ in the next time step. Using the $X$ operator this can be formulated as a constraint:

\[
\neg(\text{Occupied}(b_1) \land X(\text{Occupied}(b_3)))
\]

### 4.2.3 Division into static and dynamic predicates

The model will be divided into a static and a dynamic part.

A **static predicate** is a predicate which has a static value for each choice of argument values. It describes something that does not change. For example the predicate

\[
M(x, y) := x \text{ is the mother of } y
\]

is naturally static. In other words a static predicate is a predicate $P$ such that $P(a) \leftrightarrow X(P(a))$ for each $a$ in the domain of $P$.

A **dynamic predicate** is a predicate which is not static. These kinds of predicates describe things that may change. For example

\[
F(x, y) := x \text{ and } y \text{ are friends.}
\]
Chapter 5

Model 1: Occupied blocks

The railroad track in the system is divided into smaller parts, which will be referred to as blocks. The set of blocks is denoted by $B$ and is the first set of objects in the model. Lower-case $b_i$ with or without subscript, will be used to denote an arbitrary block.

The purpose of the model is to describe how trains can move on the railroad track, i.e. when and how blocks can be occupied from one point in time to the next. The act of occupying a block $b_i$ will be described by the predicate $Occ_i$, i.e.

$$Occ(b) \iff b \text{ is occupied by something}$$

In each moment of time, there will be a value for each $Occ(b_i)$. The goal is to define how the value of $Occ(b_i)$ can change with time.

5.1 The railway graph

The model is built up from information about how the blocks are connected. Recall from section 3.1.1 that each track in a railway net has a direction. In the pictures of this report, the direction (in cases where the direction is important) will be denoted by an arrow so that the arrowhead points in the even direction. The predicate $Next\_block$ keeps track of how the blocks are connected.

$$Next\_block(b_i, b_j) \iff b_i \text{ and } b_j \text{ are connected so that } b_j \text{ is connected to the even end of } b_i$$

**Example 7. If the system looks like this**

![Diagram showing the railway graph with nodes $b_1$ to $b_7$ and edges showing connections](image)

then $B = \{b_1, b_2\}$ and $Next\_block(b_i, b_j)$ is true if and only if $i = 1$ and $j = 2$.

With this perspective each railway system can be regarded as a directed graph $(B, Next\_block)$ where a node (block) is red if and only if something occupies the block.

**Example 8. An example graph:**

![Diagram showing a more complex railway graph with nodes $b_1$ to $b_{10}$ and various connections](image)
5.2 Static properties of the railway graph

The purpose of the static properties is to define a clear abstraction of the railway net. All static properties will be formulated in terms of the predicate \( \text{Next\_block} \).

The following constraint is made to simplify later abstractions and states that "no block is the next block of itself"

\[ \forall b \in B \ [ \neg \text{Next\_block}(b, b) ] \]

5.2.1 Block-types

The model accepts three types of blocks; end blocks, switch blocks and linear blocks. The reason for the classification of blocks is that different logic will apply to different types of blocks.

Another reason is that, by classifying the kinds of blocks, all parts of the railway graph will be recognized and well understood.

**End blocks:** The blocks at the endpoints of the track layout are entry (and exit) points into (and out of) the railway system for the trains. A block \( b \in B \) is said to be an end block if it has no more than one neighboring block. The following picture shows end blocks (node with thick border) in three very small systems.

\[ \circ \quad \circ\circ \quad \circ \circ \circ \]

In other words, a block \( b \) is an end block if and only if it has no two distinct neighbors. Let the predicate \( \text{End} \) denote whether a block is an end block or not.

\[ \text{End}(b) := \neg \exists b_i, b_j \in B \left[ b_i \neq b_j \land \left( \text{Next\_block}(b_i, b) \lor \text{Next\_block}(b_j, b) \right) \right] \]

**Linear blocks:** The linear blocks are the inner blocks in the railway system that do not have a switch. The following picture shows a linear block (nodes with thick border).

\[ \circ \circ \circ \]

They have one neighbor in the even direction and one neighbor in the odd direction.\(^1\)

\[ \text{Linear}(b) := \exists b_i \ [ \text{Next\_block}(b_i, b) ] \land \exists b_i \ [ \text{Next\_block}(b_i, b) ] \]

**Switch blocks:** A switch block is a block containing a switch. This especially means that the block has two neighbors on one of its sides. We will also require that a switch block has a third neighbor on the opposite side. We let a facing switch in the even direction be called an even switch and a facing switch in the odd direction be called an odd switch. The following picture shows the two different kinds of switch blocks (nodes with thick border).

\[ \begin{array}{c}
\text{Even\_switch} \\
\circ \quad \circ \\
\end{array} \\
\begin{array}{c}
\text{Odd\_switch} \\
\circ \quad \circ \\
\end{array} \]

\(^1\)The symbol \( \exists x \) means "there exists exactly one \( x \) such that".
The two types of switch blocks can be defined as follows:

\[
\text{Even\_switch}(b) := \exists!b_i \exists!b_j \exists!b_k \left[ b_i \neq b_j \land b_i \neq b_k \land b_j \neq b_k \land \text{Next\_block}(b_i, b) \land \text{Next\_block}(b_j, b_j) \land \text{Next\_block}(b_k, b_k) \right]
\]

\[
\text{Odd\_switch}(b) := \exists!b_i \exists!b_j \exists!b_k \left[ b_i \neq b_j \land b_i \neq b_k \land b_j \neq b_k \land \text{Next\_block}(b_i, b) \land \text{Next\_block}(b_j, b_j) \land \text{Next\_block}(b_k, b_k) \right]
\]

Now we can define what is meant by being a switch block.

\[
\text{Switch}(b) := \text{Even\_switch}(b) \lor \text{Odd\_switch}(b)
\]

**Remark 1.** Each block type definition pinpoints the number and placements of neighboring blocks. As these two properties differs between different block types, the three sets of blocks are disjoint.

Each block \( b \in B \) needs to be classified as one of the above three types. Hence we need a constraint.

\[
\forall b \in B \left[ \text{End}(b) \lor \text{Linear}(b) \lor \text{Switch}(b) \right]
\]

In addition to the static predicates already defined, we will need a predicate \( \text{Fork\_path} \). For each two blocks \( b_i, b_j \), \( \text{Fork\_path}(b_i, b_j) \) denotes that \( b_i \) is a switch which has \( b_j \) as one of its neighboring blocks on the "forking" side of the switch.

\[
\text{Fork\_path}(b_i, b_j) := \exists b_k \left[ b_j \neq b_k \land \left( \text{Next\_block}(b_i, b_j) \land \text{Next\_block}(b_k, b_k) \right) \lor \left( \text{Next\_block}(b_j, b_i) \land \text{Next\_block}(b_k, b_i) \right) \right]
\]

**Remark 2.** For each switch block \( b \) there will be two blocks \( b_i \) and \( b_j \) such that \( b_i \neq b_j \), \( \text{Fork\_path}(b_i, b_j) \) and \( \text{Fork\_path}(b_i, b_j) \).

Note that this does not hold for either an end block or for a linear block. Thus whenever \( \text{Fork\_path}(b_i, b_j) \) is true for \( b_i, b_j \in B \) we will know that \( \text{Switch}(b_i) \) is true and \( b_j \) belongs to one of the two different paths through the switch block.

### 5.3 Dynamic properties of the railway graph

The dynamic properties of the railway graph are the kind of properties that change the structure of the railway net and so affect the movement of the trains. The only such property in this model is the movement of points inside the switch blocks.

When it is possible to travel from one block to a neighboring block, the path between the blocks will be said to be open. As mentioned in section 3.1, the switch has three possible states; the out of control state and the two fixed states. We need to model these three states.

In order to know if a path is locked open or not there is a need for a new predicate, \( \text{Locked\_in\_position}(b_i, b_j) \).

\( \text{Locked\_in\_position}(b_i, b_j) \Leftrightarrow b_i \) is a switch block and the path from \( b_i \) to \( b_j \) is locked open by the switch in \( b_i \).

Using this new vocabulary it is possible to define what it means for a path between two neighboring blocks to be open. The new predicate \( \text{Open}(b_i, b_j) \) has the following definition.

\[
\text{Open}(b_i, b_j) := \left( \text{Next\_block}(b_i, b_j) \lor \text{Next\_block}(b_j, b_i) \right) \land \left( \text{Fork\_path}(b_i, b_j) \Rightarrow \text{Locked\_in\_position}(b_i, b_j) \right) \land \left( \text{Fork\_path}(b_j, b_i) \Rightarrow \text{Locked\_in\_position}(b_j, b_i) \right)
\]
This means that for $b_i$ and $b_j$ to have an open path first they need to be adjacent blocks. Second, if the two blocks together constitutes a forked path, then $\text{Locked\_in\_position}(b_i, b_j)$ needs to be true.

Recall from remark 2 that $\text{Fork\_path}(b_i, b_j)$ can only be true if $b_i$ is a switch and $b_j$ is one of the two forked neighbors.

Thus $\text{Open}$ is variable only if the blocks $b_i$ and $b_j$ together define a path forked off from a switch block and then $\text{Open}$ depends on $\text{Locked\_in\_position}$. In next section we will model the behavior of the switch blocks by defining sensible constraints on the predicate $\text{Locked\_in\_position}$.

5.3.1 Constraints
We will add two new constraints to the model.

(i) Considering the three positions that a switch can be in, it can be noted that there is one combination missing; the case where both paths are open. As this case cannot happen in reality there is a need for a constraint.

$$\forall b_i, b_j, b_k \in B \mid b_j \neq b_k \Rightarrow \neg(\text{Locked\_in\_position}(b_i, b_j) \land \text{Locked\_in\_position}(b_i, b_k))$$

(ii) If the switch moves the point between the two locked positions, there is an inevitable phase in between where the switch is out of control, i.e. the point is not fixed to either of the paths. In other words, a switch cannot transition from being locked in one position into being locked in another position from one time step to the next.

$$\forall b_i, b_j, b_k \in B \mid b_j \neq b_k \Rightarrow \neg(\text{Locked\_in\_position}(b_i, b_j) \land \text{Locked\_in\_position}(b_i, b_k))$$

Remark 3. Note that these constraints don’t demand $b_i$ to be a switch and so these constraints will be on $\text{Locked\_in\_position}(b_i, b_j)$ for all $b_i$. However, $\text{Locked\_in\_position}(b_i, b_j)$ does only influence $\text{Open}$ when $b_i$ is a switch. This means that even though the constraints affect $\text{Locked\_in\_position}$ for all blocks, it will only matter when it affects the forked paths of a switch block and so there is no need for restricting the constraints any further.

5.4 Train movements in the railway graph
The purpose of this section is to present some reasonable restrictions to $\text{Occ}(b)$ for each block $b$ so that the value of $\text{Occ}(b)$ can be said to match train movements more accurately than it would have without any restrictions.

To be able to restrict $\text{Occ}(b)$ the following assumption is needed.

$$\text{Occ}(b) \iff \text{a train is positioned so that some part of it is on the block } b$$

With this new assumption, a train is inside the system whenever a non-empty sequence of adjacent blocks ($b_i, \ldots, b_m$) are occupied. Note however that such a sequence of occupied blocks could be any number of individual trains.

5.4.1 Movements of trains
At any time a train outside the railway system has the possibility to enter into the system through an end block. Similarly, any train on an end block could at any time exit the system. Hence when $\text{End}(b)$ holds, there are no restrictions for $\text{Occ}(b)$.

If a train is already inside the system and stays there, there are three possible events.

(i) The train moves forward; the sequence of occupied blocks expands by one more block and/or contracts by one block.
(iii) The train splits into two individual trains; a block with two occupied neighbors becomes unoccupied.

(iii) The train stands still; the sequence of occupied blocks is unchanged.

This means that there is some continuity in how a train moves. In particular a train cannot jump arbitrarily, but can instead only shift its position onto adjacent blocks.

**Remark 4.** If a train only covers one block $b$ and is moving forward, the next position of the train needs to cover two blocks, of which $b$ is one of the blocks. This is a special case of (i) above, but follows the underlying principle of smooth transitions between the positions of the trains.

From the above statements about train movements, the following conclusion can be drawn.

**Conclusion 1.** The value of $\text{Occ}(b)$ can change from one point in time to the next if one of the following conditions holds.

- $b$ is an end block
- $b$ had an occupied and reachable neighbor in the first time step that stays occupied and reachable in the next time step.

**Proof.** If $b$ is an end block, there are no restrictions to the value $\text{Occ}(b)$ as a train could at any time enter into or exit out of the railyard, thus the conclusion holds. Assume now that $b$ is not an end block. There are two cases, $\text{Occ}(b)$ becomes false or $\text{Occ}(b)$ becomes true.

**Case 1** Assume $\text{Occ}(b)$ is true and $X(\text{Occ}(b))$ is false, i.e. the train on $b$ disappears from $b$.

Either (i) or (ii) applies, each of which implies that there is a neighboring block $b'$ of $b$ which is occupied in both time steps. In the case (i) the train contracts, moving in the direction from $b$ towards $b'$. In the case of (ii), the train splits apart, in which case $b$ actually has two neighboring blocks which are occupied and reachable in both time steps.

There must be a smooth transition in the movement between the two blocks. This means that if the train is moving from $b$ towards $b'$ there must be a time step where both $b$ and $b'$ are occupied. Hence $\text{Occ}(b')$ must also be true.

**Case 2** Assume $\text{Occ}(b)$ is false and $X(\text{Occ}(b))$ is true, i.e. the train arrives to $b$.

This means that there is a train which arrives onto $b$, i.e. case (i) applies, the sequence of blocks is expanding. As transitions between blocks are smooth, $X(\text{Occ}(b'))$ must hold for a neighboring block $b'$.

Regardless of whether the train arrives at $b$ from $b'$ or leaves $b$ in order to arrive at $b'$, the path between the two blocks needs to be open. Otherwise the train cannot travel between them.

**5.4.2 Constraints on $\text{Occ}(b)$**

A constraint can be formed directly from conclusion 1. Namely that in order for a train to arrive to or leave from a block $b$ a neighbor $b'$ must be occupied in the two time steps on which the transition occurs. Furthermore the path between $b$ and $b'$ must be open. The constraint can be formulated as follows.

$$\forall b \in B \left[ \text{Occ}(b) \neq X(\text{Occ}(b)) \Rightarrow \exists b' \left( \text{End}(b) \lor \text{Occ}(b') \land \text{Open}(b, b') \right) \right]$$
Chapter 6

Model 2: Trains and positions

One problem with the first model is that there are no restrictions on how big or how small a sequence of occupied blocks can become. Another problem is that the same "train" (i.e. connected sequence of occupied blocks) can decouple arbitrarily many times. The purpose of the second model is to define an extended model where these two problems don't occur.

This is done by giving a set of trains $t_1, ..., t_i$ and for each train $t_i$ a set of possible positions $p_1, ..., p_m$. The second model gives restrictions to when these positions (sequence of blocks) can be occupied, instead of as in the first model, giving restrictions to when individual blocks can be occupied.

An example is in order.

Example 9. Let the railway system include three blocks, each of length 50m, and a train of length 70m. Then there are 6 valid states. (Recall that the red node denotes that the block is occupied.)

In this small example, there are two invalid states (there is only one train which cannot decouple and the train is too long to fit inside one block):

By only allowing the train to occupy those six positions, only valid train positions will come out of the model.

The set of trains will be denoted by $T$ and the set of positions by $P$. The main interest of this model is to give correct restrictions to when a position $p \in P$ is occupied. A predicate $P_{\text{occ}}(p)$ is needed.

$$P_{\text{occ}}(p) \iff \text{a train occupies } p$$

6.1 The position-graph

Each train $t$ has a set of positions $P_t \subseteq P$, all the positions that $t$ could occupy. If we let

$$\text{Position for}(p, t) \iff p \text{ could be occupied by } t$$

be the set of possible positions for a train $t \in T$, then $P_t$ can be defined as

$$P_t := \{ p \in P \mid \text{Position for}(p, t) \}$$

Each train has now a set of valid positions. Now we need to describe how a train can move between these positions, i.e. we need to define which transitions between positions that are valid.
for each train. Let the predicate $\text{Next} \_ \text{position}(p_i, p_j)$ denote that $p_j$ is a next position in the
even direction for a train currently at position $p_i$.

Then $(P, \text{Next} \_ \text{position})$ forms a graph where each $P_t$ is a subgraph. By the following two
constraints $(P_t, \text{Next} \_ \text{position}|p_i)$ forms a directed graph for each train-part $t \in T$.

The $P_t$s are disjoint

$$\neg \exists p \in P [\exists t_i, t_j \in T [\text{Position} \_ \text{for}(p, t_i) \land \text{Position} \_ \text{for}(p, t_j)]]$$

Next \_ position stays inside the position-graph

$$\forall p_i, p_j \in P [\text{Next} \_ \text{position}(p_i, p_j) \Rightarrow \exists t \in T [\text{Position} \_ \text{for}(t, p_i) \land \text{Position} \_ \text{for}(t, p_j)]]$$

Remark 5. The purpose of the two constraints is to make sure that each train will have its own
graph.

Each node in the graph represents a position that the train could occupy and each edge repre-
sents a valid transition between two positions. Inside each position-graph, a node is colored red if
and only if a train is occupying the position that the node represents.

Example 10. Let us find the position-graph for the railway system of the previous example. Let
the train be denoted by $t$ and the valid positions of $t$ be denoted by $\{p_1, \ldots, p_6\}$.

Then the following position-graph of $t$ describes Next \_ position in the correct manner. (Note
that node $P_1$ is colored, meaning that the train is currently not inside the railway system).

6.2 Dynamic properties of the position-graph

We denote that a train-part $t_i$ occupies a certain position $p$ by $\text{Pt}_\_ \text{occ}(p, t_i)$. It seems reasonable
to define the predicate $\text{Pt}_\_ \text{occ}$ as follows.

$$\text{Pt}_\_ \text{occ}(p, t) := \text{P}_\_ \text{occ}(p) \land \text{Position} \_ \text{for}(p, t)$$

In order to ensure proper movement of trains we need to give some restrictions on $\text{Pt}_\_ \text{occ}$.

Each train occupies exactly one position

$$\forall t \in T \exists p \in P_t [\text{Pt}_\_ \text{occ}(p, t)]$$

Each train can only move to an adjacent position

$$\forall t \in T \forall p_j, p_k \in P_t \left[\begin{array}{c}
    \text{Pt}_\_ \text{occ}(p_j, t) \\
    \land \\
    X(\text{Pt}_\_ \text{occ}(p_k, t))
\end{array}\right] \Rightarrow
\left[\begin{array}{c}
    \text{Next} \_ \text{position}(p_j, p_k) \\
    \lor \\
    \text{Next} \_ \text{position}(p_k, p_j)
\end{array}\right]$$

With these constraints, each train has a proper movement inside its position-graph. Only cor-
rect placements (positions) are accepted (by the choice of the positions) and only correct transitions
between positions are allowed.
Chapter 7

Union of the two models

In order to unite the two models, we need to define what it means to be a position in the railway graph.

**Definition 2.** A position \( p \in P \) represents a set of occupied blocks in the railway graph.

In order to remember which block belongs to which position we need a static predicate \( \text{Position} \).

\[
\text{Position}(b, p) \iff \text{Block } b \text{ belongs to the position } p
\]

Finally, to bring the two models together we define the following constraint.

\[
\forall b \in B \ [ \text{Occ}(b) \iff \exists p \in P \ [ \text{Position}(b, p) \land \neg \text{Occ}(p)])
\]

With this constraint a block \( b \) is occupied if and only if there is a position \( p \) which contains \( b \) and is occupied, i.e., there is a train which is positioned on \( p \).

All together we now have two constraints on \( \text{Occ} \), of which the first constraint (from model 1) ensures that the train-parts respect switches and the second constraint (above) ensures valid placements of train-parts.

7.1 Resolve conflicts

Due to the extension of model 1 by model 2, each train positions itself correctly on the track. However, as each train move in its own position-graph and these position-graphs are completely separated from each other, nothing prevents two trains from sharing blocks. This means, for example, that a train may pass another train on the same track.

We need to make some restrictions on how the trains can move together in the railway system. Imagine that we have two trains \( t_1, t_2 \) in the railway system and that we precompute the sets of positions \( P_{t_1}, P_{t_2} \). Take two positions \( p_1 \in P_{t_1}, p_2 \in P_{t_2} \). As long as \( p_1 \) and \( p_2 \) do not share any block then both can be occupied at the same time. But there are also other valid situations. A couple of examples may be in order.

**Example 11.** Below there is a railroad track consisting of 5 consecutive blocks and two trains sharing the block \( b_3 \).

![Example 11 diagram]

**Example 12.** Below there is a railway system consisting of 4 blocks, of which one is a switch.
block, and two trains (seen from above) sharing the switch block.

Example 13. In the below picture there is a railway system consisting of 4 blocks and two trains sharing an end-block. One of the trains may be entering the railway system in this state.

This means that there are some cases when block sharing between train positions is physically possible and so should be allowed in the model. We will use the term utmost block of a position \( p \) to be a block \( b \) which belongs to the position and which has a neighboring block that doesn’t belong to the position.

\[
Utmost(b, p) := \text{Position}(b, p) \land \exists b' \left[ (\text{Next}\_\text{block}(b, b') \lor \text{Next}\_\text{block}(b', b)) \land \neg \text{Position}(b', p) \right]
\]

Two different positions \( p_1 \) and \( p_2 \) that share a block \( b \) may both be occupied at the same time if \( b \) is an utmost block for both \( p_1 \) and \( p_2 \). This can be formulated as in the following constraint.

\[
\forall p_i, p_j \in P \forall b \in B \left[ p_i \neq p_j \land \left( \bigwedge_{\text{occ}(p_i)} \land \left( \bigwedge_{\text{occ}(p_j)} \land \left( \bigwedge_{\text{Position}(b, p_1)} \land \left( \bigwedge_{\text{Position}(b, p_2)} \Rightarrow \left( \bigwedge_{\text{Utmost}(b, p_1)} \land \bigwedge_{\text{Utmost}(b, p_2)} \right) \right) \right) \right) \right] \right]
\]

Note however that this constraint only reduces the problem, but does not remove it completely. For example, say that the system contains two trains which occupy two positions that have the length of two or one block. Then all the blocks of the positions are utmost blocks of the positions. Hence the trains can stand on top of each other, and as a result also pass each other on the track.
Chapter 8

Analysis

This chapter contains an analysis of the model.

8.1 Configurations

The first model consists of 10 predicates \((\text{Occ, Next\_block, End, Linear, Even\_switch, Odd\_switch, Switch, Fork\_path, Locked\_in\_position and Open})\) which together set the restrictions on what can be said about the model. Looking at a particular railway system, we may want to add new constraints to the model in order to have a closer resemblance with the underlying system. The following is a list of examples of what we may want to add to the model for some railway systems and how this is formulated in the language of the model.

- Initialize the model with some trains already in the system, i.e. define \(I(\text{Occ}(b))\) for each block \(b\). For example \(\forall b \in B \ [ I(\text{Occ}(b)) := \text{False}]\) initializes the railway system to be empty.
- Define some blocks to be broken, so that no train can enter on them, i.e. let \(\text{Occ}(b) := \text{False}\) for some \(b \in B\). This could for example be used on some of the end blocks in order to control exactly where trains can enter into (or exit from) the system.
- Lock some of the switches into positions, i.e. let \(\text{Locked\_open}(b_i, b_j) := \text{True}\) for some \(b_i, b_j \in B\). This means that if \(b_j\) and \(b_i\) is the forked paths of a switch block \(b_i\), then no train can travel between \(b_k\) and \(b_i\) even though they are adjacent.
- Transition time for a switch, i.e. the number of cycles that a switch must be "out of control" before locking into any of its positions. Note that according to one of the constraints of the model, there must be at least one such cycle. As an example, enforcing the "out of control" state to last at least \(n \geq 2\) cycles can be done by adding the constraint.

\[
\forall b \in B \left[ \exists b' \in B \left( \begin{array}{c} \text{Locked\_in\_position}(b, b') \\
\neg X(\text{Locked\_in\_position}(b, b')) \\
\neg \exists b' \in B \ [XX(\text{Locked\_in\_position}(b, b'))] \\
\neg \exists b' \in B \ [XX...X(\text{Locked\_in\_position}(b, b'))] \\
\end{array} \right) \right] \Rightarrow \left( \begin{array}{c} \neg \exists b' \in B \ [XX(\text{Locked\_in\_position}(b, b'))] \\
\vdots \\
\neg \exists b' \in B \ [XX...X(\text{Locked\_in\_position}(b, b'))] \\
\end{array} \right)
\]

Where the number of \(X\)s in the last conjunction is \(n\).

- Some railway systems have coupled switches, so that if the points move on one of the switches, the points of the coupled switch has to move as well. For a switch \(b_i\) with forked paths \(b_j\) and \(b_k\) coupled with another switch \(b'_j\) that has forked paths \(b'_j\) and \(b'_k\) one way to enforce this is by the constraint:

\[
\text{Locked\_open}(b_i, b_j) \Leftrightarrow \text{Locked\_open}(b'_i, b'_j) \wedge \text{Locked\_open}(b_i, b_k) \Leftrightarrow \text{Locked\_open}(b'_i, b'_k)
\]
We may also want to say something about the trains, especially set restrictions on how they can couple and decouple. But this is hard since we cannot distinguish individual trains. If two adjacent blocks are occupied it may be one big train or two smaller trains. For all we know, on the same block, there may be several trains. The closest we can come to defining coupling is by letting two series of occupied blocks (i.e. the two series have an unoccupied block in between) become one, and decoupling by letting one series become two.

- **forbid decoupling**
  A series of occupied blocks cannot become two series of occupied blocks. This could be formulated as "for every series of three adjacent blocks, if the middle one changes from occupied to unoccupied, then both of the neighbors cannot be reachable and occupied in the current time step"

\[
\left( \text{Next\_block}(b_i, b_j) \wedge \left( \text{Occ}(b_j) \wedge \neg \text{X(Occ}(b_i) \right) \Rightarrow \left( \text{Occ}(b_i) \wedge \text{Open}(b_i, b_j) \right) \right)
\]

- **forbid coupling**
  Two series of occupied blocks cannot be joined into one. "for every series of three adjacent blocks, if the middle one changes from unoccupied to occupied, then both of the neighbors cannot be reachable and occupied in the next time step"

\[
\left( \text{Next\_block}(b_i, b_j) \wedge \left( \neg \text{Occ}(b_j) \wedge \text{X(Occ}(b_i) \right) \Rightarrow \left( \text{Occ}(b_i) \wedge \text{Open}(b_i, b_j) \right) \right)
\]

### 8.2 Formal verification

An instance of the model is formed by dividing the railroad track into a list of blocks \( b_1, ..., b_n \) and assigning a truth value \( \text{Next\_block}(b_i, b_j) \) on each pair \( b_i, b_j \) of blocks.

After filling in the model for a particular underlying railway system we could define some formulas to test on the model. Assuming, for example that we configure the model to be initialized as empty, i.e. no occupied blocks in the initial time step, one may want to show that all blocks are reachable. A block \( b \) is reachable if it is possible to find a scenario where the blocks in the system are occupied in the correct order from an end block, finally occupying \( b \). In other words the formula \( \text{Occ}(b_i) \) should be satisfiable for each block \( b_i \in B \).

Such a scenario is a counterexample to the sentence "if the railway system is empty in the initial time step, then the block \( b_i \) cannot be occupied".

\[
\forall b \in B \left[ \neg I(\text{Occ}(b)) \right] \Rightarrow \neg \text{Occ}(b_i)
\]

If this formula is valid for some block \( b_i \), then \( b_i \) is not reachable from any end block. On the other hand, if a counter model to this formula is found then there is a way to occupy \( b_i \) in the model.

### 8.3 Analysis of the first model

See appendix A for a summarized description of the axioms of the first model.

The model gives a straightforward abstraction of a railway system and it tries to model train movements in the given abstraction by constraining the \( \text{Occ} \) predicate. The resulting movement is continuous, a train can neither appear nor disappear out of thin air, and movement is always done in sequence, one adjacent block at a time. Movement to a block originates from an occupied neighbor, movement from a block results in an occupied neighbor.

However, the model does not include any definition of what a train is and so we cannot give any restrictions guarding how many blocks a train can occupy at the same time. So, any series of
blocks can expand or shrink arbitrarily. Another strange behavior is that a sequence of blocks in
the railway system could decouple arbitrarily many times.

Even though the model is very simple it is large enough to be able to give a good first ap-
proximation of train movements. As seen in section 8.1 we can find sensible configurations and
as seen in 8.2 we can construct formulas testing reachability in the railway system. The model is
constructed with care to not exclude any realistic behavior.

8.4 Analysis of the union

See appendix B and C for a summarized description of the axioms of the second model and the
union of the models.

The second model was built in order to solve two problems of the first model, namely that

• Trains could expand/contract arbitrarily.

• Trains could decouple arbitrarily many times.

The chosen solution was to statically define exactly how each train can move on the track by
defining valid positions and valid transitions between positions.

The two models were merged into one, the union, which put some restrictions on how the trains
can be positioned relative to the position of other trains.

The two models are kept together by the \( \text{Oc} \) predicate, but their responsibilities are quite
different; the first model controls the behavior of the switches and makes sure that no movement
can exist between two blocks if the path between them is not open, while the second model enforces
restrictions on the positioning of the trains.

Although the union of the models is an improvement of the first model, there are still some
unrealistic behaviors.

**Arbitrary many small trains can fit on the same block:** If a train occupies only one block
then this block is an utmost block and so the train can share this block with another train.
If the second train was also only occupying that block then we could add another train, and
so on, squeezing in arbitrary many small trains on the same block.

**Unrealistic speed of trains:** There are no restrictions on how fast a train may move on the
railroad track.
Chapter 9

Related work

During the work of this thesis a train model ([9]) has been studied for comparison. In this chapter, the studied model and the model of this thesis are compared.

9.1 Towards a TRain Book - for the RAilway DomaiN

The book presents a train model written in the RAISE Specification Language (RSL).

The model includes a detailed description of the basic parts of a railway net and as such it includes a whole range of object types; railway nets, rail units, lines, stations, tracks, connectors, etc.

The basic building block of the railway net are the rail units (similar to the blocks of model 1), which are defined by the connection-points between rail units, the connectors. Each rail unit can be in one of its set of states, depending on the intended traffic through the rail unit.

If a continuous sequence of rail units are in such a state that traffic is intended the whole way through the sequence of rail units, then the sequence is called a route. Trains in the model are modeled as routes and movements of the trains are handled by adding/removing rail units to/from the ends of the route, but only forward movement is accepted. The route cannot add/remove a unit on both ends at the same time.

9.2 Comparison

The two models of this thesis (model 1 and model 2) have some similarities to the train model by Bje"ner (here called the train book model, see [9]).

The description of the underlying railway net, i.e. the static description, is much more detailed than in model 1, which solely uses the set B and the predicate Next_block in order to define the static attributes of the model.

One important type in the train book is the connectors, which represents the connection points between the railway units (i.e. the blocks). The topology of the railway net is defined in terms of these connectors which could be a better choice as it is possible then to exactly define the length of a path.

Example 14. In a switch block there are two possible paths,

the straight part and the diverging path. As long as our language does not distinguish between the two paths we cannot give them different lengths.

However the lengths could be defined by introducing the three connectors of the switch, which is done in the train book model, and then defining the length of a path as the length between the two connectors of a path.

25
The definition of movement of trains in model 1 is inspired by the definition of train movements in the train book. The difference is that the trains in the train book cannot decouple. The movement of trains in the train book is similar to the movement of trains in model 1. The trains (routes) can only add or remove railroad units from the route, but cannot add railroad units at both ends at the same time, nor remove railroad units at both ends at the same time. The reason for the simpler version in model 1 was to have a simple straightforward description of train movements.

The term Open has been taken from the train book but in the train book the concepts of reachability is taken a step further. In the train book each railroad unit has a set of managed states and a set of physical states, where the managed states are the intended states, and the physical states are all states that are physically possible but are not intended states. Model 1 does not distinguish managed from physical states.
Chapter 10

Conclusions

This chapter contains a small summary of the thesis and some of the writer’s own reflections about the work process, the resulting model and about possible future work with the model.

10.1 Summary

Recall that the goal of the project was to define a train model, written in predicate logic, describing train movement in a railway system. The idea was to construct the model so that it could be used to verify railway control systems.

The railroad track was abstracted as a directed and orientable graph where the nodes where of three different kinds: end block nodes, linear block nodes and switch block nodes.

The first model includes some basic behavior of switches and a simple description of train movement. With this model the train cannot disappear in the middle of the railroad track. Neither can it suddenly appear in the middle of the railroad track. In other words, the motion of the train is constrained to be continuous.

However, the model did not distinguish between different trains and it was possible to reach a state were the complete railroad system is occupied. In order to remedy this problem, the second model was developed.

In the second model a set of train objects $t_1, \ldots, t_n$ are given. Each train $t_i$ has a certain length which means that we can statically determine the complete set of possible positions on the railroad track for the train. Each such position consists of a sequence of blocks. The movement of trains is constrained by forcing each train object to attain exactly one of its possible positions at all times.

The union of the two models is therefore a model in which the trains move without arbitrarily jumping, expanding or shrinking. However, the identification of individual trains meant that some constraints where needed to describe the interaction between trains; the trains could pass each other on the same railroad track. One constraint was added in order to tackle this problem: if two trains are occupying the same block, then this block must be an utmost block of the two trains. Unfortunately this did not remove the problem completely as small trains can still pass each other on the track.

To be able to understand the first model better, the writer also implemented a small Python script instantiating the constraints of the first model. The script takes an input consisting of a list representing the Next_block predicate and outputs an instantiation of the first model based on the input. The script was used to explore the first model using an internal theorem prover at Prover Technology.

10.2 Reflections

The goal was to find a model of train movements which does not exclude any realistic behavior. It is not possible to formally prove that the model does not exclude any realistic behavior as such
a proof would demand a definition of what one means by "realistic behavior". If we had such a definition, then the model could be replaced by this definition. Instead, the model is derived by a careful argumentation from an informal description of some behavior that the writer regards as unrealistic, which is then removed from the model by formal constraints.

The work process consisted in studying other domain models (mostly the model by Bjarne, see [9]) and discussing solutions with my two mentors. Reading up on formal verification and domain models was done at a quite late stage in the process when the model already had its final form. This was not perhaps the best order of doing things. In hindsight it seems obvious that the project should have started with collecting and reading up on these subjects.

There is still work to be done with the model. As was explained in section 8.4, the interaction between the trains is not realistic as arbitrary many small trains can fit on the same block. Also, there are no restrictions on the speed of the trains. But it is important to note here that even though these are unrealistic train movements, they might not pop up in counter models, thus making the work with excluding them from the model unnecessary.

The work with this thesis has been a quick look into formal verification of the railway domain and has covered subjects ranging from railway system designs to the mathematics behind formal verification. It has been an opportunity for the writer to think, discuss and read about formal verification. In this aspect, the work has been a success.

10.3 Future work

Additions to the railway graph The model is quite minimalistic as it only includes some very basic objects and properties of a railway system. Therefore there is room for extending the basic structure of the railway graph to include other railway junctions, signals, level crossings etc.

Verification of a particular railway control system As was discussed in the introduction, the model could be used in formal verification of a railway control system to filter away unrealistic counter models. This has not been done during the work of this thesis, but it is a natural continuation of this work. Doing so could be helpful in order to find flaws in the model and also to understand how to extend the model further. Using the model during the verification of a particular railway control system may give inspiration to a new set of predicates for the model.
Appendix A

Formulas for model 1

This appendix contains all formulas for the first model. See chapter 4 for a description of the language and vocabulary used in this appendix.

A.1 Domain

The set of railroad blocks $B = \{b_1, ..., b_n\}$.

A.2 Static predicates

The following is a list of informal descriptions for each static predicate and definitions for all static predicates except $Next\_block$. The definition of $Next\_block$ depends on the underlying railway system as it describes how the blocks of the railroad are connected. The rest of the static predicates can be defined in terms of $Next\_block$.

$Next\_block(b_i, b_j)$: $b_i$ and $b_j$ are connected so that $b_j$ lies in the even direction of $b_i$.

$End(b)$: $b$ is an end block.

$Linear(b)$: $b$ is a linear block.

$Even\_switch(b)$: $b$ is a facing switch in the even direction.

$Odd\_switch(b)$: $b$ is a facing switch in the odd direction.

$Switch(b)$: $b$ is a block containing a switch.

$Fork\_path(b_i, b_j)$: $b_i$ is a switch which $b_j$ as one of the two blocks on the forked side of the switch block.
\[
End(b) := \neg\exists b_i, b_j \in B \left[ b_i \neq b_j \land \left( (\text{Next\_block}(b, b_i) \lor \text{Next\_block}(b, b_j)) \land (\text{Next\_block}(b, b_j) \lor \text{Next\_block}(b, b_i)) \right) \right]
\]

\[
Linear(b) := \exists b_i [\text{Next\_block}(b, b_i)] \land \exists b_i [\text{Next\_block}(b, b_i)]
\]

\[
Even\_\text{switch}(b) := \exists b_i \exists b_j \exists b_k \left[ b_i \neq b_j \land b_i \neq b_k \land b_j \neq b_k \land \text{Next\_block}(b_i, b) \land \text{Next\_block}(b_j, b) \land \text{Next\_block}(b_k, b) \right]
\]

\[
Odd\_\text{switch}(b) := \exists b_i \exists b_j \exists b_k \left[ b_i \neq b_j \land b_i \neq b_k \land b_j \neq b_k \land \text{Next\_block}(b_i, b) \land \text{Next\_block}(b_j, b) \land \text{Next\_block}(b_k, b) \right]
\]

\[
Switch(b) := Even\_\text{switch}(b) \lor Odd\_\text{switch}(b)
\]

\[
Fork\_\text{path}(b_i, b_j) := \exists b_k \left[ b_j \neq b_k \land \left( (\text{Next\_block}(b_i, b_j) \land \text{Next\_block}(b_i, b_k)) \lor (\text{Next\_block}(b_j, b_i) \land \text{Next\_block}(b_j, b_k)) \right) \right]
\]

### A.3 Dynamic predicates

The following are informal descriptions of all dynamic predicates in the first model and definitions for two of them. The other two predicates (\textit{Occ} and \textit{Locked\_in\_position}) are restrained by constraints given in section A.5.

\textbf{Occ}(b): A train occupies \( b \).

\textbf{Locked\_in\_position}(b_i, b_j): b_i is a switch block and the path from \( b_i \) to \( b_j \) is locked open by the switch in \( b_i \).

\textbf{Open}(b_i, b_j): \( b_i \) and \( b_j \) are adjacent blocks and the path between them is open.

\[
Open(b_i, b_j) := \left( (\text{Next\_block}(b_i, b_j) \lor \text{Next\_block}(b_j, b_i)) \land (\text{Next\_block}(b_i, b_j) \Rightarrow \text{Locked\_in\_position}(b_i, b_j)) \right) \land (\text{Next\_block}(b_j, b_i) \Rightarrow \text{Locked\_in\_position}(b_j, b_i))
\]

### A.4 Constraints for static predicates

No block is the next block of itself

\[
\forall b \in B [\neg \text{Next\_block}(b, b)]
\]
A block is either an end block, a linear block or a switch block

$$\forall b \in B \ [\text{End}(b) \lor \text{Linear}(b) \lor \text{Switch}(b)]$$

### A.5 Constraints for dynamic predicates

A switch cannot be locked in both positions at the same time

$$\forall b_1, b_j, b_k \in B \ | \ b_j \neq b_k \Rightarrow \neg (\text{Locked}_i \text{ in } \text{position}(b_i, b_j) \land \text{Locked}_i \text{ in } \text{position}(b_i, b_k))$$

Force correct transition between locked states

$$\forall b_1, b_j, b_k \in B \ | \ b_j \neq b_k \Rightarrow \neg (\text{Locked}_i \text{ in } \text{position}(b_i, b_j) \land X (\text{Locked}_i \text{ in } \text{position}(b_i, b_k)))$$

All movement has an origin

$$\forall b \in B \ \left[ \text{Occ}(b) \neq X (\text{Occ}(b)) \Rightarrow \text{End}(b) \lor \exists b' \left[ \begin{array}{c} \text{Occ}(b') \land \text{Open}(b, b') \\ \land \\ X (\text{Occ}(b') \land \text{Open}(b, b')) \end{array} \right] \right]$$
Appendix B

Formulas for model 2

This appendix contains all formulas for the second model. See chapter 4 for a description of the
language and vocabulary used in this appendix.

B.1 Domain

The set of positions $P = \{p_1, ..., p_n\}$.
The set of trains $T = \{t_1, ..., t_l\}$.

B.2 Static predicates

$\text{Position}_{\text{for}}(p, t)$: $p$ could be occupied by $t$.

$\text{Next}_{\text{position}}(p_i, p_j)$: $p_j$ is the next position of $p_i$ in the even direction.

B.3 Dynamic predicates

$\text{P}_{\text{occ}}(p)$: a train occupies $p$.

$\text{Pt}_{\text{occ}}(p, t)$: $p$ is occupied by $t$

$\text{Pt}_{\text{occ}}(p, t) := \text{P}_{\text{occ}}(p) \land \text{Position}_{\text{for}}(p, t)$

B.4 Constraints for static predicates

$\neg \exists p \in P [\exists t_i, t_j \in T [\text{Position}_{\text{for}}(p, t_i) \land \text{Position}_{\text{for}}(p, t_j)]]$

$\forall p_i, p_j \in P [\text{Next}_{\text{position}}(p_i, p_j) \Rightarrow \exists t \in T [\text{Position}_{\text{for}}(t, p_i) \land \text{Position}_{\text{for}}(t, p_j)]]$

B.5 Constraints for dynamic predicates

$\forall t \in T \exists ! p \in P_i [\text{Pt}_{\text{occ}}(p, t)]$

$\forall t \in T \forall p_j, p_k \in P_t \left[ \left( \text{Pt}_{\text{occ}}(p_j, t) \wedge X(\text{Pt}_{\text{occ}}(p_k, t)) \right) \Rightarrow \left( \text{Next}_{\text{position}}(p_j, p_k) \lor \text{Next}_{\text{position}}(p_k, p_j) \right) \right]$
Appendix C

Formulas for the union of the two models

This appendix contains all formulas for the union. These formulas together with the formulas of the first and the second model (see appendices A and B) form the extended model. See chapter 4 for a description of the language and vocabulary used in this appendix.

C.1 Static predicates

\( Position(b, p) \): Block \( b \) belongs to the position \( p \).

\( Utmost(b, p) \): \( b \) is one of the utmost blocks in the position \( p \).

\( Utmost(b, p) := Position(b, p) \land \exists b' [(Next \_ block(b, b') \lor Next \_ block(b', b)) \land \neg Position(b', p)] \)

C.2 Constraints

\( \forall b \in B \ [Occ(b) \leftrightarrow \exists p \in P [Position(b, p) \land P \_ occ(p)]] \).

\( \forall p_i, p_j \in P \forall b \in B \ [p_i \neq p_j \land \left( P \_ occ(p_i) \land \left( Position(b, p_i) \land \left( Position(b, p_j) \Rightarrow \left( Utmost(b, p_i) \land Utmost(b, p_j) \right) \right) \right) \right] \)
Bibliography


