Verification of Train Control Systems: Tools and Techniques

Andrew Lawrence

June, 2014

A thesis submitted to Swansea University
in candidature for the degree of Doctor of Philsophy

Department of Computer Science
Swansea University

Declaration

This work has not previously been accepted in substance for any degree and is not being currently submitted for any degree.

June , 2014

Signed:

Statement 1

This thesis is being submitted in partial fulfilment of the requirements for the degree of a MRes in Logic and Computation.

June , 2014

Signed:

Statement 2

This thesis is the result of my own independent work/investigation, except where otherwise stated. Other sources are specifically acknowledged by clear cross referencing to author, work, and pages using the bibliography/references. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure of this dissertation and the degree examination as a whole.

June, 2014

Signed:

Statement 3

I hereby give consent for my thesis to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

February 18, 2011

Signed:

Contents

Chapter 1

Introduction

1.1 Introduction

This thesis is concerned with the application of formal methods within the Railway Domain. Firstly we present a new approach to develop verified SAT solving algorithms and which have been applied the verification of a real world train control system: the solid state interlocking. Secondly we present an approach to formalise the European Rail Traffic Management System (ERTMS) and apply a model checker to verify the systems safety.

1.2 Aim

We have 2 sets of aims. The first set of aims is in regard to the development of verified SAT solving algorithms and their application to the verification of solid state railway interlocking programs:

- Formalize the DPLL proof system and a proof of its completeness.
- Extract a standard DPLL SAT algorithm from the formalisation into a functional programming language and test its performance
- Modify the formalisation and completeness proof of the DPLL proof system so that it captures the behaviour conflict driven clause learning SAT algorithms.
- Extract a clause learning SAT algorithm from the formalisation of the modified DPLL proof system and show that clause learning increases the efficiency of the solver.
- Apply verified SAT algorithms to the verification of solid state railway interlocking programs.

Our second set of aims is in regard to the formal specification and verification of the European Rail Traffic Management System (ERTMS) using Real Time Maude.

- Formalise ERTMS as a hybrid automata.
- Formalise ERTMS as a Real Time Maude specification.

- Verify ERTMS using Real Time Maude's linear temporal logic model checker.

1.3 Thesis Outline

Chapter 2

Background: Traditional Railway Control Systems:

The birth of the railways occurred in the 1800s and from that point until the present day they have experienced continual growth, change and development. In the beginning the burden of safety was placed solely on human shoulders and has since been placed on mechanical systems before finally being transferred to electronics systems that are in place guaranteeing safety today. In the following we shall present a brief history of the railway followed by information on our industrial partner Invensys Rail. Following this general introduction we shall look into specific equipment found on the modern railway and other information needed to understand the domain. Most importantly we shall describe the Westrace interlocking, a modern system charge with guaranteeing safety on the railway, which is produced by Invensys. In the final part of this chapter we shall look at some previous work in this field.

2.1 A History of Railway Signalling and Control Systems

Initially in the early days of the railway there was not fixed signals as we see today. Instead Policemen stationed at junctions and railway stations, were charged with signalling train drivers using a system of flags or oil lamps and changing the points of the railway. A major problem of the time, without telecommunications, was that there was no way of locating a train once it went out of sight. This meant that in practice the only safety precaution that could be taken was to delay the departure of trains using an egg timer which would hopefully space out their positions along the track. The level of safety resulting from this precaution was acceptable as train speeds were not relatively high at the time.

The most important type of signal found in the modern railway is the **fixed signal** which are static, positioned by the side of the track and visually present information to the train driver. The first fixed signals were wooden boards shaped in such a way to provide specific information attached to poles which rotate around a vertical axis. Typically these boards would form a signal instructing the train driver to stop, however if the board was rotated side-on to the driver then it becomes inactive and the driver would proceed if it is safe to do so.

The next major enhancement of these signals came in the form of the **Semaphore** fixed signal. Instead of having 2 positions (visible/ not visible), the boards of a semaphore signal could be moved into one of several predetermined positions. The most common of these signals had 3 visible positions or aspects in which they could be placed which would rely the following information to an approaching driver: proceed if its safe, proceed with caution if its safe and stop.

The introduction of the semaphore signal also foreshadowed another major change in the system for controlling the signals behind the scenes. A new band of professional **Signallers** were employed to specifically manage the railways, replacing the Policemen that proceeded them in the progress. The efficiency of signal control was also improved by a system of wires, pulleys and levers which enabled multiple signals and points to be controlled from a central position. This central position was typically enclosed in a box giving rise to the name **signal box** which were typically manned by one or more signallers. The controlling signals and points from the centrality of the signal box also enabled for more safety mechanisms to be installed. The most prominent of these the **interlocking** shall be the subject of interest to us in later chapters. The interlocking physically prevented the control system of a railway from unsafe state by physically locking levers.

The advent of electricity brough about a further advance in railway technology, allowing for the development of the **track circuit**. When a train was on a segment of track it completed the circuit between the two rails and light up an indicator in the signal box. As more track circuits were deployed it became possible to control certain signals without human intervention. These **automatic signals** were completely operated by track circuits and prevented trains from entering a segment of track behind a signal already containing a train. Electric motors enabled the construction of electric point machines enabling the signaller to electronically control a point at a great distance with little physical effort. The area control by each signal box was greatly increased along with the number of signals controlable by one signaller. Electromechanical **relays** were introduced as a replacement for the soley mechanical relays reducing the size of each signal box and its footprint on the landscape.

Further instances of an electrical system replacing a mechanical one occurred in the 1920s and subsequently in the 1930s. New **colour light signals** replaced the mechanical semaphore signals and had the advantage of being brighter than the oil lamps of that signal. Electronic **control panels**, which contained switches and buttons, replaced the mechanical levers and removed the remaining physical effort needed to operate signalling systems. The combination of track circuits, electronic points and control panels enabled the introduction of **route setting** where by at the press of a single button a configuration of signals and points could be selected thus enabling a train to travel down a particular route along the track.

The next major advancement came some 50 years later and was enabled by the development of microprocessors. The relays and mechanical interlockings were replaced with **solid state interlockings** which were smaller still and more reliable than relay interlockings. These solid state interlockings are one of the systems verified later on in the PhD. Jump forward another 30 years to the current day and we are seeing the development and deployment of a new type of control system which makes use of recent technologies such as GPS, GMS radio communications that enable signallers to have a far finer grain of control when compared with the discrete track circuits and do not need the physical coloured light signals to function. One such instance of a control system is the European Rail Traffic Management System which is

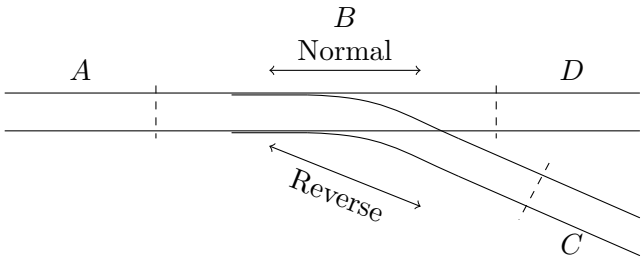


Figure 2.1: A Typical Junction

discussed in more detail in Chapter **??**

2.2 Invensys Rail

Invensys Rail [**?**] has been producing train control equipment and safety devices for over 140 years starting with its original incarnation Westinghouse Rail Systems Ltd. The first such safety device was train air brakes, if the power was cut from the train then the brake would automatically kick in, entering a failsafe state. The company was also charged with supporting British Rail in its development of the first solid state digital railway interlocking to be installed in Leamington Spa. Currently their operations are focused on supplying railway control systems globally with customers in Australia, Hong Kong, Germany , Spain and the UK. One of the products produced by Invensys that we are interesting in is a solid state interlocking called the Westrace. This interlocking continually receives input from the railway, decides whether or not a request or railway state is safe using a Ladder Logic program and prevents the railway from entering unsafe states. This ladder logic will be explained later in this chapter. Further information on the railway domain including terminology and methodology can be found in a book by David Kerr and Tony Rowbotham (See [KR01].)

2.3 An Overview of the Railway Domain

In this section we present the features of the railway domain that are in the scope of this thesis. We hope to provide the reader with the background information and terminology necessary to understand the parts of this thesis.

2.3.1 The Railway Topology: Track and Points

We will now present an overview of the physical railway from a topological point of view. To do this we will present an example of a small track plan of a junction. If the reader is interested in learning more about the topology of the railway a more detailed description can be found in [KR01]

2.3.1.1 Track Segments

A section of track is typically broken down into track segments each containing one or more track circuits to detect the presence of a train. Typically track segments become larger on long straight stretches of track without any interesting topological features such as junctions or stations. Likewise track segments become smaller around junctions and stations where control over train movement is of greater importance.

2.3.1.2 Points

A point is a physical piece of equipment that is used to form a junction. Due to the nature of the rails and trains it is not possible to physically to just join two segments of track. Instead a point is needed to act as physical switch controlling the flow of trains through a junction. A point has two positions which are referred to as **normal** and **reverse**. This presents a safety hazard, for example see figure 2.1, if a train enters the junction b from c when the junction is locked in the position for normal then the train will be derailed.

2.3.2 Railway Signalling

Signals are the main means used to communicate information regarding the state of the track ahead of the train. Typically they are placed either on the track side or over hanging the railway. Visual indications known as aspects are used to convey information to the driver. A signal will have many such aspects which can be displayed, each with a particular meaning. The main type of signal considered in this project is the coloured light signal. Typically these have between one - four aspects each conveying a different indication about the state of the track ahead. Below is a description of the aspects used for a three light signal.

Green - If this aspect is displayed it indicates that track ahead is clear for a sufficient distance and the train driver can proceed at full speed to the next signal.

Yellow - This aspect indicates the track immediately ahead in between this signal and the next is clear however the driver should proceed with caution as a train could be in the track after that.

Red - This aspect indicates that the track ahead is not clear, the driver should stop and wait at this signal.

The one aspect signal is typically a fixed red indicating that is not possible to proceed down the track at this current point in time. The two - four aspect signals are used on tracks with different speeds to convey different stopping distances. The two aspect signal for instance would be used on a low speed track segment where stopping distances are relatively short. Whereas the four aspect signal would be used on a high speed line where stopping distances are long and the driver needs information for a greater length of track. These signalling schemes are fixed in the UK however they are not fixed from country to country. On the continent, for example, they may use different conventions, colours and number of lights on each signal.

2.3.3 The Westrace Interlocking

The railway interlocking is a key component in ensuring the safety of the railway. Its job is to apply a set of rules to the requests and commands it receives from the control system and check whether or not the future state of the railway is safe. If the control signals it receives do not violate the safety of the railway then these signals are committed to the physical infrastructure. For example if the human controller requests for a route to be set the interlocking will process this request and ensure that it does not conflict with other routes before allowing the command to be passed to the physical railway.

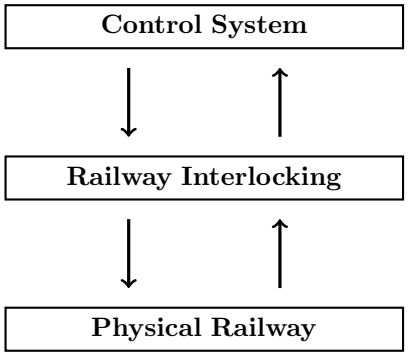


Figure 2.2: The Location of the Railway Interlocking

The railway interlocking repeatedly executes a program or set of rules over some discrete time interval. Each time it uses the set of rules it contains to process a new set of inputs before committing them as outputs. The Westrace interlocking used by Invensys Rail executes a so-called ladder logic program to perform this process. The following are the three main stages of operation in the running of an Westrace interlocking.

Reading of Inputs - Read inputs from the control systems as well as the physical railway infrastructure.

Internal Processing - Execute the ladder logic program with the above inputs and calculate outputs.

Committing of Outputs - The outputs calculated in the previous cycle are then passed on to various places including the physical railway.

2.4 Ladder Logic

The Westrace Interlocking performs calculations by executing a ladder logic program [**?**]. In the following we will look in more detail at these ladder logic programs. The main concepts behind their construction and behaviour will be presented. In later chapters we will provide a formal framework for the verification of these programs.

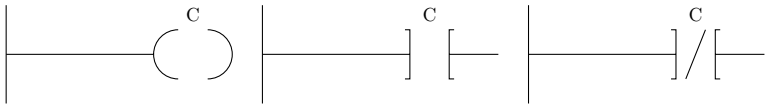
The international standard for programmable logic controllers IEC 61131 [**?**] describes the graphical language ladder logic. It gets its name from its graphical "ladder" like appearance which was chosen to suit the control engineers responsible for their design. Each rung of the

ladder is used to compute an output variable from one or more input variables in the rung. In the railway industry these input variables are referred to as contacts and the output variables are referred to as coils. A description of the entities representing these variables is as follows:

Coils : These are used to represent values that are both stored for later use and output from the program. The value of a coil is calculated when a rung fires making use of the current set of inputs, the previous set of outputs and any outputs already computed for this cycle. The coil is always the right most entity of the rung and its value is computed by executing the rung from left to right.

Open Contacts : This entity represents the value of an un-negated variable

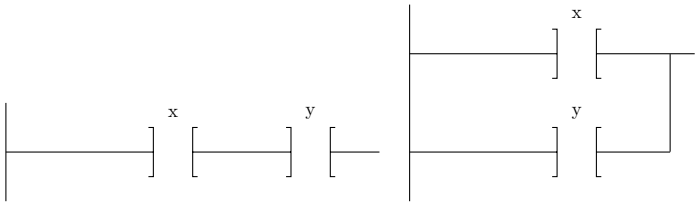
Closed Contacts : This entity represents the value of a negated variable.



(a) A coil (b) An open contact (c) A closed contact

Figure 2.3: The Entities Used In Ladder Logic

A Ladder logic rung is built using these entities and connections between them. The shapes of the connections between the contacts determines how the value of the coil is computed from them. Using propositional logic for comparison, a horizontal connection between two contacts represents logical conjunction and a vertical connection between two contacts represents logical disjunction see Figure 2.4.



(a) $x \wedge y$ Conjunction (b) $x \vee y$ Disjunction

Figure 2.4: Logical Connectives In Ladder Logic

In section 4 an approach is presented to capture the semantics of ladder logic programs using propositional logic.

2.5 Previous Work in this Field

Previously James carried out work for Invensys, applying various SAT and model checking techniques to verify the correctness of a simple pelican crossing and two existing railway interlockings consisting of approximately 500 rungs (see James [**?**]).

James used Kanso's work (See [**?**]), in particular his translation from ladder logic into propositional logic, and applied several model checking techniques in order to try and reduce the complexity of the problems. Both the work by Kanso and James was based on an early feasibility study by Fokkink and Hollingshead [FH98]. The relationship between a ladder logic program and propositional logic was discussed in great detail. A method for formulating such a ladder logic program as a formula in propositional logic was presented. This laid the ground work for all successive projects involving ladder logic. The possible application of program slicing was discussed and this was later applied in the work by James [**?**].

Some of the techniques applied by James to the verification of ladder logic are discussed below.

Bounded model checking: This was the main topic of the work of Phil James. It had the advantage that it produced counter example traces which are highly valuable to the engineers at Invensys. It allowed for the verification of 2000 iterations of the ladder logic programs provided without programming slicing and up to 20000 iterations of ladder logic programs with program slicing.

Temporal Induction: This is another technique used in the verification of the ladder logic programs, it succeeded whenever the inductive verification method Kanso applied also succeeded. It should however be stronger than inductive verification but no example was found to prove this. Temporal induction produced a counter example whenever the bounded model checking produced a counter example.

Program Slicing: This technique was combined with application of bounded model checking to reduce the state space requiring verification. This reduced the number of rungs in a ladder logic program by up to a factor of 10.

Chapter 3

Background: The European Rail Traffic Management System

Following the successful use of the solid state interlocking for over 30 years Railway Engineers are trying to employ modern technologies such as GPS and mobile data communication to gain a much finer control than the tradition discrete solid state interocking and their boolean track circuits would allow. This has led to the development of the European Rail Traffic Management System (ERTMS) which combines the tradition discrete control systems of the railway with continuous systems. This new system consists of a *radio block processor* which negotiates with the tradition interlocking in order to grant lenghts of track to a train in the form of a *movement authority* (MA). Trains are free to move along the lengths of track granted to them in the form of a MA and communicate their position, speed and requests for further segments of track over radio to the radio block processor.

ERTMS levels

ERTMS is seperated into different levels of implementation. These range from being a compliment to traditional signalling to a complete replacement for it. Currently both level 1 and 2 have been implemented in different parts of the world and level 3 is still in the conceptual phase and has yet to be implemented anywhere.

ERTMS Level 1: (Figure 3.1) Movement authorites are calucated based on the underlying signalling system and then transmitted from the trackside to the train through the use of an electronic beacon called a Eurobalise. The Eurobalise also enable train detection at the trackside and pass relevant information regarding train movement via the landside equipment unit (LEU) to the interlocking and signalling centre. While this level makes use of an interlocking, it does not make use of a radio block controller unlike later levels of ERTMS.

ERTMS Level 2: (Figure 3.2) Movement authorities are calucated based on the blocks contained in the underlying signalling system by the radio block processor which is located in the radio block centre. Data is continously transmitted via radio between the radio block centre

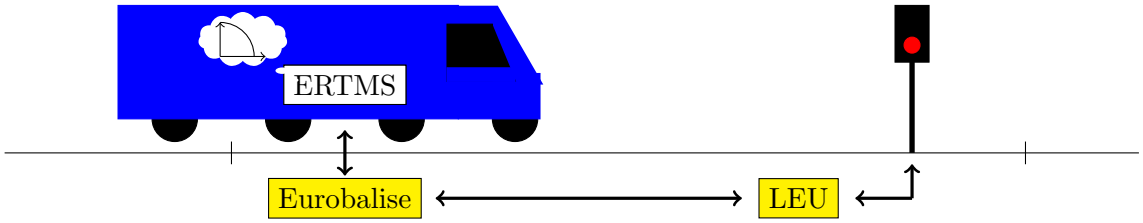


Figure 3.1: ERTMS Level 1

and the train in both directions. Movement authorities are transmitted to the train and the train gives its location relative to a given balise This transmission is complemented by on the spot transmission between the track side balise and the train. These track side balise act as both a beacon and a point of reference enabling the train to determine its location along the track. Train detection is also performed at the trackside using track circuits and axle counters, however these determine whether a segment of track is occupied for the purpose of route setting, rather than the exact location of the train. .

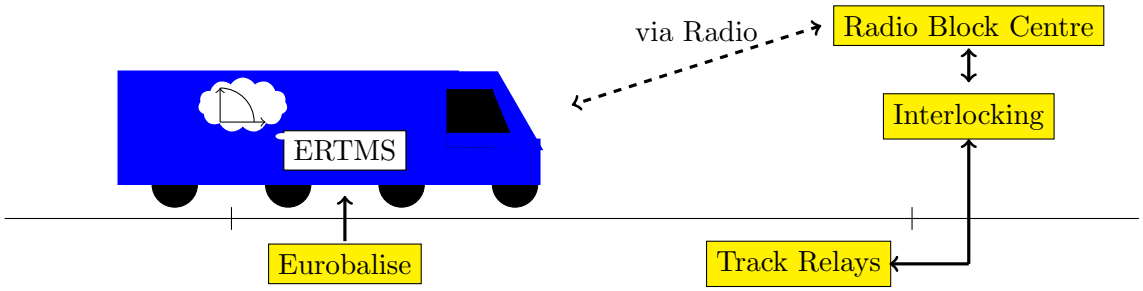


Figure 3.2: ERTMS Level 2

ERTMS Level 3: (Figure 3.3) Instead of using the block system based on track circuits from previous levels, the train itself is considered a moving block. Route locking and route releasing are performed by the radio block centre using information gathered from trains. Similarily to ERTMS Level 2 communication takes place via radio between the train and the radio block centre. Integrity checking is performed on board the train as it continously monitors its position along the track.

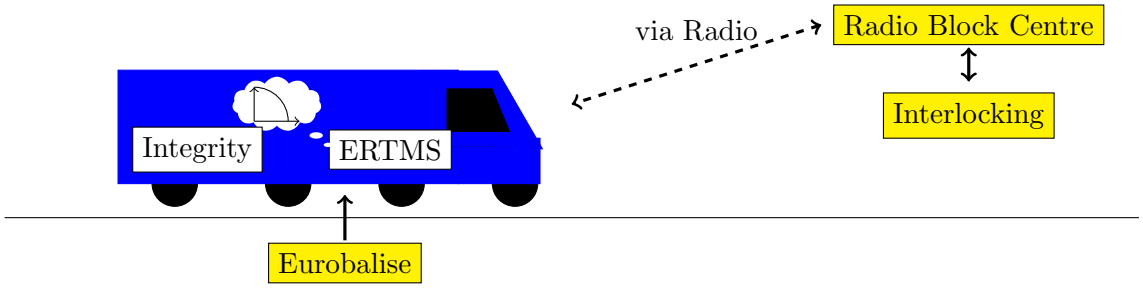


Figure 3.3: ERTMS Level 3

Movement Authorities

There are two different types of movement authorities used to restrict the movement of a train along a piece of track (See Figure 6.2).

End of Authority (EoA) is the end of the movement authority for a train on a given piece of track. It is a point which the train can not proceed past along that track.

Limit of Authority (LoA) is a trains maximum allowed velocity (VMax) at a position on a piece of track and is calculated using breaking curves.

The on board control system will automatically apply the brakes if the train exceeds either type of movement authority.

An Informal Description of the Combined System

In the following we will present a high level description of the combined interlocking and radio block processor system. The aim of specifying a combined system is to get rid of the inconsistencies and problems caused by the interface between the RBC and interlocking. A prime example of such an inconsistency occurs when a high speed ERTMS enabled train travels along a line, under the guidence of a movment authority it is possible for the train to travel through an ordinary red light.

The components of the system which we are trying to capture can be found inside the dotted line in figure 3.5. These consist of the interlocking and radio block processor. Currently the interface between them is biased in one direction, most of the information travels from the interlocking locking to the radio block processor. Some information does go in the other direction for example the radio block centre will notify the interlocking when a ERTMS enabled train is travelling along the line. However other possibilities for a partition exist in a combined system exist, for example the route setting and releasing could take place in the radio block centre. There is also different possibilities from the level of partitioning/ integration of the system, it could be the case that a fully integrated system is much simplier however such a system would have to be constructed from scratch.

We will make the following assumptions in the modelling of the combined system:

- We assume that positioning works and that the location of the trains is known at all times.
- We assume that all trains are fitted with the ERTMS system.
- We will only work with one radio block controller. Trains are assumed to enter and exit the controlled area.

As part of this project we will also have to formalise new safety properties for the combined system. Some of examples of these are as follows:

- A point should not move if it is in a movement authority
- Two trains should not have over lapping movement authorities

3.1 Typical ERTMS Scenario

Consider the following scenario involving 3 trains attempting to cross a double junction.

- Train A would like to travel in a route from D to C
- Train B would like to travel in a route from A to B
- Train C would like to travel in a route from F to C

We will assume that initially no routes are set. This means that either a controller or a time table will determine the precedence of the trains. For this example we will time table the trains in alphabetical order.

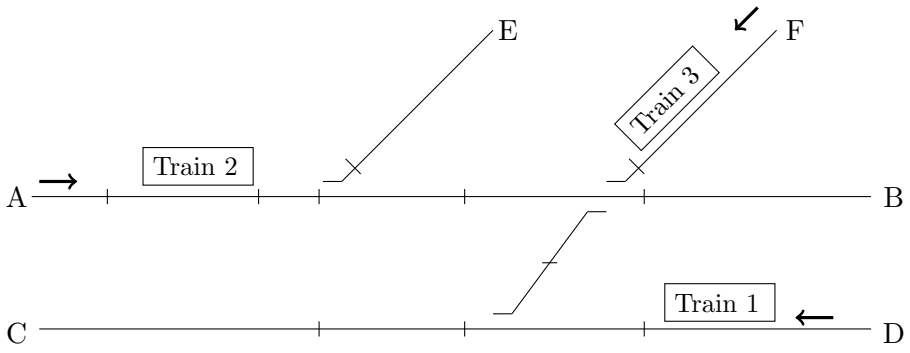


Figure 3.6: A Common Scenario

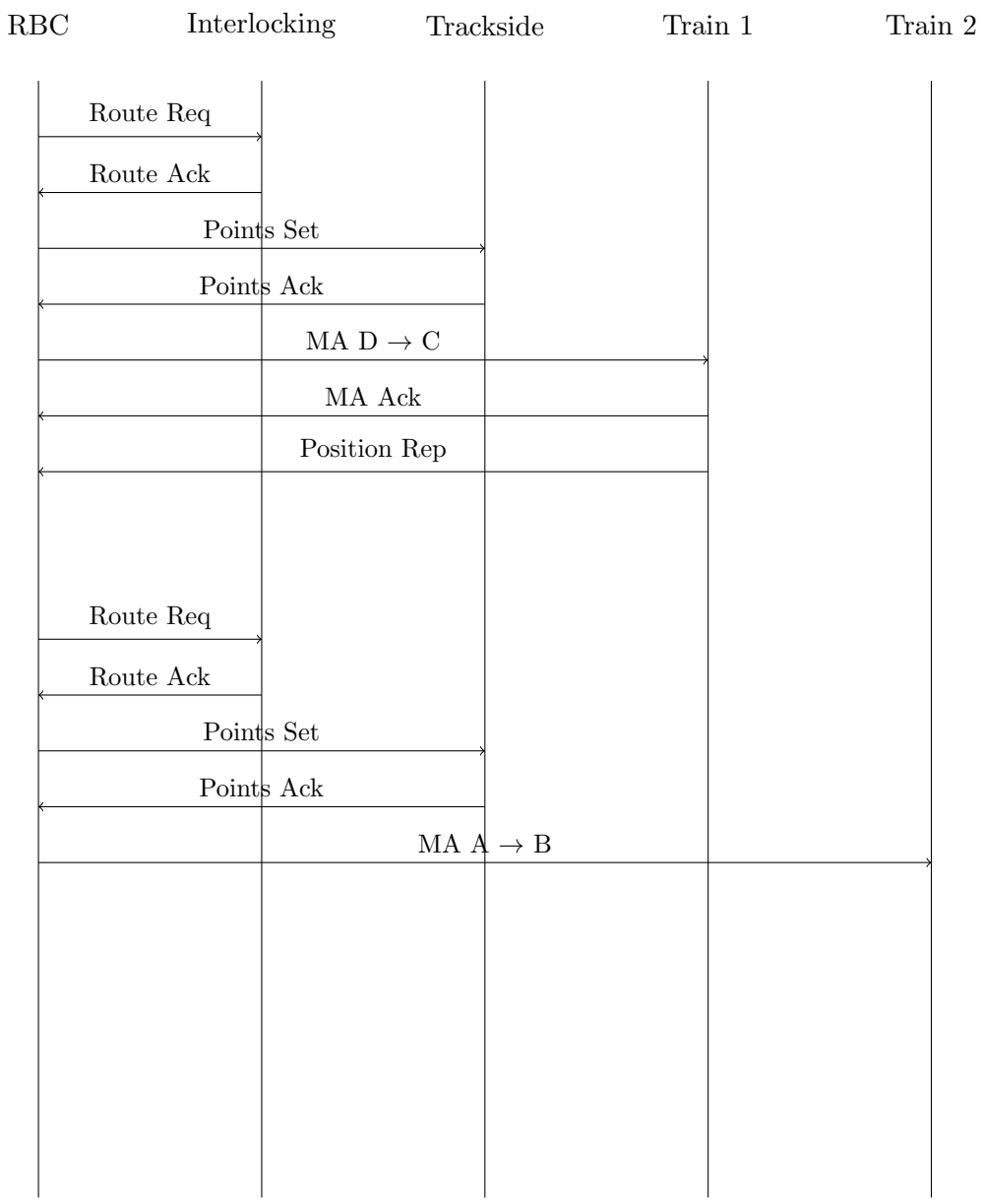


Figure 3.7: Message Sequence Chart for "A Common Scenario"

3.2 Previous Work: Attempts to Verify ERTMS

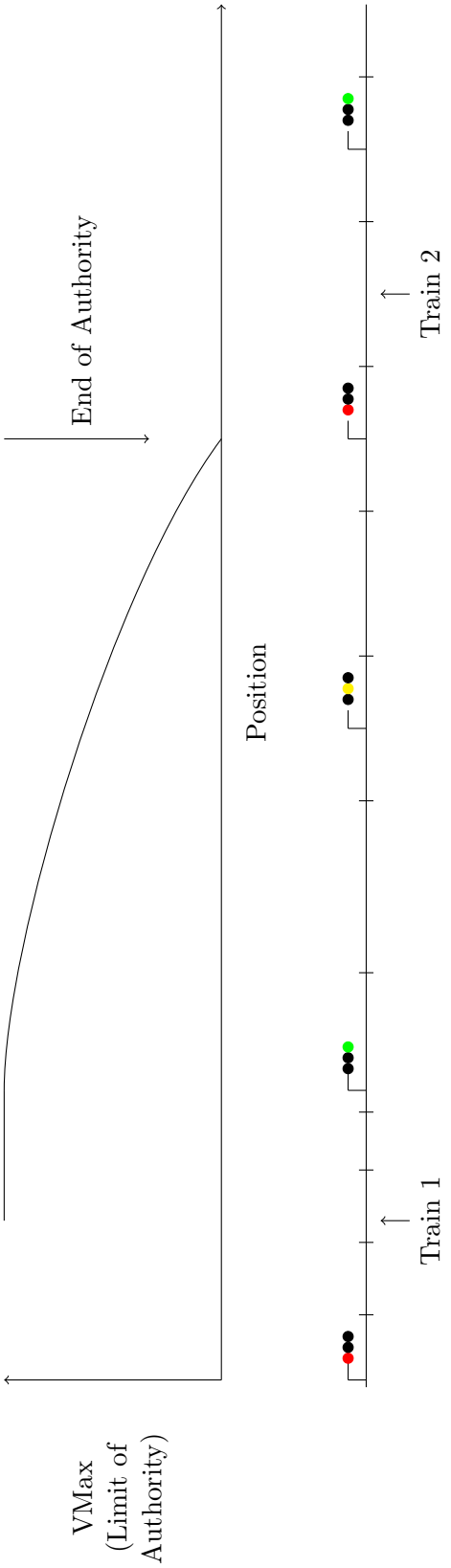


Figure 3.4: Movement Authorities in ERTMS Level 1

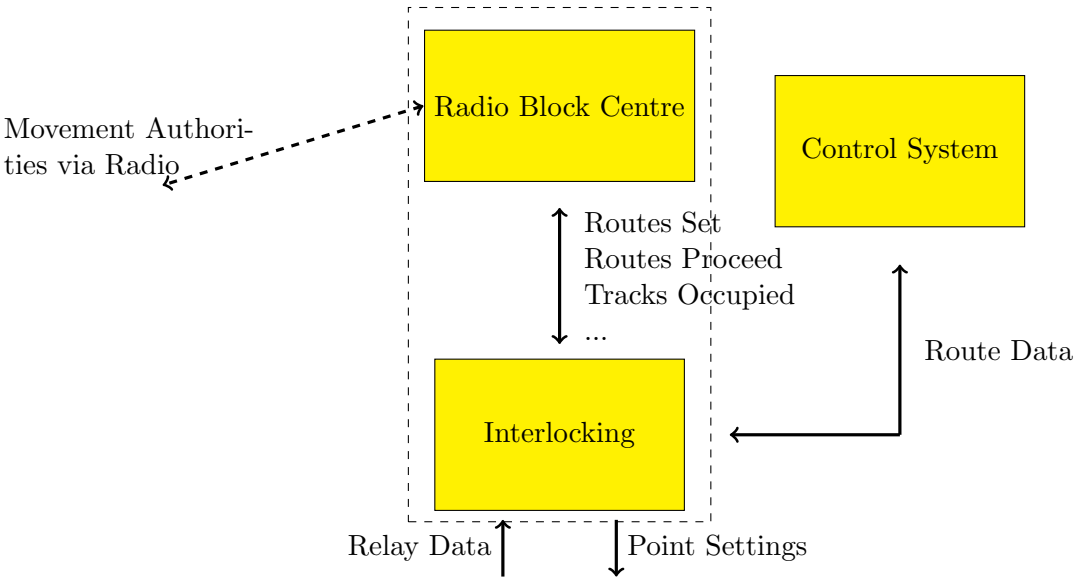


Figure 3.5: Current Situation

Chapter 4

Extracting Verified Decision Procedures

Chapter 5

Extracting a Clause Learning SAT Algorithm

Chapter 6

The Application of Real Time Maude to Model and Verify the European Rail Traffic Management System

In the following we present the Maude [CDE+03, Mau] and Real Time Maude [ÖM02, ÖM04, RTM] tools and describe an approach using these tools to model and verify ERTMS **??**.The property we verify is that no two trains on our example railway share the same movement authority which in turn prevents the trains from crashing. We begin by capturing ERTMS as a hybrid automaton which allows us to get a grip of its behaviour. We then prove that .. Finally we introduce Maude, Real Time Maude and the Real Time Maude LTL Model Checker which we shall use to model, simulate and formally verify ERTMS.

Related Work

The Maude System has been used for a variety of specification and verification tasks. Hardware such has microprocessors has been specified and verified [Har02].

On the front of hybrid automata, ERTMS has been verified in the form of Hybrid Automata by translating the automata into a program and then performing model checking using the post condition calculus [IMN13]

6.1 The European Rail Traffic Management System

6.2 Formalising the European Rail Traffic Management System Using Hybrid Automata

One formalism which we can use to reason about systems such as ERTMS is hybrid automata [Hen96]. We shall give a theoretical overview of Hybrid Automata before formalising ERTMS as 3 Hybrid automata composed in paralell. We begin by defining the syntax of a Hybrid automata before defining its semantics in terms of a labelled transition system.

Definition 1 (Hybrid Automaton: Syntax)**.** *The syntax of a hybrid automaton H comprises of the following:*

Variables *A finite set $X = \{x_1, \ldots x_n\}$ of variables which range over the real numbers. The cardinality of $|X|$ is called the* dimension *of H. The variable set has a corresponding dotted variable set $\dot{X} = \{\dot{x}_1, \ldots \dot{x}_n\}$ which represents the continuous changes of variables and a primed variable set $X' = \{x'_1, \ldots, x'_n\}$ that represents values at the conclusion of a discrete change.*

Control graph *A finite directed multigraph (V, E) consisting of a set of vertices which we shall refer to as* control modes *and a set of edges E which we shall refer to as* control switches*.*

Initial, invariant and flow conditions *Three functions* init,inv *and* flow *that label each control mode $v \in V$ with three predicates such that for all $v \in V$,* $\text{FreeVar}(init(v)) \subseteq X$, $\text{FreeVar}(inv(v)) \subseteq \dot{X}$ *and* $\text{FreeVar}(flow(v)) \subseteq X \cup \dot{X}$*.*

Jump Conditions *A function* jump *that labels each edge $e \in E$ with a predicate such that* $\text{FreeVar}(jump(e)) \subseteq X \cup X'$*.*

Events *A finite set Σ of* events*, at least one of which is assigned to each control switch by a labelling function event $: E \to \Sigma$.*

Where $\text{FreeVar}(P)$ *is the set of free variables in the predicate P.*

Definition 2 (Labelled Transition System)**.** *We define a* labelled transition system $LTS = (S, S_0, T, L)$ *as follows:*

- *S is a possibly infinite set of states*
- *S_0 is a set of initial states $S_0 \subseteq S$*
- *L is a set of* labels
- *T is a binary relation $\xrightarrow{a}$ over the state space S.*

When defining the semantics of a hybrid automata we need to be able to speak about parts of the state space. We define a *region* as being a subset of the state space $R \subseteq S$.

Definition 3 (Transition Semantics of Hybrid Automata)**.** *We define the semantics of a hybrid automaton H in terms of a* timed transition system $LTS^t_H = (S, S_0, L, T)$ *as follows:*

- *The state space S of the timed transition system is defined as, $S, S_0 \subseteq V \times R^n$. A state is in the state space $(v, x) \in S_0$ iff the closed predicate $inv(v)[X := x]$ holds. In addition*

to the previous condition a state is in the initial state space $(v, x) \in S_0$ *iff the closed predicate* $init(v)[X := x]]$ *holds. We call a subset of the state space* S *a* H-region.

- $L = \Sigma \cup R_{\geq} 0$
- *For all events* $\sigma \in \Sigma$ *such that there exists a control switch* $e \in E$*, define* $(v, x) \xrightarrow{\sigma} (v', x')$ *iif the following conditions are satisified:*
  1. *the source and target of* e *are* v *and* v' *respectively.*
  2. *the closed predicate* $jump(e)[X, X' := x, x']$ *holds.*
  3. $event(e) = \sigma$*.*
- *We define a transition* $(v, x) \xrightarrow{\delta} (v, x')$ *for all nonnegative reals* $\delta \in R_{\geq 0}$ *iff there exists a differentiable function* $f : [0, \delta] \to R^n$ *such that the following holds:*
  1. *The functions first derivate is* $\dot{f} : (0, \delta) \to R^n$
  2. $f(0) = x$
  3. $\forall \epsilon \in (0, \delta)$ *both of the predicates* $inv(v)[X := f(\epsilon)]$ *and* $flow(v)[X, \dot{X} := f(\epsilon), \dot{f}(\epsilon)]$ *hold.*

  We call the founction f *the* witness *of the transition* $(v, x) \to (v, x')$

The following is a simple example railway we shall use for the purposes of demonstrating our verification approach. It contains 5 track segments connected to form a pentagon with two trains. This captures most of the behaviours found in a larger more complicated railway as the trains and control systems view any piece of track as a set of track segments joined together to form a route.

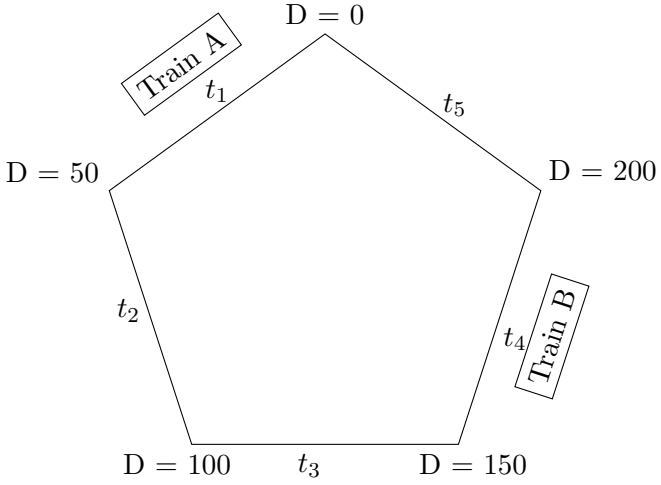
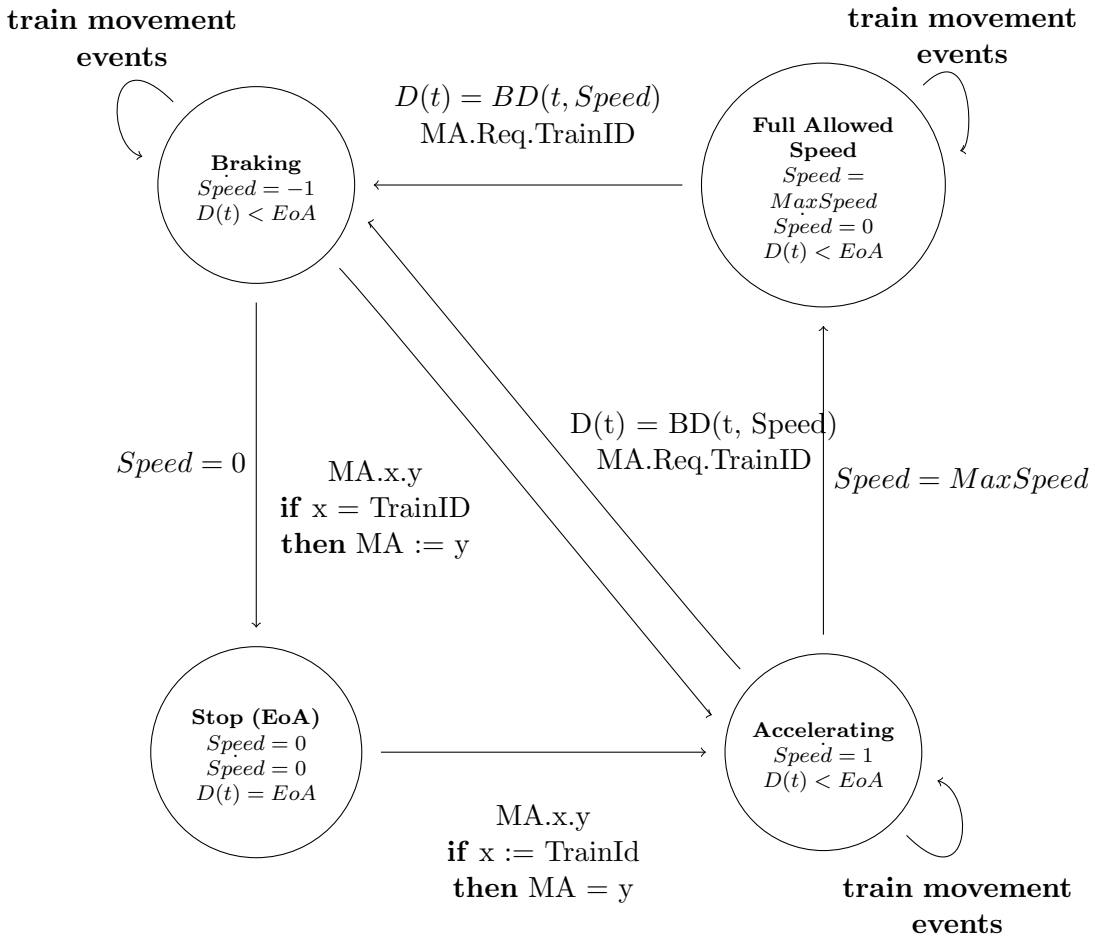


Figure 6.1: Pentagon Example

We have modelled the ERTMS system controlling the pentagon example (see fig 6.2) using several hybrid automata. In this example the value $D(t)$ represents the distance from the start point at time t. It contains 2 trains A and B and 5 track circuits $l_0, \ldots, l_4$. The track is uni-directional allowing trains to travel from $0 - 249$.

Remark 1 (Train Movement Events)**.** *The following two* train movement events *will be abbreviated in some diagrams:*

$$\begin{aligned} \textit{train movement events} := (\textbf{\textit{if}}\ D(t)\, mod\, 50 = 0) \\ \textbf{\textit{then}}\ t_{D(t)\, div\, 50}.t_{(D(t)\, div\, 50)+1} \\ Train.TrainID.Pos.D(t) \end{aligned}$$



We start by defining a hybrid automaton for the interlocking consisting of 2 states representing whether a request have been made or not, 5 boolean variables representing whether a track segment is occupied or not and a variable $ReqID$ which stores the last requested track segment. When a request for a track segment is made that track segment is stored and then if the track freedom condition is met then a request is granted, otherwise it is denied. The event $l_x.l_{x+1}$ captures the movement of a train from one track segment to the next in both the interlocking automaton and the train automaton.

Definition 4 (Interlocking Hybrid Automaton)**.** *We define a hybrid automaton H_{IL} as follows:*

Variables *The state of the interlocking automaton consists of five boolean variables* $\underbrace{l_0, \ldots, l_4}_{Occupied/Free}$

and a variable ReqID ranging over $\{0, \ldots, 4\}$.

Control Graph *The control graph of the interlocking automaton consists of two control modes* $\{Response, Idle\}$ *with four control switches connecting them;* $Response \to Idle$, $Idle \to Response$, $Response \to Response$, $Idle \to Idle$.

Initial, invariant and flow conditions

- $init(Idle) := [l_0 = Free, l_1 = Free, l_2 = Free, l_3 = Free, l_4 = Free]$.

Jump Conditions

- $jump(Idle \to Response) := Req.z, ReqId' = z$
- $jump(Response \to Idle) :=$ **if** $(l_{ReqId} = Free \wedge l_{ReqId+1 \, mod \, 5} = Free)$ **then** $Grant.ReqId$ **else** $Deny.ReqId$
- $jump(Idle \to Idle) := l_x.l_{x+1}, [l'_x = Free, l'_{x+1} = Occupied]$
- $jump(Response \to Response) := l_x.l_{x+1}, [l'_x = Free, l'_{x+1} = Occupied]$

Events

- $event(Idle \to Response) := Req.z$
- $event(Response \to Idle) := Grant.z, Deny.z$
- $event(Idle \to Idle) := l_x.l_{x+1}$
- $event(Response \to Response) := l_x.l_{x+1}$

Before we can descibe the behaviour of the train as a hybrid automaton we must first define the equations of motion on which its behaviour is based.

Definition 5 (Equations of Motion)**.** *The following are equations of motion*

1. $v = u + at$
2. $s = ut + \frac{1}{2}at^2$
3. $s = \frac{1}{2}(u + v)t$
4. $v^2 = u^2 + 2as$
5. $s = vt - \frac{1}{2}at^2$

where $s = displacement, u = initital\ velocity$, $v = final\ velocity$, $a = acceleration$ *and* $t = time$.

Equations number 1 and 2 are the mains one used throughout this formalisation as they allow for the calculation of the distance travelled and speed of a train. The train automaton modelling the trains we make use of a function BD that calculates location required to stop at the EoA based on the trains speed. The simplest way to model deceleration would be to assume that the trains speed decreases at -1 unit of distance per unit of time.

Definition 6 (Breaking Distance)**.** *We define the breaking distance for a train given its movement authority EoA and a speed Speed as follows:*

$$BD(EoA, speed) = EoA - \frac{speed^2}{2} mod 250$$

This hybrid automaton has 4 control modes namely Braking (Auth), Full Allowed Speed, Accelerating and Stopped and has 5 variables namely $Maxspeed$,$D(t)$ (Position), $Speed$, $\dot{Speed}$ and EoA. We have that $D(t) \leq EoA$ is an invariant of the stopped mode and $D(t) \leq EoA$ is an invariant of all other control modes. We define a transition with the event $l_x.l_{x+1}$ which is triggered by the condition $D \, mod \, 50 = 0$ in the modes $FullSpeed$, $Accelerating$ and $Brake$. We compose both the automata for both the interlocking and the hybrid $H_{IL}||H_T$. It is possible for a new MA EoA' with $EoA < EoA'$ to be received in the $Braking$ and $Stopped$ states. This is triggered by the event MA.x.y if $x = TrainID$ then y becomes the new movement authority. The movement event MA.Req.x is used by a train automaton request a movement authority for train x from the RBC automaton.

Secondly we define a hybrid automaton H_T which models an individual train.
Definition 7 (Train Automaton)**.** *We define a hybrid automaton H_T as follows:*

Variables *The state of the interlocking automaton consists of* $\underbrace{D(t), EoA}_{0, \ldots, 249}$,

$\underbrace{Speed, \dot{Speed}, MaxSpeed, TrainID}_{\mathbb{N}}$.

Control Graph *The control graph of the interlocking automaton consists of four control modes* $\{Stop(EoA)$, $Braking$, $Accelerating$, $Full\,Allowed\,Speed\}$.

Initial, invariant and flow conditions

- $init(Stopped) := D(t) < EoA.$
- $inv(FullAllowedSpeed) := \dot{Speed} = 0 \wedge D(t) < EoA$
- $inv(Accelerating) := D(t) < EoA$
- $inv(Braking) := D(t) < EoA$
- $inv(Stop(EoA)) := Speed = 0$
- $flow(Accelerating) := \dot{Speed} = 1$
- $flow(Braking) := \dot{Speed} = -1$

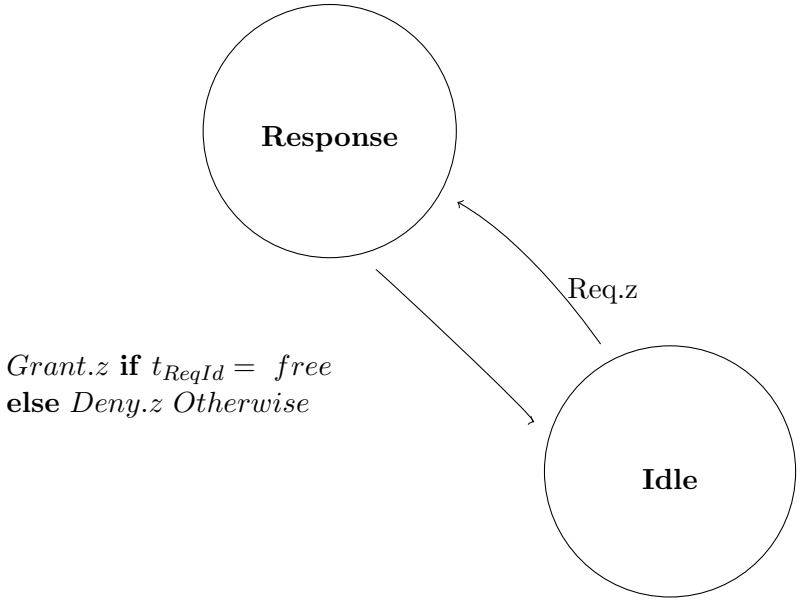
Jump Conditions

- $jump(FullAllowedSpeed \rightarrow FullAllowedSpeed) := l_{D(t)\,div\,50}.l_{(D(t)\,div\,50)+1} \wedge D(t)\,mod\,50 = 0$
- $jump(Braking \rightarrow Braking) = l_{D(t)\,div\,500}.l_{(D(t)\,div\,50)+1} \wedge D(t)\,mod\,50 = 0$
- $jump(Accelerating \rightarrow Accelerating) = l_{D(t)\,div\,500}.l_{(D(t)\,div\,50)+1} \wedge D(t)\,mod\,50 = 0$
- $jump(Stop(EoA) \rightarrow Accelerating) := MA.TrainID.y \wedge EoA' = y$
- $jump(Braking \rightarrow Accelerating) := MA.TrainID.y \wedge EoA' = y$
- $jump(Accelerating \rightarrow Braking) := D(t) = BD(t, Speed) \wedge MA.Req.TrainID$
- $jump(FullAllowedSpeed \rightarrow Braking) := D(t) = BD(t, Speed) \wedge MA.Req.TrainID$
- $jump(Accelerating \rightarrow FullAllowedSpeed) := Speed = MaxSpeed$

- $jump(Braking \rightarrow Stop(EoA)) := Speed = 0$

Events

- $event(Stop(EoA) \rightarrow Accelerating) := MA.TrainID.y$
- $event(Braking \rightarrow Accelerating) := MA.TrainID.y$
- $event(FullAllowedSpeed \rightarrow FullAllowedSpeed) := l_{D(t)\,div\,50}.l_{(D(t)\,div\,50)+1} \wedge D(t)\,mod\,50 = 0$
- $event(Braking \rightarrow Braking) = l_{D(t)\,div\,50}.l_{(D(t)\,div\,50)+1} \wedge D(t)\,mod\,50 = 0$
- $event(Accelerating \rightarrow Accelerating) = l_{D(t)\,div\,50}.l_{(D(t)\,div\,50)+1} \wedge D(t)\,mod\,50 = 0$
- $event(Accelerating \rightarrow Braking) = MA.Req.TrainID$
- $event(FullAllowedSpeed \rightarrow Braking) = MA.Req.TrainID$



Definition 8 (Radio Block Controller Hybrid Automaton)**.** *We define a hybrid automaton H_{RBC} as follows:*

Variables *The state of the radio block controller automaton consists of* $\underbrace{MA_1, Pos_1, MA_2, Pos_2}_{0,\,\ldots,\,2499}$, $\underbrace{LastTrain}_{\mathbb{N}}$.

Control Graph *The control graph of the radio block controller automaton consists of four control modes* $\{Idle, Ready\,to\,Request, Wait, Granted\}$.

Initial, invariant and flow conditions

- $init(Idle) := \ldots$

Jump Conditions

- $jump(ReadytoRequest \rightarrow ReadytoRequest) := Train.x.Pos.y \wedge MA.x = y$

- $jump(Wait \rightarrow Wait) := Train.x.Pos.y \wedge MA.x = y$
- $jump(Granted \rightarrow Granted) := Train.x.Pos.y \wedge MA.x = y$
- $jump(Idle \rightarrow ReadytoRequest) := MA.Req.x \wedge LastTrain = x$
- $jump(ReadytoRequest \rightarrow Wait) := Req.NextTrack(y)$
- $jump(Wait \rightarrow ReadytoRequest) := Deny.LastTrain$
- $jump(Wait \rightarrow Granted) := Grant.LastTrain$
- $jump(Granted \rightarrow Idle) := MA.x.EndOf(z)$

Events

- $jump(ReadytoRequest \rightarrow ReadytoRequest) := Train.x.Pos.y$
- $jump(Wait \rightarrow Wait) := Train.x.Pos.y$
- $jump(Granted \rightarrow Granted) := Train.x.Pos.y$
- $jump(Idle \rightarrow ReadytoRequest) := MA.Req.x$
- $jump(ReadytoRequest \rightarrow Wait) := Req.NextTrack(y)$
- $jump(Wait \rightarrow ReadytoRequest) := Deny.LastTrain$
- $jump(Wait \rightarrow Granted) := Grant.LastTrain$
- $jump(Granted \rightarrow Idle) := MA.x.EndOf(z)$

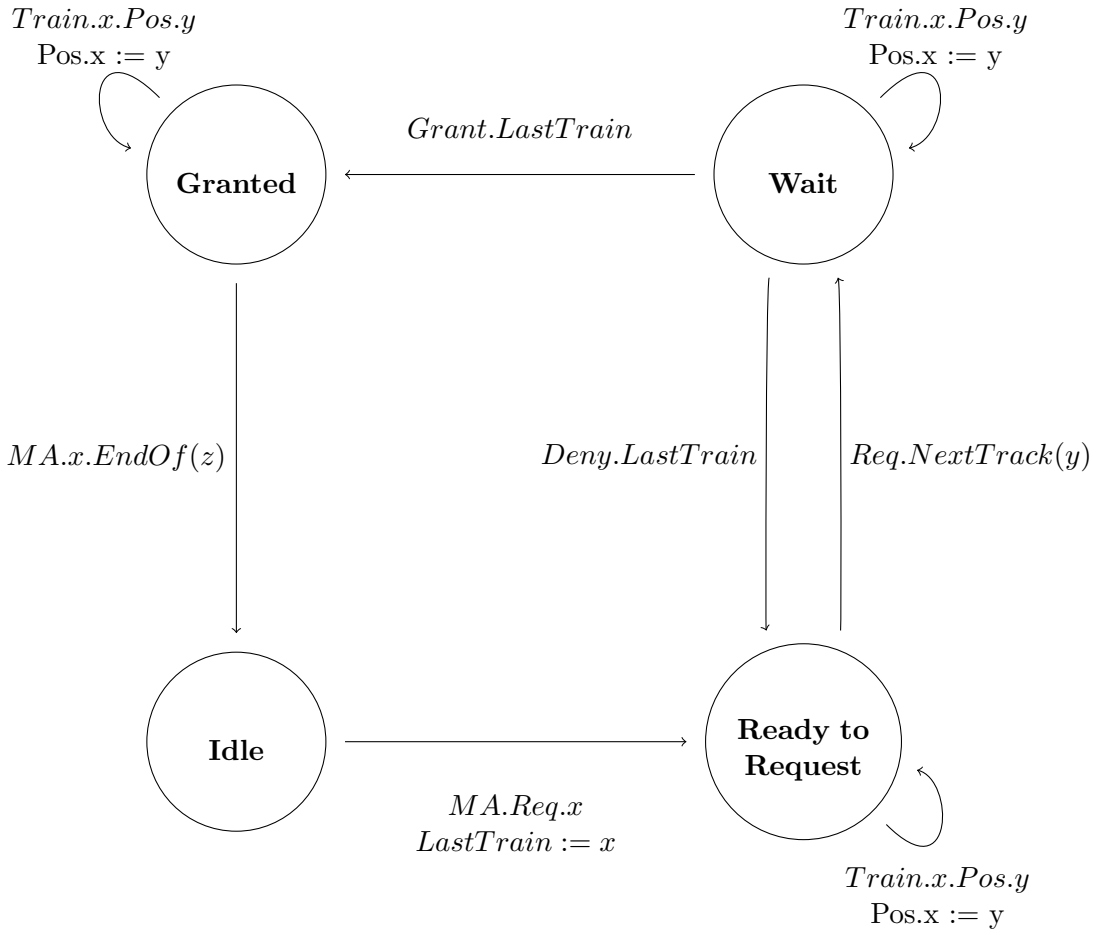
Definition 9 (q_0-rooted and initialised trajectories)**.** *Given a labeled transition system* $LTS = (S, T, S_0, L)$ *and a state* $s \in S$*. We define a* q_0-rooted trajectory *as a finite or infinite sequnce of pairs* $\langle a_i, q_i \rangle_{i \geq 1}$ *of labels* $a_1 \in L$ *and states* $q_i \in S$ *such that* $q_{i-1} \xrightarrow{a_i} q_i \in T$ *for all* $i \geq 1$*. If* $q_0 \in S_0$ *then the seqence of pairs* $\langle a_i, q_i \rangle_{I \geq 1}$ *is an* initialized trajectory*.*

Definition 10. *We define a live transition system (LTS,A) to consist of a labelled transition system LTS and a set A of infinite intialised trajectories of LTS. If for all of the finite initialised trajectories of LTS there exists an infinite trajectory in A such that the finite trajectory is a prefix of the infinite one then we call A* machine-closed*. We define a* trace $\langle a_i \rangle_{i \geq 1}$ *of (LTS,A) to consist of the labels from the either a finite initialised trajectory of LTS or a trajectory in A.*

Definition 11 (Trace semantics of hybrid automata)**.** *All transitions of a timed transition system* S_H^t *have an associated* duration $\delta \in R_{\geq 0}$*. Every event* $\sigma \in \Sigma$ *the transition* $q \xrightarrow{\sigma} q'$ *has a duration of 0. For all transition* $q \xrightarrow{\delta} q'$ *where* $\delta \in R_{\geq 0}$ *the duration is* δ*. We define an infinite trajectory* $\langle a_i, q_i \rangle_{i \geq 1}$ *of a transition system* S_H^t *to be* divergent *if the sum of the transitions* $\Sigma_{i \geq 1} \delta$ *is* divergent*.*

Safety conditions for the combined system can be separated into discrete and continuous parts. If we want to specify a safety condition which states that it is not possible for two trains to collide in our current system this could have the following components.

the continuous part would state that any movement authority issued by the radio block processor would respect the interlockings separation policy. **(Note: I should up date this part at some point. In the current Maude implementation the interlocking will**



grant a track segment if it is not occupied) The discrete part would basically state that there is at least two free track circuits in-between each train.

We will now give a proof by induction that the safety condition "The train will always break on time" holds for a single train automaton. The base case is that we are in the initial state, the stopped mode, with the invariant $D(t) \leq EoA$. If are in the stopped mode and we receive a movement authority with $EoA > D(t)$ then a jump is performed and move to the accelerating state. Since the transition of time does not cause any flow transitions to occur from the base case we do not need to consider it. In the step case we are in one of the 4 modes (Stopped,Braking,Accelerating,) with the invariant $D(t) \geq BD(EoA, speed)$. In these states jump conditions are considered by performing a case distinction is performed on whether we reach the braking point i.e. $D(t) = BD(EoA, speed)$ or we reach maxspeed the transitions result in states in which the invariants still hold. We also consider flow conditions, if time elapses by some amount δ we either reach the breaking point $D(t) = BD(EoA, Speed)$ and perform a jump into a state in which the invariant still holds or $D(t) > BD(EoA, speed)$ in which case the invariant $D(t) \leq EoA$ holds.

Since we have restricted the movement authorities to occur at discrete intervals seperated by 50 units of distances we perform induction over the time interval (0..1) in order to prevent multiple mode changes from occuring durring the induction step.

Theorem 1. *Given a live transition system* $(S^t_{H_T}, L^t_{H_T})$

$$\forall \langle a_i, q_i \rangle_{i \geq 1} \in L^t_{H_T}. \forall (a_n, (v, [D(t), EoA, Speed, \dot{Speed}, TrainID])) \in \langle a_i, q_i \rangle_{i \geq 1}$$
$$\wedge D(t) \leq BD(EoA, Speed) \leq EoA$$

under the assumption that the max speed of the train and the track segment size (new MA increments) obey the following

$$s + s^2 < tsegsize$$

Proof. The proof is performed by fixing a trace $\langle a_i, q_i \rangle_{i \geq 1}$ and a label/state pair (a_n, q_n) in the timed trace in which the property holds and then proving that for all possible successor label/state pairs (a_{n+1}, q_{n+1}). Where the state $q_n = (v, [D(t), EoA, Speed, \dot{Speed}, TrainID])$ and $q_{n+1} = (v', [D(t)', EoA', Speed', \dot{Speed}', TrainID'])$

We begin by performing induction on the control mode of the system. There are 4 cases for v in which we must argue that the transition $q_n \xrightarrow{a_{n+1}} q_{n+1}$ maintains the property $D(t) \leq EOA$. We then perform induction on the time giving us a further two sub cases in which the duration of the transition $\delta = 0$ or $\delta \in \mathbb{R}_{>0}$

v = Stop All possible transitions from the stop mode have a duration $\delta = 0$. There is one possible event that $MA.TrainID.y$ which will grant a new movement authority y such that $EoA < y$ and cause a jump to the *Accelerating* mode with $D(t)' \leq BD(y, Speed') < y$

v = Accelerating In the case that the duration of the transition $\delta = 0$ and the successor state is $q_{n+1} = (Braking, [D(t)' = D(t), EoA' = EoA, Speed' = Speed, \dot{Speed}' = \dot{Speed}, TrainID' = TrainID])$ and either $D(t) = BD(EoA, Speed)$ or $Speed = MaxSpeed$ has occurred. If $D(t) = BD(EoA, Speed)$ then $D(t)' = BD(EoA', Speed') \leq EoA'$. The other case that $Speed = MaxSpeed$, $v' = Full\ Allowed\ Speed$ and $D(t)' \leq BD(EoA', Speed') \leq EoA'$. If the successor state and the current state are the same then an event has occurred and $D(t) \leq BD(EoA, Speed) < EoA$.

In the case that $\delta \in \mathbb{R}_{<0}$ then a flow transition has occured and we are one of several possible control modes depending on whether the transition of time triggered a jump or not. The possibilities are as follows:

Acc: The train has accelerated and at not point in time has the distance become equal to the breaking distance therefore the current distance remains below the breaking distance.

Acc $\rightarrow$ Brake: The train has transitioned between three states during the transition of δ . The transition to the breaking state was causes by the condition $D(t') = BD(EoA, Speed)$ at some t' during the first duration in the Acc control mode which remains invariant during the Brake control mode //

Acc $\rightarrow$ Brake $\rightarrow$ Acc: The train has transitioned between three states during the transition of δ . The transition to the breaking state was causes by the condition $D(t') = BD(EoA, Speed)$ at some t' during the first duration in the Acc control

mode which remains invariant during the Brake control mode before the invariant $D(t'') < BD(EoA, Speed)$ takes hold at all times t'' in the second Acc control mode. //

v = Full Allowed Speed (FAS) In the FAS mode have 4 possible cases

FAS: The train is travelling at maxspeed and then condition $D(t') = BD(EoA, Speed)$ has not been met for all t' in the real interval δ. The invariant therefore continues to hold.

FAS $\rightarrow$ Brake: The time interval δ can be partitioned into two further time intervals δ' in which the system is in FAS and δ'' in which the system is in the Brake mode. The argument from the previous FAS case holds for the first time interval. The end of this time interval marks at point at which the condition $D(t) = BD(EoA, Speed)$ for some t. For the whole of the time period t'' this holds as an invariant $D(t'') = BD(EoA, Speed)$.

FAS $\rightarrow$ Brake $\rightarrow$ Acc: The time interval δ can be partitioned into 3 time intervals t_1 for the period in the FAS mode, t_2 for the period in the Brake mode and t_3 for the period in the Acc Mode. The argument for the time intervals t_1 and t_2 is the same as the argument for the previous FAS $\rightarrow$ Brake case. At the end of the time period t_2 a new movement authority EoA' has been granted such that $BD(EoA, Speed) + 50 = BD(EoA', Speed)$. For the whole of the Acc period the invariant $D(t_3) < BD(EoA', Speed)$ holds as previously $D(t'') = BD(EoA, Speed)$ and it is impossible for the train to cover a distance of 50 in one time unit.

FAS $\rightarrow$ Brake $\rightarrow$ Acc $\rightarrow FAS$: This follows the same argument as the previous however the time period has been partitioned into 4 instead of 3. The invariant holds in the final FAS state for the same reason as the preceding Acc state as the control mode jump does not have any effect on the distance of the train.

v = Braking In the Braking mode we have 5 possible cases depending on whether jumps have occurred between the different control modes.

Braking: The first case is that we remain in the braking mode for the duration of the time interval. The invariant $D(t') = BD(EoA, Speed)$ holds for the duration of the as both the braking distance and distance decrease by the same amount over the time interval.

Braking $\rightarrow$ Stopped: The time interval δ can be partitioned into two separate intervals t_1, t_2 corresponding to the duration of time spent in the two control modes. The argument for the first time interval t_1 is the same as that for the previous Braking case. The transition from the Braking to Stopped modes is triggered by the speed of the train reaching 0. Up to and including this point in time the invariant $D(t_2) = BD(EoA, Speed)$ holds. For the remainder of the interval t_2 the the train is stationary and the invariant continues to hold.

Braking $\rightarrow$ Stopped $\rightarrow$ Acc: In this case the time interval δ can be split up into 3 separate intervals t_1, t_2 and t_3 corresponding to the 3 control modes. The argument for the first 2 control modes is the same as the argument for the previous case. At the end of the interval t_2 a new movement authority EoA' has been granted and $D(t_3) <$

$BD(EoA', Speed)$ since the train has travelled at most one unit of distance during the interval and the movement authority has increased by 50.

Braking $\rightarrow$ Acc: The time interval δ can be partitioned into 2 separate intervals t_1 and t_2 corresponding to the two control modes. The argument for the first control mode is same as that as the argument for the first Braking case. At the end of the interval t_1 a new movement authority is received which greatly increases the braking distance far beyond what the train can travel in this possible time interval.

Braking $\rightarrow$ Acc $\rightarrow$ FAS: The time interval can be partitioned into 3 separate intervals t_1, t_2 and t_3. The invariant holds in the first two time intervals following the argument from the previous case Breaking $\rightarrow$ Acc. At the end of the Acc mode there is a transition to FAS however since there is no possibility of the train reaching the breaking point we know that $D(t_3) < BD(EoA', Speed)$.

□

Another interesting property of our model is that alone the model of the interlocking allows for "jumping trains" i.e. it allows for track circuits to become free and occupied in a way that does not model the normal movement of trains. However when the interlocking automaton is placed in parallel with a train automaton it behaves in a

Theorem 2. *In the composite automaton $H_{IL}||H_T$ the event $t_x.t_{x+1}$ will only occur if and only ift_x is currently occupied in which case t'_x = free and t'_{x+1} = occupied.*

Proof. We have to prove the two directions of the statement. First we prove the $\rightarrow$ direction.

Assume a train is in t_x and $((x-1)\ , mod 5) * 50 \leq D(t) < x * 50$.

There are two cases

1. If the train is in the Stopped state then a $t_x.t_{x+1}$ event will never occur as it is not possible in this state.
2. The train is moving i.e. it is in either of the *Accelerating*, *Braking*, *Full Allowed Speed* states. In this case the position of the train will be increasing and eventually it will happen that $D(t) \, mod \, 50 = 0$. When since $D(t) \, mod \, 50 = 0$ satisfies the jump condition the event $t_x.t_{x+1}$ will occur.

□

6.3 Maude

In the following section we shall describe the term rewriting system Maude. It is a mutli-purpose tool containing support for executable-specification, simulation and verification of systems and software. We make use of all 3 of these capabilities and it is this wide range which draws us towards its use. In order to full understand its use in the specifcation of a real time system we first present a formal and informal definition of a Maude specification followed by an example maude specification to give insight into theses definitions.

6.3.1 Maude Specifications

A Maude specification consists of *functional modules* declared using `fmod` and `endfm` which contain the following:

sorts	`sort` s or `sorts` s s'.
subsorts	`subsort` $s < s'$.
function symbols	`op` f : $s_1 \dots s_n$ `->` s .
variables	`vars` v v' : s'.
uncondition equations	`eq` $t = t'$.
condition equations	`ceq` t = t' `if` *cond*
membership axioms	`mb` t : s . or `cmb` t : s `if` *cond* .

Formally a Maude specification is a *rewrite theory* of the form $\mathfrak{R} = (\Sigma, E, L, R)$, where

$$[l] : t \to t' \textbf{if} \bigwedge_{i=1}^{n} u_i \to v_i \bigwedge_{j=1}^{m} w_j = w'_j$$

where l is a label $l \in L$ and t, t', u_i, v_i, w_j and w'_j are implicitly universally quantified variables representing Σ-terms. Maude theories are *order-sorted* allowing for the specification of subsorts which is achieved by including into the specification a partial order relation over sorts. Given two sorts s and s' this partial order relation $s \leq s'$ is interpretted as subset inclusion $A_s \subseteq A_{s'}$ in a model A.
Definition 12 (Rewrite Theory)**.**

6.3.1.1 Example Maude Specification

The following is a specification of the natural numbers in Maude:

```
fmod BASIC-NAT is
        sort Nat .

        op 0 : -> Nat .
        op s : Nat -> Nat .
        op _+_ : Nat Nat -> Nat .

        vars N M : Nat .

        eq 0 + N = N .
        eq s(M) + N = s(M + N) .
endfm
```

6.4 Real Time Maude

One extension of Maude that has been used to model and verify hybrid systems is Real Time Maude. This contains specifc support enabling the modelling and verification of real-timed and discrete timed systems. There is an example of a distributed sensor network that has been modelled and verified using this tool. It is stated by its developers that Real Time Maude is not first and foremost a verification tool. This is due to some inherent limitations in LTL-model checking that hinder the verification of infinite state systems. We have, as in the sensor network example, only model checked a portion of the state space of our example using this tool. This is however enough to have confidence in the correctness of the system with respect to a given safety condition.

6.4.1 Real Time Maude Specifications

A Real Timed Maude specification consists of *timed modules* that start with `tmod` and end with `endtm`. Formally a Real Time Maude specification is a real-time rewrite theory which can be thought of as a rewrite theory with an interpretation for the abstract time domain together with rewrite rules for terms of type `System` that have a time duration [ÖM02]. These rewrite rules can be seperated into two categories, the *tick* rules have a non-zero time elapse and the *instantaneous* rules have a time elapse of zero.

The interpretation of the abstract time domain is mapped to a concrete specification of time during the specification process. Real time Maude comes with two of these specifications as standard the most simple is discrete time based on the natural numbers and the more complicated of the two is dense time based on the rational numbers. We shall use discrete time during the specification as it allows for the best results during the verification process.

Definition 13 (Equational Theory Morphism)**.** *We define an* equational theory morphism $H : (\Sigma, E) \to (\Sigma', E')$ *to consist of the following*

- *a monotone map* $H : sorts(\Sigma) \to sorts(\Sigma')$ *which maps sorts in* Σ *to sorts in* Σ'
- *a mapping which maps every function symbol* $f : s_1 \dots s_n \to s$ *in* Σ *to some term* $H(f_{s_1 \dots s_n, s})$ *from* Σ' *of sort* $H(s)$ *such that the following conditions are satisfied:*
  1. $\mathrm{Var}(H(f_{s_1 \dots s_n, s})) \subseteq x_1 : H(s_1), \dots, x_n : H(s_n)$
  2. *if the operator* $f : s_1 \dots s_n \to s$ *can be subsort overloaded* $f : s'_1 \dots s'_n \to s'$ *such that* $s_i \leq s'_i, s \leq s'$ *then it is possible to substute each variable* $x_i : H(s'_i)$ *by the corresponding variable* $x_i : H(s_i)$ *in the term* $H(f_{s'_1 \dots s'_n, s})$ *and obtain* $H(f_{s_1 \dots s_n, s})$.
  3. *For every axiom:*
  $$(\forall y_1 : s_1, \dots y_k : s_k) u = v \ \textbf{\textit{if}}\ C$$
  in E *the homomorphic extension of* H^* *to terms and equations in the condition* C *causes the following to hold:*
  $$E' \models (\forall y_1 : H(s_1), \dots, y_k : H(s_k)) H^*(u) = H^*(v) \ \textbf{\textit{if}}\ H^*(C)$$

Definition 14 (Real-Time Rewrite Theory)**.** *We define a* real-time rewrite theory $\mathfrak{R}_{\phi,\tau}$ *to be a tuple* $(\mathfrak{R}, \phi, \tau)$ *containing a rewrite theory* $\mathfrak{R} = (\Sigma, E, L, R)$ *where:*

- *ϕ is an equational theory morphism $\phi : Time \to (\Sigma, E)$ that maps objects in the theory Time to objects in the equational theory (Σ, E).*
- *The time domain is functional i.e. if a term r of sort $\phi(Time)$ has a rewrite proof $\alpha : r \to r'$ then $r = r'$ and the identity proof of r is equivalent to α.*
- *There is a designated sort typically called State and a sort System, contained within (Σ, E), which has no supersorts or subsorts and only one operator that does not satisfy any non trivial equations:*

  $$\{\_\} : State \to System$$

  Further to this condition, it is all required that the sort System does not appear in $s_1, \ldots s_n$ the domain of any operator $f : s_1, \ldots s_n$.
- *For each rewrite rule with u and u' of sort System in $\mathfrak{R}$ of the following form*[1]*:*

  there is an assigned term $\tau_l(x_1 \ldots x_n)$ of sort $\phi(Time)$ within τ.

Example Real Time Maude Specification

The following a very simple model of a train Real Time Maude that moves one unit of distance in one time unit along a circular track of length 500. It defines a sort `TrainState` and a single state `move`. We have a constructor train of type `System` which consists of a train state and a natural number.

```
(tmod DISCRETE-SINGLE-TRAIN is protecting NAT-TIME-DOMAIN .
  sort TrainState .
  ops  move :  -> TrainState [ctor] .
  op train : TrainState Nat -> System [ctor] .

  vars N : Time .
  crl [travel] : {train(move,N)} => {train(move,N + 1)} in time 1 if N < 500 .
  rl [reset] : {train(move,500)} => {train(move,0)} .

endtm)
```

Executing a Real Time Maude Specification

Real Time Maude allows one to execute or simulate a real time system by applying rewriting rules to a term of type `System`. The command `(trew {System} in time <= t)` will attempt to rewrite the system to a state t time units in the future. This isnt always possible though as the system may deadlock. The following command attempts to rewrite a train, which initially has distince 0, to its state in 100 units of time in the future:

`(trew train(move,0) in time` $\leq$ `100 .)`

[1] All other rules which are not of this form are called *local* rules and have an instantaneous zero time elapse. These do not act on the system as a whole but rather on one or more of its components.

The result from this timed rewrite is as follows:

```
rewrites: 4027 in 4ms cpu (3ms real) (1006750 rewrites/second)

Timed rewrite  {train(move,0)} in DISCRETE-SINGLE-TRAIN
with mode deterministic time increase in time <= 100

Result ClockedSystem :
  {train(move,100)} in time 100
```

6.4.2 Object Orientated Specification in Real Time Maude

Real Time Maude is based on Full Maude which contains message and object constructs for object orientated specification. Using these constructs its possible to model ERTMS as several synchronously communicating processes. A Real Time Maude specification consists of *timed object orientated modules* which start with `tomod` and end with `endtom`. We can define the following:

classes : `class C | a1 : Sort-1, ... , an : Sort-n .`
objects : `< O : C | a1 : v1, ... , an : vn > .`

where C is the class identifier O is the object name $a_1 \dots a_n$ are attribute names and $v_1 \dots v_n$ are values.
To model communicating processes Real Time Maude uses a messages construct.

```
msgs M1 ... Mn : Sort-1 ... Sort-n -> Msg .
```

Configurations in Real Time Maude

Objects and messages in a specification are given a sort of `Configuration` which is a subsort of type `System`. The composition operation for these configuration allows for a combination of objects and messages to form new configurations. This composition operation is also commutative meaning the order of messages and objects in any configuration is ignored. This allows for more generalised writing rules in an object orientated specification that speak about specific objects and messages that are part of some bigger configuration that does not affect the firing of the rule.

```
mod CONFIGURATION is
    *** basic object system sorts
    sorts Object Msg Configuration .

    *** construction of configurations
    subsort Object Msg < Configuration .
    op none : -> Configuration [ctor] .
    op __ : Configuration Configuration -> Configuration
         [ctor config assoc comm id: none] .
```

Example Object Orientated Specification

The following is an example consisting of two classes of object which serves to demonstrate some of the advantages of object orientated specification as well as some of the pit falls. Firstly there is the class defining D objects which count modulo 500 and transmit a message containing the current count. Secondly there is class defining a object B that reads messages from a D object.

```
(tomod EXAMPLE1 is protecting NAT-TIME-DOMAIN .
  msgs msgD  : Nat -> Msg .
  class  D | d : Nat .
  class  B | b : Nat .
  vars N: Nat .
  var O : Oid .
  var REST : Configuration .

  rl [makeD] : {< O : D | d : N > REST} =>
  {msgD(N + 1 rem 500)
  < O : D | d : (N + 1) rem 500 > REST} in time 1 .

  rl [readB] : {msgD(N) < O : B | > REST} =>
  {< O : B | b : N > REST}  .
endtom)
```

We can execute the specification as follows:

```
(trew < myD : D | d :  0 > < myB : B | b :  0 > in time <= 2 .)
```

Resulting in the following output:

```
rewrites: 6246 in 7ms cpu (7ms real)
(805000 rewrites/second)

Timed rewrite  {< myD : D | d : 0 > < myB : B | b : 0 >}
in EXAMPLE1 with mode deterministic time
increase in time <= 2

Result ClockedSystem :
  {< myB : B | b : 2 > < myD : D | d : 2 >} in time 2
```

The following are some states reachable in time 2.

```
{< myB : B | b : 1 > < myD : D | d : 2 >}
{< myB : B | b : 2 > < myD : D | d : 2 >}
{msgD(1)< myB : B | b : 2 > < myD : D | d : 2 >}
{msgD(1)msgD(2)< myB : B | b : 0 > < myD : D | d : 2 >}
{msgD(2)< myB : B | b : 1 > < myD : D | d : 2 >}
```

Non-determinism should be avoided unless it is really is a property you want in a model. One way to remove some of these reachable states is to make the variable `REST` of sort

`ObjectConfiguration`. This forces objects of the B class to read messages after each incrementation of time. If we make this modification to the specification then the following states are reachable in time 2.

```
{< myB : B | b : 2 > < myD : D | d : 2 >}
{msgD(2)< myB : B | b : 1 > < myD : D | d : 2 >}
```

6.5 Modelling the European Rail Traffic Management System

In the following we shall give an overview of the specifications used to model the ERTMS system. Following the hybrid automata describe in **??** we have defined one class for each of the 3 hybrid automata.

Modelling the Transition of Time

In our previous Real Time Maude specifications we have had a rewrite rules that work on a specific object or state and increment time for the whole system. This presents a problem as we could have two objects that can both increment the time of the global system independently. We need a way of incrementing the time of the whole global system and then distributing this increment of time amongst the objects of the system. To solve this problem we define operator δ which describes how distributed system evolves with time.

```
op delta : Configuration -> Configuration [frozen] .

var OCREST : ObjectConfiguration .

op delta : Configuration -> Configuration [frozen] .

rl [timetrans] : {OCREST} => {delta(OCREST)} in time 1 .
rl [delta1] : delta(CON1 CON2) => delta(CON1)delta(CON2) .
```

Modelling Trains

We shall now describe the specification of trains. We define the Train class to have a state which like our hybrid automata can be in one of 4 possibilities, constant speed, accelerating, stopped and breaking. As well as the state it also has 6 fields of sort Nat representing the distance, speed, acceleration, movement authority, current track segment and max speed.

```
sort TrainState .
ops  cons acc stop break :  -> TrainState [ctor] .

class Train | state : TrainState, dist : Nat, speed : Nat, ac : Nat,
                                ma : Nat, tseg : Nat , maxspeed : Nat .
```

We capture the behaviour of trains using a number of instantaneous and tick rewriting rules. The instantaneous rules are used to capture a change of state within a train. For example if the a train is in the cons state distance to the movement authority becomes less than the distance needed to break to zero in the next moment of time one of these rules will fire and cause a state transition.

We use the following function to compute the breaking distance:

The division operation is based on repeated subtraction and finally taking the floor or the resulting number. When we divide a number by 2 in the breaking function we add 1 to change this to the ceiling of the resulting number. This prevents a situation where we have a distance of 1 from our moment authority but the breaking distance is 0.

Modelling the Radio Block Processor

We define the RBC state as follows:

```
sort RBCState .
ops rbcidle ready wait :  -> RBCState [ctor] .
```

We define the RBC class as follows:

```
class RBC | state :  RBCState, lasttrain :  Oid, ma :  MapON, pos :  MapON ,
                                curreq :  SetO .
```

We use maps to store the current movement authorities and positions of the various trains. In order to prevent an infinite amount of requests being made to an interlocking we limit the rbc to make one request on behalf of each train to the interlocking. To do this we use `curreq`, a set of oids, to store which trains have had a request made for them at a given moment of time.

An example RBC transition is as follows:

```
 rl [rbcgrant] :  {grant(N) < O : RBC | state :  wait, lasttrain :  T1, ma :
            MAP1 > REST} => { < O : RBC | state :  rbcidle, ma :
     insert(T1,endoftrack(N),MAP1) > grantma(T1, endoftrack(N)) REST} .
```

Modelling the Interlocking

An interlocking has two states, either it is idle or it has received a request for a movement authority and must respond. We define an Interlockings state in Real Time Maude as follows:

```
sort InterState .
ops idle res :  -> InterState [ctor] .
```

The interlocking class consists of a interlocking state, a request id which stores the current track segment requestions and 5 Booleans which indicate whether or not a track segment is occupied. We define an Interlocking class in Real Time Maude as follows:

```
class Inter | state :  InterState, reqid :  Nat, t0 :  Bool , t1 :  Bool , t2
                    :  Bool , t3 :  Bool , t4 :  Bool .
```

All of the rewrite rules on Interlocking objects are instantaneous and do not effect the transition of time. We can split these rules into two types. Firstly there are interlocking rules that deal with a request

Secondly there are several rewriting rules that deal with granting or denying a received request. There is currently a rule for both denying and granting a request for each track segment depending on the value of the boolean variable.

```
rl [resreq1g] :  < 0 : Inter | state :  res, t1 :  false, reqid :  1 > REST
             => < 0 : Inter | state :  idle > grant(1) REST .
```

6.5.1 Simulating the European Rail Traffic Management System Using Real Time Maude

We will now see how this executable specification can be used to simulate and obtain a better understanding of the modelled system. For this purpose we have written a small Haskell program that takes the output from multiple timed rewrites and plots them as a graph.

In the following we use an example initial state containing two trains one of which is slower (train1) than the other (train2), an interlocking and a radio block processor. The faster train will eventually catch up with the slow train and it waits for that train to clear all successive track segments before proceeding. The trains start at positions on opposing sides of the track with train1 starting at 0 and train2 starting at 150. The RBC and the interlocking are both initialised with the trains in these positions. In Maude this initial state is modelled as follows. Firstly, we define the initial states of the component objects individually:

```
eq initialintstate2 = < inter1 : Inter | state : idle, reqid : 0 ,
                                  t0 : true, t1 : false,  t2 : false,
                                  t3 : false, t4 : false > .
eq initialrbcstate2 = < rbc1 : RBC | state : rbcidle, lasttrain : train1,
                                   ma : (train1 |-> 49,train2 |-> 199),
                                   pos : (train1 |-> 0, train2 |-> 150) ,
                                   curreq : empty > .
rl [inittr1] : initialtrainstate1 => < train1 : Train | state : acc,  dist : 0,
                                                        speed : 0, ac : 1, ma : 49,
                                                        tseg : 0 , maxspeed : 4 > .
rl [inittr2] : initialtrainstate2 => < train2 : Train | state : acc, dist : 150,
                                                       speed : 0, ac : 1, ma : 199,
                                                       tseg : 3 , maxspeed : 7 > .
```

Secondly, we combine these individual components to form the complete system:

```
eq initialstate2 = {initialtrainstate1 initialtrainstate2
                        initialintstate2  initialrbcstate2} .
```

The model of ERTMS in the is executed using the following command:

```
(trew initialstate2 in time ≤ 50 .)
```

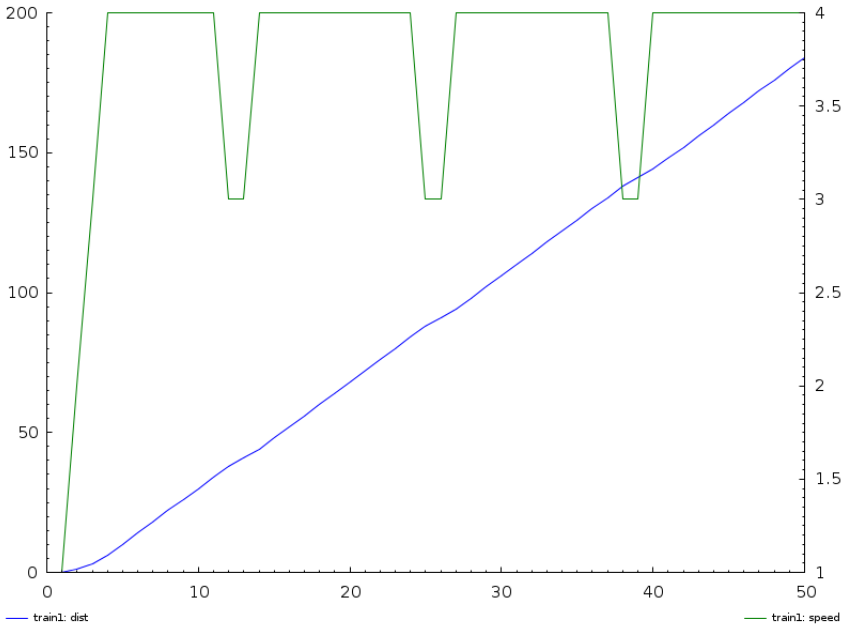


Figure 6.2: A graph comparing the distance and speed of train1

In the resulting output at time 50 both of the trains have moved a considerable distance and have requested and received new movement authorities.

```
Result ClockedSystem :
  {< inter1 : Inter | reqid : 3,state : idle,t0 : false,t1 : false,t2 : true,t3
    : true,t4 : false > < rbc1 : RBC | curreq : train2,lasttrain : train2,ma :(
    train1 |-> 199, train2 |-> 149),pos :(train1 |-> 180, train2 |-> 122),state
    : rbcidle > < train1 : Train | ac : 1,dist : 184,ma : 199,maxspeed : 4,
    speed : 4,state : cons,tseg : 3 > < train2 : Train | ac : 1,dist : 128,ma :
    149,maxspeed : 7,speed : 5,state : break,tseg : 2 >} in time 50
```

In fig **??** we see that train2 never enters the same track segment as train1.

6.6 The Maude Linear Temporal Logic Model Checker

The Real Time Maude system includes a model checker for linear temporal logic [EMS02]. This is a useful automatic tool which allows for the verification and analysis of real time systems. In addition to the positive verification results in which a safety property holds, the system produces, in the case of a negative verification resuIt, a counter example trace which shows the. In the following we shall present formal definitions for linear temporal logic and the model checking problem for formulae in this logic.

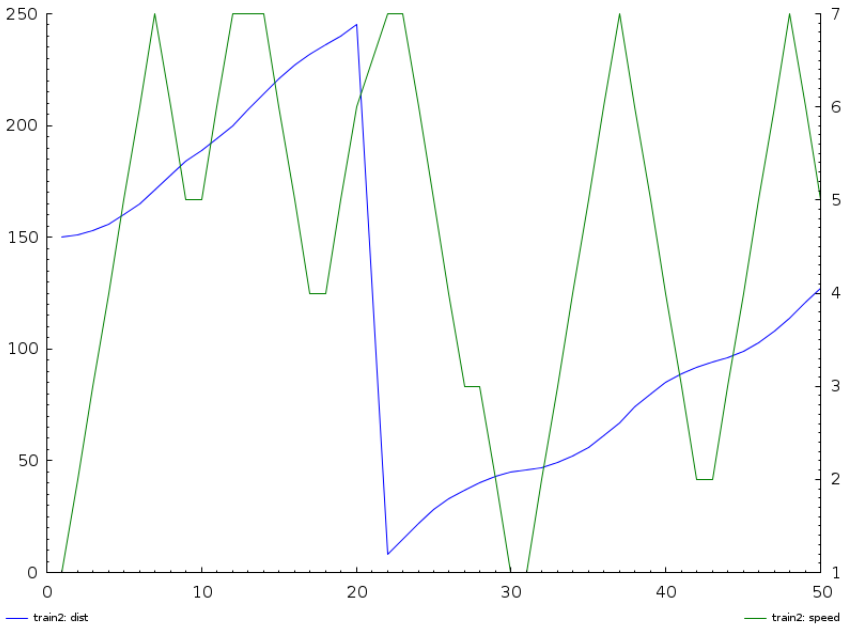


Figure 6.3: A graph comparing the distance and speed of train2

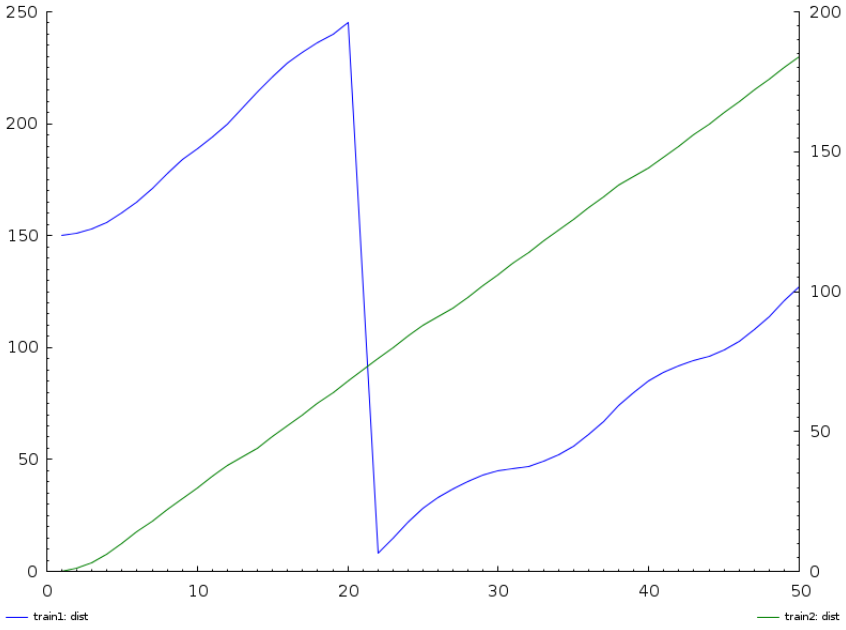


Figure 6.4: A graph comparing the distances of train1 and train2

6.6.1 Linear Temporal Logic

In order to perform model checking over a system we typically need a formal language that allows one to speak about time. We need to be able to formalise sentences such as "the next moment of time" and "all moments in time in the future". One such logic that allows us to formalise these statements is linear temporal logic (LTL)[Pnu77].

Definition 15 (Atomic Propositions)**.** *Given a set of symbols S we inductively define the set of atomic propositions AP as follows:*

- *If s is in the set of symbols $s \in S$ then $s \in AP$.*
- *if p_1 is an atomic proposition $p_1 \in AP$ then $\neg p_1 \in AP$.*
- *given two atomic propositions p_1 and p_2 then $p_1 \circ p_2 \in AP$ where $\circ$ is a propositional connective $\circ \in \{\wedge, \vee, \rightarrow\}$.*

Using these atomic propositions combined with the temporal operators it is possible to define a syntax for temporal logic formulae.

Definition 16 (Syntax of Linear Temporal Logic)**.** *Let AP be the set of atomic proposition names then:*

- *$\top$ and $\bot$ are well formed formulas.*
- *if $p \in AP$ then p is well formed formula (wff).*
- *if f and g are wff then $\star f$ and $f \circ g$ are wffs where $\star \in \{\neg, \mathbf{X}, \mathbf{G}, \mathbf{F}\}$ and $\circ \in \{\wedge, \vee, \boldsymbol{R}, \boldsymbol{U}\}$.*

We write $LTL(AP)$ to denote the LTL logic formulas for a given set of atomic propositions AP.

LTL operations can be used to speak about paths through a system specified as a Kripke structure. a *path* π is a sequence of states $s_1, \ldots s_n$ and a *path formula* is one that holds in each given state of a path. We define $Path(K, s_0)$ to be the set of paths starting at state s_0 in the Kripke structure K as the set of functions $\phi : N \rightarrow S$ such that $\phi(0) = a$ and the $\phi(n) \rightarrow \phi(n + 1)$.

We shall now look at the semantics of LTL firstly using an informal description of the LTL operations and secondly by giving a formal semantics for LTL. The following is a description of the 5 LTL operations over paths of a Kripke Structure.

- **X** f : The property f holds in the *next* moment of time.
- **G** f : The property f is *globally* true. i.e. it holds for all times on all paths.
- **F** f : The property f is *finally* true. i.e. there exists a time such that the property f holds on a path.
- f **U** g : For all paths the property globally f holds *until* property g holds.
- f **R** g : f holds up to and including the point when g holds.

The semantics of LTL is defined inductively in terms of the **X** and **U** LTL operations which can then be used in combination with operator equivalences to define the semantics of other operations.

Definition 17 (Semantics of Linear Temporal Logic)**.** *We define the semantics of a linear temporal logic formula $\phi \in LTL(AP)$ in terms of a satisfaction relation*

$$K, s \models \phi$$

over a Kripke structure $K = (S, T, L)$ and a state $s \in S$ as follows:

$$K, s \models \phi$$

holds if and only if $\forall \pi \in Path(K, s)$, $K, \pi \models \phi$ holds.

We define what it means for $K, \pi \models \phi$ to hold inductively as follows:

- *$K, \pi \models \phi$ always holds.*
- *For all atomic propositions $p \in AP$ the following always holds:*

$$K, \pi \models p \leftrightarrow p \in L(\pi(0))$$

- *For all LTL formulas $\mathbf{X}\phi \in LTL(K)$ the following always holds:*

$$K, \pi \models \mathbf{X}\phi \leftrightarrow K, succ; \pi \models \phi$$

  where succ is a successor function $succ : N_{-} \rightarrow N$ such that $succ; \pi(n) = \pi(succ(n)) = \pi(n + 1)$.

- *For all LTL formulas $\phi \mathbf{U} \psi \in LTL(K)$ the following always holds:*

$$K, \pi \models_{LTL} \phi \mathbf{U} \psi \leftrightarrow$$

$$\exists n \in N.$$

$$(K, succ^n; \pi \models_{LTL} \psi) \wedge (\forall m \in N. m < n \rightarrow K, succ^m; 1 \models_{LTL} \phi)$$

- *For all LTL formulas $\neg\phi \in LTL(AP)$ the following always holds:*

$$K, \pi \models_{LTL} \neg\phi \leftrightarrow K, \pi \not\models \phi$$

- *For all LTL formulas $\phi \vee \psi \in LTL(AP)$ the following always holds*

$$K, \pi \models_{LTL} \phi \vee \psi \leftrightarrow K, \pi \models_{LTL} \phi \text{ or } K, \pi \models_{LTL} \psi$$

The semantics for formulas containing other LTL operations can be obtained by using the following equivalences:

$$\mathbf{G}\phi \equiv False\mathbf{R}\phi$$

$$\mathbf{F}\phi \equiv True\mathbf{U}\phi$$

Combined with the equivalences for obtaining negative normal form.

Definition 18 (LTL Negative Normal Form)**.** *The negative normal form of an LTL formula can be obtained by applying the following equivalences:*

$$\neg True \equiv False$$

$$\neg False \equiv True$$

$$\neg\neg\phi \equiv \phi$$

$$\neg(\phi \vee \psi) \equiv \neg\phi \wedge \neg\psi$$

$$\neg(\phi \wedge \psi) \equiv \neg\phi \vee \neg\psi$$

$$\neg\mathbf{X}\phi \equiv \mathbf{X}\neg\phi$$

$$\neg(\phi\mathbf{U}\psi) \equiv (\neg\phi)\mathbf{R}(\neg\psi)$$

$$\neg(\phi\mathbf{R}\psi \equiv (\neg\phi)\mathbf{U}(\neg\psi)$$

from left to right until no further equivalence is applicable.

6.7 Model Checking the European Rail Traffic Management System

In the following section we will demonstrate an approach to apply the Real Time Maude LTL model checker to verify the Real Time Maude specification described previously. Firstly we shall define the property we want to check in the Real Time Maude system. To do this we have to define a satisfaction relation which describes what it means for the property to hold in a state of the system we are checking. The property we shall consider in this case is that the moment authorities of two trains in system do not overlap. Logical property we define identifies the individual trains using their object identifiers and then computes whether or not an over lap has occured using a boolean formula. We shall show that it is possible to verify this property over a time period large enough for all normal behaviours of the system to occur.

Defining a Satisfaction Relation

We shall now describe how to define the behaviour of the satisfaction relation for a given system and property as a Real Time Maude specification. The satisfaction relation itself is predefined in a specification as an operation which takes a State and a property which is of type Prop and computes a boolean. Maude defines the satisfaction relation $\models$ in the following module:

```
fmod SATISFACTION is
  protecting BOOL .
  sorts State Prop .
  op _|=_ : State Prop -> Bool [frozen] .
endfm
```

The sorts `State` and `Prop` are undefined as is the behaviour of $\models$. It is left to the user of the model checker to define the behaviour for their own purposes. The standard way to define predicates which refer to a given object is by matching object identifiers.

We can check that a train is in a specific state as follows: `ops train-cons : Oid -> Prop [ctor] .`
`eq REST < O1 : Train | state : cons > |= train-cons(O1') = (O1 == O1') .`

This predicate holds if there is a train in t he configuration with object identifier O1' and a

state cons.

We can check that a train has a certain speed as follows:

`op train-s : Oid Nat -> Prop [ctor] .`
`eq REST < O1 : Train | speed : N1 > |= train-s(O1',N1') = (N1 == N1') and (O1`

`== O1') .` Here the natural number N1 in the train object is matched with N1' in the predicate train-s. The main movement authority that we want to check is that the movement authorities of two trains does not over lap. To do this we define a Property nomaoverlap(O1,O2) of type `Oid Oid -> Prop [ctor]` as follows:

```
eq REST < O1 : Train | dist : D1 , ma : M1 > < O2 : Train | dist : D2 ,
    ma : M2 > |= nomaoverlap(O1',O2') = (O1 == O1') and (O2 == O2') and
                          noolap(D1,M1,D2,M2) .
```

Here we have matched two object identifiers and natural numbers and callled a boolean function on those variables. This definition features an external operation which computes a boolean depending on whether a set of inequalities are satisfied.

```
eq noolap(D1,M1,D2,M2) = ((D1 < M1) and (M1 < D2) and (D2 < M2)) or ((D2 <
M2) and (M2 < D1) and (D1 < M1)) or ((D1 < M1) and (M2 < D1) and (M1 < D2))
              or ((D2 < M2) and (M1 < D2) and (M2 < D1) ) .
```

6.7.1 Execution of The LTL Model Checker

We shall now describe the execution of the LTL model checker in Real Time Maude. This is done using the mc command with some initial state of type system, a satsifaction relation, the LTL formula to be checked and an optional time. The satisfaction relation can be either $\models_u$ for untimed model checking for which time is not specifed or $\models_t$ for which the optional time is specified.

The Real Time Maude LTL model checker is then called using the following command:

```
(mc initialstate2 |=t [] nomaoverlap(train1,train2) in time <= 30 .)
```

The command states that we are applying timed model checking to check that it is globally true that the movement authorities of the trains do not over lap at all moments in time less than or equal to 30. This results in the following output from Real Time Maude which indicates that the property was succesfully verified in 3 minutes and 22 seconds.

```
rewrites: 138889387 in 179663ms cpu (202024ms real) (773054 rewrites/second)

Model check initialstate2 |=t[]nomaoverlap(train1,train2)in
MODEL-CHECK-ERTMS8 in time <= 30 with mode deterministic time increase

Result Bool :
  true
```

Unfortunately the Real Time Maude LTL model checker can not detect the congruence between difference configurations of control system. Therefore the state space is infinite and it is impossible to do untimed model checking on the above model checking problem. It is however possible to do timed model checking within a reasonable time limit that allows for all behaviours of the system to be verified.

Bibliography

[CDE+03] M. Clavel, F. Durn, S. Eker, P. Lincoln, N. Mart-Oliet, J. Meseguer, and C. Talcott. The Maude 2.0 system. In R. Nieuwenhuis, editor, *International Conference on Rewriting Techniques and Applications (RTA)*, volume 2706 of *Lecture Notes in Computer Science*, pages 76–87. Springer Berlin Heidelberg, 2003.

[EMS02] S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL model checker. In F. Gadducci and U. Montanari, editors, *Fourth Workshop on Rewriting Logic and its Applications, WRLA '02*, volume 71 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.

[FH98] W. Fokkink and P. Hollingshead. Verification of interlockings: from control tables to ladder logic diagrams. In J. F. Groote, S. P. Luttik, and J. J. van Wamel, editors, *Proceedings of the 3rd Workshop on Formal Methods for Industrial Critical Systems - FMICS'98*, pages 171–185, 1998.

[Har02] N.A. Harman. Verifying a simple pipelined microprocessor using maude. In Maura Cerioli and Gianna Reggio, editors, *Recent Trends in Algebraic Development Techniques*, volume 2267 of *Lecture Notes in Computer Science*, pages 128–151. Springer Berlin Heidelberg, 2002.

[Hen96] T. A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, LICS '96, pages 278–292. IEEE Computer Society, 1996.

[IMN13] D. Ishii, G. Melquiond, and S. Nakajima. Inductive verification of hybrid automata with strongest postcondition calculus. In *Proceedings of the 10th International Conference on Integrated Formal Methods*, Lecture Notes in Computer Science, pages 139–153. Springer Berlin Heidelberg, 2013.

[KR01] D. Kerr and T. Rowbotham. Introduction to Railway Signalling. *Institution of Railway Signal Engineers*, 2001.

[Mau] Maude. `http://maude.cs.uiuc.edu/`. Accessed: 2014-01-22.

[ÖM02] P. C. Ölveczky and J. Meseguer. Specification of real-time and hybrid systems in rewriting logic. *Theoretical Computer Science*, 285:359–405, 2002.

[ÖM04] P. C. Ölveczky and J. Meseguer. Specification and analysis of real-time systems using Real-Time Maude. In T. Margaria and M. Wermelinger, editors, *Funda-*

mental Approaches to Software Engineering (FASE 2004), volume 2984 of *Lecture Notes in Computer Science*, pages 354–358. Springer, 2004.

[Pnu77] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, 1977.

[RTM] Real Time Maude. `http://heim.ifi.uio.no/peterol/RealTimeMaude/`. Accessed: 2014-01-22.