

Verification of Discrete and Real-timed Railway Control Systems

Monika Seisenberger

Joint work with Andrew Lawrence, Ulrich Berger, Phil James, Markus Roggenbach

Swansea University

CIRM, 11 January 2016

Verification of Discrete and Real-timed Railway Control Systems

2 Aims:

- 1 Discrete: Verification of Solid State Interlockings -
 - From Ladder Logic to a SAT solving problem.
 - Extraction of a verified SAT Solver in the Minlog System.
- 2 Real-Timed: Modelling the European Rail Traffic Management System (ERTMS)
 - ERTMS – what it is and how it works
 - Generic Modelling: ERTMS in Real-Time Maude
 - Verification & simulation results

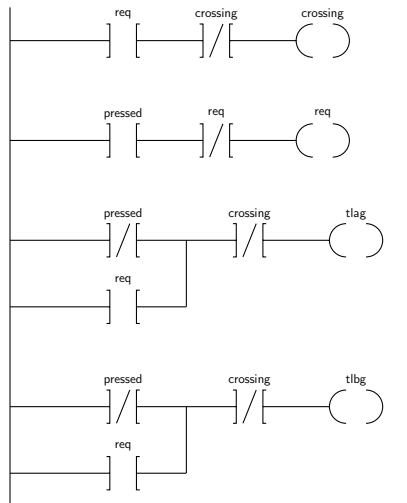
The use of Ladder Logic: a simple Crossing Example

Traditionally railway engineers use ladder logic to specify their systems.

Here a small crossing example:

- 1 input variable: pressed
- 2 internal state variables: "crossing" and "required"
- variables for the traffic lights: tlag, tlar, etc

[Example created by Karim Kanso]



Ladder Logic Program (4/8 rungs)

Translation (automated) to Propositional Logic

$$\begin{aligned} crossing' &\leftrightarrow req \wedge \neg crossing, \\ req' &\leftrightarrow pressed \wedge \neg req, \\ tlag' &\leftrightarrow (\neg pressed \vee req') \wedge \neg crossing' \\ tlbq' &\leftrightarrow (\neg pressed \vee req') \wedge \neg crossing' \\ tlar' &\leftrightarrow crossing', \quad tibr' \leftrightarrow crossing', \\ plag' &\leftrightarrow crossing', \quad plbq' \leftrightarrow crossing', \\ plar' &\leftrightarrow \neg crossing', \quad plbr' \leftrightarrow \neg crossing' \end{aligned}$$

- $crossing'$, req' , ... are new variables
- primed variables on left sides are all different.
- a primed variable may depend on earlier computed primed variables, but not on the unprimed ones.

Definition Ladder Logic Formulae

I input variables, C output variables.

Example: $I = \{pressed\}$ and

$C = \{crossing, req, tlag, tlbq, plaq, plbq, \dots\}$.

$C' = \{c' \mid c \in C\}$ to be a set of new variables (intended to denote the output variables computed in the current cycle).

$\text{unprime} : C' \rightarrow C, \text{unprime}(c') = c$.

A *ladder logic formula* ψ is a propositional formula of the form

$$\psi \equiv ((c'_1 \leftrightarrow \psi_1) \wedge (c'_2 \leftrightarrow \psi_2) \wedge \dots \wedge (c'_n \leftrightarrow \psi_n))$$

such that the following holds for all $i, j \in \{1, \dots, n\}$:

- $c'_i \in C'$
- $i \neq j \rightarrow c'_i \neq c'_j$
- $\text{Vars}(\psi_i) \subseteq I \cup \{c'_1, \dots, c'_{i-1}\} \cup \{c_i, \dots, c_n\}$

Semantics Ladder Logic Formulae

The semantics of a ladder logic formula ψ is a function that takes the current valuations for input and output variables and returns a new valuation for output variables (one time cycle later).

$$[\psi] : \text{Val}_I \times \text{Val}_C \rightarrow \text{Val}_C$$
$$[\psi](\mu_I, \mu_C) = \mu'_C$$

where

$$\text{Val}_I = \{\mu_I \mid \mu_I : I \rightarrow \{0, 1\}\} = \{0, 1\}^I$$

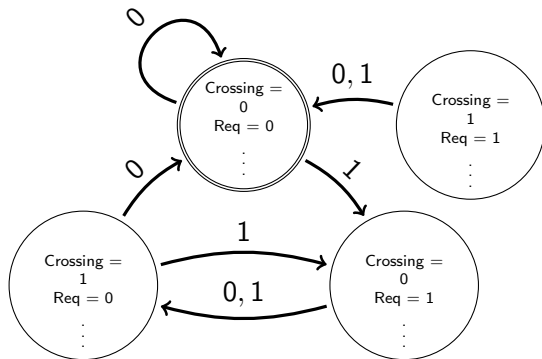
$$\text{Val}_C = \{\mu_C \mid \mu_C : C \rightarrow \{0, 1\}\} = \{0, 1\}^C$$

$$\mu'_C(c_i) = [\psi_i](\mu_I, (\mu_C)_{\upharpoonright\{c_i, \dots, c_n\}}, (\mu'_C \circ \text{unprime})_{\upharpoonright\{c'_1, \dots, c'_{i-1}\}})$$

$$\mu'_C(c) = \mu_C(c) \text{ if } c \notin \{c_1, \dots, c_n\}$$

and $[\psi_i](\cdot, \cdot, \cdot)$ denotes the usual value of a propositional formula under a valuation.

Crossing transition system



- Included one unreachable state were both Crossing and Req are true.

Ladder Logic Labelled Transition System

We define the labelled transition system $LTS(\psi)$ for a ladder logic formula ψ to be the four tuple $(Val_C, Val_I, \rightarrow, Val_0)$

where

- $\mu_C \xrightarrow{\mu_I} \mu'_C$ iff $[\psi](\mu_I, \mu_C) = \mu'_C$
- $Val_0 = \{\mu_C \mid \mu_C \text{ initial valuation}\}$

A state s is called *reachable* if $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \dots \xrightarrow{t_{n-1}} s_n$, for some states s_0, \dots, s_n , and labels t_0, \dots, t_{n-1} such that $s_0 \in Val_0$ and $s_n = s$.

Definition (Safety Conditions)

Given a ladder logic formula ψ over the variables in $I \cup C$ a **verification condition** is a propositional formula formed from the variables in $I \cup C \cup C'$.

Examples of Safety conditions: In crossing not all light not green at the same time, no two trains on the same track segment on the same time,...

Definition (The Verification Problem)

We define the verification problem for a ladder logic formula ψ for a verification condition ϕ

$$\text{LTS}(\psi) \models \phi$$

iff for all triples μ_C, μ_I, μ'_C such that $\mu_C \xrightarrow{\mu_I} \mu'_C$ and μ_C is reachable in $\text{LTS}(\psi)$, we have $[\phi](\mu_C, \mu_I, \mu'_C) = 1$.

Real World Case Studies and Technologies applied

Invensys Rail UK (now Siemens UK) provided Ladder Logic Programs for several stations. Size: 600 variables, 350 rungs, for a small London Underground Station. largest: 8166 variables, 14726 clauses.

- 1 SAT solving using an industrial Tool: SCADE (Prover).
 - Our tool automatic translates to SCADE language.
 - Several optimization methods, no control on methods.
 - All 109 safety conditions together take less than 1s.
 - 55 produced counter examples, which need to be eliminated by adding invariants → 100 invariants added.
- 2 Extracting a SAT solver in the Minlog system.
 - Extracted SAT solver can easily be integrated in the Minlog system, i.e. will allow for a combination of SAT solving and interactive theorem proving.
 - Provides in each case either a model or a derivation why not satisfiable.
 - Can deal with all the above safety conditions (8s/12s)
 - Variant: Extension to backtracking and clause learning.

Extraction of a SAT solving algorithm

Basic definitions:

- A *literal* l is either a positive variable $+v$ or a negative variable $-v$. The *opposite* value of a literal is defined as: $\overline{+v} = -v$, $\overline{-v} = +v$.
- A *clause* C is defined as a set of literals $\{l_1, \dots, l_k\}$ (representing their disjunction).
- A *formula* Δ is a set of clauses (representing their conjunction).

An example of a formula:

$$\Delta = \{\{l_{11}\}, \{l_{21}\}, \{\overline{l_{11}}, \overline{l_{21}}\}\}$$

to be read as

$$l_{11} \wedge l_{21} \wedge (\neg l_{11} \vee \neg l_{21})$$

SAT problem: is there a valuation for these variables satisfying the formula?

DPLL Proof System $\Gamma \vdash \Delta$

Most modern SAT solvers are based on DPLL algorithm (Davies, Putnam, Logemann, Loveland 1960/1962).

We use the DPLL proof system, consisting of 5 rules:

$$\frac{\Gamma, I \vdash \Delta}{\Gamma \vdash \Delta, \{I\}} \text{ (Unit)} \quad \frac{\Gamma, I \vdash \Delta, C}{\Gamma, I \vdash \Delta, (\bar{I}, C)} \text{ (Red)}$$

$$\frac{\Gamma, I \vdash \Delta}{\Gamma, I \vdash \Delta, (I, C)} \text{ (Elim)}$$

$$\frac{}{\Gamma \vdash \Delta, \emptyset} \text{ (Conflict)} \quad \frac{\Gamma, I \vdash \Delta \quad \Gamma, \bar{I} \vdash \Delta}{\Gamma \vdash \Delta} \text{ (Split)}$$

(Γ is a valuation (set of literals) and Δ is a formula (clause set). $\Gamma \vdash \Delta$ essentially means that Δ is not satisfiable, using the literals from Γ)

Valuations and Models

- A valuation Γ , i.e. set of literals $\{l_1, \dots, l_k\}$, is *consistent* iff $l \in \Gamma \rightarrow \bar{l} \notin \Gamma$. Let Cons be the set of all consistent Valuations.
- A *model* is a total function M which maps literals to booleans and satisfies the following property $\forall l M. l \leftrightarrow \neg(M \bar{l})$

Two abbreviations:

- For a given valuation Γ , $\forall l \in \Gamma M l$ is abbreviated as $M \models \Gamma$.
- For a given formula Δ , $\forall C \in \Delta \exists l \in C M l$ is abbreviated as $M \models \Delta$.

We call a valuation Γ and a formula Δ *compatible* if there exists a model satisfying both, i.e.

$$\exists M. M \models \Gamma \wedge M \models \Delta$$

Formalising and Proving Completeness

The expected statement of completeness is: $\forall \Gamma \in \text{Cons}, \forall \Delta.$

$$\text{incompatible}(\Gamma; \Delta) \rightarrow \Gamma \vdash \Delta$$

We proved the following classically equivalent but constructively stronger statement: $\forall \Gamma \in \text{Cons}, \forall \Delta.$

$$\text{compatible}(\Gamma; \Delta) \vee \Gamma \vdash \Delta$$

Program extraction yields a program that either yields a **model** if Γ and Δ are compatible ($\exists M. M \models \Gamma \wedge M \models \Delta$) or a **derivation** if incompatible.

Proof of Completeness Theorem

Theorem: $\forall \Gamma \in \text{Cons}, \forall \Delta, \Theta. \emptyset \notin \Theta \wedge \text{Var}(\Gamma) \cap \text{Var}(\Theta) = \emptyset \rightarrow$
 $(\Gamma \vdash \Delta \cup \Theta) \vee \exists M. M \models \Gamma \wedge M \models \Delta \cup \Theta,$

We aim to perform the proof in such a way that an efficient program is extracted:

1. Since performing a split is the only computational expensive operation, we only apply it when it is absolutely necessary.
2. We perform an optimisation on the proof level by partitioning the clauses into 'clean' and 'unclean' clauses, where a clause is called clean if we cannot apply Elim, Reduce or Unit to that clause.

Program Extraction - Extracted Solver

The proof has been formalised in the Interactive Proof System Minlog, and - via modified realisability - a program has been extracted.

Example run: We run the extracted solver using *pigeon hole formulae*

$$\begin{aligned}
 PHP(n, m) := & \{ \{ l_{i,1}, \dots, l_{i,m} \} \mid 1 \leq i \leq n \} \\
 & \cup \{ \{ l_{i,k}^-, l_{j,k}^- \} \mid 1 \leq i < j \leq n, 1 \leq k \leq m \}
 \end{aligned}$$

Intuitively, e.g. $PHP(n, n - 1)$ states "it is not possible to put n pigeons into $n - 1$ holes and only have one pigeon in each hole"

Extracted Program (cont.)

On satisfiable formulae:

$PHP(2,2)$	$PHP(3,3)$	$PHP(4,4)$	$PHP(5,5)$	$PHP(6,6)$
< 1 Sec	< 1 Sec	5.45	26.09	1:34.11

On unsatisfiable formulae:

$PHP(2,1)$	$PHP(3,2)$	$PHP(4,3)$	$PHP(5,4)$	$PHP(6,5)$
< 1 Sec	1.17	33.62	13:54	5:35:41

1. Improvement: Non-computational Quantifiers

Comparison on Unsatisfiable Formula

Solver	$PHP(2, 1)$	$PHP(3, 2)$	$PHP(4, 3)$
\forall	< 1 Sec	1.17	33.62
\forall_{nc}	< 1 Sec	< 1 Sec	11.61

$PHP(5, 4)$	$PHP(6, 5)$
13:54	5:35:41
2:41	37:25

The \forall_{nc} solver is significantly faster on unsatisfiable formulae!

Realisability with Non-Computational Quantifiers

As well as the usual quantifiers, \forall and \exists , Minlog offers *non-computational* (nc) quantifiers \forall_{nc} and \exists_{nc} .

The definitions of the type for the ordinary quantifiers:

$$\tau(\forall x^\rho A) = \rho \rightarrow \tau(A)$$

$$\tau(\exists x^\rho A) = \rho \times \tau(A)$$

The definitions of the type for the **nc** quantifiers:

$$\tau(\forall^{nc} x^\rho A) = \tau(A)$$

$$\tau(\exists^{nc} x^\rho A) = \tau(A)$$

For the nc-quantifiers the realizers do not depend on the quantified variables:

2. Further improvements: Extraction to Haskell, etc

Formula	Minlog \forall	Minlog \forall_{nc}	Haskell		Haskell (-f11vm)	
	Witness	Witness	Witness	Yes/No	Witness	Yes/No
PHP(4,3)	33.62s	11.61s	0.019s	0.006s	0.015s	0.004s
PHP(4,4)	5.45s	5.25s	0.019s	0.010s	0.014s	0.007s
PHP(5,4)	13m54s	2m41s	0.055s	0.020s	0.036s	0.012s
PHP(5,5)	26.09s	25.03s	0.024s	0.015s	0.020s	0.010s
PHP(6,5)	5h35m41s	37m25s	0.367s	0.066s	0.279s	0.039s
PHP(6,6)	1m34.11s	1m24.88s	0.035s	0.025	0.025s	0.015s
PHP(8,8)	-	-	0.054s	0.029s	0.040s	0.025s
PHP(9,8)	-	-	-	1m21.915s	-	32.062s
PHP(9,9)	-	-	0.064s	0.042s	0.052s	0.030s
PHP(10,9)	-	-	-	102m 16s	-	15m 5s

[Extraction to Haskell done in collaboration with Fredrik Nordvall Forsberg]

Performance compared to Versat

Versat was formalized and verified in the dependently typed programming language Guru and translated into C-code.

Formula	\forall_{nc} compiled (Yes/No)	Versat
PHP(7,6)	0.226s	0.089s
PHP(8,7)	2.42s	0.794s
PHP(9,8)	32.062s	17.217s
PHP(10,9)	15m 5s	15m 46s

Part 2: Real-Timed Railway Control Systems

To investigate how a Centralized Traffic Control System, the European Rail Traffic Management System (ERTMS) can be modelled and verified using the Real-Time-Maude system

Overview Part 2:

- I: ERTMS – what it is and how it works
- II: Modelling of ERTMS in Real-Time Maude
- III: Validation and Verification results

European Rail Traffic Management System (ERTMS) I

What it is:

- European standard of signalling, control and train protection
- To replace the many incompatible safety systems (20!) currently used by European railways
- Offers possibility for traffic management
- Originally designed for Europe, has rapidly become a global standard.

Some facts:

- Europe: Switzerland (1200km, full coverage by 2017), Denmark (4000km), Germany, Belgium, Spain, Austria;
UK's first line is in Wales: Cambrian Coast Line, 215km
- World wide: China: 8000km.

European Rail Traffic Management System (ERTMS) II

Traditional railway interlockings control the rail traffic via signals.
In short: ERTMS removes the signals, and replaces them by communication between trains and interlockings.

ERTMS shall achieve:

- interoperability
- ease of maintenance (less track equipment)
- higher capacity

European Rail Traffic Management System (ERTMS) II

Traditional railway interlockings control the rail traffic via signals.
In short: ERTMS removes the signals, and replaces them by communication between trains and interlockings.

ERTMS shall achieve:

- interoperability
- ease of maintenance (less track equipment)
- higher capacity (40%)

European Rail Traffic Management System (ERTMS) II

Traditional railway interlockings control the rail traffic via signals.
In short: ERTMS removes the signals, and replaces them by communication between trains and interlockings.

ERTMS shall achieve:

- interoperability
- ease of maintenance (less track equipment)
- higher capacity (40%)

Open research questions include:

- 1 How can safety be verified?
- 2 How can capacity be measured and improved?
- 3 How can reliability be measured and estimated?

European Rail Traffic Management System (ERTMS) II

Traditional railway interlockings control the rail traffic via signals.
In short: ERTMS removes the signals, and replaces them by communication between trains and interlockings.

ERTMS shall achieve:

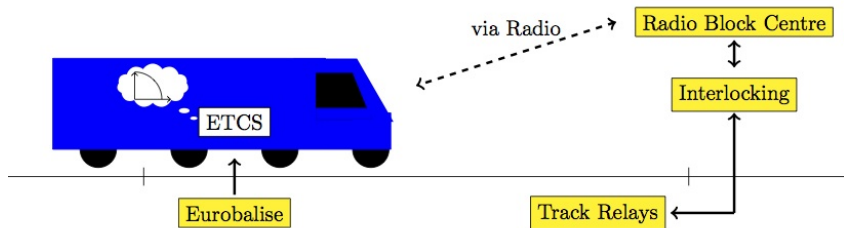
- interoperability
- ease of maintenance (less track equipment)
- higher capacity (40%)

Open research questions include:

- ① How can safety be verified?
- ② How can capacity be measured and improved?
- ③ How can reliability be measured and estimated?

Here: 1 and, partially, 2.

System components of ERTMS, level 2



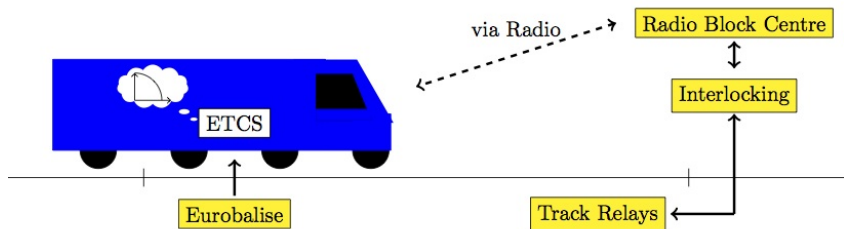
Main Responsibilities:

Trains - communicate position/speed, and receive movement authorities.

RBC - grants MAs/denies MA requests, consults with Interlocking

Interlocking - allows for setting new routes, responsible for safety.

System components of ERTMS, level 2



Main Responsibilities:

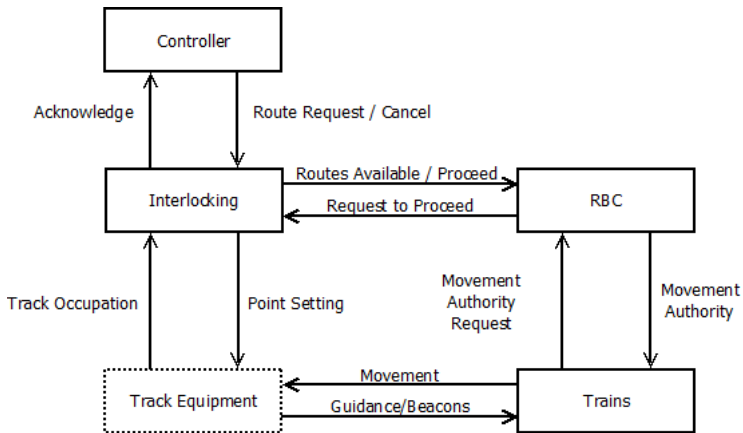
Trains - communicate position/speed, and receive movement authorities.

RBC - grants MAs/denies MA requests, consults with Interlocking

Interlocking - allows for setting new routes, responsible for safety.

Controller (not in picture) - requests new routes.

Information flow in ERTMS, level 2



Object Oriented Modelling in Real-Time-Maude

- Real-Time Maude (Peter C. Ölveczky and José Meseguer 2004) is a language and tool extending Maude, that allows for simulation and formal analysis of real-time and hybrid systems.
- Object based systems are modelled as multisets of objects and messages of a sort `Configuration`, a subset of Maude's built-in in sort `System`.

Object Oriented Modelling in Real-Time-Maude

- Real-Time Maude (Peter C. Ölveczky and José Meseguer 2004) is a language and tool extending Maude, that allows for simulation and formal analysis of real-time and hybrid systems.
- Object based systems are modelled as multisets of objects and messages of a sort `Configuration`, a subset of Maude's built-in in sort `System`.
- A real-time specification consists of
 - the sort `Time` (in our case `NNegRat`),
 - the constructor `{_} : System -> Globalsystem`
 - instantaneous rewrite rules,
 - a so-called tick rule that defines how time elapses.

Object Oriented Modelling in Real-Time-Maude

- Real-Time Maude (Peter C. Ölveczky and José Meseguer 2004) is a language and tool extending Maude, that allows for simulation and formal analysis of real-time and hybrid systems.
- Object based systems are modelled as multisets of objects and messages of a sort `Configuration`, a subset of Maude's built-in in sort `System`.
- A real-time specification consists of
 - the sort `Time` (in our case `NNegRat`),
 - the constructor `{_}` : `System -> Globalsystem`
 - instantaneous rewrite rules,
 - a so-called tick rule that defines how time elapses.

```
cr1 [tick] : {CURRENT} => {delta(CURRENT,T)} in time T
           if T <= mte(CURRENT) [nonexec] .
```

where `delta` defines the effect of time elapse on a configuration.

`mte` defines the maximal possible time elapse.

Modelling 1: location specific data & messages

Encoding of the rail topology:

```
sort RouteName . ops RouteName1A ... : -> RouteName .
sort Track .     ops AA AB AC ... : -> Track .
sort Point .     ops P1 P2 : -> Point .
```

Messages to be exchanged between the ERMTS components:

```
msg routerequest : RouteName -> Msg .
msg marequest : Oid Track -> Msg .
```

Modelling 2: Instantaneously reacting sub-systems

No time-constraints:

```
eq mte(< 01 : Controller | >) = INF .
```

Interlocking – a class with internal states:

```
class Inter | routeset : MapRouteName2Bool,
              pointslocked : MapPoint2Bool,
              occ : MapTrack2Bool,
              pointPositions : MapPoint2PointPos .
```

Ignoring a route request:

```
cr1 routerequest(RN1)
  < 0 : Inter | occ : MAPTB1, pointslocked : MAPPB3 >
=>
  < 0 : Inter | > if (not checkClear(RN1, MAPTB1)) or
                    pointsLocked(RN1, MAPPB3) .
```

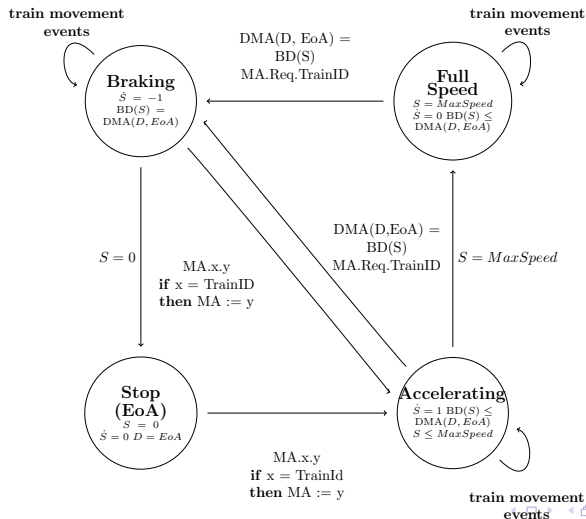
Modelling 3: Trains with ERTMS equipment

```

crl [acc] :
  ...
  delta(< 0 : Train | state : acc, dist : DT, speed : S,
        ac : A, ma : MA , tseg : AN, maxspeed : MAX >, T)
=>
  ...
  < 0 : Train | state : if (S + T * A == MAX)
                        then cons
                        else (if T == mteMA(DT,S,A,MA)
                              then brake
                              else acc fi) fi,
        dist : DT + S * T + A * T * T * 1/2,
        speed : S + A * T > ) .

```

Modelling 3: Trains with ERTMS equipment



III) Validation through Simulation

We have validated our model through exploring various train movements.

For example, rewriting a train starting on track AA:

```
(trew { < inter1 : Inter | pointPositions : (P1 |-> normal,
P2 |-> normal) , ... > < train1 : Train | state : acc, dist :
2, speed : 0, ac : 1, ma : 1498, tseg : AA , maxspeed : 60 >
} in time <= 39 .)
```

shows that it accelerates until it is required to begin braking due to its MA:

```
... < train1 : Train | ac : 1, dist : 1499446241/2000000, ma
: 1498, maxspeed : 60, speed : 38671/1000, state : brake,
tseg : AA >... in time 38671/1000
```

Validation through Simulation (2)

It then makes a movement authority request:

```
marequest(train1,AA) < inter1 : ...> < train1 : Train | speed  
: 37671/1000, ... > in time 39671/1000
```

However at this point the system will not progress until we add an RBC to deal with the request...

Error Injection: Train - Incorrect braking parameters

Our modelling is able to find errors, for example:

Deceleration for used for computation: 1; physical deceleration: 8/10.

```
< train1 : Train | ... dist : 3249, ac : 1, ma : 6499, tseg
: AD , maxspeed : 20 > < train2 : Train | ... ac : 8/10, ma
: 1, tseg : Entry , maxspeed : 60 > ...
```

Model checking is able to produce a counter example

```
< train1 : Train | ac : 1,dist : 15662341/2500,ma : 6499,
maxspeed : 20,speed : 20,state : cons,tseg : AF > < train2 :
Train | ac : 4/5,dist : 968593576867/156250000, ma :
7999,maxspeed : 60,speed : 60,state : cons, tseg: AF > ...
```


Error Injection: RBC Design Error

Our modelling is able to find errors, for example:

Assume the RBC is designed with incorrect EoA values,
e.g. EoA of Route 1A = 3449m

Model checking is able to produce a counter example where train 1
overruns and hence is able to get within 100m of train 2:

```
...< train1 : Train | ac : 1,dist : 3449,ma : 3449,  
maxspeed : 20,speed : 0,state : stop,tseg : AD >  
< train2 : Train | ac : 1,dist : 12433788921/4000000,  
ma : 6499,maxspeed : 60, speed : 60,state : cons, tseg  
: AC > ...
```

Safety Verification through Model-checking

Verification that trains cannot be within 100 metres of each other, e.g.:

```
mc initState |=t [] nocrashDistance(train1,train2) .
```

Scheme Plan	Round Robin Controller Unbounded	Random Controller in Time 300
Junction	2.4s / 5,767,435 rewrites	268.3s / 208,715,358 rewrites
Pass-through Station	3.0s / 7,135,987 rewrites	439.2s / 308,629,500 rewrites
Three Platform Station	2.8s / 6,624,578 rewrites	2697.1s / 729,201,878 rewrites

Table: Verification results of model checking three scheme plans.

Conclusion and Future Work

Part1: Verification of Traditional Interlockings: all translation processes can be automated; method included in industrial process between design and testing. Industry will still do testing (involves humans), but the burden of guaranteeing is completeness and correctness greatly reduced.

Second, we presented a conceptually new approach to the synthesis and verification of SAT algorithms.

- does not require the formalisation of the algorithm, but obtains it by program extraction.
- interesting point: do optimisations not on the programme level, but on the proof level.
- Future work: Extension to include backtracking and clauselearning.

Conclusion: Part 2: Real-Timed Railway Control Systems

The firsts:

- First use of Maude / Real-Time Maude in the railway domain.
- First formal model comprising of all ERTMS subsystems required for the control cycle.
- Rail control modelled as a hybrid system,
- Safety properties in physical rather than in logical terms.


Future work: Improving the models:


- Bi-directional rail-yards.
- Further controller strategies.
- More complex train progression behaviour.

Reflecting on the models:

- Address completeness/Use of Real Numbers
- Include further safety properties.
- Develop abstractions to increase in verification speed.

References

-  James, P., Lawrence, A., Moller, F., Roggenbach, M., Seisenberger, M. and Setzer, A. Chadwick, S. ,P. Kanso, K., :
Verification of solid state interlocking programs.
In SEFM'13, LNCS 8368, Springer 2014.

-  Berger, U., Lawrence, A., Nordvall Forsberg, F. , Seisenberger, M.
Extraction of Verified Decision Procedures.
LMCS 11(1:6), 2015.

-  James , P., Lawrence, A., Roggenbach, M., Seisenberger, M.
Towards Safety Analysis of ERTMS/ETCS Level 2 in Real-Time Maude.
FTSCS 2015, to appear, Springer, 2016.