A Journey in Technology Transfer & Developing a Verification Tool for Railway Interlockings

Markus Roggenbach
Swansea Railway Verification Group, UK

IFIP WG 1.3 – 2/24

UK's Research Excellence Framework (REF)

In the UK, every ~ 7 years, reserach excellence is evaluated.
Computer Science is one of the units of assessment.
$\rightsquigarrow$ determines amount of reasearch funding
Next round: 2029.

Part of this is on *'impact'*, defined as

> *"an effect on, change or benefit to the economy, society, culture, public policy or services, health, the environment or quality of life, beyond academia."*

My Impact Case Studies (mostly "benefit to the economy"):

- 2014: Improving processes and policies in the UK railway industry (**)
- 2021: Improving performance, safety and software development of railway signalling (***)

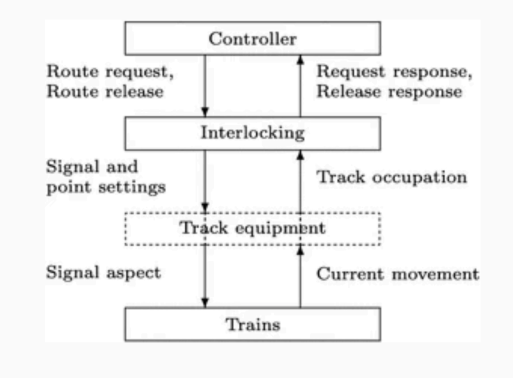
The Object Under Discussion: Interlocking Computer

Each installation is unique:

- unique software
- unique hardware configuration
  (built from standard hardware components)

Realised as a Progammable Logic Controller (PLC)

What Are Interlocking Used For?

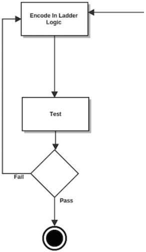


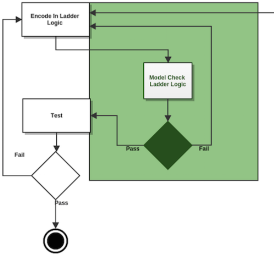
Interlocking = safety layer between controller and trains & track

Example of a safety property: "the operator is not allowed to let two different trains use the same route"

Hypothesis: Formal Method Could Be Of Help

SW Development "gets started by a hypothesis that a particular operational mission (or set of missions) could be improved by a software effort." B Boehm, 1988.





There appears to be room for improvement:

- There are many develop-test cycles: 4–6
- Testing takes long – in the order of several weeks
- Formal Methods might be more 'thorough' than testing

Begin of the Journey:

What Does Theory Say On PLC Verification?

Some Examples of PLC Applications

Nuclear Power Plant

Wasching Machine

Car

Water Park Slides

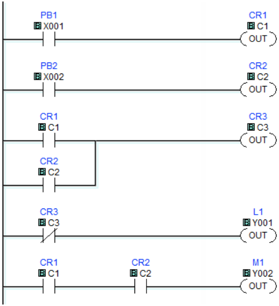
Railway Interlocking Computers

Algorithm 5: PLC Operation

input : Sequence of values
output: Sequence of values

```
initialisation
while (true) do
    read (Input)
(*) State' ← ControlProgram(Input, State)
    write (Output') & State ← State'
```

A PLC Control Program in Ladder Logic



Ladder Logic: Graphical Programming Language

Standardized by the International Electrotechnical Commission in document IEC 61131-3 "Programmable controllers – Part 3: Programming languages"

Note: In our version of Ladder Logic, everything is of type Boolean.

Finite transition system defined by propositional logic

Let $\bar{x} = (x_1, \ldots, x_n)$, $\bar{x}' = (x_1', \ldots, x_n')$, and $\bar{i} = (i_1, \ldots, i_m)$ be vectors of Boolean variables, for some $m, n \geq 0$.
Given propositional formulae

- $I(\bar{x})$ – the initialisation condition – and
- $T(\bar{x}, \bar{i}, \bar{x}')$ – the transition condition) –

we define a *labelled transition system* $\mathcal{S} = (S, \longrightarrow, \mathit{Init})$ as follows:

- The set of all Boolean vectors $S = \{0,1\}^n$ is the set of states;
- $\longrightarrow \subseteq S \times \{0,1\}^m \times S$ is the transition relation given by

$$s \stackrel{i}{\longrightarrow} s' :\Longleftrightarrow T(s, i, s') \text{ evaluates to } 1;$$

- $\mathit{Init} = \{s \in S \mid I(s) \text{ evaluates to } 1\}$.

Example: Ladder Logic Defines a Finite Transition System

Variables:

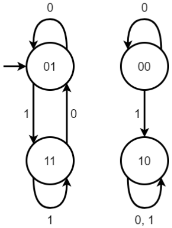
- Two Boolean state variables: b, s
- One Boolean input variable: i

Characterization of initial states: $\neg b \wedge s$

A Ladder Logic program (written as two Boolean Formulae):

$$\begin{aligned} b' &\iff i \vee (\neg s \wedge b) \\ s' &\iff s \end{aligned}$$

Resulting Transition System (state: first b, then s; i label)



Model Checking Problem

Definition

Given a safety property P,
compute if P holds
in all reachable states of the transition system.

Example: $\neg b \vee s$

Model checking can realised, e.g., via 'Inductive Verification':
Provided

- $I(\bar{x}) \longrightarrow P(\bar{x})$ and
- $P(\bar{x}) \ \wedge \ T(\bar{x}, \bar{i}, \bar{x}') \longrightarrow P(\bar{x}')$

hold, then $\mathcal{S}$ has safety property P.
Requires 2 calls to a SAT-Solver (e.g., Z3).

The problem is decidable as the state space is finite,
though possibly large.

Theoretical Framework Defined

Recipe:

- Model check the transition system
- Powerful off-the-shelf SAT solvers are available

Comparison to established practice:

- Testing: considers *some* of the (reachable) states
- Model checking: considers *all* (reachable) states

$\rightsquigarrow$ This all is text book knowledge these days

Unclear:

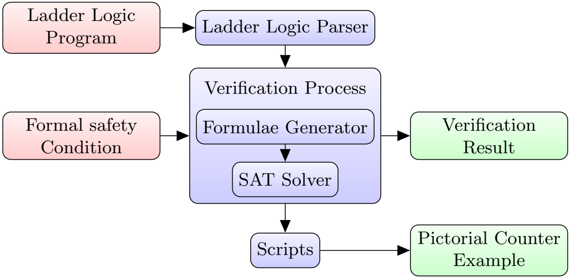
- Does it scale to, say, 12,000 rungs, i.e., $2^{12,000}$ states?
- Can we express the properties we are interested in?

1st Implementation

It's all clear,

isn't it?

Software Architecture



Realised in two MRes Projects (research degree, duration 1 year).

Added Features

Optimisation:

- Reading off the Ladder Logic Program using the 'Tseitin transformation'
- Reducing size of the program through 'slicing'

Adding 'Bounded Model Checking' to generate counter example traces

Evaluation of the 1st Prototype

Positive Outcome:

- Fully automatical verification of Ladder Logic programs

Scalability challenge:

- With slicing, can effectively handle small interlocking programs (300 rungs)

$\rightsquigarrow$ Off by two orders of magnitude: need $12,000$ rungs

Usability:

- Only ‘academic’ users, running several parametrised scripts from a terminal
- No ‘pre-modelled’ safety properties

Interoperability:

- ‘Wild-West’ of interfaces

2nd Implementation

Now we will get everything right, won't we?

Safety Requirements – informal

Before a movement authority can be given, the following conditions are required:

		Route Class	
Ref	Control	Main	Shunt
1	Route set and locked to exit signal	YES	YES
6	Points in route are in the correct position, locked, and detected.	YES	YES
12	All train detection devices in the route indicate the line is clear.	YES	Where specified by infrastructure controller
22	Junction and route indicators required to be proved are alight (see section C7.3).	YES	YES
32	'All-signals-on' or signal group replacement controls not operated.	YES	YES
35	Approach and route locking has been applied.	YES	YES

Excerpt from "Interlocking Principles", Railway Group Standard, 2003.

Variable Naming Scheme in Ladder Logic Programs

Element	Property	822 Prefix	822 Suffix
Track	Occupied	T<SEGMENT>	.OCC(IL)
Track	Clear	T<SEGMENT>	.CLR
Track	Locked	NUSR	.T<SEGMENT>
Point	Reverse (Signal Level)	P<POINT>	.RL
Point	Normal (Signal Level)	P<POINT>	.NL
Point	Detected Reverse	P<POINT>	.RWK
Point	Detected Normal	P<POINT>	.NWK
Point	Locked (Reverse)	NUSR	.(R)P<POINT>
Point	Locked (Normal)	NUSR	.(N)P<POINT>
Route	Set (Signal)	S<SIGNAL>	.RU
Route	Set (Route)	S<SIGNAL(ROUTE)>	.U
Route	Released	S<SIGNAL(ROUTE)>	.ALS
Signal	Approach Locking	S<SIGNAL>	.APPR
Signal	Shows Proceed	S<SIGNAL>	.G
Signal	Proved Alight	S<SIGNAL>	.EC
Signal	Signal Group Replacement Controls	MSDP6	.SGRC

Verification

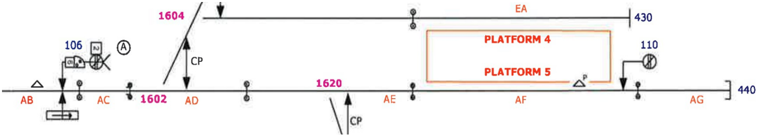
SIEMENS
Ingenuity for life

Safety requirement:

Proceed Aspect with Exit Signal

A signal shall only display a ‘proceed’ aspect if the exit signal of the signalled path from the signal is proved alight.

$$S106.HGE\_1 \lor S106.DGE\_1 \Rightarrow S110.EC\_0$$



Complete Rewrite of the Verification Engine

Change of implementation language:

- Was: Haskell
- Now: C$\sharp$ – adhering to industrial software standards

Main add-on: Encoding of $\sim$ 300 safety principles

- Optimised for efficient verification, e.g.,

  $$\forall x \,.\, \varphi(x) \rightarrow \big(\forall y \,.\, \psi(x,y) \rightarrow \xi(x,y)\big)$$

  leads to faster verification than a property of the form

  $$\forall x, y \,.\, \big(\varphi(x) \wedge \psi(x,y)\big) \rightarrow \xi(x,y)\big)$$

  (experimental evidence)

Realised in about 3 years: 1 MRes project, then team of 2 industry SE and 2 seconded academics.

Evaluation of the 2nd Prototype

Positive Outcomes:

- Fully integrated in the Siemens Mobility ecosystem
- User interface ‘managable’ for rail engineers
- Scalability successfully challenge addressed:
  - 300 properties in 2 hrs for large interlocking programs

Challange to the verification methods:

- 35–40% of safety properties cannot be decided

The Journey continues –

but hopefully not
ad infinitum :-)

Where We Are Now

Use of Ladder Logic Verifier has the potential
to reduce time and cost,
whilst increasing software quality.

Identified potential:

- Faster turn-around of railway signalling projects
- New work practices during design cycles
  ('trial and error-problem-solving')
- Experts in quality assurance are less disturbed by 'noise'
- Shorten the critical path in signalling software development

Lessons Learnt

Once more confirmed:

- The journey is longer than expected, to be measured in years
- Challenges are hard to predict – comprehendable only when they appear

Some lessons learnt:

- There appears to be a systematic, that can't be ignored: foundations – academic trial – industrial trial – business case – in production
- Methods from Empirical Software Engineering and Management are inevitable in the later phases

Current Challenges as of Today – Mostly Empirical

- Safety properties that the 2nd prototype can't deal with
  $\rightsquigarrow$ Positive results with 'IC3' algorithm
  (1 MRes project completed, 4 months postgrad time)
- How 'good' are the 300 safety properties in uncovering mistakes?
  $\rightsquigarrow$ Verification with error injection
  (1 ongoing MRes project; 1 PhD project to start 10/24)
- Are Formal Methods at least as 'thorough' as testing?
  $\rightsquigarrow$ Systematic comparative study on historic developments
  (3 months postgrad time)
- Cost/Benefit analysis
  $\rightsquigarrow$ Modelling with management methods
  (3 months postgrad time)
- Shadowing a live development
  (6 months project applied for)

Teaching Resources

https://sefm-book.github.io/lab-classes.html