Formal Verification of Safety Properties in Systems Defined with Ladder Logic

Karim Kanso

Swansea University

9^{th} April 2008

Prepared for BCTCS 2008

This work is funded by *Invensys Rail Group, UK*

Similar Research

Over the past 10 years formal verification has been researched by many different individuals, the following list highlights research that is similar to my own.

- **L.H. Eriksson**. Formalising Railway Interlocking Requirements. *Swedish National Rail Administration Technical Report*, 3, 1997.
- **L.H. Eriksson**. Formal Verification of Railway Interlockings. *Swedish National Rail Administration Technical Report*, 4, 1997.
- **W.J. Fokkink**, **P.R. Hollingshead**, **J.F. Groote**, **S.P. Luttik**, and **J.J. van Wamel**. Verification of interlockings: from control tables to ladder logic diagrams. *Proceedings 3rd Workshop on Formal Methods for Industrial Critical Systems (FMICS98)*, pages 171–185, 1998.

There are too many to list in full but these papers are directly applicable.

Ladder Logic

- Used for programming integrated circuits (*IC's*)
- Ladder logic is defined by the standards
  - **IEC 61131-3** and
  - **BSI EN 61131-3:2003**
- Consists of rungs
  - Rungs are propositional formulæ

Ladder Logic: Rungs

Rungs are propositional formulæ of the form

$$\varphi \leftrightarrow a$$

where

φ is a propositional formula built from atomic propositions, conjunctions, disjunctions and negation.

a is an atomic proposition, can also be thought of as the result of the rung.

Ladder Logic: Semantics

- A ladder consists of an ordered sequence of rungs
- The results of "previous" rungs can be used as propositions in subsequent rungs
- The ladder is repeated indefinitely
  - Condition necessary as ladder logic is used in critical systems such as a railway interlocking.
- Typically the initial states of the propositions are defined

Propositional Ladder

To convert a ladder $\mathcal{L}$ into a propositional formula $\varphi_{\mathcal{L}}$ the following is performed:

For each $(\varphi \leftrightarrow a) \in \mathcal{L}$. Create a "fresh" atomic proposition, a', for the result of the rung and replace all occurrences of a by a' in all subsequent rungs of $\mathcal{L}$.

Finally to generate $\varphi_{\mathcal{L}}$ requires that a conjunction of all rungs be taken, order of the rungs is not important here.

Safety Conditions

Given a safety condition $\varphi_{\mathcal{S}}$, I want to show

$$\varphi_{\mathcal{L}_0} \wedge \varphi_{\mathcal{I}} \rightarrow \varphi_{\mathcal{S}_1}$$

and

$$\varphi_{\mathcal{L}_n} \wedge \varphi_{\mathcal{S}_n} \rightarrow \varphi_{\mathcal{S}_{(n+1)}}$$

where $\varphi_{\mathcal{I}}$ is the initial state of the variables.

This is *Hoare* Logic, the first formula states from the initial state the safety condition $\varphi_{\mathcal{S}}$ holds; the second states that from an arbitrary state the safety condition holds.

SAT Solvers

The Boolean satisfaction problem as described by Stephen Cook is as follows:

Given a propositional formula φ with atomic propositions p_0, p_1, ...,p_n. An assignment of values to the atomic propositions such that φ is satisfied.

This problem was the first problem in the complexity class **NP**.

SAT Solvers: Classes

SAT Solvers come in different flavours, all based upon **DPLL**:

Conflict Driven

- Looks for cases where a proposition has to satisfied and falsified
- Backtracking can use a heuristic to select a branch
- Good for solving industrial problems (*large but simple*)

Look Ahead

- High level of reasoning, analyses the problems
- Good for solving hard problems (*small but hard*)

Conceptual Model

A system specified using ladder logic can be viewed as a Boolean program where the atomic propositions are either:

- Input,
- Output *or a*
- Latch (Storage)

This Boolean program is executed indefinitely.

Latches are used for passing information between the execution cycles.

Consistency of Cycles

When validating the safety conditions there are scenarios where two or more cycles are dependant upon the input propositions.

E.g. The first cycle looks at the input propositions (*state of the world*) and decides what to do, then the second cycle will assume certain conditions to be upheld with respect to the input propositions.

When verifying the ladder logic these dependencies need to be specified.

Invariance

Invariants for verification fall into various categories:

- Logical
  - Conclusions from the formulæ that can be proven using natural deduction.
- Input Constraints
  - Physical constraints from the real world inputs
    - A switch that can only be in one of three positions, or none
    - Typically real world devices that can only be in one state at a time
- Operational / Reachable States
  - Consistency of Cycles (as discussed on last slide)
  - Linking cycles together, such that only true successors are examined

Questions Raised

The major research aim of the project is to identify and define invariants such that they are sufficient to allow safety properties to be formally verified.

The choice of SAT Solver; conflict driven or look ahead? Currently working with **OKSolver2002** which is look ahead.

An interesting question is whether the safety conditions should only refer to inputs and outputs or latches as well. This is raised as an internal "*violation*" of the system does not necessarily produce a violation in the outputs.

Thankyou for Your Time

Questions?