Logical Methods in the Formal Verification of Safety-Critical Software

C.Pulley and G.V.Conroy
Department of Computation, University of Manchester Institute of Science and Technology

1 Introduction

This paper reports on some work that is currently in progress under the SERC/DTI Advanced Technology Program Safety Critical Systems initiative (project number IED4/1/9035). The project covers research into the design and checking process for data written in application specific languages and is a collaborative effort between UMIST, GEC ALSTHOM Signaling Limited and Westinghouse Signals Limited.

The project concentrates on one particular language - the **S**olid **S**tate **I**nterlocking (SSI) data language. This language is used to define the functionality of safety-critical railway signaling interlocking installations based on the SSI system (see [1]).

The purpose of the work reported in this paper is to describe a formal procedure by which one may automatically verify the safety of a specific SSI interlocking. Unlike other research work reported on elsewhere (see eg. [2] and [3]), we shall explicitly model the IFPs polling cycle within the formalism to be presented shortly. This has the added benefit of enabling us to more correctly model the SSIs behavior and to also formally verify the safety of the IFP. Unlike [3], we are able to express and verify much more general safety properties. Our work also differs from the above in that we have concentrated on trying to improve the efficiency of our decision procedure rather than by concentrating on the *modes* by which we may verify certain types of safety property.

This paper contributes to current research on the verification of safety properties for SSIs in a number of specific ways. Firstly, the use of our modeling techniques (ie. use of second-order monadic arithmetic with the successor function) enables us to provide a precise logical classification of the types of safety properties that may be verified, along with a means by which

communicating groups of interlockings may also be verified. The explicit modeling of the IFP polling cycle is an additional contribution of our work in this area. Finally, by relating our verification strategy to garbage collection, we also contribute to this area by indicating how one may parallelise the verification of interlocking safety properties.

After giving an overview of the SSI system in section 2, we go on to show how SSIs may be modeled using finite state machines in section 3. In section 4, we introduce a safety property language and then go on to give a logical classification of the types of safety properties that may be expressed within the language. This section is finished by demonstrating the relationship between garbage collecting and verifying the safety of an SSI. Section 5 discusses some ongoing research work, with section 6 finishing the paper by detailing some relationships between the approaches adopted in this paper and those used elsewhere.

Finally, thanks go to Bob for allowing the authors to *plagiarize* the introductory sections of [4].

2 Overview of the SSI System

An SSI (**S**olid **S**tate **I**nterlocking) is a generic railway signal control system designed to replace the more expensive and less flexible electromechanical systems that are currently in use. Each SSI installation has the function of controlling the points and signals on one section of a railway network. Interlocking between these controls ensures that safety of train movements is maintained at all times.

2.1 Hardware

The overall concept of an SSI may be summarised as follows:

- A centralised safety-critical microcomputer system issues commands to, and receives status indications from, up to 63 safety-critical controllers (the **T**rackside **F**unctional **M**odules or TFMs) distributed along the railway and up to 15 other SSI interlockings (thus enabling train movements between different interlocking control authorities).
- Each TFM contains a dual-channel microcomputer system, for safety purposes, and it drives and monitors either one or two colour-light signals, or up to four sets of points, as well as monitoring the status of train detection systems, etc..

- In the interests of availability, the centralised equipment is fault-tolerant, with active redundant 1-out-of-2 and 2-out-of-3 hardware configurations used for non-safety and safety portions of the system respectively.
- Communication between the centralised and distributed parts of the system is achieved by securely-coded digital links, duplicated for availability, carried on screened twisted-pair copper cables for distances of up to 40km. In addition, 64 kbit/s channels in optical fiber PCM systems may be used for longer distance communication.
- The detailed functional requirements of each installation are defined by the applications data, which is compiled into about 60k bytes of object data, and stored in EPROMs in each channel of central equipment.

2.2 SSI Data Language

For every new application of SSI, data must be prepared for, among other things, the interlocking system. The prepared data is in the form of a series of source files, which are then passed through several compiler programs to create sets of object files used to program EPROMs for each type of hardware module. Once these data EPROMs are installed in the hardware, the fixed interlocking program uses the SSI data to define the conditions to be applied at each stage of interlocking operation.

2.2.1 Fixed Interlocking Program

For the purposes of this paper the fixed interlocking program consists of the **I**nterlocking **F**unctional **P**rogram (IFP). The IFP is broken down into a series of processing stages.

First, there is the initialisation stage. Of these there are three possible start up modes:

Start up Modes 1, 2 and 3

For the purposes of this project we have taken start up mode 3 as our initialisation stage. The remaining two start up modes can be thought of as special cases of this start up mode.

The IFP then repeats a processing stage called the major cycle. A major cycle is further broken down into 64 minor cycles. Each minor cycle is essentially used to interact with a particular TFM. Minor cycles are themselves broken down into 5 message cycles. The processing activity of these message cycles are detailed below:

- **Processing of Incoming Telegrams** - an incoming telegram from a TFM and possibly another interlocking are processed on this message cycle.
- **Processing of Flag Data** - roughly a $\frac{1}{64}^{th}$ of the flag data is processed on this message cycle.
- **Processing of Outgoing Telegrams** - an outgoing telegram to a TFM and possibly another interlocking are processed on this message cycle.
- **Updating of Timers** - 2 signal timers, 4 track circuit timers and 1 elapsed timer are processed on this message cycle.
- **Processing of Panel Requests** - if time is left (a minor cycle may only take a maximum of 30 ms), then a panel request is processed on this message cycle.

Each major cycle exercises the whole of the flag data and input and output telegram data. All other data is accessed as and when needed.

2.2.2 SSI Applications Data

The SSI applications data consists of a series of SSI source data files. The SSI source data files include the logical conditions which must be applied at various stages of interlocking operation. These include:

- Input and output telegrams (defining the logical relationship between bits in the telegrams sent to, and received from, trackside controllers and signaling functions within the interlocking).
- Flag operations (logical conditions for releasing routes and for implementing user-defined functions).
- Panel requests (logic associated with the operator interface).
- Points free to move (conditions affecting point operation)
- Map data (a simplified map of the layout, used by the interlocking program to search through the layout).

The interlocking program accesses some of these files regularly, others as required (see the previous subsubsection).

3 Modeling the SSI System Using Finite State Machines

In this section we describe how we intend to formally model our SSI as a finite state machine. Before doing this we note the following important details of the implementation that allow us to simplify our presentation:

- Any SSI can only communicate with at most 63 TFMs, at most 15 other interlockings and receive at most 1023 different panel requests (ie. there is a finite range of possible inputs and outputs to an SSI).
- An SSI can only be in one of a finite number of internal memory states (ie. finite range of states).

These two facts together suggest that an SSI may indeed be modeled by a finite state machine. The following simply formalises this intuition.

Definition 1. *Fix a nonempty finite set S of states, a finite set I of inputs, a finite set O of outputs, an accessibility relation R[1] $\subseteq S \times I \times S \times O$ and a nonempty set $I_0 \subseteq I$ of initial states. A finite state machine is then the tuple (S, I, O, R, I_0)*[2].

We may now model an SSI as a finite state machine as follows:

- I is taken to be the set of all possible tuples $(l, i_0, .., i_{78})$, where l is a partial map from the set $\{0, .., 63\}$ to the set of all possible panel requests for this SSI[3], and $i_0, .., i_{63}$ are incomming telegrams from the TFMs and $i_{64}, .., i_{78}$ are incoming telegrams from other SSIs.
- O is taken to be the set of all possible tuples $(o_0, .., o_{78})$, where $o_0, .., o_{63}$ are outgoing telegrams to the TFMs and $o_{64}, .., o_{78}$ are outgoing telegrams to other SSIs.
- S is taken to be the set of all possible internal memory states of the particular SSI.

[1] Notice that R may here be nondeterministic. This is something we shall discuss briefly, later on in this paper.

[2] Note that we have intentionally not included any final states for our finite state machine.

[3] This partial map models the fact that an SSI may only process at most one incoming panel request per minor cycle.

- R is defined by the major cycle of the particular SSI. So we have that:

$$R(\sigma, i, \sigma', o)$$

  holds precisely when our SSI is started in state σ with inputs i and, after one major cycle, is in state σ' and has produced outputs o.

- $I_0 \subseteq S$ is taken to be the set of valid mode 3 start up states for the particular SSI.

Our previous section and remarks ensure that these sets are indeed finite. For reasons of mathematical simplicity we shall find it more convenient to model our inputs and outputs as being part of our finite state machines state. Thus, given a finite state machine $M = (S, I, O, R, I_0)$, we shall in fact work with the finite state machine $M' = (I \times S \times O, \emptyset, \emptyset, R', I \times I_0 \times O)$, where R' is given by:

$$R'((i, \sigma, o), (i', \sigma', o')) \text{ iff } R(\sigma, i, \sigma', o)$$

Since such machines no longer have any need to refer to their input and output sets, we shall simply talk about the finite state machine $M' = (I \times S \times O, R', I \times I_0 \times O)$.

4 Formally Reasoning about the Safety of an SSI

Our aim is to formally verify that an SSI never allows a railway to enter an unsafe state. What exactly we shall mean by the terms *safe* and *unsafe* shall not be described here (this is the subject of some research work that is currently being carried out by Westinghouse Signals Limited). Instead we shall define a formal logical language that is sufficiently expressive enough to capture our intuitive notions of *safe* and *unsafe*. Using this safety property language, we may then go on to concentrate on the problem of how we may formally verify that our SSI satisfies a given property. We shall here be working in the main with the dual problem of that presented in [3]. It shall shortly be made clear that our approach is to perform a behavioral analysis of our SSI (see [2] and [3]).

Definition 2. *Given a finite state machine* $M = (S, R, I_0)$[4]*, we define an* M*-stream* $\sigma_0, \sigma_1, ...$ *to be an infinite sequence of members of* S *such that:*

$$\sigma_0 \in I_0 \text{ and } \forall i \geq 0 \cdot R(\sigma_i, \sigma_{i+1})$$

[4]Recall that in the previous section we made the assumption that all inputs and outputs are to be modelled as state items. Thus, finite state machines can be thought of as finite directed graphs with at least one distinguished node.

For simplicity, we insist on dead end states as being modeled by repeating the state ad infinitum.

When we talk of wishing to perform a behavioral analysis of our SSI, we mean that we wish to constrain the possible M-streams of our SSI (where M is the finite state machine that models our specific SSI). So that we may formalise this further we introduce the following logical language that shall enable us to express various behavioral properties of our SSI.

Definition 3. *Let TL be the set of terms built from the constant term 0, the variables $x_i\,(i \in \,)$ and the unary function s. We now define FL to be the set of first-order formulae built using the terms TL, the unary predicates $p_i\,(i \in \,)$, the binary connective $\leq$, the propositional connectives $\neg$, $\vee$, $\wedge$ and $\Longrightarrow$ and the first-order quantifiers $\forall$ and $\exists$.*

As is usual, we shall adopt the convention of allowing the symbols $p, q, r, ...$ (with or without indexing) to also denote unary predicate symbols.

The formulae in FL shall be used to specify our required behavioral properties. The following definition describes how this may be done:

Definition 4. *Fix a finite state machine M (with $S \subseteq (\,)$), an M-sequence $\sigma : \longrightarrow S$ and an assignment function $A : \{x_i | i \in \,\} \longrightarrow$. We define an interpretation function $i_A : TL \longrightarrow$ by structural induction on TL as follows:*

$$i_A(0) = 0$$
$$i_A(x_i) = A(x_i)$$
$$i_A(s(t)) = i_A(t) + 1$$

and the M-satisfaction relation $\sigma, i_A \models_M \theta$ by structural induction on $\theta \in FL$ as follows:

$$\sigma, i_A \models_M t_1 \leq t_2 \text{ iff } i_A(t_1) \leq i_A(t_2)$$
$$\sigma, i_A \models_M p_i(t) \text{ iff } i \in \sigma(i_A(t))$$
$$\sigma, i_A \models_M \neg\varphi \text{ iff } \sigma, i_A \not\models_M \varphi$$
$$\sigma, i_A \models_M \varphi \wedge \psi \text{ iff } \sigma, i_A \models_M \varphi \text{ and } \sigma, i_A \models_M \psi$$
$$\sigma, i_A \models_M \varphi \vee \psi \text{ iff } \sigma, i_A \models_M \varphi \text{ or } \sigma, i_A \models_M \psi$$
$$\sigma, i_A \models_M \varphi \Longrightarrow \psi \text{ iff } \sigma, i_A \models_M \varphi \text{ implies } \sigma, i_A \models_M \psi$$
$$\sigma, i_A \models_M \forall x_l \cdot \varphi \text{ iff for all } j \simeq_{x_l} i_A,\ \sigma, j \models_M \varphi$$
$$\sigma, i_A \models_M \exists x_l \cdot \varphi \text{ iff there exists } j \simeq_{x_l} i_A \text{ such that } \sigma, j \models_M \varphi$$

where $j \simeq_{x_l} i_A$ means that the interpretation function $j : TL \longrightarrow$ satisfies the following condition:

$$\forall m \in \,\cdot\, m \neq l \text{ implies } j(x_m) = i_A(x_m)$$

M-validity is now defined using the above definition of M-satisfiability as usual by:

$$\models_M \theta$$
iff
$$\forall M\text{-sequence } \sigma, \text{ assignment function } A : \{x_i | i \in \} \longrightarrow \cdot \ \sigma, i_A \models_M \theta$$

The intuition behind the above definition is that the unary predicates $p_i\,(i \in)$ enable us to assert simple (ie. propositional) statements about particular states of our given finite state machine M (hence the reason why a state consists of the indexes of those unary predicates that are true at the given state). The terms in TL enable us to specify particular points, within our state sequences (eg. an initial state, a next state or an arbitrary state), at which our unary predicates are to hold. The remaining logical connectives enable us to assert statements constraining the way a proposed M-sequence may develop.

The following proposition (due originally to [5]) provides one with a means by which the majority of our first-order quantifiers may be eliminated:

Proposition 1. *Fix a finite state machine $M = (S, R, I_0)$ with $S \subseteq ()$. Then for any $\theta \in FL$, there is a $\theta' \in FL$ such that,*

$$\models_M \theta \text{ iff } \models_M \theta'$$

and what is more, θ' has the following format:

$$\sigma_0 \wedge \forall t \cdot \rho_{t,s(t)} \Longrightarrow \forall x \cdot \tau_x$$

where σ_0, $\rho_{t,s(t)}$ and τ_x are quantifier free formulae that do not contain the relation symbol $\leq$ and only contain the terms indicated by their indexes.

Proof. By [6] we have a θ'' such that,

$$\models_M \theta \text{ iff } \models_M \theta''$$

with θ'' having the following format:

$$\sigma_0 \wedge \forall t \cdot \rho_{t,s(t)} \Longrightarrow (\exists y \forall x \cdot y \leq x \Longrightarrow \tau_x)$$

where σ_0, $\rho_{t,s(t)}$ and τ_x are quantifier free formulae that do not contain the relation symbol $\leq$ and only contain the terms indicated by their indexes. Again by [6],

$$\neg\theta'' \text{ is } M\text{-satisfiable}$$
iff
$$\neg\theta'' \text{ is } M\text{-satisfiable by an ultimately periodic } M\text{-sequence}$$
$$\text{of phase } \phi \text{ and period } p$$

What is more we have that $\phi + p \leq 2^{\text{number of distinct unary predicates in } \neg\theta''}$. Thus, since we have an ultimately periodic M-sequence (see [6]), we have an $n \in$ such that,

$$\begin{gathered}\neg\theta'' \text{ is } M\text{-satisfiable} \\ \text{iff} \\ \sigma_0 \wedge \forall t \cdot \rho_{t,s(t)} \wedge (\exists x \cdot s^n(0) \leq x \wedge \tau_x) \text{ is } M\text{-satisfiable}\end{gathered}$$

where $s^0(0) = 0$ and $s^{n+1}(0) = s(s^n(0))$. Now let q_0,..,q_n be distinct unary predicates that do not appear anywhere in θ''. Then we have that,

$$\begin{gathered}\sigma_0 \wedge \forall t \cdot \rho_{t,s(t)} \wedge (\exists x \cdot s^n(0) \leq x \wedge \tau_x) \text{ is } M\text{-satisfiable} \\ \text{iff} \\ \sigma_0 \wedge \mu_0 \wedge (\forall t \cdot \rho_{t,s(t)} \wedge \lambda_{t,s(t)}) \wedge (\exists x \cdot q_n(x) \wedge \tau_x) \text{ is } M\text{-satisfiable}\end{gathered}$$

where μ_0 is the formula;

$$q_0(0) \wedge \bigwedge_{i=1}^{n} \neg q_i(0)$$

and $\lambda_{t,s(t)}$ is the formula;

$$\bigwedge_{i=0}^{n-1} (q_i(t) \Longrightarrow q_i(s(t)) \wedge q_{i+1}(s(t))) \wedge (q_n(t) \Longrightarrow q_n(s(t)))$$

We now obtain our required result by taking θ' to be the formula,

$$\sigma_0 \wedge \mu_0 \wedge (\forall t \cdot \rho_{t,s(t)} \wedge \lambda_{t,s(t)}) \Longrightarrow (\forall x \cdot q_n(x) \Longrightarrow \tau_x)$$

□

The previous proposition thus shows that if we are satisfied with FL as being expressive enough to capture all our required safety properties, then we need only express safety properties in the form:

$$\sigma_0 \wedge \forall t \cdot \rho_{t,s(t)} \Longrightarrow \forall x \cdot \tau_x$$

In fact [6] demonstrates that a similar result holds even if we allow second-order quantifiers! But this more general result is beyond the needs of this paper.

Example 1. *Consider the following simple track layout consisting of a single signal (S), that may display one of two aspects (green(g) and red(r)), and*

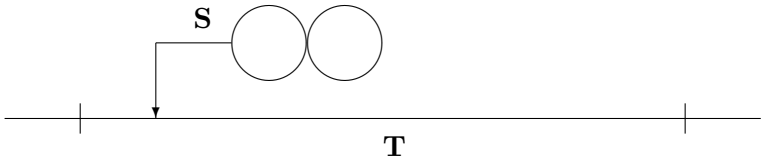

Figure 1:

a single track circuit (T), that may be in one of two states (occupied and unoccupied). The signal S is an entry signal to a number of routes;

By way of an example of the sort of safety properties that we have to verify, we shall consider a special case of approach locking. A signal may become approach locked when it displays a proceed aspect (ie. green). Approach locking ensures that in the event of a signalman requesting (with a panel request Q) that a route (R), starting at our given signal S, be cleared, then any train that may not yet have cleared or reached the route is given sufficient time to either clear the route or stop at the signal S. The route may not be cleared until the signal has been cleared of approach locking. Release of approach locking may occur once a signal has changed to a red aspect and a timeout period has expired (for us this shall be a period of 180 major cycles). The following formulae describe the conditions for release of approach locking to occur;

1. $R_{set}(x) \wedge S_{aspect,g}(x) \wedge Q_{clear}(x) \Longrightarrow$
   $S_{state,applock}(s(x)) \wedge S_{timer,180}(s(x)) \wedge S_{aspect,r}(s(x))$
2. $S_{state,applock}(x) \wedge \neg S_{timer,stopped}(x) \Longrightarrow S_{state,applock}(s(x))$
3. $\bigwedge_{n=1}^{255}(S_{timer,n+1}(x) \Longrightarrow S_{timer,n}(s(x)))$
4. $S_{timer,1}(x) \Longrightarrow S_{timer,stopped}(s(x))$
5. $S_{timer,stopped}(x) \Longrightarrow S_{timer,stopped}(s(x))$

Property 1 ensures that a signal shall become approach locked whenever the signal S is green, a route R is set from this signal and a panel request Q has been sent to clear the route. Under these conditions, the signal is to become approach locked, have its timer set for a 180 major cycle wait and its aspect set to red; Property 2 ensures that a signal maintains approach locked as long as the signals timer has not expired (ie. been stopped); Properties 3, 4

and 5 describe how a signals timer is updated over a given major cycle (these properties are contained within the logical description of the given SSI). Denoting property 1 and property 2 by the expressions $property_1(x, s(x))$ *and* $property_2(x, s(x))$ *respectively we get the following safety property that is to be verified of our SSI (properties 3, 4 and 5 already being incorporated into the logical description of our SSI and thus do not need to be checked);*

$$(\forall x \cdot property_1(x, s(x)) \wedge property_2(x, s(x)))$$

Placing this property into the normal form as given by proposition 5 gives us the following safety property that is to be verified (where P_1 *and* P_2 *are two new unary predicates);*

$$\begin{array}{c}(\forall x \cdot (property_1(x, s(x)) \iff P_1(x)) \wedge \\ (property_2(x, s(x)) \iff P_2(x))) \Longrightarrow \\ (\forall y \cdot P_1(y) \wedge P_2(y))\end{array}$$

Using the decidability algorithm originally due to [5], we can now (at least in principle) solve our original problem of verifying that a given safety property holds of an SSI. The remainder of this section describes a simplification of this original algorithm.

[2] and [3] appear to be concerned with demonstrating that a safety property is M-valid, whereas we shall instead mainly work with the dual problem of M-satisfiability. Thus, we shall be interested in showing that an SSI can never evolve into a state that fails our given safety property. The remainder of this section shall demonstrate the connection between this problem and the one of *garbage collection*.

Definition 5. *Let* Mem *be a (nonempty) set of memory cells,* $\emptyset \neq Root \subseteq Mem$ *and* $Link \subseteq Mem \times Mem$*. Then for* $m \in Mem$*, we say that:*

$$m \text{ is garbage iff there is no } r \in Root \text{ such that } (r, m) \in Link^*$$

where $Link^*$ *is the reflexive, transitive closure of the relation* $Link$*.*

Lemma 1. *Fix a finite state machine* $M = (S, R, I_0)$ *and a state* $\sigma_f \in S$*. Then the following problems are equivalent:*

$$\begin{array}{c}\sigma_f \text{ is not garbage} \\ \exists M\text{-stream } \sigma_0, \sigma_1, ... \exists i \in \cdot \ \sigma_i = \sigma_f\end{array}$$

Proof. Take $Mem = S$, $Link = R$ and $Root = I_0$ □

The previous lemma demonstrates that if we view unsafe states as being garbage, then garbage collection can be thought of as providing the basis of a decision procedure for showing that an SSI satisfies a given property. This lemma is nothing more than a statement of the well known connection between the problem of reachability and garbage collection. Unfortunately, as we shall see in the next section, the memory spaces over which we wish to garbage collect are *astronomically* large. So usual garbage collecting strategies such as mark and sweep, copy collection, etc. just can not in general be applied in a computationally efficient manner.

However, we may exploit techniques such as lazily evaluating our garbage collecting algorithms, in order to help improve on the efficiency of our garbage collecting strategies. In addition we may also exploit this connection with garbage collection to help parallelise our decision procedure. To further improve our computational efficiency, some other techniques are required. Some possibilities are discussed in the next section.

5 Further Research

We have seen in the previous section that any safety property expressed by a formula of FL may be placed in the form,

$$\sigma_0 \wedge \forall t \cdot \rho_{t,s(t)} \Longrightarrow \forall x \cdot \tau_x$$

with quantifier free formulae σ_0, $\rho_{t,s(t)}$ and τ_x containing no relation symbols $\leq$ and only containing terms given by their indexes.

Since we are here interested in the problem of M-satisfiability, we are interested in demonstrating the existence of an M-sequence σ that M-satisfies $\sigma_0 \wedge \forall t \cdot \rho_{t,s(t)}$ and that fails to M-satisfy $\forall x \cdot \tau_x$. Now if we think of M as being a finite state machine that models a given SSI, then assuming that all predicates that occur in our safety property relate to simple assertions about M's state, we may think of $\sigma_0 \wedge \forall t \cdot \rho_{t,s(t)}$ as placing further constraints upon the evolutionary behavior of M. We could even think of $\sigma_0 \wedge \forall t \cdot \rho_{t,s(t)}$ as *pruning* some of the IFP's possible execution paths. This is a particularly attractive way of thinking about these formulae, since it suggests a means by which we may further improve the computational efficiency of our verification problem.

Unfortunately, it is not possible in general to think of $\sigma_0 \wedge \forall t \cdot \rho_{t,s(t)}$ in this manner, since it can contain predicates that do not make assertions about the state of M but rather are more concerned with simulating the behavior of the $\leq$ relation, repeated applications of s, etc..

Some ongoing research work is to classify the roles of these additional predicates in an attempt to better understand this relationship between our safety properties and the way they may constrain the evolutionary behavior of an SSI.

Due to the *astronomical* size of our search spaces (typically they contain more states than the number of electrons in the universe! - see [7]) some care needs to be exercised in how we implement our decision procedures. Thus, the obvious mark and sweep garbage collecting strategy proposed in the previous section is in general too costly in terms of our time and space resources. This leads one to consider some means by which we may break our overall problem into a collection of simpler ones.

One approach that has attracted some interest by other researchers (see [3] and [8]) is that of decomposition by the use of topological properties of our finite state machines. One aspect in which we differ from their work, is that we intend to exploit the structure of the IFP in order to decompose our verification problem into simpler portions. The intuitive idea is that it is easier to verify that a property holds for each of the message cycles of an arbitrary minor and major cycle than to show that the same property holds of an arbitrary major cycle. Since a major cycle is a sequential composition of message cycles, one would like to have some way of *sequentially* composing safety properties that hold over particular message cycles. The following proposition establishes this property for some restricted types of safety property. However, we first need a technical lemma.

Lemma 2. *Let $S \subseteq ()$, $\theta_{x,s(x)}$ be a quantifier free formula that only contains the terms given by its index. Furthermore let $A, A' : \{x_i | i \in \} \longrightarrow$ and $\sigma, \sigma' : \longrightarrow S$ be such that;*

$$\sigma(i_A(x)) = \sigma'(i_{A'}(x)) \text{ and } \sigma(i_A(x)+1) = \sigma'(i_{A'}(x)+1)$$

Then;

$$\sigma, i_A \models \theta_{x,s(x)} \text{ iff } \sigma', i_{A'} \models \theta_{x,s(x)}$$

where $\models$ means that we work not with M-sequences for some finite state machine M, but with arbitrary sequences drawn from the set S.

Proof. Proof proceeds by an easy induction on $\theta_{x,s(x)} \in FL$. □

Definition 6. *Let $\theta_{x,s(x)}$ be a quantifier free formula that only contains the terms given by its indexes. We say $\theta_{x,s(x)}$ is transitive if the relation,*

$$\{(\sigma(n), \sigma(n+1)) | n \in \ \wedge \sigma : \ \rightarrow () \wedge \sigma \models \theta_{x,s(x)}[s^n(0)/x]\}$$

over $()$ is transitive.

Proposition 2. *Let* $M = (I_0, S, R)$ *and* $M' = (I'_0, S, R')$ *be finite state machines with* $S \subseteq ()$ *and such that the following conditions hold;*

$$R(a,b) \text{ implies } \exists M\text{-sequence } \mu,\ i \in \ \cdot \mu(i) = a \wedge \mu(i+1) = b$$
$$R'(a',b') \text{ implies } \exists M'\text{-sequence } \nu,\ j \in \ \cdot \nu(j) = a' \wedge \nu(j+1) = b'$$

for any $a, a', b, b' \in S$[5]. *Then for any transitive quantifier free formulae* $\theta_{x,s(x)}$ *that only contains the terms given by its indexes, we have that;*

$$\models_M \forall x \cdot \theta_{x,s(x)} \text{ and } \models_{M'} \forall x \cdot \theta_{x,s(x)} \text{ implies } \models_{M;M'} \forall x \cdot \theta_{x,s(x)}$$

where $M;M' = (I_0, S, R \circ R')$ *and* $R \circ R'$ *is forward relational composition*[6].

Proof. Proof proceeds by showing that for each $n \in$;

$$\models_M \forall x \cdot \theta_{x,s(x)} \text{ and } \models_{M'} \forall x \cdot \theta_{x,s(x)} \text{ implies } \models_{M;M'} \theta_{x,s(x)}[s^n(0)/x]^7$$

Our required result then follows easily from this. Now let σ be an $M;M'$-sequence and $n \in$ then the following two cases hold;

1. $R(\sigma(n), \tau)$ holds
2. $R'(\tau, \sigma(n+1))$ holds

where $\tau \in S$
By our initial assumptions we have that;

- we have an M-sequence σ' and an $m \in$ such that,

$$\sigma'(m) = \sigma(n) \text{ and } \sigma'(m+1) = \tau$$

- we have an M'-sequence σ'' and an $m' \in$ such that,

$$\sigma''(m') = \tau \text{ and } \sigma''(m'+1) = \sigma(n+1)$$

[5]ie. essentially there are no redundant paths in the finite state machines M and M'.
[6]ie. $R \circ R'(x,y)$ iff $\exists z \cdot R(x,z) \wedge R'(z,y)$
[7]By this we mean that for any $M;M'$-sequence μ,

$$\mu, j \models_{M;M'} \theta_{x,s(x)} \text{ holds}$$
$$\text{for every interpretation } j : TL \longrightarrow \mathsf{N} \text{ such that } j(x) = n$$

Similarly for $\mu \models_{M;M'} \theta_{x,s(x)}[s^n(0)/x]$.

So by the hypothesis,

$$\models_M \forall x \cdot \theta_{x,s(x)}$$

we have that,

$$\sigma' \models_M \theta_{x,s(x)}[s^m(0)/x]$$

holds. By a similar argument we get that;

$$\sigma'' \models_{M'} \theta_{x,s(x)}[s^{m'}(0)/x]$$

holds. Using our previous lemma and the fact that $\theta_{x,s(x)}$ is *transitive* we get our required result. □

Exploiting this intuitive idea directly is further complicated by the theoretical possibility that we may enter an *unsafe* state at the end of one message cycle and then find ourselves in a *safe* state again several message cycles later! And since time has been quantised into major cycles, this behavior would not be immediately observable to us. However, identifying such points of execution in our code might in itself be a useful way of performing a preliminary analysis of where we are likely to evolve into an *unsafe* state.

Within the theoretical framework of this paper, it is possible to model control authorities consisting of more than one SSI. This is possible since we have allowed the possibility that our finite state machines may be nondeterministic. However, such models drastically increase the complexity of our verification procedures, and place a greater degree of emphasis on ensuring that we have good decomposition techniques to help deal with these extra complexity issues.

6 Other Related Work

This paper has focused on the use of finite state machines and their properties to the modeling of a particular computing problem. The approach adopted in this paper is by no means the only one. Indeed there are a number of other perfectly valid ways of modeling the problems addressed in this paper. This section discusses some of these alternative approaches.

6.1 Work Related to the Underlying Logical Theory

As indicated in the body of this paper, the underlying logical theory that we have utilised is that of Monadic Second-Order Arithmetic, equipped with just the constant 0 and the successor function. However, this is not the only way of viewing the underlying logical theory. One could have equally well

have used Linear Propositional Temporal Logic (see eg. [9]) or have concentrated more on the underlying Büchi automata models (see [5]). However, we have avoided both these approaches here, preferring instead to work with the apparently more expressive system of second-order monadic successor arithmetic and to utilise properties of its automata models. This means that the safety properties that we express are thus closer to formulae of the first-order predicate calculus than they might otherwise be. This is of use when we consider that the eventual implementation of our decision procedures shall be used by people who not necessarily familiar with temporal operators or automata theory.

These different viewpoints do have their benefits though. They provide us with resolution techniques for verifying our safety properties (see [10]) and with worst case complexity results concerning our decision procedures (see [7]).

6.2 Work Related to the Modeling of SSIs

[11], [2] and [3] are the main approaches adopted so far to modeling the problems addressed within this paper.

[3] suggests that one should verify the safety of an SSI by using an inductive proof (ie. safety-transitivity), whilst relying on decomposition techniques to help break the overall problem into smaller problems (see [8]). Unfortunately, this work only appears to deal with universal safety properties and there appears to be a need to verify stronger safety properties than might otherwise be necessary.

[11] suggests using CCS to model our SSIs with the concurrency workbench providing the tool by which we may proof check that a given safety argument is indeed valid.

[2] uses HOL to code-up a program logic for an SSI geographic data language with the intention of using HOL tactics to try and automate some of the proof search strategy. This appears to be a much more promising approach than that taken in [11]. HOLs tactic language should hopefully help to reduce the need for significant prompts for help from a user.

Both [11] and [2] suffer from the lack of any formal model of the IFP. [11], [2] and [3] all concentrate more on the *modes* by which one may verify certain types of safety property.

References

[1] Cribbens, A., H. (1987) Solid State Interlocking (SSI): an intergrated electronic signaling system for mainline railways. *IEE Proceedings, 134(3) pp.148-158, May 1987*

[2] Morley, M., J. (1993) Safety in Railway Signaling Data: A Behavioral Analysis. *Presented at the HOL User Group Meeting '93*

[3] Ingleby, M. and Mitchell, I. (1992) Proving Safety of a Railway Signaling System Incorporating Geographic Data. *Proceedings SAFECOMP '92, 28^{th} – 30^{th} October 1992, Zurich, Switzerland*

[4] Barnard, B. (1993) Tool Support for an Application-Specific Language. *Safety-Critical Systems Club Conference, Feburary 1993*

[5] Büchi, J., R. (1960) On a Decision Method in Restricted Second-Order Arithmetic. *Logic Methodology and Philosophy of Science, Proceedings of 1960 Stanford International Congress, Stanford 1962, pp. 1-11*

[6] Siefkes, D. (1970) Büchi's Monadic Second Order Successor Arithmetic. *Lecture Notes in Mathematics 120, Springer-Verlag*

[7] Sistla, A., P. and Clarke, E., M. (1985) The Complexity of Propositional Linear Temporal Logics. *Journal of the Association for Computing Machinery, Vol. 32, July 1985, pp.733-749*

[8] Ingleby, M. (1993) A Galois Theory of Local Reasoning in Control Systems with Compositionality. *See this proceedings.*

[9] Gough, G., D. (1984) Decision Procedures for Temporal Logic. *Phd Dissertation, Department of Computer Science, University of Manchester, October 1984*

[10] Fisher, M. (1992) A Normal Form for First-Order Temporal Formulae. *Proceedings of 11^{th} International Conference on Automated Deduction, Saratoga Springs, New York, June 1992, Lecture notes in Computer Science Vol. 607*

[11] Morley, M., J. (1991) Modeling British Rail's Interlocking Logic: Geographic Data Correctness. *LFCS Report ECS-LFCS-91-186, University of Edingburgh, November 1991*