

Automatic Verification of Industrial Designs¹

V. Hartonas-Garmhausen

Union Switch & Signal and Engineering & Public Policy, Carnegie Mellon University

T. Kurfess

George W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology

E. M. Clarke

School of Computer Science, Carnegie Mellon University

D. Long

AT&T Bell Labs, Murray Hill

Abstract

This study presents new risk analysis tools and demonstrates the feasibility and applicability of these tools in the design verification of railway interlocking control systems and deadlock prevention in automated manufacturing systems. Our verification methodology consists of the following stages. First, we analyze the rules executed by the controller and extract a state machine model of the controller. Second, we compose safety, reliability, and operability system specifications using a propositional temporal logic. Finally, we use the model checker to check the state machine model of the system against its requirements. The verification approach allows an exhaustive search of all possible behaviors and scenarios. We verified two real railway interlocking control applications with 125 and 452 constraints respectively. Checking two opposing signal protection specifications involving 3 signals ranged between 74 and 1223 seconds depending on the size and the complexity of the interlocking. Traditional verification methods typically require several person weeks.

1 Introduction

This study investigates whether a formal methods approach to system specification and verification is feasible for real

industrial control systems. Verification in product development is the process by which we answer the question: "Are we building the product correctly?" The chapter on formal methods in the FAA Digital Systems Validation Handbook defines verification as the process of determining whether each level of specification, and the final design itself, fully and exclusively implement the requirements of its superior specification [1]. Formal methods are mathematical techniques that have been used in the specification and verification of computer systems. They have been developed primarily by mathematicians and computer scientists. Some of the methods have seen a tremendous growth and maturity within the last five to six years. The question is whether these methods can be effectively applied to real industrial designs.

This study was motivated by several factors. There are complex systems where reliability and the verification of reliability are vital to their development and operation. Public safety and high social and economic stakes depend on the safe and reliable operation of automated control systems in medical machines, air traffic control, railway signalling, controllers in nuclear plants, computer-controlled financial services, and so on. The systems used in our case studies are highly capital intensive for the respective industries, must be competitive in world markets, require effective design and operational strategies, and must meet strict safety, reliability, and

¹ This research is sponsored in part by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under Grant F33615-93-1-1330, and in part by the National Science Foundation under Grant No. CCR-9217549 and in part by the Semiconductor Research Corporation under Contract 92-DJ-294. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed, or implied, of the U.S. government.

performance specifications. Traditional verification methods, such as Hazard and Operability Analysis, Fault Tree Analysis, simulation, prototype testing, and manual inspection are limited because of the large combinatorial search problems caused by the complexity of the systems and the large number of the possible states.

To investigate the usefulness of formal methods, we performed case studies in two industries: railway signalling and manufacturing. The systems that are verified are solid state railway interlocking control systems and automated manufacturing systems. These systems belong in the domain of Discrete Event Dynamic Systems (DEDS) in which the evolution of the system in time depends on the complex interactions of the timing of various discrete events, such as the arrival of a train, or the failure of a switch or a lamp in the case of the interlocking control, or the arrival of a part, departure of a finished part, or failure of a machine in the case of an automated manufacturing system (AMS). The formal methods approach selected is called Symbolic Model Checking [5] and was developed at the Computer Science Department of Carnegie Mellon University.

The paper is organized as follows. Section 2 presents the formal logic and the model checking verification methodology. Sections 3 and 4 describe respectively the railway interlocking verification problem and the manufacturing control system design problem, related verification methods and tradeoffs, and use of model checking with its strengths and limitations.

2: Verification Methodology

The methodology that we use for verifying state machines is known as temporal logic model checking [5]. Temporal logic [8] is a logic for expressing the ordering of events in time without introducing time explicitly. It contains operators for specifying properties such as “condition p will eventually hold” and “conditions p and q will never hold simultaneously.” Temporal logic is a form of modal logic. The origins of modal logic can be found in Aristotle and later medieval logicians who conceived many modes of truth, such as necessary truths, things that ought to be true, things that we know are true, or things that we believe are true. Modal logics also include theorems of propositional calculus and were embellished with necessary operators to achieve a greater degree of expressiveness.

The particular temporal logic that we use is called CTL (Computation Tree Logic). Formulas in CTL are built from three components: atomic propositions, boolean connectives, and temporal operators. In our application, atomic propositions are used to refer to the values of the variables used in the controller program. The boolean

connectives are the standard ones, such as AND, OR, XOR, and NOT. Each temporal operator consists of two parts: a path quantifier (A or E) and a temporal modality (F, G, or X). All of the operators are interpreted relative to an implicit “current state.” There are in general many execution paths (sequences of state transitions) of the controller starting at this current state, both because the environment may supply arbitrary inputs, and because we may have nondeterministic state transitions due to the order of evaluation effects in the rule set defining the controller. The path quantifier indicates whether the modality denotes a property that should be true of all of those possible paths (the universal path quantifier A) or whether the property need only hold on some path (the existential path quantifier E). The modalities describe the ordering of events in time along an execution path and have the following intuitive meanings:

1. $F \phi$ (ϕ “holds sometime in the future”) is true of a path if there exists a state on the path for which a formula ϕ is true.
2. $G \phi$ (ϕ “holds globally”) means that ϕ is true at every state in the path.
3. $X \phi$ (ϕ “holds in the next state”) means that ϕ is true in the second state on the path (the state reached immediately after the current state).

Figure 1 displays graphically some basic CTL expressions. A black circle indicates that a specification ϕ is TRUE in the corresponding state. A white circle indicates the reverse.

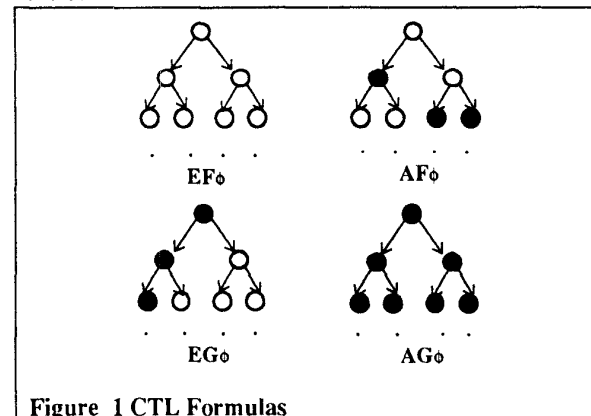


Figure 1 CTL Formulas

In our controllers, a state will consist of a valuation for the controller variables, plus a valuation for the inputs. Each formula of the logic is either true or false in a given state. An atomic proposition x corresponding to the controller variable x is true in a state if x has the value 1 in that state, and is false if x has the value 0. The truth of a formula built from boolean connectives depends on the truth of its subformulas in the usual way. A formula whose

top level operator is a temporal operator with a universal (existential) path quantifier is true whenever all paths (some path) starting at the state have the property required by the operator's modality. A formula is satisfied by a controller if it is true for all the initial states of the controller.

The objective of temporal model checking is essentially to check whether a state transition system corresponding to a system description satisfies a desired specification expressed in temporal logic. The desired system specifications such as safety, reliability, operability, and performance are compiled from industrial quality standard checklists, field test checklists, process design specifications, hazard analysis, and other sources. The state transition system that describes the system architecture is a triple $\langle S, I, R \rangle$ where S is the set of all possible states, I is the set of initial states of the system, and R is a binary relation on S which defines the possible transitions between states.

Given a controller and a specification in temporal logic, an existing temporal logic model checker called SMV (Symbolic Model Verifier) [7] is used to determine whether the controller satisfies the specification [4], [5], [6], [10]. In those cases where the specification is false, the model checker will produce a counterexample execution path that shows why it is false. The counterexample is very useful in debugging the controller. SMV uses a symbolic representation for state machines based on Ordered Binary Decision Diagrams (OBDDs) [2] which makes it possible to check finite state concurrent systems with more than 10^{120} states [3].

3: Railway Interlocking Control

3.1: Problem Formulation

A Railway Interlocking is an arrangement of signals and signal appliances interconnected so that their movements must succeed each other in proper sequence and for which interlocking rules are in effect. It may be operated manually or automatically².

A wayside interlocking controller performs the following main functions:

- receives and stores physical and serial inputs,
- performs logical and timing operations on the inputs and the internal state variables,
- produces physical and serial outputs,
- communicates with other controller units,
- accepts operating requests from the Central Office,

- constantly runs multiple diagnostics to detect hardware errors to respond to the errors in timely fashion.

A simple model of a turnout interlocking control (Figure 2) is used to illustrate the automatic model checking approach.

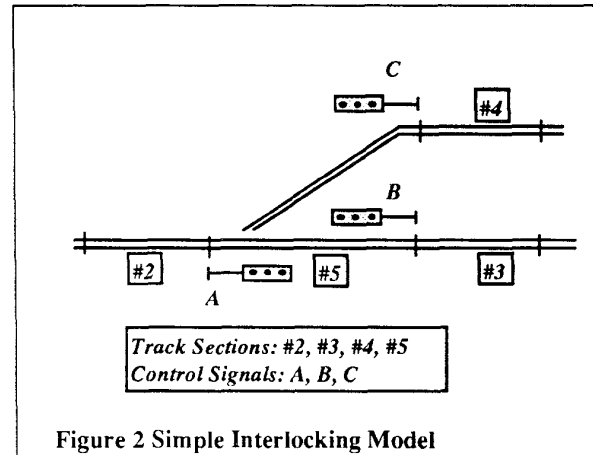


Figure 2 Simple Interlocking Model

Signals A, B, C are controlled by several factors such as condition of rails, presence and proximity of trains, route requests, the operating state of signals and switches, etc. We have designed this interlocking using simple rules and the minimum number of inputs and outputs required to achieve a simple and clear demonstration of the automatic verification approach. In real interlocking design, several additional types of locking are present such as approach locking, detector locking, route locking, switch locking, traffic locking, etc.

The design of an interlocking is required to meet high safety and efficiency requirements. Interlocking control safeguards against conflicting movements of trains throughout the system. The interlock is controlled based on the following rules:

- In the absence of trains, signals remain red.
- Track switch operation and correspondence precedes signal operation.
- An alternate route is established when the requested route is busy.
- Broken tracks are treated as occupied tracks.
- An alarm is activated when a switch is detected broken.

Table 1 and Table 2 define the inputs and outputs of the simple interlocking controller. The 'normal' position, which appears in the definitions of the system inputs and outputs, is the position in which signal and other devices are assumed to lie normally according to rule or convention. In Figure 2 the switch must lie in the

² As defined by the Association of American Railroads

'normal' position to allow the traffic from track block 2 to block 3 and from block 3 to block 2. The 'reverse' position allows traffic from track block 2 to 4 and from block 4 to 2.

Table 1 Inputs

2T	0	No train in track block 2
	1	Track block 2 is occupied by a train or track is broken in this section
3T	0	No train in track block 3
	1	Track block 3 is occupied by a train or track is broken in this section
4T	0	No train in track block 4
	1	Track block 4 is occupied by a train or track is broken in this section
5T	0	No train in track block 5
	1	Track block 5 is occupied by a train or track is broken in this section
L_2RC	0	Request to route train from track block 2 to track block 3
	1	Request to route train from track block 2 to track block 4
IE F	00	Broken/Inoperative Switch
	01	Normal
	10	Reverse
	11	In-transit

Table 2 Outputs

V W	00	Do not operate (Keep previous state)
	01	Operate 'Normal'
	10	Operate 'Reverse'
	11	Activate Alarm
SignalA	0	Operate RED
	1	Operate GREEN
SignalB	0	Operate RED
	1	Operate GREEN
SignalC	0	Operate RED
	1	Operate GREEN

3.2: Traditional Verification Methods

The objective of verification is to prove that (1) the operation of each subsystem and the system as a whole complies with the design and (2) the system as designed performs in accordance with the specifications. The

subsystems that are verified are route operation, switch operation, approach locking, time locking, opposing signal protection, and overrun protection.

The traditional verification methods include extensive inspection of the software by the programmer and other engineers, lab tests, and ultimately field tests. The basic tools include a custom-made simulator to test for logical errors and an equivalent circuit generator to produce and verify an equivalent relay circuit representation. These methods have been practiced in the railway signalling industry and the railroads for many years. Some of the problems are: normal interlocking control applications are too large to test all possible scenarios, verification is costly due to the time and resources needed, manual inspection is error-prone since "no human being can operate in a fail-safe fashion" [13].

3.3: Model Checking Verification

The verification approach we will describe next can complement the above methods offering a greater degree of confidence, automation, and efficiency.

The automatic verification of solid-state interlocking controller logic consists of the following steps. First, a translator compiles the application logic into the verifier's specification language. Second, an analyzer extracts the input-output behavior of the set of rules that are executed by the controller in the form of a state machine. The analyzer also checks for non-stabilization and non-determinism. Third, a model checker determines whether the controller meets a set of specifications expressed in CTL.

The basic operation cycle of an interlocking controller is:

- collect inputs from the environment,
- repeatedly execute the rules (in some order) until stability is achieved, and
- output the final value of externally-visible variables.

Our ultimate goal is to extract the input-to-output behavior of the set of rules in the form of a state machine. During this process, we also check some basic consistency conditions for the rules. While the controller has a deterministic scheduling policy for determining the order in which to execute the rules, the exact details of this process are somewhat complex. As a result, the design engineers generally do not use these details when deriving the rules. In fact, we would like the rules to be such that the order of execution does not matter. This means that

- the order should not affect whether stability is achieved, and
- the order should not affect the final values of the outputs.

If either of these conditions are violated, the analyzer displays diagnostic traces to illustrate the problem.

The interlocking controller, used in our example, must meet the following design specifications to ensure safe interlocking control.

1. There is no state where both signals A and B are green simultaneously (opposing signal protection).
2. There is no state where both signals A and C are green simultaneously (opposing signal protection).
3. For all states where the switch is broken the next action will be to activate the alarm.
4. It is always true that the alarm will never be activated unless the switch is broken.
5. It is always true that the absence of a train in section 2 will turn signal A red.
6. It is always true that the absence of a train in section 3 will turn signal B red.
7. It is always true that the absence of a train in section 4 will turn signal C red.
8. For all states where blocks 2 and 3 are occupied, the next state for the switch is reverse or broken.
9. It is always the case that when signal A is green then the switch will be normal, reverse, or broken.
10. Once the controller gives a green light to the train in section 2, the switch should not move until the train leaves section 2.

These specifications are expressed in the CTL Temporal Logic as follows:

```

SPEC
AG!(SignalA=1 & SignalB=1)
SPEC
AG!(SignalA=1 & SignalC=1)
SPEC
AG(((IE=0) & (F=0)) → AX ((V=1) & (W=1)))
SPEC
AG(!((IE=0) & (F=0)) → AX!((V=1) & (W=1)))
SPEC
AG (2T=0 → AX SignalA=0)
SPEC
AG (3T=0 → AX SignalB=0)
SPEC
AG (4T=0 → AX SignalC=0)
SPEC
AG (2T=1 & 3T=1 → AX (SignalA=1 → (IE=1 & F=0) |
(IE=0 & F=0)))
SPEC
AG (SignalA=1 → ((IE=0 & F=1) | (IE=0 & F=0) | (IE=1 &
F=0)))
SPEC
AG (SignalA=1 & IE=0 & F=1 → !E[5T=1 U ((IE=1 &
F=1)|(IE=1 & F=0))])

```

The model checker found the first 9 out of 10 specifications to be TRUE and generated a counterexample (Figure 3) to demonstrate a scenario where the last specification is FALSE:

while the train in track block 2 is crossing the interlocking in the normal direction to block 5 and then block 3, a train appears in block 4 and the controller gives the command to operate the switch to the reverse position before the first train has left block 5.

Such a scenario is undesirable because it can lead to train derailment and/or train crash. Approach and detector lockings must be designed in the controller operation to prevent such switch movement.

```

--specification AG (SignalA=1 & IE=0 & F=1 →... is false
--as demonstrated by the following execution sequence
state 1.1:
2T=0
3T=0
4T=0
5T=0
L_2RC=0
IE=0
F=1
V=0
W=0
SignalA=0
SignalB=0
SignalC=0

state 1.2:
2T=1

state 1.3:
2T=0
5T=1
SignalA=1

state 1.4:
4T=1
SignalA=0

state 1.5:
4T=0
V=1

state 1.6:
5T=0
IE=1
F=0
V=0

```

Figure 3 Counterexample

An exhaustive verification of our simple turnout model via the traditional methods, i.e. manual inspection and simulation is possible because of the relatively small state space (10^7) of the system. The ten specifications listed above were tested by SMV within 0.3 seconds. We used the above simple example with the sole purpose of

demonstrating the model checking verification approach in a clear manner.

The model checking verification method becomes more significant in the verification of real interlocking controllers where the size of the state space is so large that traditional methods can never fully cover it. SMV was used to verify specifications in two real interlocking applications: the first one had 44 inputs, 114 outputs, and 125 equations, the second had 132 inputs, 203 outputs, and 452 equations. Verification times (running SMV on a RISC 6000 workstation) ranged from seconds to hours.

The computational complexity is a function not only of the number of state variables but also of the complexity of the relationships among the state variables. The complexity of the interlocking rules depends on the type and size of the interlocking, types of locking subsystems implemented, number of neighboring track circuits and interlocking controllers. In the moderate case with 125 constraints, which is the typical size application implemented in the field, SMV checked two opposing signal protection specifications involving 3 signals within 74 seconds. In the very large application with 452 equations, similar specifications required 1222.23 seconds. Manual inspection or use of simulated scenarios (in the factory and the field) typically take several person weeks. Most importantly, none of the traditional methods can check the above specifications exhaustively and as reliably.

4: Automated Manufacturing Systems

4.1: Problem Formulation

An Automated Manufacturing System (AMS) is often a complex interconnection of various subsystems such as computer controlled machining centers, assembly stations, inspection systems, and material handling devices such as automated guided vehicles, robots, and conveyors [25]. The design requirements of an AMS are: flexibility to survive several product lines, fault-tolerance to sustain desired levels of production in the presence of possible subsystem failures, and resilience to design, demand, and product mix changes. It is important to note that the construction is so capital intensive that a manufacturing system will not be constructed unless the design analysis shows that the system is financially viable. Hence, performance and reliability verification during the design phase are vital to the actual construction of a manufacturing system.

Here we introduce a new approach using a temporal logic model checking verification system to the design of AMSs with no deadlocks. A deadlock is a situation where each of a set of two or more parts keeps waiting

indefinitely for the other parts in the set to release resources such as machines, tools, automated guided vehicles, etc. Deadlocks are undesirable because they involve high labor costs for clearing the deadlock. Also resetting the system leads to degraded performance and ultimately zero throughput. Deadlocks are difficult to predict because they may happen after a complex sequence of operations on parts passing through the system concurrently.

In the next section we discuss related verification methods used in manufacturing.

4.2: Related Verification Methods

The methods used in the design of manufacturing systems include simulation, analysis (Markov Chains, Queuing Theory, Petri-Nets), operations research, and artificial intelligence (expert systems and neural nets).

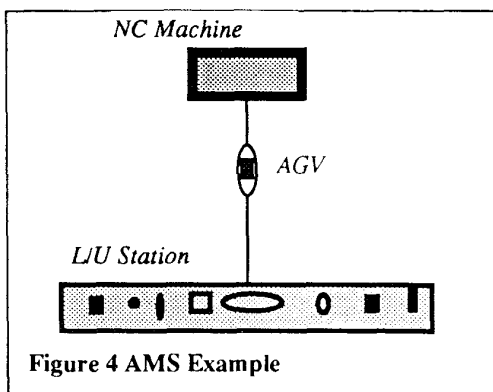
Simulation involves the development and validation of a model, writing and debugging a computer program, creating the inputs, and conducting enough tests to produce statistically significant performance measures. There are several simulation languages such as SIMAN, SIMFACTORY, and CINEMA that are specific to manufacturing. Simplicity, modelling, and computational power, and availability of languages, tools, and manuals (almost every factory floor has at least a custom-made simulator) make simulation a very effective technique. There are three problems in simulation-based design verification testing: how does one generate the input stimuli, how does one know that the results are correct, and how good are the applied input stimuli i.e. how complete is the testing experiment [23]. Some limitations of simulation are the high computational cost when greater accuracy requires more detailed models and a large number of tests, the very large state-space which prevents an exhaustive verification, and design errors that may be missed if they occur after a complex sequence of input changes.

Research and development on manufacturing system analytical models started in the 1970s and grew significantly in the 1980s. Analytical Methods involve the development and validation of a model which may be solved in closed form or via numerical analysis. Accuracy depends on the formulation of the model. Petri Nets [24] can be very powerful as performance modeling tools because of their ability to model concurrency, sequential execution, synchronization, conflict, and randomness. Besides the representational power of the analytical methods, other advantages they possess are the ability to provide quick feedback in the evaluation of design alternatives as well as the abundance of software packages for solving analytical models. CAN-Q, MVAQ, and

MANUPLAN are a few example software packages specific to manufacturing. On the other hand, the limitations or difficulties of the analytical methods are: the state explosion problem, the high computational cost, numerical stiffness, numerical ill-conditioning, and numerical instabilities.

4.3: Model Checking Verification

We use the example (Figure 4) from the manufacturing literature [25] to illustrate the formal verification approach.



The example consists of a Load/Unload (L/U) station with raw and finished parts, a numerically controlled (NC) machine which processes the parts, and an Automated Guided Vehicle (AGV) which carries raw parts from the L/U station to the NC machine and finished parts from the NC machine to the L/U station. We assume that there are always parts available in the L/U station. The AGV can only carry one part at a time and the NC machine can only process one part at a time. It takes a certain amount of time for the AGV to transport a part from the L/U station to the NC machine or from the NC machine to the L/U station (the transit time is less when the AGV does not carry anything). The objective is to design the system and its operational policies such that deadlocks will never occur. We tested the following three specifications:

SPEC

AG!(agv.state=AGV_available_for_finished_part & ncm.state=ncm_available)

SPEC

AG!(agv.state=AGV_with_raw_part_waiting_for_NCM & ncm.state=NCM_waiting_for_AGV)

SPEC

AG!(agv.state=AGV_available_for_finished_part → **AX!**(cmd=assign_raw_part))

The first specification reads “it is never the case that simultaneously the AGV is available to carry a finished part and the NC machine is available waiting for a raw part.” The second specification reads “it is never the case that the NC machine is waiting for the AGV after processing a part while the AGV is waiting for the NC machine carrying a raw part.” The third specification reads “it is always true that when the AGV is available to carry a finished part it is not assigned to a raw part.”

The model checker found the first two specifications to be false and generated the counterexamples listed in Figure 5 and Figure 6. The third specification was satisfied.

The first deadlock occurs when the AGV is assigned a finished part while the NC machine is waiting for a raw part. The second deadlock occurs when the following scenario takes place: the AGV is assigned to a raw part, the AGV carries the raw part to the NC machine and is released when it finds the machine free. While the machine is processing the part, the AGV returns to the L/U station and loads another raw part which it starts delivering to the NC machine. A deadlock occurs because the AGV is waiting for the machine to be freed so that it drops off the raw part but the machine can not be freed because it is waiting for the AGV to pick up the finished part.

The first deadlock can be prevented if we always assign the AGV a raw part when no finished part is waiting. There are several resource allocation policies that will prevent the second deadlock. The first policy dictates that the AGV will not be released until the machine finishes processing and the AGV unloads the finished part. The second policy is to assign the AGV to a finished part when a finished part is waiting.

```

--specification AG!(agv.state=AGV_available_for_fini
--in module c) is false
--as demonstrated by the following execution

state 1.1:
raw_material=1
timer.c11=0
timer.count=c11
c.agv.state=agv_available
c.cmd.state=ncm_available
c.cmd=do_not_operate

state 1.2:
c.agv.state=AGV_available_for_finished_part
c.cmd=assign_finished_part

```

Figure 5 First Deadlock

```

--specification AG(!(agv.state=...(in module c) is false
--as demonstrated by the following execution sequence

state 2.1:
raw_material=1
timer.count=c11
c.agv.state=agv_available
c.ncm.state=ncm_available
c.cmd=do_not_operate

state 2.2:
c.agv.state=AGV_available_for_raw_part
c.cmd=assign_raw_part

state 2.3:
timer.count=c4
c.agv.state=AGV_carries_raw_part_to_NCM
c.cmd=transport_raw_part

state 2.4:
timer.count=c11
c.agv.state=AGV_with_raw_part_waiting_for_NCM
c.cmd=carry_raw_part_to_machine

state 2.5
c.agv.state=agv_available
c.ncm.state=NCM_processing_AGV_release
c.cmd=release_finding_machine_free

state 2.6:
c.agv.state=AGV_available_for_raw_part
c.cmd=assign_raw_part

state 2.7:
timer.count=c4
c.agv.state=AGV_carries_raw_part_to_NCM

state 2.8:
timer.count=c7
c.agv.state=AGV_carries_raw_part_waiting_for_NCM
c.cmd=carry_raw_part_to_machine

state 2.9:
timer.count=c11
c.ncm.state=NCM_waiting_for_AGV
c.cmd=release_machine_processing_part

```

Figure 6 Second Deadlock

Next we make the necessary design changes to the AMS controller to prevent the deadlocks identified by the model checker. The result is a deadlock-free AMS (Figure 7).

```

--specification AG(!(agv.state=AGV_available... is true
--specification AG(!(agv.state=AGV_with_raw... is true
--specification AG(!(agv.state=AGV_available... is true

resources used:
user time: 0.483333s, system time: 0.0666667s
BDD nodes allocated: 5753
Bytes allocated: 917504
BDD nodes representing transition relation: 1018+1

```

Figure 7 SMV output for deadlock-free AMS

5: Conclusions and Future Work

This study presents new risk analysis tools and demonstrates the feasibility and applicability of these tools in the verification of industrial control designs. We demonstrated the automatic model checking verification approach using case studies in two industries, the railway signalling industry and the manufacturing industry. Hence, we showed that industries other than the computer industry can benefit from recent developments in formal verification. The systems selected for this study are real industrial systems. For the purposes of this paper we simplified and reduced our examples in size to illustrate the methodology in an efficient manner. This demonstrates that formal verification techniques are not limited to ‘toy problems’.

Logic verification of railway interlocking control systems and deadlock prevention in automated manufacturing systems via model checking have several advantages. The approach presented in this paper allows an exhaustive search of all possible behaviors and scenarios, is fast, automatic, powerful, and allows the designer to learn the system better via the generation of counterexamples and experiment with alternate designs before a prototype is built.

The methodology presented in this paper is limited to discrete event systems. No automatic formal verification tools are currently available for continuous systems. However, research of formal specification and verification of hybrid systems is currently a very active area.

The method we presented depends on model accuracy, exhaustiveness, and correctness of the specifications being verified. Furthermore, the computational requirements depend on the complexity of the specific system. Both the size of the state space and the complexity of the system architecture dictate the practical feasibility of the proposed methodology. In the worst case, the computation complexity is exponential to the size of the system.

The case studies we performed demonstrate the feasibility and power of model checking and formal verification in real industrial systems. There are great benefits in using these techniques when applicable. Changes in the industry [20] provide additional motivation. For example, the railroad and transit industries now demand that safety characteristics of new systems be numerically or mathematically provable. Performing the safety analysis and verification concurrently with the system design allows greater efficiency in the development process since several design errors can be detected prior to the traditional test phase. Furthermore, due to its complexity, a complete design in the traditional development cycle cannot be exhaustively verified even with sophisticated state-of-the-art techniques.

The transfer of the methodology presented in the study from research to product development is not trivial. We propose to develop methodologies of how and when formal verification can be most effectively incorporated in the engineering design process. The ultimate goal is to produce a correct method of combining the design and verification phases.

Important issues which require further work are identifying the necessary level of training of design engineers in formal methods, state transition systems, temporal logic, and model checking to produce correct system model and design specifications, and designing user-friendly tool interfaces which offer

- the necessary libraries of useful models (timers, counters, etc.),
- a high level specification language,
- strategies for implementing techniques such as modular system description and abstraction which may be required to handle large systems,
- techniques for analyzing the computational complexity - it is important to be able to reasonably predict the numerical complexity and required run time at the start of the verification process - and incorporating counterexamples in revising and ultimately producing a correct design.

Acknowledgment: Part of this work was funded by the National Science Foundation under the Engineering Design Research Center, an NSF Engineering Research Center. The government has certain rights on this material. Any opinions, findings, and conclusions or recommendations are those of the authors and do not necessarily reflect the views of the National Science Foundation. The railway application was developed at Union Switch & Signal which has the rights to this work.

6: References

1. J. Rushby. Formal Methods and Digital Systems Validation for Airborne Systems. SRI Technical Report. April 1993.
2. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.* C-35(8), 1986.
3. E. M. Clarke, O. Grumberg, D. Long. Verification Tools for Finite-State Concurrent Systems. 1993.
4. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142-170, June 1992.
5. E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Logic of Programs: Workshop*, Yorktown Heights, NY, May 1981, volume 131 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.
6. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244-263, 1986.
7. K. L. McMillan. Symbolic Model Checking: An Approach to the State Explosion Problem. Ph.D. Thesis, Carnegie Mellon University, 1992.
8. Z. Manna, A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems - Specification*. Springer Verlag, 1992.
9. A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45-60, 1981.
10. J. P. Quielle and J. Sifakis. Specification and verification of concurrent programs in CESAR. In *Proceedings of the Fifth International Symposium in Programming*, 1981.
11. Hopcroft-Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley.
12. D. P. Wood, W. G. Wood. Comparative Evaluations of Four Specification Methods for Real Time Systems. Technical Report CMU/SEI-89-TR-36, December 1989.
13. M. Brignano, H. McCullough. THE SEARCH FOR SAFETY: A history of Railroad Signals and the People Who Made Them. Pittsburgh PA, May 1981.
14. G. M. Kichenside, A. Williams. *British Railway Signalling*. Second Edition 1968.
15. Association of American Railroads. SIGNAL MANUAL of Recommended Practice, 1992.
16. J. R. Egnot, D. R. Disk. The Application of Microprocessors to Interlocking Logic. Union Switch & Signal.
17. R. J. Mathews. Microprocessor Control of Small Interlockings: The Next Step. Union Switch & Signal Technical Paper, Bulletin 436.
18. Union Switch & Signal SERVICE MANUAL 6400A. Vital Application Logic Programming - MICROLOK Vital Interlocking Control System - MICROLOK PLUS Vital + Not Vital Control Package, October 1991.
19. Union Switch & Signal SERVICE MANUAL 6400B. Hardware Installation, Power and Data Interfacing - MICROLOK Vital Interlocking Control System - MICROLOK PLUS Vital + Not Vital Control Package, October 1991.
20. W. C. Vantuono. Transit Signalling: The Next Generation. *Railway Age*, October 1993.
21. W. Keats, D. H. Newing. The Use of Microprocessors in Automatic Train Operation. *IEEE Conference Proceedings*, September 1978.
22. R. S. Sandige. *Modern Digital Design*. McGraw Hill.
23. Abramovici, Breuer, Friedman. *Digital Systems - Testing and Testable Design*. Computer Science Press.
24. J. Peterson. *Petri Net Theory and the Modeling of Systems*.
25. N. Viswanadham, Y. Narahari. *Performance Modeling of Automated Manufacturing Systems*. PRENTICE HALL, 1992.
26. G. Chryssolouris. *Manufacturing Systems Theory and Practice*. Springer-Verlag, 1992.