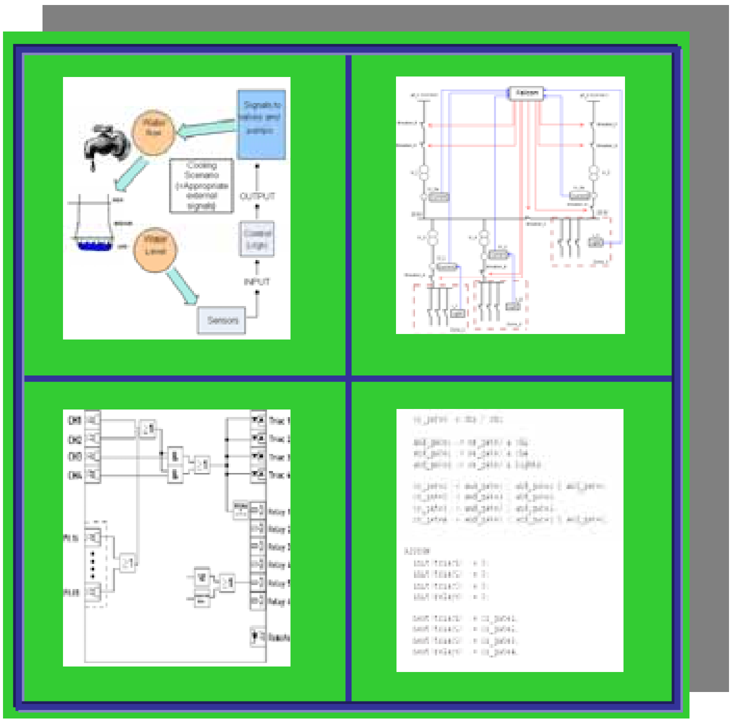
ESPOO 2008

VTT WORKING PAPERS 94



NPP Safety Automation Systems Analysis

State of the Art

Janne Valkonen & Ilkka Karanta

VTT Technical Research Centre of Finland

Matti Koskimies, Keijo Heljanko & Ilkka Niemelä

Helsinki University of Technology TKK

Dan Sheridan & Robin E. Bloomfield

Adelard LLP

ISBN 978-951-38-7158-1 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 3, PL 1000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 4374

VTT, Bergsmansvägen 3, PB 1000, 02044 VTT
tel. växel 020 722 111, fax 020 722 4374

VTT Technical Research Centre of Finland,
Vuorimiehentie 5, P.O. Box 1000, FI-02044 VTT, Finland
phone internat. +358 20 722 111, fax +358 20 722 4374

VTT, Vuorimiehentie 3, PL 1000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 6027

VTT, Bergsmansvägen 3, PB 1000, 02044 VTT
tel. växel 020 722 111, fax 020 722 6027

VTT Technical Research Centre of Finland,
Vuorimiehentie 5, P.O. Box 1000, FI-02044 VTT, Finland
phone internat. +358 20 722 111, fax +358 20 722 6027

Series title, number and report code of publication

VTT Working Papers 94
VTT–WORK–94

Author(s) Valkonen, Janne, Karanta, Ilkka, Koskimies, Matti, Heljanko, Keijo, Niemelä, Ilkka, Sheridan, Dan & Bloomfield, Robin E.		
Title **NPP Safety Automation Systems Analysis** **State of the Art**		
Abstract This report describes the state of the art of formal methods and models applied in safety evaluation of nuclear and other industrial safety systems. Special attention is drawn to a technique called model checking that is a set of methods for analysing whether a model of a system fulfils its specifications by examining all of its possible behaviours. The report describes the scope and requirements for safety evaluation and introduces typical safety assessment approaches. The Safety Case concept is also described and discussed how it could be combined with model checking.		
ISBN 978-951-38-7158-1 (URL: http://www.vtt.fi/publications/index.jsp)		
Series title and ISSN VTT Working Papers 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)		Project number 23743
Date April 2008	Language English	Pages 62 p.
Name of project MODSAFE	Commissioned by	
Keywords model checking, safety automation, safety case, formal verification, programmable logic controller, SAFIR2010	Publisher VTT Technical Research Centre of Finland P.O. Box 1000, FI-02044 VTT, Finland Phone internat. +358 20 722 4520 Fax +358 20 722 4374	

Preface

This report has been prepared under the research project Model-based safety evaluation of automation systems (MODSAFE) which is part of the Finnish Research Programme on Nuclear Power Plant Safety 2007–2010 (SAFIR2010). The aims of the project are to develop methods for model-based safety evaluation, apply the methods in realistic case studies, evaluate the suitability of formal model checking methods for NPP automation analysis, and to develop recommendations for the practical application of the methods.

The report describes the state of the art of formal methods and models applied in safety evaluation of industrial and nuclear safety systems. The methods are evaluated with respect to their applicability in the analysis of NPP safety automation systems.

Espoo, March 2008,

Authors

Contents

List of Acronyms

ALARP As Low As Reasonably Practicable

ASCAD Adelard Safety Case Development Manual

BMC Bounded Model Checking

CAE Claims-Argument-Evidence

COTS Commercial Off-The-Shelf

DCS Distributed Control System

FBD Function Block Diagram

FMEA Failure Mode and Effects Analysis

GSN Goal-Structuring Notation

HAZOP Hazard and Operability Analysis

HMI Human-Machine Interface

I&C Instrumentation and Control

IFE Institute for Energy Technology (Norway)

IL Instruction List

IRSN Institute for Radiological Protection and Nuclear Safety (France)

ISTec Institute for Safety Technology (Germany)

LD Ladder Diagram

MTTF Mean Time To Failure

NPP Nuclear Power Plant

(O)BDD (Ordered) Binary Decision Diagram

OTS Off-The-Shelf

PLC Programmable Logic Controller

PSA Probabilistic Safety Analysis

QA QA Technology Company, Inc.

QAC C compiler by QA company

RETRANS REverse TRAnsformation of Normed Source code

SAT	Propositional Satisfiability
SCA	Software Criticality Analysis
SFC	Sequential Function Chart
SMV	Simple Model Verifier
ST	Structured Text
STUK	The Finnish Radiation and Nuclear Safety Authority
VTT	Technical Research Centre of Finland

1 Introduction

1.1 Scope of the Report

This report is part of yearly reporting in the MODSAFE (Model-based safety evaluation of automation systems) project which belongs to the Finnish National Research Programme on NPP Safety 2007–2010. The report describes the state of the art of formal methods and models applied in safety evaluation of industrial and nuclear safety systems. The methods are evaluated with respect to their applicability in the analysis of NPP safety automation systems. Special attention is drawn to a technique called model checking [40], which is a set of methods for analysing whether a model of a system fulfils its specification by examining all of its possible behaviours. The compatibility of the methods with the practices and models used in different phases of NPP safety automation design is also an important issue.

The basic objective of licensing safety critical systems is to assess if they are adequately safe and if not, what are the reasons for that and what can be done to achieve the required level of safety. Licencing process includes collecting evidence supporting the safety claims of the system under consideration, and based on this evidence assessing the achieved safety of the system. That is why safety cases and the role of model checking in the NPP automation safety case creation are discussed in the report. The report aims at giving an introduction to the research area with a large number of references to be used in more detailed investigation of the topics introduced here.

1.2 NPP Automation Systems

Instrumentation and Control (I&C) systems play a crucial role in the operation of a nuclear power plant. The most important tasks of I&C systems are to control and supervise processes inside the power plant and the interfaces with the operators. The control and supervising tasks are performed with human interactions or automatically according to predetermined rules.

The main objectives of I&C systems in nuclear power plants are to ensure safety, availability and performance of the plant. The following definitions are based mostly on [65]. Nuclear safety means the achievement of proper operating conditions, prevention of accidents or mitigation of accident consequences, resulting in protection of workers in the plant, the public and the environment from undue radiation hazards. Availability is the fraction of time for which a system is capable of fulfilling its intended purpose. Reliability represents essentially the same information, but in a different form: reliability is the probability that a system or a component will meet its minimum performance requirements when called upon to do so. Performance means the accomplishment of a task in accordance with a set standard of completeness and accuracy.

The requirements for I&C systems are derived from the main objectives (listed above) and they are presented as functional and technical requirements. As [116] describes, functional requirements are implemented by the I&C systems. The protection functions prevent systems and process components from damages, the interlocking functions prevent unwanted operational conditions of process, the closed loop controls keep process parameters within predefined limits, and the automatic start and shutdown processes and the measurement functions acquire process values and present them in the control room. Technical requirements influence mostly on the selection, design and implementation of equipment.

I&C systems need lots of information from the process itself and also from control commands. This information is provided by different kinds of measurements or measuring systems. The HMI (Human-Machine Interaction) is realised through different types of indicators, push buttons and computer workstations in control rooms.

1.3 Special Characteristics of the Nuclear Field

There are many characteristics making the nuclear field different from other safety critical industries. [112] lists several of them. The most important one is the fact that it is not enough to have a safe nuclear installation, but it also has to be proved to the licensing authorities that the installation really is safe and meets all the necessary requirements. One special characteristic in the nuclear field is the concept of *defence in depth*. Defence in depth means several levels of protection, ensuring that if a failure were to occur, it would be detected and the release of radioactive material to the environment would be prevented. The concept has been described in several nuclear related documents (e.g., [64]) and it is applied to all safety activities: organisational, behavioural and design related. Defence in depth helps to preserve the three basic safety functions (controlling the power, cooling the fuel and confining the radioactive material), and ensures that radioactive materials do not reach people or the environment. Defence in depth helps to achieve higher reliability with unreliable components. In a more concrete level, it relies on redundancy, separation and diversity, which aim to ensure that no single failure will pose a threat to safety.

Guide YVL 5.5 “Instrumentation systems and components at nuclear facilities” [102] says that “With the testing and analyses it shall also be ensured that there are no unintentional functions in the system or its equipment that could be detrimental for safety.” The principle of defence in depth should be used to provide sufficient proofs that I&C will follow YVL 5.5, meaning that the system provides all intended and no unintended functions. However, the hardware and software platforms used for I&C are complex, and proving the absence of unintended functions is almost impossible because it is always possible to argue that design errors in the hardware and software platforms may cause simultaneous failures of several critical functions [112]. In this context an unintentional function is one that is unnecessary for the actual functioning of a system or piece of equipment. Functions not required for accomplishing a task, but whose safety significance has been analysed and considered in system design, are not unintentional [102].

The I&C systems of the first generation NPPs were based on analogue technology. Now the trend is to start using digital I&C systems with programmable logic controllers. The increasing shortage of suitable spare parts for analogue systems is also accelerating the transfer towards digitalised systems.

One important characteristic differentiating nuclear power plants from conventional industry is their very long life-cycles. Nuclear power plants were typically designed to operate forty years and nowadays it is common to apply for a lifetime extension of twenty years or so. This usually includes renewals and upgrades of some systems. As information technology and computers are developing rapidly, the lifetime of a typical computer or software is usually less than five years, sometimes even less than three years. Comparing the huge difference between the lifetimes of the plant and a typical computer brings up the fact that the computer systems in NPPs have to be upgraded a several times during the lifetime of a plant.

1.4 Structure of the Report

Section 2 introduces some safety assessment approaches and explains the essentials of safety cases. The scope of making safety evaluations is also discussed along with sketching the different system levels which can be the target of evaluation. Section 3 describes formal methods for digital automation systems analysis. Some previous approaches and experiences from both research and regulatory perspectives are introduced as well. Section 5 sums up the report with conclusions and discussions.

2 Safety Assessment Approaches

2.1 Scope and Requirements for Analysis

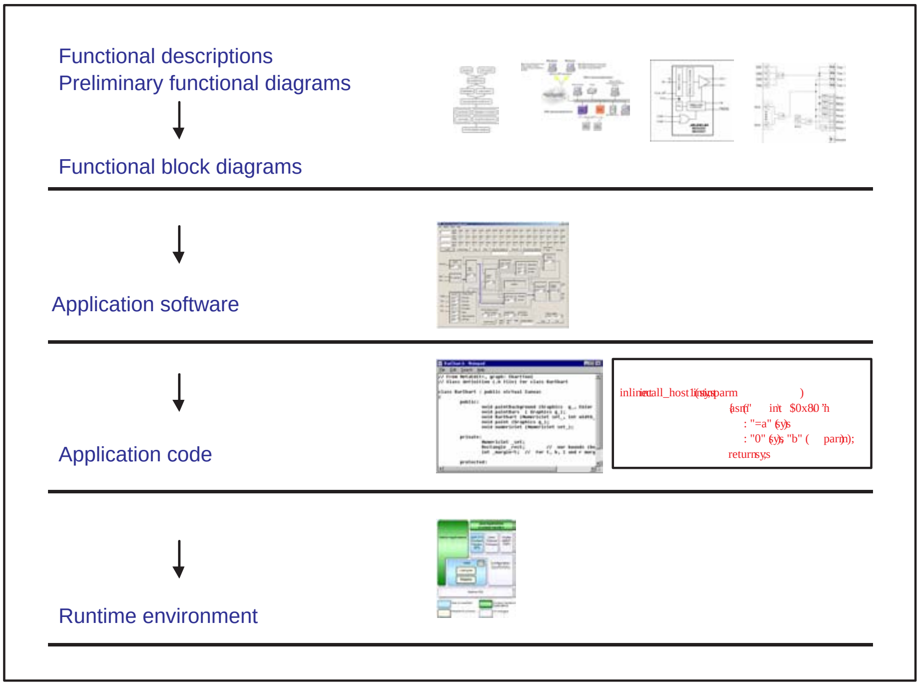
Safety analysis is difficult, labour- and knowledge-intensive work that can benefit from computerised support tools in several ways, such as reducing the time and effort involved in the analysis and keeping track of the system's analysed parts and their relationships. Following [56], the evidence demonstrating the acceptability of any safety related system can be divided into a) evidence addressing the quality of the development process and b) evidence addressing the quality of the product.

The quality of the system development process, testing, and analysis of products are typical parts of system safety assessment. [94] suggests that the safety assessment rests on a tripod made up of testing, analysis, and a certification of the personnel and the production process. In every case, the use of several methods in safety assessment is a significant and positive aspect, and will naturally lead to a more realistic safety estimate than putting one's faith on a single method [56].

One type of analysis is testing, which can show the presence of bugs. However, testing is not practical for showing that software is free from design errors. In the design phase of system or software development, simulation can be used to show presence of desired behaviour, but simulation does not cover all situations and possible behaviours. Instead, a technique called model checking can provide convincing evidence that the system design under inspection has or has not certain behaviour (model checking is introduced in Section 3.1.1).

Figure 1 below presents an example of possible "levels" at which a system can be analysed, and at which a safety assessment of I&C systems can be performed. On the first level, the system specifications, starting from functional descriptions and diagrams acting as requirements, are developed into more detailed specifications and designs. This results in production of function block diagrams – a more rigorous way of specifying the behaviour. The increasingly growing complexity in design may be achieved by starting with the main functions of the system and adding more specific details, redundant channels, voting units etc. little by little. One target of early phase safety analysis is the validation of function block diagram descriptions before the development gets to the implementation level.

The second level of the system is the application software, in the form in which it is developed (e.g., source code). This may be validated formally against the function block diagrams or informally against the original requirements. This validation exercise determines whether the developer has correctly transformed the requirements into the software design, implementing all of the required functionality without introducing unintended behaviours. This is a very challenging task, and so far no suitable meth-



```
inline int call_host1(int sys, int parm)
{   asm("int $0x80"
        : "=a" (sys)
        : "0" (sys), "b" (parm));
    return sys;
```

Figure 1. Example of system levels.

ods for making a complete safety analysis for software have been identified. There exist tools that perform automatic symbolic execution of source code in order to detect runtime errors. However, those are only assessing the code quality. They can be used to make searches for dangerous, incorrect or complex constructions but the assessment is not complete.

The third level is the application code, in the form in which it will execute on the platform (e.g., binary or bytecode). Validating this against the second level (application software) establishes the correctness of the compiler or code generator. In some circumstances it may be more convenient to validate the application code directly against the function block diagrams, for example due to tool availability; this combines two sources of bugs (the developer and the compiler).

The fourth level is the running system, formed by the application code and its runtime environment, such as a language interpreter or operating system. Arguments for the correctness of the runtime environment are typically based on process arguments (that the interpreter has been developed by a trustworthy development team) or proven-in-use statements (observing successful deployments of the same software elsewhere). The fourth level can be validated to a certain extent by testing it against a model derived from an earlier level (see 3.1.3).

2.2 Reliability analysis

Reliability can be defined as:
The ability of an item to perform a required function, under given environmental and operational conditions and for a stated period of time [68].

Reliability has been the subject of scientific study since 1930's. *Structural reliability analysis* concentrates on structural elements such as beams and bridges using the methods of mechanics and strength of materials, and will not be considered further here. In the *actuarial approach* [10] to reliability analysis, all information about the object system is included in the probability distribution function $F(t)$ of the time to failure T (the time interval from the present to the next failure of the system); the aim of the analysis is to obtain a good estimate of this function. When the system under consideration consists of several components, the analysis is called *system reliability analysis.*

Any complex technological system consists of three main types of components:

- *Hardware:* this has long been the main subject of study in reliability theory.
- *Software:* this has several distinguishing characteristics that render reliability methods developed for hardware components meaningless. For example, the reliability of a software component does not deteriorate with age, but rather stays the same once the software has been updated; the uniqueness of each piece of software makes the reliability data of earlier versions mostly meaningless in considering the reliability of a piece of software; software errors manifest themselves only under particular conditions.
- *Human:* in particular, considering the effects of limited rationality and perception; considering the effects of stress and fatigue; etc.

In addition to the above three types, environmental conditions and reliability of structures (weather, earthquakes, buildings, dams, etc.) have to be taken into account. This report will not consider either human reliability analysis or the environmental conditions.

The reliability of complex systems is often analysed with the help of a fault tree or an event tree. A fault tree is a presentation of the various faults in the system and how they (may) lead to system failure when combined. An event tree is a presentation of sequences of events that lead to a failure. A branching in the tree corresponds to points where events can take more than one course.

Some active research areas in the field are common cause failures (where one event, e.g. a fire, can cause several faults, e.g. burning of electric cables and leakage of pipes); restricting the number of events in event trees to avoid combinatorial explosion; human reliability; reliability of digital systems; elicitation and incorporation of expert opinions to the analyses; and the use of reliability analyses in decision making.

In the following, a commonly used method for risk-based decision making – failure modes and effects analysis (section 2.3) – is briefly described. There are other methods such as:

- *Pareto analysis:* this is a prioritisation technique that identifies the most significant items among many.
- *Checklist analysis:* here, the risks related to a system or operation are evaluated in the light of a preconstructed checklist.
- *Change analysis:* it looks systematically for possible risk impacts and appropriate risk management strategies in situations where a change is occurring. This includes situations in which system configurations are altered, operating practices or policies are changed, new or different activities will be performed, etc.
- *What-if analysis:* it is a brainstorming approach that uses broad, loosely structured questioning to postulate potential upsets that may result in accidents or system performance problems, and ensure that appropriate safeguards against those problems are in place.
- *HAZOP:* a systematic qualitative process in which a multi-discipline team performs a systematic study of a process using guide words to discover how deviations from the design intent can occur in equipment, actions, or materials, and whether the consequences of these deviations can result in a hazard. It focuses on identifying single failures that can result in accidents of interest. It is most commonly used to identify safety hazards and operability problems of continuous processes.

2.3 Failure analysis

Failure modes and effects analysis (FMEA) is a qualitative analysis that is often carried out before probabilistic risk analysis. Failure mode refers to the way a failure might occur, e.g. the failure of a valve to prevent flow or a leak of the valve. Failure effect is the consequence of failure from the system's (or customer's) point of view, e.g., lack of flow in a pipe.

FMEA is often carried out early in the development life cycle to find ways of mitigating failures and thereby enhancing reliability through design; in this case, coarse probability estimates for the failure causes and effects are usually derived. However, for the purposes of this report, FMEA is considered as part of reliability and risk analysis, and in that context it is a purely qualitative technique. FMEA consists of the following tasks:

- Potential failure modes are identified. It is not possible to anticipate every mode that a component etc. might fail, but as many modes as possible are identified.
- Their effects on the operation of the system are determined.
- Actions to mitigate the failures are identified.

FMEA's may be carried out from many perspectives, the following being the most common:

- *System:* the focus is on global system functions.
- *Design:* the focus is on components and subsystems.
- *Process:* the focus is on manufacturing and assembly.
- *Service:* the focus is on service and maintenance.
- *Software:* the focus is on the software subsystem functions.

FMEA can be used for different purposes depending on the phase of the development life cycle:

- Development and evaluation of requirements from reliability point of view.
- Identification of design characteristics that contribute to failures, thus aiding design.
- Help to develop useful tests to assess reliability.
- Ensure that potential failures will not injure or seriously impact people.

FMEA consists of the following steps:

- Describe the system (product, process or other) and its function. This covers also the uses of the system, both intentional and unintentional.
- Create a block diagram of the system. This shows the components or subprocesses of the system as nodes connected by lines that indicate how the nodes are related logically to each other.

- Create a table that lists components of subprocesses, and the failure modes and effects associated with each. A numerical ranking is assigned to each failure effect. Also the potential causes of each failure mode should be listed. Also the mechanisms that identify the failure or prevent it from causing system failure should be listed. The table should also contain a header that identifies the system considered, persons responsible etc.

- The table should be updated on a regular basis during FMEA, bearing in mind that a failure mode in one component may induce a failure in another.

- Compute risk priority numbers that are the product of failure severity, failure probability and failure detection likelihood (the probability that the failure will be detected, given that one has occurred). This and the following step are not carried out when using FMEA as part of probabilistic risk analysis.

- Determine potential actions for failures that have a high risk priority number.

An IEC standard exists for the conduct of FMEA [67].

A traditional FMEA uses potential equipment failures as the basis of analysis. Therefore, human errors and especially those that do not produce equipment failure are often overlooked in FMEA. In FMEA, equipment failures are analysed one by one, and therefore important combinations of equipment failures might be overlooked. Environmental conditions, external impacts and other such factors are analysed in FMEA only to the extent that they produce equipment failures; external influences that do not produce equipment failures (but may still produce system failure) are often overlooked. A single FMEA typically accounts for an equipment failure in one mode of operation (e.g. use, maintenance), and therefore more than one FMEA may be necessary to obtain a full picture of hazards.

2.4 Safety Cases

Safety cases are a way of presenting a clear, defensible argument that a system is adequately safe. A safety case document contains all of the necessary information for justifying the safety of a system together with an argument that explains how the available evidence supports the safety claim. The structure can take a number of forms. Historically, safety cases were structured so as to show compliance with the relevant safety standards. Vulnerability-based arguments are based on demonstrations that vulnerabilities within a system do not constitute a problem—this is essentially a "bottom-up" approach. Goal-based approaches (see Section 2.4.1) are the converse “top-down” approach, in which a general claim is supported by layers of increasingly detailed argument. These approaches are not mutually exclusive, and a combination of these approaches can be used to support a safety justification, especially where the system consists of both off-the-shelf and application-specific elements.

Although safety case documents are mandated in many sectors (e.g., railways, off-shore), in the nuclear sector there is no single defining piece of legislation. The requirement for a safety case arises from several Licence Conditions. Importantly, a safety case must demonstrate, by one or other means, the achievement of ALARP[1]. Other Licence Conditions, e.g. the establishment of Operating Rules, in the satisfaction of their requirements draw upon the contents of the safety case. In the UK Health and Safety Commission’s submission to the Government’s ‘Nuclear Review’[2] a Safety Case is defined as “a suite of documents providing a written demonstration that risks have been reduced as low as reasonably practicable”. It is intended to be a living dossier which underpins every safety-related decision made by the licensee.

The core of a nuclear system safety case is

- a deterministic analysis of the hazards and faults which could arise and cause injury, disability or loss of life from the plant either on or off the site, and
- a demonstration of the sufficiency and adequacy of the provisions for ensuring that the combined frequencies of such events will be acceptably low.

Risk-reducing provisions include safety systems, so the safety case will include arguments of compliance with appropriate standards and probabilistic analyses of reliability. Other techniques that may provide inputs to the safety case include fault and event tree analysis, failure mode and effects analysis (FMEA) and hazard and operability studies (HAZOP).

The safety case traditionally contains diverse arguments (or “legs”) that support its claims based on different evidence. Just as there is defence in depth in employing diversity at the system architecture level, there is an analogous approach within the safety case itself. Independent assessment is also an important part of the safety case process. The objective of independent assessment is to ensure that more than one person or team sees the evidence, to overcome possible conflicts of interest and blinkered views that may arise from a single assessment. The existence of an independent assessor can also motivate the assessed organisation. The relationship between

[1]The *ALARP* principle mandates a demonstration that risks are *As Low As Reasonably Practicable*, and is usually assessed by comparing the cost of further risk reductions with the postulated “cost” of a fatality or injury.

[2]The review of the future of nuclear power in the UK’s electricity supply industry.

independent assessments and legs can, however, be complex. Probabilistic Safety Analysis (PSA) is now an accepted means of demonstrating safety. The PSA is based on the overall plant design and operation and covers all initiating events that are in the design bases. It is performed using best-estimate methods and data to demonstrate the acceptability of the plant risk. The PSA provides a comprehensive logical analysis of the plant and the roles played by the design for safety, the engineered safety systems and the operating procedures. The PSA also demonstrates that a balanced design has been achieved: no particular class of accidents of the plant makes a disproportionate contribution to the overall risk. The PSA provides information on the reliability, maintenance and testing requirements for the safety and safety-related systems. There are various techniques used for safety analysis with fault trees, event trees and FMEA being the dominant methods, and HAZOP being used in fuel reprocessing applications.

In Finland, the concept of Safety Case is not yet widely used. Instead, the Finnish nuclear sector uses the term Final Safety Analysis Report (FSAR) which defines the principles of functioning and testing of safety related systems and equipment. FSAR presents design principles, tasks, and parameters that have an effect on safety. In addition, it summarises the tests of systems and equipment and it should be updated regularly after maintenance and update activities (just like Safety Cases).

2.4.1 Goal-based safety cases

Goal-based approaches are flexible, in that they focus directly on the safety requirements for the system, for example, requirements on functional behaviour, accuracy or fail-safe behaviour. This research has been adopted by several industries and goal-based cases are routinely developed in some sectors (e.g., civil aviation [33] and UK defence industries [109]), although they are still a departure from current licensing practices in the nuclear industry.

The flexibility of the goal-based approaches and the focus on safety properties makes them applicable when a standards compliance case cannot be made. This is often the case for off-the-shelf components, where typically, development follows industrial good practice and does not necessarily conform to a recognised safety lifecycle (e.g., IEC 61508). Alternative evidence can be presented to justify the safety properties depending on the characteristics of the device being justified and the process followed. Evidence can also be related to a range of different safety standards by identifying how the standards' requirements support the various claims.

One of the difficulties with safety cases is in ensuring their validity: does the safety case accurately show that the safety requirements are met? This difficulty is not linked to the specific approach taken to argue safety—it is applicable regardless of the approach being prescriptive or goal-based. For example, if a prescriptive approach is followed, where a given failure integrity is argued based on compliance with a safety process (as in the case of IEC 61508), we must consider how one can ensure that

- following the prescribed process guarantees that the integrity target has been achieved, and
- the deployed process is an adequate interpretation of the prescribed process.

A similar difficulty arises when a goal-based approach is applied. In a goal-based approach, top-level claims are decomposed into more specific claims and further decomposed until evidence can be supplied to support the claims. However, there is currently no systematic way of evaluating the validity and completeness of this decomposition. Similarly, no guidance is currently available on how to achieve an adequate decomposition of top-level claims. How can one ensure that the claim decomposition is appropriate and that we have confidence that the evidence presented supports the claims? Obviously, the validity of the safety case strongly depends on the sub-claims being adequate and sufficient to form an argument in support of a higher-level claim - that is, that the decomposition is valid.

2.4.2 Notation

The original goal-based argumentation structures were developed by Toulmin [106]. Toulmin's scheme addresses all types of reasoning whether scientific, legal, aesthetic, colloquial or management. The general shape of arguments consists of grounds, claims, warrants and backing:

- *Claims*, as the name suggests, are assertions put forward for general acceptance.
- The justification for the claim is based on some *grounds*, the "specific facts about a precise situation that clarify and make good the claim".
- The basis of the reasoning from the grounds (the facts) to the claim is articulated. He coins the term *warrant* for this. These are "statements indicating the general ways of arguing being applied in a particular case and implicitly relied on and whose trustworthiness is well established".
- The basis for the warrant might be questioned and here Toulmin introduces the notion of *backing* for the warrant. Backing might be the validation for the scientific and engineering laws used.

We need to consider that the implication from grounds to claims may not be deterministic: it may be possible or probable that the claim follows from the grounds. These are captured by the modality of the argument.

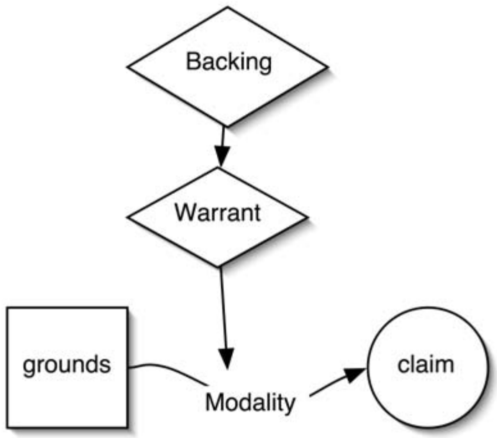


Figure 2. Toulmin's approach to argumentation.

2.4.2.1 Claims-Argument-Evidence (CAE) The work of Toulmin is the basis of the Adelard goal-based justification approach ASCAD [4], [17], where the claims-argument-evidence (CAE) structure is closely related to Toulmin's argument components:

- *Claims:* these are the same as Toulmin's claims.
- *Evidence:* is the same as Toulmin's grounds.
- *Argument:* is a combination of Toulmin's warrant and backing.

This relationship is illustrated in Figure 2.

Modalities are not captured by the arrows in the ASCAD notation (the dashed arrows in Figure 3). However, there is an informal convention to specify a set of assumptions or preconditions. This is similar to modality: the claim is only valid if the preconditions hold. The ASCAD notation encourages a series of arguments to be joined together in sequence by allowing claims to take a set of sub-claims as their grounds. We can thus demonstrate the top level claim by showing that

- The lowest level grounds can be shown to hold.
- All of the arguments in the structure are valid.

2.4.2.2 Goal-Structuring Notation (GSN) GSN [74] is a graphical approach to presenting safety cases, with a more involved graphical syntax than ASCAD. Node types are Goal, Solution, Strategy, Assumption, Justification, Context, Model, Notes and Option; GSN also includes facilities for

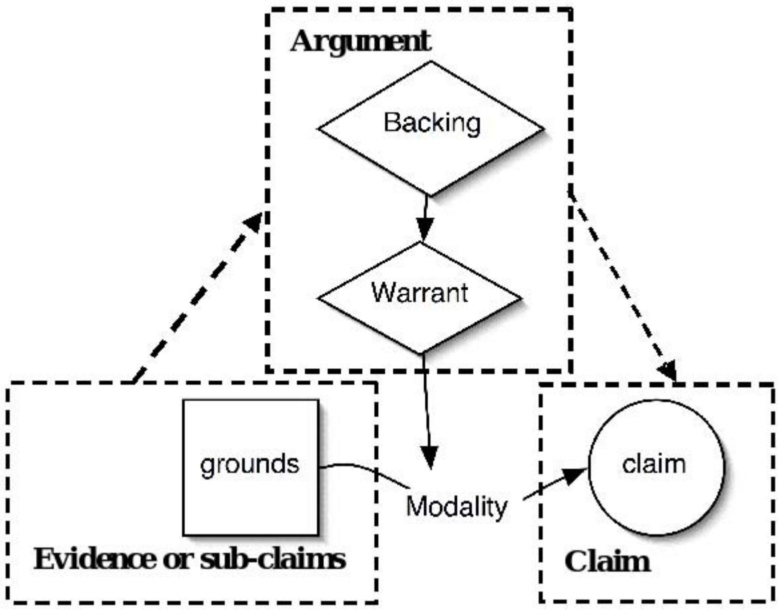


Figure 3. Relationship between Toulmin's scheme and ASCAD CAE approach.

templates and notation indicating how a template should be expanded to form a suitable safety case.

A relationship can be demonstrated between GSN and the original Toulmin concepts, where a GSN "goal" is equivalent to a claim, which is "solved" by strategies, "sub-goals" and "solutions" (which can be related to Toulmin's warrants and grounds).

2.4.3 Safety case formalisms

2.4.3.1 Govier support patterns Weaver et al. [114] argue that safety cases can be made more convincing by restructuring them so that each argument used fits one of the three support pattern types proposed by Govier [54] (Figure 4):

- A *single support pattern* has one premise supporting one conclusion.
- A *linked support pattern* has multiple premises which interdependently support the conclusion - the conclusion can only be held to be true if all of the premises hold.
- A *convergent support pattern* also has multiple premises, but each supports the conclusion separately.

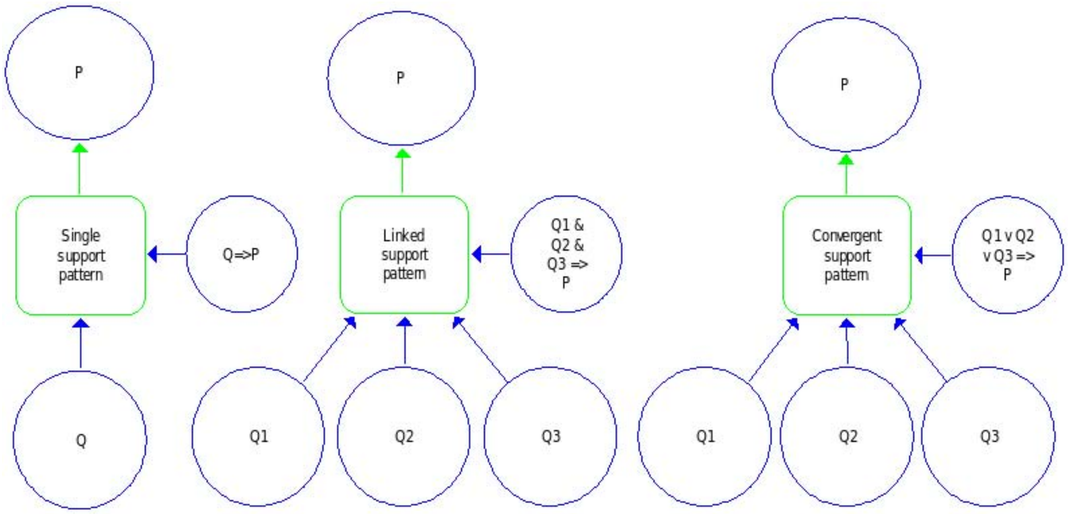


Figure 4. Govier's support pattern types in ASCAD Fog notation.

A claim that an "adequate supply of oxygen" is available might be supported by the sub-claims "main oxygen subsystem adequate"; "monitor detects main oxygen supply failure"; and "backup oxygen subsystem adequate". This is neither a linked support pattern (only the main or the backup supply is required, not both) nor a convergent support pattern (the monitor on its own cannot provide an oxygen supply). Weaver et al. "refactor" this structure into a two level tree: the adequate supply of oxygen is provided either by the main subsystem or by the backup subsystem (convergent support); the backup subsystem operates when the monitor detects failure in the main subsystem and the backup subsystem itself is adequate (linked support pattern).

This type of *normal form* for argumentation has its origins in mathematical approaches to logic and proof. Reformulating a complex argument in a normal form is a convenient way of clarifying its structure. In mathematical logic, this conversion can be performed automatically, while in the context of safety cases, arguments are rarely specified precisely enough for this to be possible. Manual conversion to a normal form therefore has the effect of forcing the safety case author to be more precise.

2.4.3.2 The Cemsis approach In the Cemsis [95] project, a formal approach to safety justification was developed which aimed at addressing some of these issues. The approach provides a "hierarchical structure for constructing arguments" [43]. Safety claims and evidence are presented in structured layers, where each layer has a corresponding model. There is no argument as such: evidence is related to the claim through some form of model and a set of agreed assumptions, i.e. the argument is self-evident given the evidence, the model and the sub-claims. Claims are made at the following levels:

0. Top level (the plant).
1. Plant / safety system interface.
2. Safety systems architecture.
3. Safety system design.
4. Operational environment.

Initial claims are made at Level 0, and can be expanded into sub-claims at lower levels. A claim at a given level can be satisfied by evidence at that level or by sub-claims at subsequent levels.

The same structure can be used for generic components, where Level 1 is a claim about the component interfaces, and subsequent claims are related to different levels of detail of the component. Claims about components can be used within larger system justifications.

2.4.3.3 The Fog approach The Fog project is partly concerned with ensuring that only valid arguments are used in the construction of the claim tree. The ASCAD notation represents the warrant and backing through the use of informal arguments in the narrative. To allow us to increase the formality of the claim structure, we draw the key parts of the warrant and backing out into side-claims (Figure 5). These may have their own decompositions, giving support and evidence for the argument itself.

Side-claims tell us how to combine the grounds together, and under what circumstances the argument is valid. In this way, for example, the various support patterns of Govier are representable. In each case, the side claims (to the right of the argument node) explain the relationship between the sub-claims or grounds ($Q, Q1, Q2, Q3$) and the conclusion claim P. For example, the single support pattern is used when a single sub-claim Q supports a single conclusion claim P. A side-claim is required to explain why $Q \Rightarrow P$ holds.

2.5 Regulatory Perspective

From the nuclear regulator's viewpoint, the problem with the current safety assessment methods is that they often do not address the behaviour of the system but try and infer this from indirect evidence such as compliance with standards or good development processes. Safety assessment methods which are feasible, repeatable and can be performed within reasonable amount of resources and time and directly address the behaviour of the system have clear advantages.

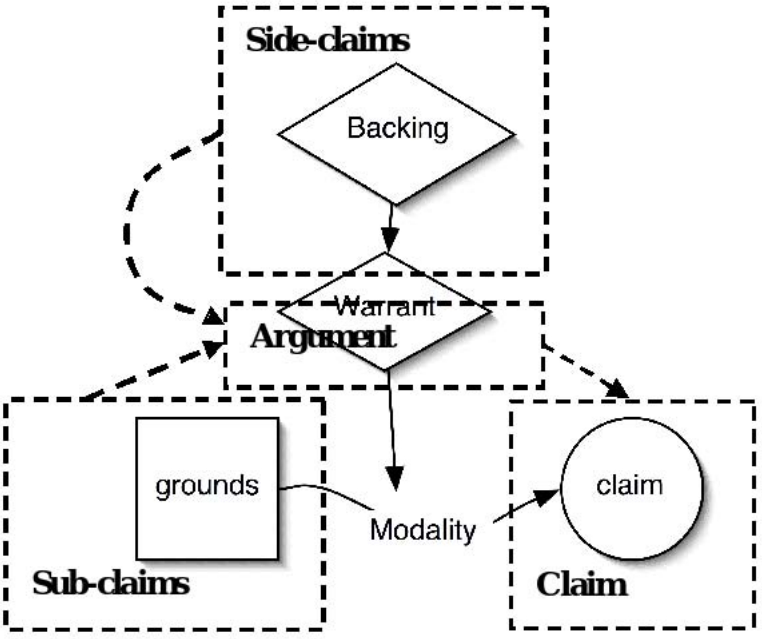


Figure 5. Relationship between Toulmin's scheme and the Fog normal form.

From STUK's (The Finnish Radiation and Nuclear Safety Authority) viewpoint, model checking can be seen as a method for finding indicators of items and entities to draw more careful attention to. It can also be used for providing direct, objective evidence for the assessment process based on the models of the system / software behaviour. Especially the phases of safety assessment that concern formalised design documentation and requirements specification are seen as potential applications for model checking.

The biggest obstacles of model checking to be part of safety assessment are related to building valid models. Creating the model and selecting the properties and system behaviour to be checked are crucial tasks and it depends on the skills and experience of the person making the model and checking it. Nevertheless, model checking can be seen as complementary technology to support testing and other more traditional assessment methods.

In the UK there are formal requirements for "Independent confidence-building" and this should provide an independent and thorough assessment of a safety system's fitness for purpose [99]. Model checking may be a technique that can be effectively used to satisfy this requirement (see [99] Safety Assessment Principles for Nuclear Facilities, HSE 2006).

Following [91], considerable progress has been made over the past 15 years or so in increasing the state space capacity of model checkers, to the point where specifications containing hundreds of state variables can often be verified automatically in a few hours. However, realistic designs often contain thousands or millions of state variables, far exceeding the reach of current model checking algorithms. Another class of verification tools called theorem provers can be used to overcome the capacity limitations of model checking. A theorem prover does not search the state space of a specification directly, but instead searches through the space of correctness proofs that a specification satisfies for a given correctness property. In general, model checking and theorem proving are complementary technologies and should be integrated to successfully tackle realistic system designs.

3 Formal Methods for Digital Automation Systems Analysis

3.1 Computer Aided Verification Methods

The traditional way of ensuring the reliability of distributed systems has relied on the two main techniques of *manual testing* using human- or machine-generated test cases and *simulation*. However, when the systems contain parallel and distributed components, the effectiveness of these techniques does not scale at the rate of the system size growth. The use of *computer aided verification* has been suggested as an aid to supplement these methods.

3.1.1 Model Checking

Model checking [40] is a set of methods for analysing whether a model of a system fulfils its specification by examining all of its possible behaviours. Model checking was introduced in the early 1980s simultaneously by two different groups [97, 41]. Good introductory books to the topic are [40, 12, 96]. In model checking, at least in principle, the analysis can be made fully automatic with computer aided tools. The specification is expressed in a suitable specification language, temporal logics being a prime example, describing the allowed behaviours of a system. Given a model and a specification as input, a model checking algorithm decides whether the system violates its specification or not. If none of the behaviours of the system violate the given specification, the (model of the) system is correct. Otherwise the model checker will automatically give a counterexample execution of the system demonstrating why the property is violated.

In *symbolic model checking* the main idea is to represent the behaviour of the system in a symbolic form rather than explicitly. There are several variations to symbolic methods. The most well-known is the use of data structure called ordered binary decision diagrams (OBDDs), which are a canonical representation of Boolean functions [30, 32, 31, 87]. The *bounded model checking* method [14] was introduced to further improve the scalability of symbolic model checking by replacing OBDDs with methods based on propositional satisfiability (SAT) checking. The main idea in bounded model checking is to look for counterexamples that are shorter than some fixed length n for a given property. If a counterexample can be found which is at most of length n, the property does not hold for the system. It seems that the bounded model checking procedures can currently challenge OBDD based methods on digital hardware designs both in terms of memory and time required to find counterexamples [15, 24, 42]. A weakness of bounded model checking is that if no counterexample can be found using a bound, using the basic method the result is in general inconclusive. However, the basic method can be extended to a complete model checking method for safety properties by employing temporal induction [100, 49]. The Laboratory for Theoretical Computer Science at Helsinki University of Technology and staff at Adelard have contributed to the development of the NuSMV symbolic model checker [38, 76, 77, 59, 60, 39, 69, 16].

The area of model checking of real source code has recently become a prominent field of research. A large part of the interest has been triggered by the success of the SLAM project [7] at Microsoft. SLAM is a model checker for *sequential* C programs aimed at the verification of the use of locking primitives of Windows device drivers. As a matter of fact, the SLAM system has been transferred from Microsoft research into the Windows group and has been integrated in a beta version of the Windows driver development kit [6], and is thus in production use. The success of the SLAM project has raised interest in using model checking methods also for analysing C source code. Similar approaches could be used to e.g., analyse the embedded software of smart sensors and other small devices with embedded software written in C. The academic tools in this area are, however, less mature than model checkers for hardware systems such as NuSMV.

For concurrent programs such as data communications protocols the analysis techniques have so far mostly concentrated on modelling the concurrent program in the input language of a traditional explicit state model checker. These tools use simpler models of concurrency and include tools such as SPIN [61] or Murφ [48]. The Laboratory for Theoretical Computer Science at Helsinki University of Technology has significantly contributed to the research on model checkers by creating tools such as PROD [111] and Maria [85].

For systems with real-time constraints model checking approaches based on analysing models with real time valued clocks such as timed automata is natural, and can be sometimes much more efficient than modelling the clocks with counters updated at discrete time intervals. One of the most prominent model checking tools in this area is Uppaal [11].

3.1.2 Deductive Verification

Deductive verification is a set of methods for determining whether a program fulfils its specification by analysing its formal semantics. The specification defines a relationship between the state of the computer when the program begins (the precondition) and the state of the computer when the program terminates (the postcondition). The way that each program statement in turn transforms the state is analysed either manually (e.g., as described by Kaldewaij [73]) or using a theorem provers or proof assistants (e.g., the Caduceus and Krakatoa tools [51]), in order to determine whether the specification is met.

Deductive verification has been successfully applied in safety applications [86, 27] but it remains a relatively expensive technique. Both manual proof and conventional theorem provers require a degree of understanding of the underlying semantic model and proof approaches as well as the NPP domain at hand. Although this requirement for skilled application of the method makes deductive verification unsuitable for MODSAFE at least from NPP I&C systems developer perspective. However, some techniques used in the field may still be quite relevant. For example, the approach could be used in verification efforts done by third parties with sufficient knowledge of both deductive verification methods as well as NPP domain expertise.

3.1.3 Model-based Testing

Testing is one of the most time consuming parts in the development of complex systems. Reports from the telecommunications industry tell us that the number of man hours devoted to testing is in many cases more than half of the effort spent in a typical development project. However, in many cases the testing is very ad-hoc and not based on a solid theoretical foundation. Thus there is a lot of room for improvement to be gained from a model-based approach to testing.

A promising way to use formal methods in connection with testing was proposed in [107, 45, 28]. The main idea is to test a black-box implementation with test cases and test verdicts automatically generated from an assumed to be correct “golden design” abstract system model, which is in this context called the specification. How this could be further applied in automation setting is an open research question.

The first commercial model-based testing tools are available for UML models [62] and for Simulink/Stateflow models [103]. However, the academic tools available for model-based testing are currently less well developed than academic model checking tools.

3.2 Formal methods for Digital Automation systems

In this section we review existing work on applying formal methods to the analysis of digital automation systems. However, as the subject area is very wide, we have restricted our focus on the key areas of interest in the view of the MODSAFE project. Therefore we discuss only studies concerning automation systems based on programmable logic controllers (hereafter PLCs). Consequently we have mostly excluded systems which are based e.g., on softPLCs (PLCs based on standard PCs) and distributed control systems (DCSs). This restriction seems reasonable, since the conclusions made on PLCs can be extended to cover also softPLCs. On the other hand, covering also the subject of DCS would have required a survey of much larger scale. However, the PLC domain can be seen as a logical stepping stone towards analysing of DCS based automation systems. On the method side we are focusing mainly on applying model checking methods to PLC applications.

The analysis of PLC based systems with formal methods can be done on different levels. In the most comprehensive approach the validity of the PLC program is verified against the specification of the entire system which consists of the combined specification of the controller and its environment. Another option is to restrict only on verifying the correctness of the controller. In this case the specification of the system as a whole is divided into the specifications of the environment and the controller part, and the program of the controller is analysed with respect to the specification of the controller.

Another classification of approaches on applying formal methods to PLC applications can be made based on the initial objective of the process. That is, the goal might be to analyse an existing application or to design a completely new one. In the first approach an existing PLC program is first transformed into some formal modelling language and then, based on the model, the validation of properties is carried out with a model checker. This approach is often referred to as modelling or formalising existing PLC programs [80]. The related studies are often—but not always—restricted to only validating the controller against its specification. In the second approach a system is designed from the beginning by using a formal modelling method. After the model (which in this case often consists of both, the model of the controller and the environment) is finished, it can be used, alongside of validating properties, to derive a PLC program automatically. This approach is usually referred to as model based design or program synthesis [53].

In this report we review studies on both approaches, the modelling of existing PLC programs and the model based design. We start by listing some earlier surveys made on the subject. From these we found especially the papers [80, 63] as a good starting point for our own review. After the list of existing surveys we proceed to present references to the most relevant studies in the view of MODSAFE project. However, before going into the actual survey we describe in the following section some classification criteria for models of PLC programs which were originally presented in [80]. We use this terminology to describe the references that we list in this paper, but not all the referred studies are classified according to all these criteria.

3.2.1 Classification Criteria for PLC Models

Here we describe the three orthogonal criteria for classifying PLC models originally presented by Mader in [80]. The discussion is intentionally kept brief and an interested reader is advised to turn to [80] for more in-depth coverage on the issue.

Modelling of the cyclic operation mode

The most fundamental characteristic of PLCs is their cyclic operation mode. Therefore the first logical choice in classifying PLC models can be made on the basis of how the scan cycle of the PLC is modelled. There are three possible choices:

- Explicit modelling of the scan cycle.
- Implicit modelling of the scan cycle.
- Abstracting from the scan cycle.

In the explicit modelling of the scan cycle the exact duration of the cycle in actual time units is modelled. Instead, in the implicit modelling the existence of the cycle itself is modelled, but the actual duration of it is not measured in time units, and moreover, it is considered to be constant. The third option is to abstract from the scan cycle entirely so that the time model of the PLC is considered to be continuous instead of discrete.

Modelling of timers

The use of timers is a fundamental characteristic of PLC programs. However, not all of the PLC applications need timers and in many cases the use of them can be avoided, either by modelling techniques or by altering the system design. Therefore, it is justifiable that there are studies concerned with modelling of timers as well as those which abstract them out.

The language fragment considered

By the language fragment criteria a choice is made on which parts of the PLC programming language are considered in the modelling process. The first obvious question is that which of the five languages defined in the IEC 61131-3 standard is considered. The options are:

- Instruction List (IL).
- Structured Text (ST).
- Ladder Diagrams (LDs).
- Function Block Diagrams (FBDs).
- Sequential Function Charts (SFCs).

Moreover, usually only a restricted part of the features of the chosen target language is considered. This is because the IEC 61131-3 standard lacks definition of formal semantics on the languages of the standard. Therefore, from the viewpoint of academic research, it simply is not reasonable to consider all possible semantic interpretations of all parts of the languages.

Finally, the language fragment considered can also be restricted on the possible data types, i.e., which of the data types of booleans, integers and real numbers are allowed.

3.3 Survey of Studies on Model Checking PLCs

In this section the survey of studies on applying model checking to PLCs is presented. The section is organised as follows. In Subsection 3.3.1 a list of already existing surveys is given, after which approaches to modelling of PLC programs are listed in Subsection 3.3.2, and finally in Subsection 3.3.3 the studies on model based design are reviewed.

3.3.1 Previous surveys

- The paper [80] by Angelika Mader presents a classification of different PLC models. It first classifies an orthogonal set of criteria on which PLC models can be classified on. The paper also introduces an extensive list of publications which are classified against the presented criteria. The dissertation [63] by Ralf Huuck presents quite a similar survey which also includes more recent studies and an extended list of the classification criteria.

- The paper [84] also by Mader discusses the application of formal methods to PLC applications in general. It presents a schema on the structure of a general PLC application and based on this framework analyses the possibilities for applying different formal methods on PLCs.

- The paper [120] by Frey introduces four criteria on which studies considering formalisation of existing PLC programs can be categorised. It also presents references to studies falling in each of these categories. The study is by no means as thorough as the survey presented in the papers [80, 63] but contains some additional references and shows another way of classifying the existing research.

- The paper [53] by Frey presents a general framework on the different phases of verification and validation and discusses what formal methods can be used in these different phases. Therefore, it not only focusses on transforming existing PLC programs to models, but it also discusses other formal methods in addition to model checking.

3.3.2 Modelling PLC applications

In the following a list of studies considering the modelling of PLC applications is given. The references are organised according to the PLC programming language.

Research on modelling SFC programs

- The Doctoral Thesis of Ralf Huuck [63] shows how SFC programs can be given formal semantics, including a translation of the untimed semantics of PLCs to the Cadence SMV model checker input language. In the paper [9] this research is extended by giving formal semantics also for timed SFCs and in the paper [8] it is shown how timed SFCs can be translated into timed automata. The latter study illustrates in both, timed and untimed cases, the complete verification procedure from model transformation to identifying errors with model checking tools UPPAAL and Cadence SMV.

  The research papers described here seem to take the closest approach to that originally sketched in the MODSAFE research proposal.

Research on modelling IL programs

- The widely referred paper [82] by Mader and Wupper shows how a fragment of IL programs can be translated into timed automaton. Based on this study a tool is presented in the paper [115] by Willems which translates IL programs automatically into timed automaton format accepted by UPPAAL model checking tool. This tool chain allows model checking of real time properties with explicit modelling of the scan cycle. The Willems's tool also allows IL programs to contain bounded integer variables. Moreover, UPPAAL tool can be used to model the environment of the PLC as well. In an unpublished paper [83] Mader presents two examples on modelling IL programs and performing their verification with the UPPAAL tool.

- In the paper [58] it is shown how IL programs can be transformed into Petri nets. The method allows usage of data structures up to length of 8-bits and it takes into account all standard instructions excluding commands from libraries. However, the real-time aspects cannot be modelled.

- In the paper [66] it is shown how IL programs can be transformed into Timed Net Condition/Event systems. The scan cycle is modelled explicitly and timers are taken into account at some level. However, the possible data structures are restricted to boolean values and only **load**, **store**, **and**, and **or** instructions are considered.

- The paper [34] deals with the same fragment of IL programs as the paper [66] described above. In addition, the loop operations are considered. However, [34] is concerned with translating IL programs directly to the input language of the SMV model checker tool. As SMV cannot directly handle real-time issues, these aspects of the IL program cannot be analysed.
- The paper [78] uses BDD and BMC based symbolic model checkers to model check two small PLC based automation systems written in IL.

Research on modelling LD programs

- In paper [90] LD programs are modelled with SMV tool without taking timing aspects into account. Based on this study there exists a research paper [108] presenting two comprehensive case studies on existing chemical processing systems. In these case studies the model of the environment is also presented. The verification process revealed numerous faults and the results could be used to improve the designs.
- In the paper [98] a large fragment of LD programs are modelled with the SMV model checker. The scan cycle is modelled implicitly and it is shown how a particular type of timers can be modelled in non-real time manner so that certain liveness and safety properties can still be verified.

Research on modelling ST programs

- The paper [71] by Jimnez-Fraustro and Rutten considers of modelling a fragment of the ST language with the synchronous language SIGNAL. The fragment includes at least assignments, conditionals and bounded loops. Scan cycles are modelled implicitly and real-time behaviour is not considered. The follow up study [72] considers also the FBD language. Unfortunately, there does not seem to exist any related studies on actual model checking based on the SIGNAL model.

3.3.3 Methods for synthesising PLC programs from models

In the following a list of studies considering the synthesis of PLC programs from models is given.

- In the paper [47] Henning Dierks presents a new modelling formalism named PLC-automata especially designed for modelling PLC applications. PLC-automata allows explicit modelling of the scan cycle but it is possible to model only particular type of timers in which an input signal is ignored for a certain time. Dierks shows also how PLC-automata models can be transformed automatically into language of Structured Text.

In the paper [46] it is shown how PLC-automata models can be transformed into timed automata models which makes it possible to perform model checking with a real-time model checker such as KRONOS or UPPAAL. Moreover, in the paper [104] a tool MOBY/PLC is presented which can be used to model PLC-automata, validation, and code generation.

- As a continuation for the research based on PLC-automata formalism Olderog presents in paper [92] an approach for designing valid PLC applications. His method is based on formulating design specifications with PLC-automata and specifying requirements in Constraint Diagrams. Olderog also presents a case study from industry for which he applies his approach. In the paper [93] Dierks and Olderog present a tool Moby/RT which is based on the design approach of [92].
- The Doctoral Thesis of Georg Frey [52] tries to formalise PLC control algorithms and their verification through the use of a Petri net based formalism.
- A quite interesting research project from the view of MODSAFE project is reported in the papers [81, 29] by Mader, Brinksma et al. In [81] a systematic design and validation of a PLC control program for a batch plant by using formal methods is reported. This plant was selected as a case study for the EC project on Verification of Hybrid Systems (VHS). In the follow up paper [29] it is reported how the SPIN model checker was used for both the verification of a process control program and the derivation of optimal control schedules.
- The paper [117] discusses the use of formal methods in designing PLC controllers to be employed in Korean nuclear power plants. The approach is based on first modelling the required control using a combined timed automata and tabular notation like model (NuSCR), and then mapping the design through automated tools to a PLC implementation based on the IEC 61131-3 Function Block Diagram FBD notation. In the actual implementation the FBD design is further hand optimised by domain experts to minimise time needed for the PLC cycle. The authors claim that they can employ model checking on the initial timed automaton model as well as verify the soundness of the hand optimisations used. However, actually performing this model checking is not reported in the paper.
- The paper [119] gives details of the NuSCR specification approach discussed above together with further references.

- The paper [37] gives the tool support used by the Korean NuSCR specification formalism. It also includes an SMV backend that supports the model checking of NuSCR specifications. The paper contains a small case study utilising model checking.
- The paper [118] describes the synthesis approach for NuSCR described above in slightly more detail.
- The paper [70] describes the methods used to test PLC programs in FBD format for Korean nuclear power plants.
- The paper [101] shows how the SMV model checker can be used to model check a Korean safety critical system for a nuclear power plant.

3.4 Overall Status of the Research on Model Checking PLCs

On the study field of modelling PLC programs most research papers seem to be available on the language of Instruction Lists. On Ladder Diagram programs there is also quite a lot of research but in this area there seems to be many gaps to be filled, especially on covering timers and real time issues. Instead, in the case of the SFC programs the situation is the opposite: there hasn't been that many research projects, but the coverage of existing ones is quite extensive.

Considering the last two languages of the IEC 61131-3 standard, i.e., on Structured Text and Function Block Diagrams, there seems to exist only very few studies. We presume that the reason for this in the case of the ST language might be that the relevant problems on the modelling issues are present also in the IL language which, in its brevity, better suits academic research. Instead, in the case of the FBD language, the similar but more evolved SFC language provides features not existing in FBD, and therefore, might be more appealing for research.

For the synthesising approach there has also clearly been a lot of research activity. The usefulness of this approach has especially been shown in the Korean research project described above. As this particular research project is very close to the scope of the MODSAFE project, it is further described in Section 3.5.1.

3.5 Previous Approaches to Software and Automation Analysis (in NPPs)

3.5.1 Computer Aided Analysis of Korean and Canadian NPP Automation Systems

As already mentioned in Section 3.3.3, the Korean approach to NPP automation systems analysis is based on using a finite state machine model of the PLC control given formally in a tabular notation called NuSCR [117].

The idea is that first this finite state machine model is proved correct using, e.g., model checking methods [101]. After the model is known to be correct, it can be mapped into an implementation using an automatic translation [118]. It is not known to us whether any control systems designed in this manner are in production use at Korean NPPs. The drawback of this approach is that it requires the design approach of the control systems to be modified to accommodate for computer aided analysis unlike in the MODSAFE project where no such design process changes are needed.

A similar approach to the one employed in Korea has been used to design real reactor shutdown systems of Canadian NPPs of Ontario Hydro using a tabular notation software design documents and proving the design documents correct using a theorem prover. A very nice overview of the approach taken in Canada can be found from [113].

3.5.2 French approaches

The French approach to reliability determination is basically deterministic and supplemented by PSAs. PSAs are not requested before licensing: they are used only to gain a global overview of the plant safety and to determine possible weaknesses. Besides, the PSAs do not cover software.

3.5.2.1 SCADE The SCADE tool [13, 105], based on the synchronous high-level LUSTRE language [36], is an industrial tool developed by Esterel. It has been used in nuclear software development by Merlin-Gerin and Schneider-Electric for software development. Two facilities provided by SCADE are of interest:

- *Validation facilities:* Design verification supporting formal analysis and simulation.
- *Automatic Code Generation:* SCADE produces some code (C) automatically. Originally problems in the tool were found by checking against the code; these problems have been corrected and Merlin-Gerin now argue that their feedback of two years experience with non-nuclear systems has stabilised the tool.

3.5.2.2 Caveat Caveat [1] is a C static analysis tool aimed at control systems code—code which may be long but is typically not very complex. It is a combination of an interactive theorem prover and code analysis tool, covering:

- Code navigation (code structure, call graphs and data-flow graphs).
- Support for formal proof of properties in first-order logic.

- Calculation of the subroutine outputs as a function of their inputs, and the calculation of weakest preconditions.
- Test case generation.
- Dead code detection.
- Software execution metrics (worst-case execution time, stack depth).

3.5.3 BE-SECBS Project

A FP5 project called "Benchmark Exercise of Safety Evaluation of Computer Based Systems BE-SECBS" aimed at making a comparative evaluation of existing methodologies in safety critical computer based systems assessment among regulators and technical support organisations in some EU Member States. The project had three assessor teams: Institute for Radiological Protection and Nuclear Safety (IRSN, France), Institute for Safety Technology (ISTec, Germany), and a Finnish consortium Technical Research Centre of Finland / Radiation and Nuclear Safety Authority (VTT/STUK) which assessed a reference study case provided by Framatome. The project was coordinated by Joint Research Centre—Institute for Energy (JRC-IE). The following three sections describe the methods used in the BE-SECBS project [75].

3.5.3.1 IRSN's Methods IRSN's method of safety critical software evaluation consists of five major steps:

1. Development process and associated documentation analysis. The first step of the evaluation consists in understanding the development process and assessing its conformance to the principles and requirements of national regulation (French basic safety rules RFS IV.1.a, IV.2.b and V.2.d) and international standards (IAEA safety guides, IEC 61226, IEC 61513, IEC 60880). The crucial part of this examination focuses on the evaluation of the consistency, completeness and correctness of the software specifications against user requirements and system design requirements.
2. Source code analysis. IRSN uses QAC and McCabe tools to assess the code quality. They can be used to make searches for dangerous, incorrect or complex constructs. Polyspace Verifier tool automatically performs symbolic execution of a source code in order to detect runtime errors such as array out of bounds violations using static analysis techniques.
3. Determination of critical software components. Failure Modes and Effects Analysis (FMEA) adopted to software systems is used at this

assessment step. An index of relative importance is established for each function by taking into account the number and severity of the consequences of the failures and categorising them. Once the potential failures that could lead to dangerous malfunctions are identified, it is checked that there are no errors in the software that could lead to these failures. This is done by verifying that there is at least one validation test performed that would have covered this postulated failure mode.

4. Dynamic analysis. In order to perform dynamic analysis, IRSN has developed a set of tools, called Claire simulation tool [35], which can simulate operation by execution of a binary program without recourse to equipment used on site. The goal is to perform non-intrusive test and observation of the actual binary codes of a multi processor system. Next, a consistency study is used to verify the output values from the channels (e.g. controlling a reactor trip) when the input values are selected by the analyst from the nominal operating range of the system. This study verifies the most significant aspects of the behaviour of the binary program that is actually operational on site. Eventually, a robustness study aims at judging the behaviour of the programs of the representative set subjected to series of tests. These tests are defined in advance. They represent abnormal situations for the system or its environment. The series of tests are focused on the critical or sensitive components detected during the previous steps.

5. Development of test cases and choice of testing strategy. For the testing analysis purposes, IRSN has developed the tool Gatel. Test cases (input/output sequences) are generated, covering a specification working together with its environment. These test cases exhaustively cover a given test objective, derived by the assessor from the safety requirements of the system. Gatel produces “abstract tests”, corresponding to a functional view of the software, so they must be “concretised” to the binary code level in order to be run by Claire. This translation is made by Claire and requires additional knowledge about the binary code under test.

3.5.3.2 ISTec’s Methods Institute for Safety Technology (ISTec) participated in BE-SECBS project as well. The safety assessment performed at ISTec was based on IEC 60880 and German national rule KTA 3503 (“Type Testing of Electrical Modules for the Reactor Protection System”). To verify the automatically generated code within a reasonable amount of time, a methodology was developed by ISTec within the framework of a research program from 1994 to 1997. This methodology is based on a tool supported static analysis of the generated code which is independent from the generation rules of the code generator. The tool is called RETRANS (REverse

TRAnsformation of Normed Source code). The methodology is restricted to the configuration part of software systems, where the configuration part consists of normed source code. RETRANS transforms the normed source code into analysis items. These analysis items should have a correspondent item in the original specification data set and vice-versa (consistency and completeness). In the case of an error-free run RETRANS

- demonstrates the functional equivalence of the specification data set with the source code,
- demonstrates the existence of consistent data in the specification data set,
- demonstrates the existence of consistent data in the generated normed source code,
- carries out a plausibility check: analyses items' comparison between redundancies and highlights differences.

In general, ISTec applied a two-phase qualification approach consisting of generic and plant-specific elements and made use of the fact that the example case was type-tested according to the mentioned German national rule KTA 3503, which is a generic qualification of hardware and software components. Within BE-SECBS, ISTec performed the corresponding plant-specific system qualification phase. Since 1993, type testing of software components has been applied by ISTec according to KTA 3503 with the objective to qualify and assess the developed software modules on the basis of the mentioned phase model and to demonstrate compliance of the type tested software components with their software requirements specification. The plant-specific system qualification is performed within the respective licensing process of the individual plant with emphasis on the system architecture, V&V measures, functional tests, and application code analysis. Besides functional testing, the analysis of the application code is a necessary verification step in the whole qualification procedure. It grants the compliance with requirements of generally accepted standards like IEC 60880 for the automatically generated application software. During the plant specific software assessment it is not necessary to evaluate the type tested software modules again; it only needs to verify that the correct versions of the type tested software modules are used and that they are used in an appropriate way.

3.5.3.3 VTT/STUK's Methods The VTT/STUK safety evaluation method is based on the Finnish regulatory guide YVL-5.5 which describes the principles for licensing automation systems for NPPs. The basic principle of the methodology is the critical review of the evidence and analyses provided by the system vendor or the power utility applying for the license. The VTT/STUK method aims at evaluating the quality of evidence (i.e. evaluation of the target product and its design and implementation process with respect to standards, authority requirements etc.). YVL-5.5 requires also quantitative reliability analyses. In the VTT/STUK method, the quantitative reliability estimates are produced by using a Bayesian network model.

The qualitative analysis of a programmable automation system has the following major tasks or phases: overall analysis of the evidence/material provided; analysis of the requirements specification; FMEA of safety functions; analysis of test coverage, operating experience, application code; analysis of the platform development process and application development process. Each of the above mentioned items is evaluated and an assessment is made on the level of quality. This analysis yields a map of safety evidence or safety arguments.

The results of the above tasks are used in the quantitative reliability analysis. There are no well-established and generally accepted methods for quantitative reliability analysis of programmable systems. However, when performing a PSA study, also these systems have to be taken into account and probability values that programmables do not perform their task properly when demanded, have to be evaluated. The direct estimation of such probabilities on the basis of operating experience is not possible due to limited statistical information. However, the information developed in different phases during licensing process reflects the reliability of the system and it should be utilised. One approach to take this information into account is the use of Bayesian networks with the following modelling phases:

1. Development of the map of evidence,
2. Definition of the structure of the Bayesian network model (variables, metrics, probabilistic relationships and dependencies),
3. Quantification of the model,
4. Interpretation of the results (sensitivity and importance analysis).

3.5.4 Model Checking Operator Procedures

Some very interesting work has also been done on model checking operators procedures in the avionics context. NASA has been using model checking to check for mode confusion errors [88, 44, 79]. The idea is to look for potential cases where the operator might lose his/her situation awareness (be confused about the mode the system is in) by using a formal modelling of the user interface of the operator and the system being controlled. So this is an interesting piece of work using formal methods to analyse human computer interaction factors of operators.

In the NPP area operator procedures have been formally checked by Institute for Energy Technology (IFE) which has been using model checking in its research for the OECD Halden Reactor Project in Halden Norway. The research has considered designing and verifying operator procedures which are documents telling operators what to do in various situations. They are widely used in nuclear power industry and in process industries. The correctness of such procedures is of great importance to assure safe executions of the procedures.

The paper [121] describes how model checking can be used to detect potential errors and to verify properties of such procedures. Types of correctness specifications, types of errors that can be detected by model checking and basic techniques for modelling operator procedures and for detecting errors are discussed.

The paper [122] presents verification approaches based on different model checking techniques and tools for the formalisation and verification of operating procedures. Possible problems and relative merits of the different approaches are discussed. The paper [122] also presents a case study of one of the approaches to show the practical application of formal verification. Application of formal verification in the traditional procedure design process can reduce the human resources involved in reviews and simulations, and hence reduce the cost of verification and validation.

3.5.5 Adelard's Methods and Approaches

A safety case is a type of logical argument in which a claim about the safety of a system is argued by decomposing it into sub-claims about simpler properties and simpler subsystems. Model checking and other formal approaches are therefore applicable in two ways: as evidence that a subsystem has certain properties, and as evidence that a safety case forms a valid argument.

3.5.5.1 Use of Formalisation of Safety Case Arguments Adelard's Fog project (see Section 2.4.3.3) is concerned with developing a formal argument structure for safety cases. By restricting safety cases to a simple logical argument in a normal form, it becomes possible to use automated tools to prove the validity (or otherwise) of the argument represented. The approach taken in Fog is to produce safety case "building blocks" which have been shown in a theorem prover to be valid; these building blocks can then be composed into a safety case. Provided the evidence at the lowest level is established, knowing the validity of the safety case argument allows us to deduce the top level claim.

3.5.5.2 Use of Formalisation in Safety Case Claims

Software Reliability Modelling The expected reliability of software can be modelled as a function of the faults detected through testing. That is, by extrapolating future reliability from past failures. Using relatively standard assumptions it can be shown [22] that the expected worst-case value of the failure rate after a usage time t is bounded by:

$$\bar{\lambda}_t \leq \frac{N}{et}$$

where N is the initial number of faults and e is the exponential constant. Less pessimistic results can be obtained if additional assumptions are made about the distribution of failure rates over the N faults. These predictions turn out to be relatively insensitive to assumption violations over the longer term. The theory offers the potential for making long term software reliability growth predictions based solely on prior estimates of the number of residual faults (e.g. using the program size and other software development metrics).

Given a fixed amount of testing, this formula can be rearranged to determine the maximum number of faults permitted for a target MTTF[3]:

$$N \leq \frac{et}{MTTF}$$

This is, of course, a worst case bound model, and better results would be obtained with more accurate models. For example, a worst case bound model that assumes a log normal distribution is described in [23].

Software Process Modelling Having related the system reliability to the number of faults, the ability of the development process to detect faults or avoid creating them becomes of interest. A “barrier model” of the development process may be used [26], where in each development phase faults are created or detected. This model is parameterised by the rates of fault creation and detection at each phase of the software lifecycle. It can then be used to estimate the number of residual faults (i.e. those that escape the last barrier).

In [26], an analysis of fault creation and detection estimates for a number of projects was carried out, in order to calibrate the statistical models for a given development process. Figure 6 shows a probability distribution for a particular development process. The bar graph marked “Judgement” indicates the predicted probabilities of producing a product with a given number of faults; gamma and log-normal best-fit curves are also shown.

[3]Mean Time To Failure (MTTF), the time before the next system failure occurs, is the reciprocal of reliability—which is the expected number of failures in a given time period.

Two main conclusions can be drawn from this type of work: recommendations can be made to a manufacturer for process improvements based on typical sources of faults found through the modelling; and the expected reliability of a new product can be deduced based on the results of previous applications of the same process and the level of testing undertaken.

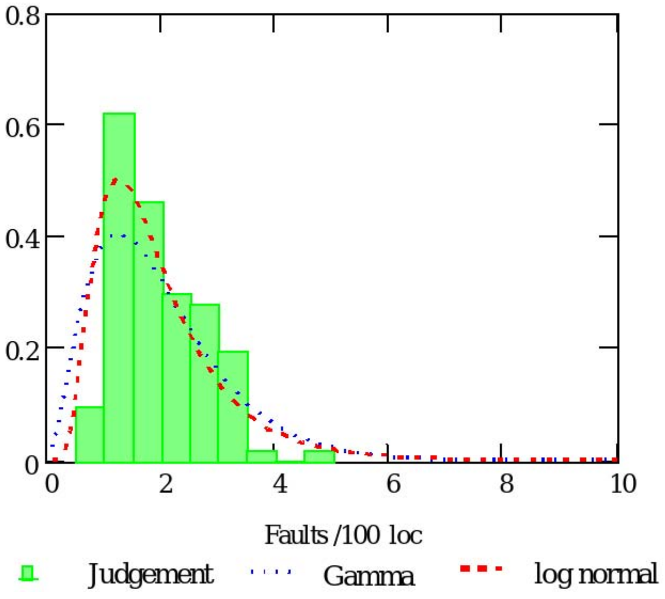


Figure 6. Fitting of judgement on fault rates.

Software Criticality Analysis A software criticality analysis [21] identifies disjoint software components and evaluates the criticality of each component to the safety of the overall system function. SCA is useful during development and configuration of systems as it can demonstrate the absence of failure propagation, and the criticality of components can be used to aid decisions on implementing additional barriers. For pre-developed software, the opportunity to change the code is limited, and retrospective assurance of safety integrity will be required on the existing code base. If the software was developed in a non-nuclear context, it is necessary to demonstrate that safety integrity of the software is consistent with nuclear software standards. This can be a major undertaking requiring additional documentation, reviews, analysis and testing. The results of the SCA can be used to focus safety assurance activities on the most critical components and hence minimise the cost of compliance.

The main activities of the SCA are:

- Identifying the software concerned and establishing an appropriate level of documentation.

- Assessing the impact on the safety function of failure of the software components by a form of Hazops.
- Ranking software components according to impact on the safety function. A possible approach is to use the Software Criticality Index technique described in [21]—this is a control flow analysis technique which assigns criticality levels to code modules depending on the criticality of the code that invokes them.
- Showing non-interference from non-critical functions and between software components and validating the SCA.

Integrity Static Analysis Integrity static analysis [18] is a lightweight static analysis technique which focuses on unsafe language constructs and covert flows. This analysis approach was motivated by two main factors: the presence of extensive field experience for many COTS[4] systems, and limited analysis resources available. The field experience is likely to detect most large and obvious faults that occur during typical execution of the program. However, specific vulnerabilities of the languages used and the particular domain of application have a less frequent manifestation that could remain undetected even after many hours of field experience. A full compliance analysis involving a formal specification of a large system is extremely expensive and time-consuming.

The analyses undertaken as part of integrity static analysis are shown in Table 1.

Table 1. Activities in software criticality analysis.

Unsafe Language Constructs	**Covert flows**
Function prototype declaration missing.	Resource sharing violations (semaphores, interrupts, etc.).
Use of "=" in conditional expressions.	Violation of program stack and register constraints (in assembler code).
No return value defined for a non-void function.	Pointer or array access outside the intended data structure.
No break between case statements.	Run-time exceptions (e.g. divide by zero).
Uninitialised variables (Possibly but not necessarily covert flows).	

The assessment of unsafe language constructs identifies potential vulnerabilities in C code by looking for deviations from published recommendations for C programming in safety-related applications [89, 57] and use of features of C identified in the ISO and ANSI standards as ill-defined or dangerous. It also includes checks for a variety of specific issues, such as

[4]Commercial Off-The-Shelf (COTS) systems are provided by third parties as generic systems, rather than being designed for the application in question.

the use of commonly misused constructs in C (such as "=" in conditional expressions).

Covert flow analysis examines the potential interference between different code functions. The most obvious covert mechanism in C or assembler code is the use of pointers (including their implicit use in array indexing). An incorrectly calculated pointer to a variable can give a procedure access to anywhere in the program's address space. Similarly, incorrect pointers to functions allow transfer of control to anywhere in the address space. The sharing of resources on a single processor and sharing of the stack give rise to other covert mechanisms. Static analysis was used to support an argument of the absence of these covert channels.

We observe that the existence of these covert mechanisms represents an error in the code, irrespective of its intended function. That is, we can argue that "whatever it is meant to do, it should not do this". We can thus carry out these analyses without a formal specification of the intended function, which is fortunate since this may not be available.

The overall procedure used in integrity static analysis is as follows:

1. *Identification of preliminary findings.* Preliminary findings identified as a result of tool analysis. In some cases (especially uninitialised variables), these can be further processed by automatic filters that remove some of the more obvious cases where there was no real problem.

2. *Provisional sentencing.* Manual or automatic pattern-based inspection of the preliminary findings to assign a provisional sentence. This preliminary sentencing is based on code inspection with limited domain knowledge. Where it does not seem possible to sentence the finding without extensive domain knowledge, an open sentence is recorded together with any information that could help the sentencing.

3. *Domain expert sentencing.* All the findings with a provisional sentence other than "no problem" are reported to the client for resolution.

4. *Review of the final sentencing.* The analysis team reviews the domain experts' sentencing, but the final decision on the solution to adopt is taken by the domain experts.

Use of Model Checking at Adelard At Adelard, model checking with SMV [3] has been used to complement analysis of the PLC design for a fuelling machine turret [19]. Our approach has been to demonstrate specific properties of the system rather than compliance with a full functional model. We demonstrated the following properties:

- *Stability* – that the control logic cannot be made to oscillate between states, and does not change state without a control input.

- *Mutual exclusion* between conflicting outputs.
- *Safety properties* – that movement is inhibited by limit stops and no movement is possible during maintenance.
- *Specific functional properties* – individual assertions that aspects of the behaviour are as required.

Focused Proof Adelard have successfully applied a lightweight formal methods approach called focused proof [25], in which only the most critical parts of the software are subject to formal proof, and the remaining parts are addressed by a number of other analyses to demonstrate either lack of interference or sufficiently correct operation. The technique is viable because the most critical modules turn out to be among the simplest. This code was proved using a manual Hoare-logic style approach; the remaining code was treated using the following methods:

- *Shared variable analysis* – identifying where non-critical modules can influence critical modules, using a modified version of Sparse [55].
- *Control flow analysis* – using a control flow graph (also constructed using Sparse) to identify all code potentially reachable in executing a given function, to ensure that the coverage of the proof is correct. Control flow analysis is also used to quickly identify any loops used in interrupts service routines which might lead to non-deterministic interrupt handling times.
- *Code slicing and code chopping* – used in two ways:
  - *Analysis of menu code* – a menu state machine may be used to set variables which control the critical part of the code. A code chop of each control variable can be used to show that the value chosen by the user corresponds to the value passed to the critical code, removing any need for further justification.
  - *Domain-specific proof* – identifying the code paths used in a particular application has the effect of hiding much of the complexity, especially where multiple configurations are accommodated through switch statements.
- *Monotonicity analysis* – if a module can be shown to implement a monotonic function, its testing becomes significantly simpler: behaviour between test points can be bounded by the monotonic behaviour. For functions with a binary output (e.g., alarm functions), the number of test points can be reduced even further. This approach is discussed in more detail in [20].

4 Model Checking in Safety Cases

One way of structuring a software safety case is to break down the safety argument by attribute, as shown below (these are derived from [5]).

Attributes of the service that are provided when the component functions correctly

- *Functional properties*: The primary functional behaviour of the component.
- *Timing properties:* The time allowed for the software to respond to given inputs or to periodic events, and/or the performance of the software in terms of transactions or messages handled per unit time.
- *Accuracy:* The required precision of the computed results.

Attributes dealing with the overall likelihood of malfunction

- *Reliability:* continuity of correct service. The probability that the software will perform to a specified requirement for a specified period of time under specified conditions.
- *Availability:* fraction of time for which a system is capable of fulfilling its intended purpose.

Attributes dealing with recovery from failure and changes

- *Maintainability:* ability to undergo modifications and repairs.

Behaviour with respect to certain classes of fault

- *Robustness:* The behaviour of the component in the event of external events: spurious (unexpected) inputs, hardware faults and power supply interruptions, either in the component itself or in connected devices.
- *Failure integrity:* The probability that an internal failure can be detected externally.
- *Usability:* Avoidance of error and delay when the component interfaces with a human operator.

Consequences of behaviour, different types of loss

- *Security:* Prevention of loss of component confidentiality, availability and integrity due to external attack.
- *Non-interference:* The absence of interference with other systems from an operational component.

Of these attributes, some are more appropriate than others to address through model checking. Conventional model checking approaches can deal with functional properties, and approaches based on timed automata may be able to handle timing properties sufficiently provided a strong enough model of the execution environment can be created. It may be possible to handle accuracy using infinite-state or SMT[5] - based techniques.

Maintainability, as a meta-property, may be better handled by analysis of the safety case itself. Does the safety case demonstrate that every attribute continues to meet its requirements over the lifetime of the system? A semi-formal approach to structuring a safety case using an inductive argument over system changes due to age or damage was presented in the ASDES project [20].

Reliability and availability are not immediately amenable to model-checking style approaches, although discovery and elimination of other bugs will improve the worst case bounds generated by reliability modelling techniques [22, 23]. Robustness and failure integrity may be handled by techniques which deliberately perturb the inputs or the internal logic (failure mode injection) and measure the resulting faults; one such technique is FSAP/NuSMV-SA, based on the NuSMV model checker [2].

The remaining attributes (usability, security, and non-interference) are too application-specific to be immediately considered, although certain aspects of each will be amenable to modelling (e.g., absence of certain interfering behaviours).

[5]Satisfiability Modulo Theorems (SMT) is an extension of Satisfiability (SAT) solver technology to handle propositional formulae in which some symbols are defined by assertions in some other theory, typically linear inequalities.

5 Conclusions

This report covers the state of the art in using formal methods in connection with NPP I&C systems and other safety critical systems. We have covered topics on safety assessment, formal methods in analysis of safety critical systems, as well as previous approaches to software and automation analysis in NPPs.

The main focus of the MODSAFE project has been to evaluate the applicability of model checking methods to the analysis of safety critical NPP I&C systems. The two case studies done in 2007 have demonstrated that the approach is feasible for the design validation of the two systems concerned: the arc protection system, and the emergency cooling system [110]. In both case studies the controller was modelled using the NuSMV model checker, and placed in an environment model modelling an abstraction of the process being controlled. The models were then model checked, and proved to work according to their specifications formalised in temporal logic from the design documents used. For the systems considered, the scalability of the model checker to analyse the system on all relevant input sequence combinations was sufficient. On the NPP I&C systems analysis side the approach closest to the one applied in the MODSAFE case studies is [117], where model checking is used as a design aid for developing the control system in the first place, instead of analysing its design after the development.

The range of software based systems used in NPP I&C systems is very large, from PLCs and small smart devices with small embedded programs to larger software systems like UPSs and automation systems (DCS) with a large platform software component. We have so far focused on small safety critical control systems where model checking seems to be fairly well applicable, at least with the kinds of systems that have only limited process feedback like emergency shutdown systems. Scaling up to larger systems is an interesting challenge, and requires more research. For example, more advanced methods for dealing with real-time aspects of systems as well as methods for comparing a system at different levels of abstraction are called for. It may be also the case that model checking should be applied in conjunction with other formal safety critical systems analysis techniques to scale up to larger systems.

References

[1] Caveat, `http://www-list.cea.fr/labos/gb/LSL/caveat/`.

[2] FSAP/NuSMV-SA. Available from `http://sra.itc.it/tools/fsap`.

[3] The SMV system. `http://www.cs.cmu.edu/~modelcheck/smv.html`.

[4] *ASCAD – Adelard Safety Case Development Manual*, 1998. ISBN 0 9533771 0 5.

[5] Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[6] Ball, T., Cook, B., Levin, V., and Rajamani, S. K. SLAM and static driver verifier: Technology transfer of formal methods inside Microsoft. In *Proceedings of the 4th International Conference on Integrated Formal Methods (IFM 2004)*, volume 2999 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 2004.

[7] Ball, T. and Rajamani, S. K. The SLAM project: debugging system software via static analysis. In *Conference Record of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2002)*, pages 1–3. Association for Computing Machinery, 2002.

[8] Bauer, N., Engell, S., Huuck, R., Lohmann, S., Lukoschus, B., Remelhe, M., and Stursberg, O. Verification of PLC programs given as sequential function charts. In Ehrig et al. [50], pages 517–540.

[9] Bauer, N., Huuck, R., Lukoschus, B., and Engell, S. A unifying semantics for sequential function charts. In Ehrig et al. [50], pages 400–418.

[10] Bedford, T. and Cooke, R. *Probabilistic risk analysis – foundations and methods.* Cambridge University Press, 2001.

[11] Behrmann, G., David, A., Larsen, K. G., Håkansson, J., Pettersson, P., Yi, W., and Hendriks, M. UPPAAL 4.0. In *QEST*, pages 125–126. IEEE Computer Society, 2006.

[12] Berard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., and Schnoebelen, P. *Systems and Software Verification. Model-Checking Techniques and Tools.* Springer, 2001.

[13] Bergerand, J. L. and Pilaud, E. Design and qualification of software for protection and control systems: the use of SAGA. In *SFEN International Conference on Operability of Nuclear Systems in Normal and Adverse Environments*, Lyon, 1989.

[14] Biere, A., Cimatti, A., Clarke, E., and Zhu, Y. Symbolic model checking without BDDs. In *TACAS'99*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999.

[15] Biere, A., Clarke, E., Raimi, R., and Zhu, Y. Verifying safety properties of a Power PC microprocessor using symbolic model checking without BDDs. In *CAV'99*, volume 1633 of *LNCS*, pages 60–71. Springer, 1999.

[16] Biere, A., Heljanko, K., Junttila, T., Latvala, T., and Schuppan, V. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5:5), 2006. (doi: 10.2168/LMCS-2(5:5)2006).

[17] Bishop, P. G. and Bloomfield, R. E. A methodology for safety case development. In *Safety-Critical Systems Symposium (SSS '98)*, Birmingham, UK, February 1998.

[18] Bishop, P. G., Bloomfield, R. E., Clement, T. P., Guerra, A. S. L., and Jones, C. Integrity static analysis of COTS/SOUP. In *Proceedings of SAFECOMP 2003*, pages 63–76, Edinburgh, UK, September 2003. Springer-Verlag.

[19] Bishop, P. G., Bloomfield, R. E., and Guerra, A. S. D. L. Quantification of safety system failure probability – task a: Reliability assessment of the siarls example. Technical Report D/189/4306/2, Adelard, June 2001.

[20] Bishop, P. G., Bloomfield, R. E., Guerra, A. S. L., Sheridan, D. J., and Tourlas, K. Asdes: Assessment of smart devices. Technical Report D/352/10502/1, Adelard, December 2006.

[21] Bishop, P., Bloomfield, R., Clement, T., and Guerra, S. Software criticality analysis of COTS/SOUP. In *Proceedings of SAFECOMP 2002*, pages 198–211, Catania, Italy, September 2002. Springer-Verlag.

[22] Bishop, P. and Bloomfield, R. A conservative theory for long-term reliability growth prediction. *IEEE Trans. Reliability*, 45(4):550–560, 1996. ISSN 0018-9529.

[23] Bishop, P. and Bloomfield, R. Using a log-normal failure rate distribution for worst case bound reliability prediction. In *ISSRE 2003*, pages 237–245, Denver, Colorado, USA, November 2003.

[24] Bjesse, P., Leonard, T., and Mokkedem, A. Finding bugs in an Alpha microprocessor using satisfiability solvers. In *Proceedings of 13th International Conference on Computer Aided Verification, (CAV'2001)*, pages 454–464. Springer-Verlag, 2001. LNCS 2102.

[25] Bloomfield, R. and Sheridan, D. Applying formal methods at all SILs. In *INUCE Control and Instrumentation conference*, Manchester, September 2007.

[26] Bloomfield, R. E. and Guerra, S. Process modelling to support dependability arguments. In *2002 International Conference on Dependable Systems and Networks (DSN 2002)*, pages 113–122, Washington, DC, USA, June 2002.

[27] Bradley, A. R. and Manna, Z. *The Calculus of Computation: Decision Procedures with Applications to Verification.* Springer, 2007.

[28] Brinksma, E. and Tretmans, J. Testing Transition Systems: An Annotated Bibliography. In *Summer School MOVEP'2k – Modelling and Verification of Parallel Processes*, pages 44–50, Nantes, July 2000.

[29] Brinksma, H. and Mader, A. H. Verification and optimization of a PLC control schedule. In Havelund, K., Penix, J., and Visser, W., editors, *7th Int. SPIN Workshop on Model Checking of Software, Stanford Univ., California*, volume LNCS 1885, pages 73–92, Berlin, August 2000. Springer-Verlag.

[30] Bryant, R. E. Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM Computing surveys*, 24(3):293–318, 1992.

[31] Burch, J., Clarke, E., McMillan, K., Dill, D., and Hwang, L. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.

[32] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L. J. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[33] CAA Safety Regulation Group. *CAP 670—ATS Safety Requirements*, 1998.

[34] Canet, G., Couffin, S., Lesage, J.-J., Petit, A., and Schnoebelen, Ph. Towards the automatic verification of PLC programs written in Instruction List. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC 2000)*, pages 2449–2454, Nashville, Tennessee, USA, October 2000. Argos Press.

[35] Carloganu, A. and Raguideau, J. Claire: An event-driven simulation tool for test and validation of software programs. In *International Conference on Dependable Systems and Networks (DSN'02)*, page 538. IEEE, June 2002.

[36] Caspi, P., Pilaud, D., Halbwachs, N., and Plaice, J. Lustre: A declarative language for programming synchronous systems. In *POPL*, pages 178–188, 1987.

[37] Cho, J., Yoo, J., and Cha, S. D. NuEditor - A tool suite for specification and verification of NuSCR. In Dosch, W., Lee, R. Y., and Wu, C., editors, *SERA*, volume 3647 of *Lecture Notes in Computer Science*, pages 19–28. Springer, 2004.

[38] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. NuSMV 2: An opensource tool for symbolic model checking. In *CAV'02*, volume 2404 of *LNCS*, pages 359–364. Springer, 2002.

[39] Cimatti, A., Roveri, M., and Sheridan, D. Bounded verification of past LTL. In Hu, A. J. and Martin, A. K., editors, *Formal Methods in Computer-Aided Design; 5th International Conference, FMCAD 2004*, Austin, TX, USA, November 2004. Springer.

[40] Clarke, E. M., Grumberg, O., and Peled, D. A. *Model Checking.* The MIT Press, 1999.

[41] Clarke, E. and Emerson, E. Design and synthesis of synchronization of skeletons using branching time temporal logic. In *Proceedings of the IBM Workshop on Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1981.

[42] Copty, F., Fix, L., Fraer, R., Giunchiglia, E., Kamhi, G., Tacchella, A., and Vardi, M. Y. Benefits of bounded model checking at an industrial setting. In *Proceedings of 13th International Conference on Computer Aided Verification, (CAV'2001)*, pages 436–453. Springer-Verlag, 2001. LNCS 2102.

[43] Courtois, P.-J. A dependability justification framework for NPP digital instrumentation and control systems. Technical report, Cemsis Project, March 2004.

[44] Crow, J., Javaux, D., and Rushby, J. Models and mechanized methods that integrate human factors into automation design. In Abbott, K., Speyer, J.-J., and Boy, G., editors, *HCI-AERO 2000: International Conference on Human-Computer Interaction in Aeronautics*, pages 163–168, Toulouse, France, sep 2000.

[45] de Vries, R. G. and Tretmans, J. On-the-fly conformance testing using SPIN. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(4):382–393, 2000.

[46] Dierks, H., Fehnker, A., Mader, A. H., and Vaandrager, F. W. Operational and logical semantics for polling real-time systems. In Anders, P. R. and Rischel, H., editors, *7th Int. Symp. on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT), Lyngby, Denmark*, volume LNCS 1486, pages 29–40, Berlin, September 1998. Springer-Verlag.

[47] Dierks, H. PLC-Automata: A new class of implementable real-time automata. In Bertran, M. and Rus, T., editors, *ARTS*, volume 1231 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 1997.

[48] Dill, D. L. The Murφ verification system. In *Proceedings of the 8th International Conference on Computer Aided Verification (CAV 1996)*, volume 1102 of *Lecture Notes in Computer Science*, pages 390–393. Springer-Verlag, 1996.

[49] Eén, N. and Sörensson, N. Temporal induction by incremental SAT solving. In Strichman, O. and Biere, A., editors, *Electronic Notes in Theoretical Computer Science*, volume 89(4). Elsevier, July 2003.

[50] Ehrig, H., Damm, W., Desel, J., Große-Rhode, M., Reif, W., Schnieder, E., and Westkämper, E., editors. *Integration of Software Specification Techniques for Applications in Engineering, Priority Program SoftSpez of the German Research Foundation (DFG), Final Report*, volume 3147 of *Lecture Notes in Computer Science*. Springer, 2004.

[51] Filliâtre, J.-C. and Marché, C. The why/krakatoa/caduceus platform for deductive program verification. In Damm, W. and Hermanns, H., editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 173–177. Springer, 2007.

[52] Frey, G. *Design and formal Analysis of Petri Net based Logic Control Algorithms*. PhD thesis, Universität Kaiserslautern, 2002.

[53] Frey, G. and Litz, L. Formal methods in PLC programming. In *IEEE International Conference on Systems, Man and Cybernetics (SMC 2000)*, pages 2431–2436, Nashville (TN), USA, October 2000.

[54] Govier, T. *A Practical Study of Argument*. Wadsworth, 1988.

[55] Group, L. K. Sparse—a semantic parser for C. `http://www.kernel.org/pub/linux/kernel/people/josh/sparse/`.

[56] Haapanen, P., Korhonen, J., and Pulkkinen, U. Licensing process for safety-critical software-based systems. Technical Report STUK-YTO-TR 171, STUK, Helsinki, 2000.

[57] Hatton, L. *Safer C.* McGraw-Hill, 1994.

[58] Heiner, M. and Menzel, T. A Petri net semantics for the PLC language Instruction List; IEEE Workshop on Discrete Event Systems, 1998. Available from `http://citeseer.ist.psu.edu/heiner98petri.html`.

[59] Heljanko, K., Junttila, T., and Latvala, T. Incremental and complete bounded model checking for full PLTL. In *CAV'05*, volume 3576 of *LNCS*, pages 98–111. Springer-Verlag, July 2005.

[60] Heljanko, K., Junttila, T. A., Keinänen, M., Lange, M., and Latvala, T. Bounded model checking for weak alternating büchi automata. In Ball, T. and Jones, R. B., editors, *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 95–108. Springer, 2006.

[61] Holzmann, G. J. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[62] Huima, A. Implementing Conformiq Qtronic. In Petrenko, A., Veanes, M., Tretmans, J., and Grieskamp, W., editors, *TestCom/FATES*, volume 4581 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2007.

[63] Huuck, R. *Software Verification for Programmable Logic Controllers.* PhD thesis, Christian-Albrechts-Universität zu Kiel, Germany, 2003.

[64] IAEA (International Atomic Energy Agency, Vienna. *Safety of Nuclear power plants: Design—Requirements*, 2000. NS-R-1.

[65] IAEA (International Atomic Energy Agency, Vienna. *IAEA Safety Glossary: Terminology Used in Nuclear Safety and Radiation Protection*, 2007.

[66] *Modelling of PLC behaviour by means of timed net Condition/Event systems*, 1997.

[67] International Electrotechnical Commission. *IEC 60812 Analysis techniques for system reliability—procedure for failure mode and effects analysis (FMEA)*, 2 edition, January 2001.

[68] International Standards Organization. *ISO 8402. Quality vocabulary*, 1986.

[69] Jackson, P. and Sheridan, D. Clause form conversions for boolean circuits. In *Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*. Springer-Verlag, December 2004.

[70] Jee, E., Yoo, J., and Cha, S. D. Control and data flow testing on function block diagrams. In Winther, R., Gran, B. A., and Dahll, G., editors, *SAFECOMP*, volume 3688 of *Lecture Notes in Computer Science*, pages 67–80. Springer, 2005.

[71] Jimnez-Fraustro, F. and Rutten, E. A synchronous model of the PLC programming language ST. Available from `citeseer.ist.psu.edu/393006.html`.

[72] Jimnez-Fraustro, F. and ric Rutten. A synchronous model of IEC 61131 PLC languages in signal. In *ECRTS*, pages 135–142. IEEE Computer Society, 2001.

[73] Kaldewaij, A. *Programming: the derivation of algorithms.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[74] Kelly, T. P. and McDermid, J. A. Safety case construction and reuse using patterns. In Daniel, P., editor, *SAFECOMP97: the 16th International Conference on Computer Safety, Reliability and Security York, UK, 7-10 September 1997*, pages 55–69. Springer, 1997.

[75] Kopustinskas, V., Kirchsteiger, C., Soubies, B., Daumas, F., Gassino, J., Pron, J., Rgnier, P., Märtz, J., Baleanu, M., Miedl, H., Kersken, M., Pulkkinen, U., Koskela, M., Haapanen, P., Järvinen, M., Bock, H., and Dreves, W. Benchmark exercise of safety evaluation of computer based systems (BE-SECBS). In *FISA-2003 / EU Research in Reactor Safety*, Luxembourg, November 2003.

[76] Latvala, T., Biere, A., Heljanko, K., and Junttila, T. Simple bounded LTL model checking. In *Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2004)*, volume 3312 of *Lecture Notes in Computer Science*, pages 186–200. Springer-Verlag, 2004.

[77] Latvala, T., Biere, A., Heljanko, K., and Junttila, T. Simple is better: Efficient bounded model checking for past LTL. In *Proceedings of the 6th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI 2005)*, volume 3385 of *Lecture Notes in Computer Science*, pages 380–395. Springer-Verlag, 2005.

[78] Loeis, K., Bani Younis, M., and Frey, G. Application of symbolic and bounded model checking to the verification of logic control sys-

tems. In *10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2005)*, pages 247–250, Catania, Italy, September 2005.

[79] Lüttgen, G. and Carreño, V. Analyzing mode confusion via model checking. In Dams, D., Gerth, R., Leue, S., and Massink, M., editors, *Theoretical and Practical Aspects of SPIN Model Checking (SPIN '99)*, volume 1680 of *Lecture Notes in Computer Science*, pages 120–135, Toulouse, France, September 1999. Springer-Verlag.

[80] Mader, A. H. A classification of PLC models and applications. In Boel, R. and Stremersch, G., editors, *5th Int. Workshop on Discrete Event Systems (WODES) – Discrete Event Systems, Analysis and Control, Ghent, Belgium*, pages 239–247, Boston, Massachusetts, August 2000. Kluwer Academic Publishers.

[81] Mader, A. H., Brinksma, H., Wupper, H., and Bauer, N. Design of a PLC control program for a batch plant - VHS Case Study 1. *European Journal of Control*, 7(4):416–439, 2001.

[82] Mader, A. H. and Wupper, H. Timed automaton models for simple programmable logic controllers. In *11th Euromicro Conf. on Real-Time Systems, York, UK*, pages 114–122, Los Alamitos, California, June 1999. IEEE Computer Society.

[83] Mader, A. Precise timing analysis of PLC applications two small examples. Unpublished manuscript. Available from `http://citeseer.ist.psu.edu/mader00precise.html`.

[84] Mader, A. and Wupper, H. What is the formal method for PLC applications? In *Proceedings ADPM 2000: 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems,* September 18-19, 2000, Dortmund, Germany, 2000.

[85] Mäkelä, M. Maria: Modular reachability analyser for algebraic system nets. In *Proceedings of the 23rd International Conference on Application and Theory of Petri Nets (ICATPN 2002)*, volume 2360 of *Lecture Notes in Computer Science*, pages 434–444. Springer-Verlag, 2002.

[86] Manna, Z. and Pnueli, A. *Temporal Verification of Reactive Systems: Safety.* Springer, 1995.

[87] McMillan, K. L. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[88] Miller, S. P., Barber, S., Carlson, T., Lempia, D., and Tribble, A. A methodology for improving mode awareness in flight guidance de-

sign. In *Proceedings of the 21st Digital Avionics Systems Conference (DASC'02)*, Irvine, CA, 2002.

[89] MISRA. *Guidelines for the use of the C language in vehicle based software*, 1998.

[90] Moon, I. Modeling programmable logic controllers for logic verification, 1994.

[91] Oak Ridge National Laboratory. *Emerging Technologies in Instrumentation and Controls: An Update*, January 2006.

[92] Olderog, E.-R. Correct Real-Time Software for Programmable Logic Controllers. In *Correct System Design - Recent Insights and Advances*, volume 1710 of *Lecture Notes in Computer Science*, pages 342–362. Springer, 1999.

[93] Olderog, E.-R. and Dierks, H. Moby/RT: A Tool for Specification and Verification of Real-Time Systems. *Journal of Universal Computer Science*, 9(2):88–105, February 2003.

[94] Parnas, D., van Schouwen, A., and Kwan, P. Evaluation of safety-critical software. *Communications of the ACM*, 33(6):636–648, 1990.

[95] Pavey, D. J. Final public synthesis report. Technical report, CEMSIS (Cost Effective Modernisation of Systems Important to Safety), 2004. `http://www.cemsis.org/`.

[96] Peled, D. A. *Software Reliability Methods*. Springer, 2001.

[97] Quielle, J. and Sifakis, J. Specification and verification of concurrent systems in CESAR. In *Proceedings of the 5th International Symposium on Programming*, pages 337–350, 1981.

[98] Rossi, O. and Schnoebelen, Ph. Formal modeling of timed function blocks for the automatic verification of Ladder Diagram programs. In Engell, S., Kowalewski, S., and Zaytoon, J., editors, *Proceedings of the 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems (ADPM 2000)*, pages 177–182, Dortmund, Germany, September 2000. Shaker Verlag.

[99] Safety Assessment Principles for Nuclear Facilities, 2006 Edition. Available from `http://www.hse.gov.uk/nuclear/saps/saps2006.pdf`.

[100] Sheeran, M., Singh, S., and Stålmarck, G. Checking safety properties using induction and a SAT-solver. In Jr., W. A. H. and Johnson, S. D., editors, *Proceedings of the 3rd International Conference on Formal*

Methods in Computer-Aided Design (FMCAD '2000), volume 1954 of *Lecture Notes in Computer Science*, pages 108–125, Austin, Texas, USA, November 2000. Springer.

[101] Song, M. J., Koo, S. R., and Seong, P.-H. Development of a verification method for timed function blocks using ESDT and SMV. In *HASE*, pages 285–286. IEEE Computer Society, 2004.

[102] STUK. *Radiation and Nuclear Safety Authority Finland, Instrumentation systems and components at nuclear facilities*, 2002. Available on-line: `http://www.stuk.fi/saannosto/YVL5-5e.html`.

[103] Systems, R. Model-based testing and validation with Reactis. Technical whitepaper, available from `http://www.reactive-systems.com/papers/bcsf.pdf`.

[104] Tapken, J. and Dierks, H. MOBY/PLC - graphical development of PLC-Automata. In Ravn, A. P. and Rischel, H., editors, *FTRTFT*, volume 1486 of *Lecture Notes in Computer Science*, pages 311–314. Springer, 1998.

[105] Technologies, E. SCADE tool. `http://www.esterel-technologies.com`.

[106] Toulmin, S. E. *The Uses of Argument.* Cambridge University Press, 1958.

[107] Tretmans, J. *A Formal Approach to Conformance Testing.* PhD thesis, University of Twente, Enschede, The Netherlands, 1992.

[108] Turk, A. L., Probst, S. T., and Powers, G. J. Verification of real time chemical processing systems. In Maler, O., editor, *HART*, volume 1201 of *Lecture Notes in Computer Science*, pages 259–272. Springer, 1997.

[109] UK Ministry of Defence. *Def Stan 00-56—Safety Management Requirements for Defence Systems*, 3 edition, December 2004.

[110] Valkonen, J., Pettersson, V., Björkman, K., Holmberg, J.-E., Koskimies, M., Heljanko, K., and Niemelä, I. Model-Based Analysis of an Arc Protection and an Emergency Cooling System – MODSAFE 2007 Work Report. VTT Working Papers 93, 2008.

[111] Varpaaniemi, K., Heljanko, K., and Lilius, J. PROD 3.2 - An advanced tool for efficient reachability analysis. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV 1997)*, volume 1254 of *Lecture Notes in Computer Science*, pages 472–475. Springer-Verlag, 1997.

[112] Wahlström, B., Ventä, O., and Valkonen, J. Proving absence of CCFs; a case for open source. In *Technical Meeting on Common-Cause Failures in Digital Instrumentation and Control Systems of Nuclear Power Plants*, Bethesda, Maryland, USA, June 2007.

[113] Wassyng, A. and Lawford, M. Lessons learned from a successful implementation of formal methods in an industrial project. In Araki, K., Gnesi, S., and Mandrioli, D., editors, *FME*, volume 2805 of *Lecture Notes in Computer Science*, pages 133–153. Springer, 2003.

[114] Weaver, R., Fenn, J., and Kelly, T. A pragmatic approach to reasoning about the assurance of safety arguments. In *Proceedings of the 8th Australian workshop on Safety critical systems and software*, pages 57–67, Canberra, Australia, October 2003.

[115] Willems, H. X. Compact timed automata for PLC programs. Technical report, CiteSeer [`http://cs1.ist.psu.edu/cgi-bin/oai.cgi`] (United States), 1999.

[116] Winter, M. Instrumentation and control system, yk1 - basic professional training course on nuclear safety finland, part iv, lecture 11, 2003.

[117] Yoo, J., Cha, S., Son, H. S., Kim, C. H., and Lee, J.-S. PLC-Based safety critical software development for nuclear power plants. In Heisel, M., Liggesmeyer, P., and Wittmann, S., editors, *SAFECOMP*, volume 3219 of *Lecture Notes in Computer Science*, pages 155–165. Springer, 2004.

[118] Yoo, J., Cha, S., Kim, C. H., and Song, D. Y. Synthesis of FBD-based PLC design from NuSCR formal specification. *Reliability Engineering and System Safety*, 87(2):287–294, 2005.

[119] Yoo, J., Kim, T., Cha, S. D., Lee, J.-S., and Son, H. S. A formal software requirements specification method for digital nuclear plant protection systems. *Journal of Systems and Software*, 74(1):73–83, 2005.

[120] Younis, M. B. and Frey, G. Formalization of existing PLC programs: A survey. In *Proceedings of CESA 2003*, pages Paper No.S2–R–00–0239, Lille, France, July 2003.

[121] Zhang, W. Verification of operator procedures by model checking. Technical Report HWR-582, OECD Halden Reactor Project, April 1999.

[122] Zhang, W. Model checking as an aid to procedure design. Technical Report HWR-648, OECD Halden Reactor Project, January 2001.

ISBN 978-951-38-7158-1 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)