Model Checking Operator Procedures

Wenhui Zhang

Institute for Energy Technology, P.O.Box 173, N-1751 Halden, Norway
Wenhui.Zhang@hrp.no

Abstract. Operator procedures are documents telling operators what to do in various situations. They are widely used in process industries including the nuclear power industry. The correctness of such procedures is of great importance. We describe how model checking can be used to detect potential errors and to verify properties of operator procedures. As there could be problems with modelling and model checking large systems, incremental modelling and verification is proposed as a strategy to help overcome these problems. A case study is presented to show how model checking (with the model checker Spin [5]) and the incremental strategy work in practise.

1 Introduction

Operator procedures are documents telling operators what to do in various situations. There are procedures to be used in normal operation situations, in case of disturbances and in case of emergencies. Operator procedures are widely used in process industries including the nuclear power industry. The correctness of such procedures is of great importance. The following is a quote from the report on the accident at the Three Mile Island [6]: "A series of events - compounded by equipment failures, inappropriate procedures, and human errors and ignorance - escalated into the worst crisis yet experienced by the nation's nuclear power industry." Such inappropriate procedures included procedures for loss of coolant accident and for pressurizer operations [7].

To assess the correctness of operator procedures, we need correctness requirements. The correctness requirements of an operator procedure include goals that must be achieved upon completion of an execution of the procedure, safety constraints that must be maintained during an execution of the procedure and general structural requirements. For achieving goals, necessary and effective instructions must be explicitly specified in the procedure. In addition, time and resource constraints for carrying out the instructions must be satisfiable. For maintaining safety constraints, preparation steps for the goal-oriented instructions must be included in the procedure. It is also important that instructions are specified in the correct order and time and resource constraints for carrying out the preparation steps are satisfiable. Structural requirements make a procedure simple and easy to understand. They include no undefined references and unreachable instructions.

Violation of the correctness requirements could result in many types of errors. Although all types of violations are important with respect to correctness, formal verification techniques only offer a partial solution to the verification of procedures. Problems related to time and resource constraints are more complicated and are not considered for verification in this paper. We divide the types of errors to be considered for verification into three categories: structural errors, execution errors and consistency errors. *Structural errors* include undefined references and unreachable instructions as mentioned earlier. A reference is undefined, if it refers to no instruction in the procedure. The cause of an undefined reference could be that the reference is wrong or the instruction to which it refers is missing. An instruction is unreachable, if it cannot be reached from any potential execution of the procedure. For instance, if there is a block of instructions without any reference from outside of the block, the instructions in the block will be unreachable (unless one of the instructions in the block is the starting point of the procedure). A more complicate situation is that there are instructions refer to this block, but the conditions for using the references never become true in all potential executions of the procedure. *Execution errors* are errors that prevent an execution to be completed. They include deadlocks and infinite loops. There is a deadlock, if the operator stops at some point and is not able to do anything in order to complete the procedure. For instance, the operator may be waiting for a condition which never becomes true. There is an infinite loop, if the operator is captured by a loop and cannot jump out of the loop by following the instructions of the procedure. The causes for infinite loops could be wrong actions, wrong conditions or wrong references in goto instructions, and other types of problems. *Consistency errors* indicate inconsistency between the set of instructions of the procedure and given specifications. They include violation of assertions (representing conditions for performing actions), violation of invariants and unreached goals. The basic question is how to detect potential errors in operator procedures. In this paper, we discuss using model checking for this purpose.

2 Verification Approach

To begin with, we have a procedure that needs to be verified. It is however not sufficient to analyse the procedure alone. We have to take into consideration the environment in which the procedure is used. As our work is related to procedure correctness in power plants, in the following, we refer to a power plant as the environment of the procedure.

2.1 Model Checking

The idea of using model checking is as follows. When we design a procedure, we do so in a natural or semi-formal language. When the procedure is executed, the operator (in the following, the meaning of an operator is defined as a person or a robot that strictly follows the procedure) gets responses from the plant as the procedure is being carried out. Such a procedure, the set of the possible initial states of the plant, the plant processes and the interaction between these processes determine a logical structure. On the other hand, there could be correctness conditions for the plant and the interaction between the operator and the plant. For instance, we may want to express the following: "Pump-A" must not be running at the time one starts repairing "Valve-A" (an assertion at a point of an execution of the procedure), "Valve-B" is closed unless "Valve-C" is closed (an invariant in an execution of the procedure), and the state of the plant will become "Normal" after starting the procedure (a liveness property of the procedure). These conditions can be verified against the logical structure.

2.2 Verification and Error Detection

As explained above, in order to be able to verify a procedure, there are two main tasks: modelling the logical structure and verification with respect to correctness conditions. The modelling task is to create the following processes:

– A model-procedure process, which is used to model the procedure (or the operator in a process of executing the procedure).
– A set of plant processes, which are used to model the physical processes of the plant (or assumptions on the physical processes).
– A set of interaction processes, which are used to model the interaction between the model-procedure process and the plant processes.
– An initialisation process, which is a process for choosing an initial state from the set of possible initial states of the plant.

The following are the principles for modelling:

– The model-procedure process communicates only with the interaction processes and does not directly read or update variables representing the system state. We impose these restrictions to the model-procedure process because it is very helpful with a clear separation of the consequences of the actions of the model-procedure process and the consequences of the actions of other processes.
– The interaction processes communicate with the model-procedure process and the plant processes. They may read and update the system state and may also initiate and activate some of the plant processes.
– The plant processes communicate with the interaction processes and may read and update the system state. Sometimes synchronisation mechanism is necessary in order to ensure the ability of the interaction processes to take appropriate actions immediately after changes are made to the system state by the plant processes.
– The initialisation process updates the system state before other processes read or update the system state.

After these processes are created, we can use a model checker to check errors of the types described in the previous section. The techniques for checking errors are as follows:

– For detecting undefined references, we check syntactic errors.
– For detecting deadlocks and infinite loops, we check execution errors of the procedure (using an auxiliary variable to indicate the completion of an execution of the procedure).
– For detecting unreachable instructions, it is not necessary to use any special technique, since such problems can also be reported by checking execution errors (unreachable instructions can be detected in cases where no execution errors are found).
– For detecting violation of conditions for performing actions, we need assertions describing conditions for relevant actions and we check whether the procedure is consistent with respect to the assertions.
– For detecting violation of invariants, we need invariants and we check whether the procedure is consistent with respect to the invariants (i.e. whether they hold in all reachable states of the model).
– For detecting unreached goals, we need goals and we check whether the procedure is consistent with respect to the goals (i.e. whether they can be reached in all reachable paths of the model).

As the verification can be carried out by using fully automated model checking tools, the verification task is simple. *The major problems* of model checking operator procedures are the complexity of the modelling of the relevant processes of the power plant and the complexity of the models. Because there are many processes that interfere with each other in the power plant, a model of the relevant processes of the power plant can be very complicated. On the other hand, it is not necessary to have a complete model for using model checking. The necessary and optimal complexity depends on the properties to be verified. In order to limit the complexity, one has to make a decision on how complicated the plant process model should be. This is a difficult decision, because a complicated model may not be suitable (too large) for automated model checking and a too much simplified model may not be sufficient for model checking of certain properties. There has been a lot work on how to overcome the complexity of model checking by using algorithmic techniques (e.g. as in [2] and [12]) to minimise the size of the internal representation of the models and the transition relations and by using rule based methods (e.g. as in [1] and [3]) for decomposing the verification tasks. In the following, we take a practical approach and propose an incremental strategy for modelling and verification.

2.3 Incremental Modelling and Verification

The motivation for incremental modelling and verification is to overcome the complexity of modelling and the problem of deciding how complicated the plant process model should be. The basic idea is to divide the modelling process in different phases and perform model checking within each phase.

Phase 1 We create a model of the procedure (or an operator in a process of executing the procedure) and check syntactic errors in order to detect undefined references.

Phase 2 We derive assumptions on the interaction between the operator and the power plant from the procedure text and create an interaction model. In this phase, we check whether there are execution errors, in order to detect deadlocks and infinite loops, and to detect unreachable instructions in case there are no deadlocks and infinite loops.

Phase 3 We add an initialisation process and refine the interaction model by revising the set of relevant values for each of the variables representing involved plant units and other relevant aspects. We check whether there are execution errors and whether there are consistency errors with respect to given specifications (assertions, invariants and goals).

Phase 4 We add a plant process model and refine the interaction model accordingly. The plant process model should capture essential aspects of relevant physical processes and should be modelled as simple as possible. If the number of relevant processes is large, some of the (possibly less important) processes may be left to later phases. In this phase, we also check whether there are execution errors and consistency errors.

Phase 5, 6, ... In phase 5, we refine the plant process model and check the same categories of errors as in phase 4. The refinement may include two aspects, to add details into existing plant processes and to extend the model with new processes. In the latter case, we also need to modify the interaction processes. We may continue to refine the plant process model in phase 6, 7 and so on, until we are satisfied with the plant process model or until the model is too complicated for model checking.

2.4 Process Knowledge

Process knowledge is very important for verification of procedures. We need process knowledge for formulating correctness requirements such as safety constraints and goals. In addition, there are different needs of process knowledge for modelling procedures and for modelling relevant plant processes in the different phases.

- Phase 1: The need for process knowledge is relatively moderate (some knowledge for understanding terminologies is needed). The need may depend on how well procedures are structured and whether the terminologies used in the procedures are standardised.
- Phase 2: The need for process knowledge is basically the same as in phase 1.
- Phase 3: The need for process knowledge is increasing. One needs to know under what conditions the procedure are assumed to be applied. One also needs to know how many different values are necessary for the states of certain plant objects.
- Phase 4: One needs to know how the states of different plant objects change and must be able to catch the essence of these changes with simple models.
- Phase 5, 6, ...: The need for process knowledge is basically the same as in phase 4.

2.5 Summary

The tasks of modelling in the different phases of the incremental modelling and verification are as follows:

- Phase 1: Create a model of the procedure.
- Phase 2: Add an interaction model.
- Phase 3: Add an initialisation process and refine the interaction model.
- Phase 4: Add a model of the physical processes.
- Phase 5,6,...: Refine the previously created plant and interaction models.

The verification sub-tasks to be performed in the different phases are summarised in the following table:

	phase 1	phase 2	phase 3	phase 4	phase 5,6,...
check syntactic errors	x				
check execution errors		x	x	x	x
check assertion violations			x	x	x
check invariant violations			x	x	x
check unreached goals			x	x	x

Remark The model of the procedure remains the same in all phases (unless an error is detected and the procedure has to be changed). Only the models of physical processes and the interaction processes are modified in the different phases. The sub-steps (verification sub-tasks) within each phase are ordered according to increasing difficulty of correctness specifications. It is possible to rearrange the sub-steps or to combine all or some sub-steps into one big step, if it is desirable.

Discussion Theoretically, if the model in phase 4 is accurate enough and there is no limitation on computational resources, all errors of the considered types could be detected. However, it is difficult to decide how accurate a model should be. Therefore we create a simple model (or make simple assumptions) to begin with and increase the accuracy of the model step by step and detect potential errors at the same time. The following arguments explain why it is sensible to divide the modelling and verification in so many phases.

- There is a clear distinction between the verification sub-tasks in phase 1 and that in phase 2.
- There is a clear distinction between the need for process knowledge in phase 2 and that in phase 3. In addition, depending on how many plant states are possible initial states and how many values are relevant for each of the involved plant units, the verification sub-tasks in phase 3 can be much more time consuming than that in phase 2.
- The modelling task is much more complicated in phase 4 than that in phase 3 and it also requires much more process knowledge in phase 4. In addition, the verification sub-tasks in phase 4 are usually much more time consuming than that in phase 3.

– Phases 5, 6 and so forth are meant to be additional phases for refining the previously created plant process model and performing model checking in order to increase confidence on the correctness and to detect potential errors that might have escaped the previous analysis.

This strategy is especially helpful in early phases of procedure evaluation when there could be many errors and a large portion of these errors can be detected and removed at early phases of the strategy (with simple models and short model checking time).

3 A Case Study

The case study is verification of an operator procedure with seeded errors. The purpose of this case study is to show the general usefulness of model checking and the incremental strategy of modelling and verification. The operator procedure considered here is "PROCEDURE D-YB-001 — The steam generator control valve opens and remains open" [11]. It is a disturbance procedure to be applied when one of the steam generator control valves (located on the feed water side) opens and remains open.

3.1 The Procedure

The primary goal of the procedure is to repair a defected control valve (there are 4 control valves) that has a problem (i.e. opens and remains open). The basic action sequence of the operator is as follows:

– Start isolating the steam generator.
– Manipulate the primary loop.
– Continue isolation of the steam generator.
– Repair the steam generator control valve and prepare to increase the power.
– Increase the power.

The description of the development of the disturbance is as follows:

– Trouble signal is initiated on steam generator level rise $> 3.30m$.
– Valve RL33/35/72/74S003 closes automatically.
– After manual disconnection of the main circulation pump, the reactor limiting system automatically reduces the reactor power to a power level permissible with the number of main circulation pumps still in operation.
– When the main circulation pump is disconnected, the plant controller reduces the turbine output to a power level permissible by the number of pumps still in operation. The primary and secondary circuit parameters stabilise within permissible limits. Similarly the disconnected loop's steam generator water level stabilises.
– If the steam generator level rises $> 3.54m$, an emergency shut-down of both turbines will take place.

The size of the procedure is at 14 pages and 1139 lines (according to the textual version of the procedure from the Computerized Operation Manuals System [11]). There are 93 instructions (each of the instructions consists of one or several actions) involving around 40 plant units such as valves, steam level meters, pumps, pump speed meters and protection signals.

3.2 Correctness Specifications

Correctness specifications include goals, invariants and assertions. For the case study, we have formulated one goal, one invariant and one assertion for performing open-valve actions.

The Goal: "Every execution of the procedure terminates and upon the completion of the execution, all control valves (RL33S002, RL35S002, RL72S002 and RL74S002) are normal or the supervisor is notified (in case the power level at a certain check-point is not normal)".

The Invariant: "During an execution of the procedure, the valve YA13S002 must be open, unless the pump YD13D001 is stopped".

The Assertion: "Before performing an open-valve action to a valve, the state of the valve must be different from open".

3.3 Seeded Errors

We have seeded the following 10 errors into the procedure. The errors cover all types (7 types, 3 categories) of errors described in section 1.

1. A wrong reference - leading to an undefined reference.
   Problem: the instruction "gosub 7" is mistakenly written as "gosub 8".
2. A missing case - leading to a deadlock.
   Problem: the case "if RESULT is None then goto ..." is missing in a list of such goto-instructions.
3. A wrong condition in a wait-instruction (or an action that does not achieve its intended function, depending on interpretation) - leading to a deadlock.
   Problem: the condition "YD13Y013 $\geq$ 1400" in a wait-instruction is written as "YD13Y013 $\geq$ 2000".
4. A wrong condition in a goto-instruction - leading to an infinite loop.
   Problem: the condition "not (RL13S001 is Opening or RL13S001 is Open)" is written as "RL13S001 is Opening or RL13S001 is Open".
5. A missing instruction (or an action that does not achieve its intended function) - leading to an infinite loop.
   Problem: the instruction "if the valve is not opening, address the YZ63 (reset protection) signal ..." is missing (this instruction is specified as a comment in the original procedure).

6. A wrong reference - leading to an unreachable instruction.
   Problem: the instruction “goto 7 2” is written as “goto 7 3”.
7. A wrong action - leading to violation of the condition for the action.
   Problem: the action “open RL13S001” is written as “close RL13S001”.
8. A wrong sequence of actions - leading to violation of the invariant.
   Problem: the two actions in the sequence “stop YD13D001; ...; close YA13S002” are swapped.
9. A missing instruction - leading to an unreached goal.
   Problem: the action “notify Supervisor” is missing.
10. An action that does not achieve its intended function - leading to an unreached goal.
    In this case, we assume that the action “notify Supervisor” is implemented by using an unreliable message system which may not achieve the intended function of the action. Although it may not be a problem of the procedure design, it is interesting with respect to the correctness of the procedure whose execution is an integrated part of the physical processes of the plant.

3.4 Model Checking with Spin

The model checking tool used in the verification is Spin developed by the formal methods and verification group at Bell Laboratories [5]. For modelling, the language Promela is used. Promela is a process meta-language provided by the model checker Spin. With respect to modelling operator procedures, it has the advantage that it can easily be used to describe actions, sequences of actions, waiting for given conditions, and conditional and unconditional jumps. In addition, we can define a set of macros to represent actions, so that the model of a procedure would have the same structure and similar contents as the original procedure. For instance, an instruction to open the valve “RL33S001” can be modelled as openValve(RL33S001) with the macro openValve(id) defined as a sequence of statements that send a message to an interaction process and wait until receiving a proceed-signal from the interaction process. Upon receiving a message, the interaction process interprets the message in order to take appropriate actions (for instance to change the state of the system and to activate or initiate plant processes) and provides appropriate feedback to the model-procedure process. Plant processes could run by themselves or could be activated or initiated by interaction processes. Assuming that opening (respectively closing) a valve takes time and makes the state of the valve evolve from Closed to Opening and Open (respectively from Open to Closing and Closed), and that valves have protection signals such that they cannot be opened before their associated protection signals are switched off, a simple plant process for opening and closing valves could be as follows:

```
proctype OpenCloseValve(byte id,action)
{
    if
    :: action==Close;
```

```
        do
        :: Valve[id]==Open; Valve[id]=Closing;
        :: Valve[id]==Closing; Valve[id]=Closed; break;
        :: Valve[id]==Closed; break;
        od;
    :: action==Open && Protection[ValveToProtection[id]]==On; skip;
    :: action==Open && Protection[ValveToProtection[id]]==Off;
        do
        :: Valve[id]==Closed; Valve[id]=Opening;
        :: Valve[id]==Opening; Valve[id]=Open; break;
        :: Valve[id]==Open; break;
        od;
    fi;
}
```

where ValveToProtection[] is an array (a function) that maps a valve identifier to its protection signal identifier, Valve[] is an array that maps a valve identifier to its current state and Protection[] is an array that maps a protection signal identifier to its current state. This process can be initiated by an interaction process when the latter receives a message from the model-procedure process for opening or closing a valve. The statements "Valve[id]==Closed; break;" in the process seem to be redundant. However these statements are useful if instructions for closing a valve are allowed in situations where the valve is already closed. This example illustrates how one might model a process in Promela. In the case study, we have made assumptions and simplifications in order to minimise the number of processes.

Formalising Correctness Specifications

For correctness specifications, we use propositional linear temporal logic formulas (translated to never-claims) to represent goals, monitoring processes to represent invariants and assertions to represent conditions for performing actions. We outline our approach to express properties of operator procedures as follows.

Structural Errors No formula is needed for checking structural errors. Undefined references can be detected by checking syntax and unreachable instructions can be detected by checking execution errors (described below).

Execution Errors Presence of deadlocks or infinite loops can be checked by asking whether the instruction indicating the end of an execution is reachable in all potential executions of the procedure. For verification, we add an auxiliary statement "ProcedureCompleted=Yes" right before finishing an execution and the formula representing the fact that there are no deadlocks and infinite loops is as follows:

$$<> \text{(ProcedureCompleted==Yes)}$$

Conditions for Performing Actions Conditions for performing actions can be specified as local safety properties related to specific instructions. Let the macro openValve(id) represent open-valve actions. The condition for performing open-valve actions in section 3.2 is added by replacing the statement openValve(id) with the following statements:

```
        assert(Valve[id]!=Open); openValve(id)
```

Invariants Invariants can be specified as global safety properties (not related to any specific instruction). The invariant in section 3.2 is specified as follows:

```
proctype monitor()
{
        assert(Valve[YA13S002]==Open || Pump[YD13D001]==Stopped);
}
```

Goals Goals can be specified as liveness properties using propositional linear temporal logic formulas. The goal in section 3.2 is specified as follows:

```
<>   (ProcedureCompleted==Yes && (
     (CheckPointPowerLevel !=NOTnormal ->
      Valve[RL33S002]==Normal && Valve[RL35S002]==Normal &&
      Valve[RL72S002]==Normal && Valve[RL74S002]==Normal) ||
     (CheckPointPowerLevel==NOTnormal -> Supervisor==Notified)))
```

Detecting the Errors

We first created a model of the procedure (with some modifications to the original procedure in order to avoid errors not caused by the seeded errors) and then created abstract models of the relevant plant processes (for simplicity, we had assumed that there was at most one defected control valve at the beginning of an application of the procedure) and used the model checker Spin to verify the models. The seeded errors were detected as follows:

Phase 1 A model of the procedure (with the seeded errors) was created and error 1 was detected by performing syntactic check.

Phase 2 A set of assumptions on the environment was identified and 3 interaction processes modelling the assumptions were created. Errors 4 and 6 were detected. Error 4 was detected by checking execution errors. Error 6 was an unreachable instruction which was detected by re-checking execution errors after error 4 was corrected.

Phase 3 A process for modelling the possible initial states of the plant was added and modification to the set of relevant values for a variable was made. Error 2 was detected by checking execution errors. Error 7 was detected by checking the assertion. Error 8 was detected by checking the invariant. Error 9 was detected by checking the goal.

Remark Error 2 (a missing case) could not be detected as a deadlock in the previous phases, because it depends on a correct set of relevant values of the variable involved in the case analysis. Violation of assertions (error 7), violation of invariants (error 8) and unreached goals (error 9) were not checked in the previous phase.

Phase 4 A simple model of plant processes (consisting of 13 processes, one for each steam level meter, one for each pump speed meter, one for each cycling valve, and one for opening and closing of valves with protection signals) was created and errors 5 and 3 were detected by checking execution errors.

Remark Error 5 (a missing instruction) could not be detected in the previous phases, because the missing instruction is an instruction used to correct an action that in some executions of the procedure do not achieve its intended function and to detect this kind of problems requires a model of the consequences of the action. Error 3 (a wrong condition) could not be detected in the previous phases, because it depends on a process that controls the value of the variable in the condition.

Phase 5 One process for sending messages to the supervisor with an unreliable communication method (the consequence is that a message could be lost) was added and error 10 was detected by checking execution errors.

Remark Error 10 (an action that does not achieve its intended function) could not be detected in the previous phases, because lack of a model of the action causes the purpose of the action to be achieved by default.

Summary The following table shows the phases and the verification sub-tasks in which the errors were detected:

	phase 1	phase 2	phase 3	phase 4	phase 5
check syntactic errors	error 1				
check execution errors		errors 4,6	error 2	errors 5,3	
check assertion violations			error 7		
check invariant violations			error 8		
check unreached goals			error 9		error 10

Computation Times and Memory Usages

The verification was carried out by using Spin (version 3.2.4) on an HP-computer (machine and model numbers 9000/778) with an HP-UX operating system (release B.10.20). The times for syntactic check varied from 3 to 5 seconds in the different phases depending on the complexity of the models. The times for compilation varied from 10 to 15 seconds in the different phases. In the following, we present times used for model checking.

Error Detection The following table shows the model checking times (in seconds, exclude times for syntactic check and compilation) for detecting the errors in the different verification phases. The table is sorted by the order in which the errors were detected.

	phase 1	phase 2	phase 3	phase 4	phase 5
error 1	-				
error 4		0.3			
error 6		0.4			
error 2			0.6		
error 7			0.3		
error 8			0.4		
error 9			0.5		
error 5				27	
error 3				10	
error 10					4612

There were respectively 1, 4, 5, 18 and 19 processes (excluding monitoring processes and never-claims) in the 5 phases. The table shows that in the early phases the model checker was very fast and in the later phases it needed significantly more time to run model checking.

Verification Errors were removed as detected. In order to be sure that there were no more errors, one had to re-run the model checker. The following table shows the model checking times (in seconds) for the re-running of the model checker (with no error reports) in the different tasks and phases:

	phase 1	phase 2	phase 3	phase 4	phase 5
check syntactic errors	-				
check execution errors		0.4	0.5	13145	13974
check assertion violations			0.3	146	157
check invariant violations			0.3	433	496
check unreached goals			0.5	13125	13692

The verification in the later phases took much more time than that in the early phases (the interesting aspect is the relative model checking times, not the absolute values, since the latter depends on many factors such as the computer on which the model checker was running). There were also big differences between verification times for different types of tasks. Generally, it takes more time to verify liveness properties than to verify safety properties. The table also shows that the verification took much more time than the error detection when the number of processes was relatively large.

Memory Usage for Verification The following table shows the associated memory usages (in megabytes) for carrying out the verification of the procedure in the different phases and for the different tasks:

	phase 1	phase 2	phase 3	phase 4	phase 5
check syntactic errors	-				
check execution errors		3	3	13	13
check assertion violations			3	23	25
check invariant violations			3	59	63
check unreached goals			3	13	13

Discussion As time and memory usage are considered, the tables show that the differences could be very large in different phases. Therefore incremental modelling and verification is practical, since many errors can be detected when the models are simple and model checking times are short, so that the need for detecting errors and re-running model checking for large models is reduced.

3.5 Limitations

As the example shows, model checking can be used to check errors of the considered types. In fact, we have found all of the seeded problems with the help of model checking. However, there are limitations of model checking. One of which is that the set of detectable problems depends on the accuracy of the model of the plant. We have already seen that many of the seeded problems are not detectable with a too much simplified model. On the other hand, the complexity of useful models with respect to model checking is limited by the capacity of model checking tools and computational resources. Another limitation is that the set of detectable problems depends on the set of correctness specifications consisting of safety constraints and goals. In the example, there are one assertion, one invariant and one goal. This set of correctness specifications is of course not complete. Assuming that there was a wrong sequence of actions where the two actions in the sequence “stop YD15D001; ...; close YA15S002” were swapped, and that the models of the plant were the same, additional correctness specifications would be needed for detecting the problem. Generally, there is no guarantee that the set of correctness specifications to be used in model checking is sufficient to detect all problems.

4 Related Work

In the recent years, procedures have been computerised in order to reduce operator stress and to enhance safety [9] [8] [4]. Computerised procedures provide a better opportunity for formal verification as they are written in formal or semi-formal languages with better structure and formality. Earlier work on using formal methods for verification of operator procedures includes investigating techniques based on algebraic specifications and related theorem proving tools [10] [13]. Generally, the weakness with theorem proving is that the verification process is basically interactive. One of the disadvantages is that the user has to find a strategy for proving theorems. It is a time-consuming process. Another disadvantage is that if the user fails to prove a theorem, we cannot be sure whether the theorem is not provable or the strategy is not correct. Model checking has the advantage that it could utilise fully automated tools, although there are other limitations with model checking as explained earlier.

5 Concluding Remarks

This paper has discussed potential for applying model checking to the verification of operator procedures and to detect potential problems in such procedures. The result of the research has shown that model checking can be used to check errors of many types, including deadlocks, infinite loops, unreachable instructions, violation of conditions for performing actions, violation of invariants, and unreached goals. By analysing detected errors, we could find and remove different types of potential problems in such procedures.

The major problems for model checking operator procedures are the complexity of the modelling of the relevant processes of the power plant and the complexity of the models. As it is not easy to find the right complexity level for modelling and model checking, incremental modelling and verification was proposed. It is worth emphasizing that the practical interest of verification is to find errors that could otherwise be hard to detect and not necessary to prove the absolute correctness of procedures. Incremental modelling and verification is a practical strategy focussing on error detection and helps overcome the complexity of modelling and the problem of deciding how complicated the plant process model should be.

For many reasons, model checking (in general, formal methods) does not provide an absolute guarantee of perfection. First, model checking cannot guarantee that the set of correctness specifications is correct and complete. Second, model checking cannot guarantee that the process model of a physical device such as a pump speed meter is accurate with respect to the physics of the device. In addition, there are limitations on computational resources and the capacity of model checking tools, and there may be errors in the model checking tools themselves. Nevertheless, model checking provides a significant capability for discovering large portions of the design errors as demonstrated in the case study.

Further Work: The incremental strategy divides the process of modelling and verification into several phases with increasing complexity of plant process models. One advantage of the strategy is that it avoids full verification versus no verification approach and allows verification to be performed to a certain degree. The degree depends on the size of procedures and plant process models. With respect to scalability, it is important to push the limit as far away as possible. As the case study is considered, the processes were created with emphasis on readability, uniformity of instruction representations and clean interfaces between processes, and not on optimal model checking times. Directions for further research include investigating techniques that can be used to reduce model checking times without sacrificing too much of the mentioned desirable properties of procedure models and plant process models and investigating other approaches for verification and error detection.

Acknowledgements

The research described in this paper was carried out at the OECD Halden Reactor Project at the Institute for Energy Technology. The author thanks G. Dahll, T. Sivertsen, K. Stølen and other members of the project for helpful comments and discussions. The author also thanks anonymous referees for their comments.

References

1. M. Abadi and L. Lamport. Conjoining specifications. ACM Transactions on Programming Languages and Systems 17(3):507-534. May 1995.
2. J. R. Burch, E. M. Clarke and D. E. Long. Symbolic Model Checking with Partitioned Transition Relations. International Conference on Very Large Scale Integration, pp. 49-58. North-Holland, August 1991.
3. O. Grumberg and D. E. Long. Model Checking and Modular Verification. ACM Transactions on Programming Languages and Systems 16(3):843-871. May 1994.
4. D. G. Hoecker, K. M. Corker, E. M. Roth, M. H. Lipner and M. S. Bunzo. Man-Machine Design and Analysis System (MIDAS) Applied to a Computer-Based Procedure-Aiding System. Proceedings of the Human Factors and Ergonomics Society 38th Annual Meeting 1: 195-199. 1994.
5. G. J. Holzmann. The Model Checker Spin. IEEE Transaction on Software Engineering 23(5): 279-295. 1997.
6. J. G. Kemeny. Report of the President's Commission on the Accident at Three Mile Island. U.S. Government Accounting Office. 1979.
7. N. G. Leveson. Software System Safety and Computers. Addison-Wesley Publishing Company. 1995.
8. M. H. Lipner and S. P. Kerch. Operational Benefits of an Advanced Computerised Procedure System. 1994 IEEE Conference Record: Nuclear Science Symposium and Medical Imaging Conference:(1068-1072). 1995.
9. L. Reynes and G. Beltranda. A Computerised Control Room to Improve Nuclear Power Plant Operation and Safety. Nuclear Safety 31(4):504-511. 1990.
10. T. Sivertsen and H. Valisuo. Algebraic Specification and Theorem Proving used in Formal Verification of Discrete Event Control Systems. OECD Halden Reactor Project Report: HWR-260, Institute for Energy Technology, Norway. 1989.
11. J. Teigen and J. E. Hulsund. COPMA-III - Software Design and Implementation Issues. OECD Halden Reactor Project Report: HWR-509, Institute for Energy Technology, Norway. 1998.
12. H. J. Touati, H. Savoj, B. Lin, R. K. Brayton and A. Sangiovanni-Vincentelli. Implicit State Enumeration of Finite State Machines Using BDDs. IEEE International Conference on Computer-Aided Design, pp. 130-133. IEEE Computer Society Press, November 1990.
13. K. Ylikoski and G. Dahll. Verification of Procedures. OECD Halden Reactor Project Report: HWR-318, Institute for Energy Technology, Norway. 1992.