Reliability Engineering and System Safety 139 (2015) 120–130

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress

Verifying large modular systems using iterative abstraction refinement

Jussi Lahtinen [a,*], Tuomas Kuismin [b], Keijo Heljanko [b]

[a] VTT Technical Research Centre of Finland Ltd., Systems Research, P.O. Box 1000, FI-02044 Espoo, Finland
[b] Helsinki Institute for Information Technology HIIT and Department of Computer Science, School of Science, Aalto University, P.O. Box 15400, FI-00076 Aalto, Finland

ARTICLE INFO

Article history:
Received 7 June 2013
Received in revised form
5 March 2015
Accepted 6 March 2015
Available online 14 March 2015

Keywords:
Model checking
Verification
Validation
Iterative abstraction refinement

ABSTRACT

Digital instrumentation and control (I&C) systems are increasingly used in the nuclear engineering domain. The exhaustive verification of these systems is challenging, and the usual verification methods such as testing and simulation are typically insufficient. Model checking is a formal method that is able to exhaustively analyse the behaviour of a model against a formally written specification. If the model checking tool detects a violation of the specification, it will give out a counter-example that demonstrates how the specification is violated in the system. Unfortunately, sometimes real life system designs are too big to be directly analysed by traditional model checking techniques. We have developed an iterative technique for model checking large modular systems. The technique uses abstraction based over-approximations of the model behaviour, combined with iterative refinement. The main contribution of the work is the concrete abstraction refinement technique based on the modular structure of the model, the dependency graph of the model, and a refinement sampling heuristic similar to delta debugging. The technique is geared towards proving properties, and outperforms BDD-based model checking, the k-induction technique, and the property directed reachability algorithm (PDR) in our experiments.

1. Introduction

Digital instrumentation and control (I&C) systems are increasingly used in the nuclear engineering domain. The exhaustive verification of these systems is challenging, and verification methods such as testing and simulation are typically insufficient.

Model checking [1,2] is a formal method that is able to exhaustively analyse the behaviour of a model against formally written specifications. If the model checking tool detects a violation of a specification, it will give out a counter-example that demonstrates how the specification is violated in the system.

In this work, we are primarily using the model checker NuSMV 2.5.4 [3], which was originally designed for synchronous digital hardware model checking. The NuSMV model checker does not have a notion of continuous time but instead the timing elements in our models are modelled with discrete time steps using explicit counter variables. In NuSMV, the formal correctness specification can be written as a simple state invariant clause that should hold in each individual reachable state of the system, or in a more complex specification language such as the Linear Temporal Logic (LTL) and the Computation Tree Logic (CTL) [1,2]. In addition to NuSMV, we also utilise another model checking tool called ABC/ZZ by Niklas Eén [4]. Since our models are written in the NuSMV modelling language, we translate the NuSMV models into the AIGER format [5] used in ABC/ZZ.

The specifications used in this work are formalised as state invariant specifications, but we are exploiting a procedure compatible with our approach that reduces LTL property model checking into state invariant model checking [6], thus enabling all LTL properties to be model checked.

The classical algorithm for model checking state invariant specifications is based on symbolically representing and exploring the reachable state space of the system by using Binary Decision Diagrams (BDDs) [1], which are a highly efficient data structure for representing and doing operations with large state spaces. Another way to check state invariants is to use a propositional satisfiability solving (SAT)-based approach. This line of work employs a propositional satisfiability solver in a bounded model checking (BMC) procedure [7,6], which looks for counter-examples shorter than a user provided maximum length, called the bound. An advanced variant of this procedure we employ in this work is called *k-induction* [8,9]. In that approach the state invariants are proved using induction, and the base step and the induction step of the proof are basically reduced to bounded model checking problems. Another SAT-based technique for checking safety properties is the IC3 algorithm by Bradley [10], also known as property directed reachability (PDR). The technique inductively searches for an invariant that holds in the initial state and implies the examined specification.

* Corresponding author.
E-mail address: jussi.lahtinen@vtt.fi (J. Lahtinen).

http://dx.doi.org/10.1016/j.ress.2015.03.012
0951-8320/

The traditional model checking algorithms including the ones described above have been successfully used to analyse individual nuclear domain safety I&C systems, see e.g. [11,12], as well as satellite onboard software designs [13]. However, in the nuclear domain it is common to cope with hardware failures by implementing several subsystems that execute the same physical function using design diversity in software and/or hardware. It may be necessary to examine these diverse subsystems simultaneously, e.g. because the specifications may in fact cover their combined behaviour, and to also additionally check that the diverse subsystems have no unintended interactions. Unfortunately, the currently available classical model checking methods by themselves do not always scale to analysing these large and complex combined systems. In our experience, the Binary Decision Diagram (BDD)-based techniques by themselves have proven to be inadequate in the analysis of our models in some of these larger system configurations. The alternative SAT-based bounded model checking (BMC) techniques [7,6] can often find counter-examples in many large systems. Some BMC techniques such as the k-induction technique [8,9], and the PDR algorithm [10] can prove properties, but in our experience the necessary CPU time and memory needed for a proof can make the verification impractical for many real life designs.

One classical approach to avoid the scaling problem is to use abstraction. Intuitively, *abstraction* is the act of simplifying a model with the intention of making the verification of the model more efficient, whereas adding more detail to the model is called a *refinement*. In systems where multiple diverse subsystems are present, the whole system functionality is rarely needed to verify a system property. Depending on the exact specification some subsystems or parts of subsystems may be irrelevant for proving the specification, and can be abstracted. The abstractions we use in this work are over-approximations. Over-approximation techniques tend to relax constraints, e.g., by allowing a variable to also have values that are not realistic. Over-approximation leads to a model that has more behaviour but can be less complex to analyse. Because of the possible unrealistic behaviour in the model, over-approximation can lead to spurious counter-examples. [14] Since the over-approximated model has more behaviour than the concrete model, the correctness of the resulting abstract model implies also the correctness of the full model when universal properties such as state invariants are examined.

Unfortunately, creating such an abstract model for each checked specification is non-trivial and requires a lot of manual work, becoming tedious and thus also error prone. For the best efficiency gains, the abstraction is tailored for each specification separately.

In this paper, we describe how these kinds of over-approximating abstractions can be created *fully automatically* by using an iterative abstraction refinement technique exploiting the modular structure of the system. Our technique will (i) significantly reduce the amount of manual work needed to create these abstractions, (ii) prove the system correctness based on verification runs automatically performed on these abstractions, and (iii) reduce the overall computational effort required for model checking, enabling the model checking of larger system models. Our approach is designed to be efficient at finding proofs for properties, as we expect most of the designs at this stage of inspection to actually be correct. For all properties that hold, the algorithm will find some abstraction of the system.

In our technique, we require that the system is structured into modules. Abstractions of the model are created by automatically replacing a subset of the modules with stubs that can at each time point non-deterministically give any value from any of their outputs. These simplified modules are called *interface modules*. This approach is similar to the compositional minimisation [15] technique.

In iterative abstraction refinement an initial abstraction is first generated and model checked. If the examined property produces a counter-example, the model is refined and the resulting new model is verified again. The process is continued until the property is proved or no further model refinement is possible.

In the model checking step of our abstraction refinement technique three model checking algorithms (BDD-based, k-induction, PDR) are run in parallel in a portfolio-based manner, similar to what is described in [16].

The abstractions in our technique are refined using a two-phase procedure. First, in the preliminary refinement phase, we obtain a computationally manageable subset of the modules in which the previously found counter-example becomes infeasible. This part of the refinement procedure is based on traversing the dependency graph of the modules. After the preliminary refinement phase we attempt to minimise the size of the needed model refinement using an iterative sampling procedure similar to delta debugging. Delta debugging [17,18] is originally a technique for isolating failure causes of software errors automatically. The technique works by systematically narrowing down failure-inducing circumstances until a minimal set remains. In our work we use the same principles in order to minimise the size of the refinement. The feasibility of the candidate refinements is repeatedly checked during both model refinement phases. These feasibility checks are performed using k-induction [8,9] to see whether the spurious counter-example of length k has been removed.

We have tested our technique, and report experimental results from verifying two different systems with it. The first system is a fictional case study that consists of two diverse safety systems. The fictional system is used to demonstrate the technique in practice, and to show more detailed examples of the system implementation. The second system is an actual industrial emergency diesel generator control system that is used for evaluating the performance of our technique on a real life system. The system is safety-critical, as emergency diesels are used e.g. in nuclear power plants to provide electricity in case of power failures. In both case studies we have compared our technique against three other model checking approaches: classical BDD-based model checking, SAT-based k-induction, and PDR-based model checking. In the comparisons these approaches used the full concrete model for verification. The results show that for most of the properties our technique is able to find a proof of correctness of the system more efficiently than the other three approaches.

2. Verified systems

We have tested our algorithm by applying it on two case study systems. The first is a fictional safety system we have constructed for demonstration purposes. The other is a model of an actual emergency diesel generator. The purpose of the emergency diesel model is simply to provide some additional benchmark information on the performance of the algorithm on a real-life industrial model.

2.1. Fictional system description

We have created a fictional system model for demonstration purposes. The NuSMV model of the system is available online [19]. The fictional model consists of two safety systems. Safety system 1 reads temperature measurements and controls an actuator device (a pump) if the measurements exceed a certain limit. Safety system 2 has pressure measurements as input, and it controls two other actuator devices. The systems are redundant, and the purpose of the pumps is to cool down a process so that the temperature and pressure of the underlying process remain sufficiently low.

The basic functionality in both of the safety systems is such that if the measurements exceed the given limits, the safety system is initiated and this fact is memorised. The safety systems are also associated with particular timing sequences that are given to the actuators when they are started.