Model Checking

Dragana Cvijanovic
d.cvijanovic@cs.ucl.ac.uk

1 Introduction

Computerised systems pervade more and more our everyday lives. Digital technology is now used to supervise critical functions of cars, airplanes, medical instruments, industrial plants etc. Where important investments and human lives are at risk, quality assurance for the underlying systems becomes paramount. Frequently, we hear of incidents where some failure occurred due to an error in a hardware or software system. Just recall the incident with the Ariane 5 rocket [4], which exploded seconds after the launch in June 1996. It was later found that the incident was caused by a software error in the computer that was in charge of rocket's movement. During the launch, an exception occurred when a large 64-bit floating-point number was converted to a 16-bit signed integer. This part of the code was not protected by an exception handling routine, which caused the failure of the back-up computer as well. As a consequence, a wrong altitude data was delivered to the on-board computer, causing the destruction of the rocket. Obviously, nowadays, the need for reliable reactive systems becomes critical.

2 Systems Verification

There are four principal models for ensuring the correctness of such systems: simulation, testing, deductive verification and model checking.
The most widely used verification techniques are simulation and testing. Simulation is usually performed on the abstraction of a kind of the system, whereas testing is performed on the actual product. However, in the case of complex, asynchronous systems, these techniques can cover only a limited set of possible behaviours [5].
Deductive verification employs use of axioms and proof rules to verify the correctness of the system. However, this process can be quite lengthy and time-consuming, and therefore is rarely used [5].
Model checking differs from these traditional program verification methods in several crucial aspects: firstly, it does not aim of being fully general and secondly it is fully algorithmic and of low computational complexity. Model checking, developed independently by Clarke and Emmerson, and Queille and Sifakis in 1980s, has been conceived as an automatic verification technique for finite state systems, performing an exhaustive search of the state space of the system in order to determine if some specification is true or not. Since subtle errors in the design of safety critical systems that often elude traditional verification techniques can be discovered in this way, and since it has proven to be cost-efficient, model checking is being adopted as a standard procedure for the quality assurance of reactive systems.

3 The Process of Model Checking

The main building blocks of model checking process can be separated into Modelling, Specification and Verification.
When embarking on the mission of model checking, the initial step is to convert a design into a formal form accepted by the model-checking tool. Before verification, Specification stage ensures that the properties that the design must satisfy are stated. Usually this is achieved by using temporal logic, which models the behaviour of the system over time. Then, by exhaustively exploring the state space of the state transition system during the Verification stage, it is possible to check automatically if the specifications are satisfied. The termination of model checking is guaranteed by the finiteness of the model. However, one of the most important features of model checking is that, when a specification is found not to hold, a counterexample (e.g., a witness of the offending behaviour of the system) is produced.

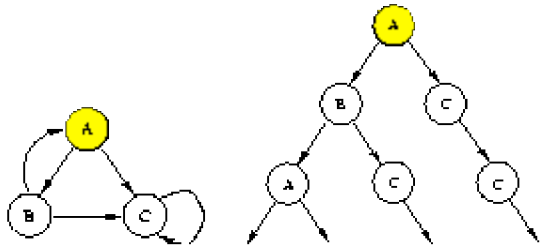
4 Systems and Properties

A common framework for representation of reactive systems is provided by the concept of transition systems. (Reactive systems are systems that maintain an ongoing interaction with their environment.) In order to be able to perform feasible model checking, usually the transition system we require should be a finite state system.

4.1 Temporal Logic

A finite state system can be described as a tuple: $M = \{S, I, A, \_\}$, where S is a finite set of states, $I \subseteq S$ is the set of initial states, and $A \subseteq S \times S$ is the transition relation, specifying the possible transitions from state to state. $\_$ is a function that labels states with the atomic propositions from a given language. Such a tuple is called state transition graph or Kripke structure (in honour of logician Saul A. Kripke).
Temporal logics are then used to predicate over the behaviour defined by Kripke structures. This behaviour is obtained starting from a state $s \in I$, and then repeatedly appending states reachable through A. Consequently, all the behaviours of the system are infinite. Since a state can have more than one successor, the structure can be thought of as unwinding into an infinite tree, representing all possible executions of the system starting from the initial states. *Figure 1* shows the unwinding of the state transition graph from that state labelled "A".

Figure 1: A State Transition Graph and its unwinding.



Two commonly used temporal logics are Computation Tree Logic (CTL) and Linear Temporal Logic (LTL). They differ in how they handle branching in the underlying tree structure. In LTL, operators are intended to describe properties of all possible computation paths, whereas in CTL temporal operators it is possible to quantify over the paths departing from a given state.
The model-checking problem can be formally formulated as follows. Given a Kripke structure M and a temporal logic formula $\_$, find the set of all states that satisfy $\_$, namely the set of states
$\{s \in S \mid M, s \_ \}$
We say that the system satisfies the specification provided that all the initial states are in the set $\{s \in S \mid M, s \_ \}$:

$$I \subseteq \{s \in S \mid M, s \_ \}$$

5 Algorithms for Model Checking

Given a transition system M and a formula φ, the model-checking problem is to decide whether $M \_ \_$ holds or not. If not, the model checker should provide an explanation why, in the form of counterexample. In accordance with two main parameters of the model checking problem, M and $\_$, there are two basic approaches to designing a model checking algorithm: "global" exploited by model checkers for CTL and other branching-time logics, and "local" used by LTL model checking.

In the linear approach, the model checking problem is defined slightly differently: one checks that all linear sequences that can be generated by the transition system M (all possible executions of the program) satisfy the linear temporal logic formula $\_$. Linear approach comes natural when the properties to be checked are expressed in terms of the possible executions of the program. Pnueli and Lichtenstein analysed the complexity of linear-time formulas, discovering that the time

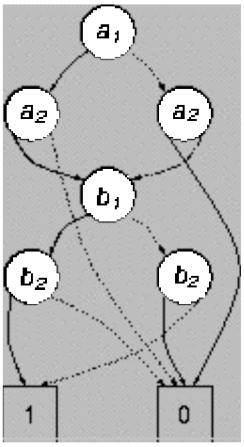
complexity of the LTL model-checking algorithm is exponential in the size of formula _, nevertheless linear in the size of the global state graph [4].

The branching approach to model checking was introduced in the early 1980s by Clarke and Emmerson [2], [3], [4]. This approach proved to be suitable for checking properties that are thought of in terms of the structure of the program. The algorithm they developed was polynomial in both size of the model determined by the model-checking program and in the size of its specification in temporal logic. They also introduced the notion of fairness without increasing the complexity of the algorithm [4]. These early model checking systems were able to check state transition graphs with between 10^4 and 10^5 states at a rate of about 100 states per second for typical formulas.
Another type of branching-time logic is CTL*, introduced by Clarke, Emmeson and Sistla; combining both branching-time and linear time approaches. This technique proved to have time complexity as the LTL model checking algorithm.

5.1 Symbolic Model Checking

The initial implementation of the model checking problem using symbolic model checking (SMC) used an explicit representation of the Kripke structure as a labelled, directed graph. For systems with limited number of processes, where the number of possible states was rather small, this approach proved to be practical. However, this method was unable to handle extensive number of states generated by complex concurrent systems (“state explosion problem”). The major improvement was achieved with introduction of Binary Decision Diagrams (BDDs) in 1987 by McMillan [4], based on the use of OBDDs (Ordered Binary Decision Diagrams). *Figure 2* [6] depicts the BDD for the boolean formula *(a_1 _ a_2) □ (b_1 _ b_2)*, using the variable ordering a_1, a_2, b_1, b_2. Solid lines represent ``then'' arcs (the corresponding variable has to be considered positive); dashed lines represent ``else'' arcs (the corresponding variable has to be considered negative). Paths from the root to the node labelled with ``1'' represent the satisfying assignments of the represented boolean formula (e.g., *a_1_ 1*, *a_2_ 1*, *b_1_ 0*, *b_2_ 0*).

Figure 3: *A BDD for the formula (a_1 _ a_2) □ (b_1 _ b_2)*



Intuitively, a state of the system is symbolically represented by an assignment of boolean values to the set of state variables. A Boolean formula (and thus its BDD) is a compact representation of the set of the states represented by the assignments, which make the formula true. Similarly, the transition relation can be expressed as a boolean formula in two sets of variables, one encoding the current state and the other encoding the next state. Then, model-checking algorithm is based on computing fixpoints (sets of states that represent system’s temporal properties) of predicate transformers that are obtained from the transition relation. Therefore, the explicit construction of the global state graph for the entire concurrent system could now be avoided.
Later on, McMillan developed a model checking system called SMV, which extracts a transition system represented as a BDD from a program and uses a BDD-based search algorithm to determine whether the system satisfies its specification [4]. The same as most of the model

checking systems, if it determines that a certain property has not been satisfied, it will produce the counterexample proving the invalidity of the specification.

5.2 Partial Order Reduction

Whereas SMC derives its power from efficient data structures for representation and manipulation of large sets of states, algorithms based on the explicit state enumeration could be improved if only a fraction of the reachable pairs are explored. This idea has been applied most successfully in the case of asynchronous systems that are composed of concurrent processes with relatively little interaction. The model employed, called *interleaved model* has all the actions of the individual processes arranged in a linear order called an *interleaving sequence*. Most logics for specifying properties of concurrent systems can distinguish between interleaving sequences in which two independent events are executed in different orders. Therefore, the full transition system considers all the possible interleavings of these sequences, resulting in an enormously large state space. There are several model checking algorithms that employ the partial order reduction approach: Stubborn sets, Persistent sets, Ample sets, unfolding technique and Sleep sets.

5.3 Alternative Approaches

Symbolic representations and partial-order reduction have increased the size of the systems that can be verified to 10^{20} – 10^{30} states, but still, there are a vast number of systems that are too complex to handle. Several techniques that can be used in conjunction with existing approaches have been introduced, allowing for complex systems to be model checked:
Abstraction technique proved to be extremely important when it comes to analysis of systems that involve data paths. The specifications of these kinds of systems contain a fairly simple relationship between data values. An attractive way of presenting abstraction in this case would be in the form of mapping between concrete data values to a small set of abstract data values. By extending this mapping to all states and transitions we could produce an abstract version of the entire system.
Compositional reasoning is a technique that exploits the modular structure of systems, composed of multiple processes running in parallel. To simplify the specification of such a system, it can be decomposed into set of properties that describe behaviours of different parts of the system. Then, the complete system must satisfy the overall specification:
- if each local property holds;
- and if the conjunction of the local properties holds.

Symmetry reduction approach applies to systems that are composed of components that are replicated in a regular manner. For example, some protocols involve set of processes communicating in some fashion, or some hardware devices contain parts that have replicated elements. These facts can further be used to reduce the complexity of the system specification.
Induction involves reasoning automatically about entire sets of finite-state processes. Generally, the problem proves to be undecidable, but it is sometimes possible to produce a form of invariant process that can be used to represent the behaviour of an arbitrary member of the set.

6 Fields of Application:

SMC has been successfully used in various fields, allowing discovery of design bugs that were difficult to highlight with traditional techniques:

- In 1992 Clarke and his students used SMV to verify the IEEE Future+ cache coherence protocol. They found a number of previously undetected errors in the design.
- In 1992 Dill and his students found several errors, ranging from unitialised variables to subtle logical errors during the verification of the cache coherence protocol of the IEEE Scalable Coherent Interface.
- In [1] the authors verified part of preliminary version of the system requirement specifications of TCAS II (Traffic Alert and Collision Avoidance System II). TCAS II is an aircraft collision avoidance system required on most commercial aircrafts in United States.

- A High-level Data Link Controller was being designed at AT&T in Madrid in 1996. Using FormalChecker verifier, 6 properties were specified and five verified. The sixth property failed, uncovering a bug that would have reduced throughput or caused lost transmissions.

7 Conclusion

In this paper I gave a brief overview of the main system verification techniques, with a particular focus on the basic concepts behind the model-checking problem. These concepts have proved to be very useful and are now being employed by large companies such as AT&T, Fujitsu, Intel, IBM, and Motorola etc. as part of their development process. Obviously, the need for automated, fast and reliable model checking tools is increasing, laying down the directions for future research in system verification. Most of this research is now being focused on the use of abstraction, compositional reasoning and symmetry for resolving the state explosion problem. Methods for verification of parameterised designs are being developed, as well as practical tools for real-time systems.

8 References:

1 Richard J. Anderson, Paul Beame, Steve Burns, William Chan, Francesmary Modugno, David Notkin, and Jon D. Reese.
Model checking large software specifications.
In David Garlan, editor, *Proceedings of the Fourth ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 156-166, San Francisco, CA, USA, October 1996. ACM.
A full version to appear in IEEE Transactions on Software Engineering

2 E. M. Clarke and E. A. Emmerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logic of Programs*: Workshop, Yorktown Heights, NY, May 1981

3 E.A. Emmerson. *Branching Time Temporal Logic and the Design of Correct Concurrent Programs*. PhD thesis, Harvard University, 1981.

4 E. M. Clarke, O. Grumberg, D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999

5 W. Nam. Course notes on model checking. Theory and Formal Methods Lab. Korea University

6 A.Cimatti, E. Clarke, F. Giunchiglia , M. Roveri. *NUSMV: a new symbolic model checker*

7 S. Merz. *Model Checking: A tutorial Overview*. Institut fur Infomatik, Universitat Munchen