Model Checking Safety Critical Systems*

C. Bernardeschi[3], A. Fantechi[1,2], S. Gnesi[2]

[1] Dip. di Sistemi e Informatica, Univ. di Firenze, Italy
[2] Istituto di Elaborazione dell'Informazione, C.N.R. Pisa, Italy
[3] Dipartimento di Ingegneria della Informazione, Univ. di Pisa, Italy

Abstract. In this paper we argue that safety critical systems are the next candidate to share the success that formal verification techniques based on model checking are enjoying in the field of hardware verification. We motivate this statement first reviewing how formal methods have so far been applied in the development of safety critical systems, and then discussing the critical points for the success of the introduction of formal methods in such a field. The results presented in this paper are based on our experience of formal verification of fault-tolerant safety critical systems by means of model checking techniques. In particular we show how fault tolerant systems posses some characteristics that help in limiting the "state explosion problem" when model checking algorithms are used on industrial cases studies.

1 Introduction

The application of formal methods, in the rigorous definition and analysis of the functionality and the behaviour of a system, should promise the ability of showing that a computer system controlling a safety critical process is correct, so that it cannot harm in any case.

Given such a promise, that is already out since several years, it is astonishing to see how little formal methods are actually used in the safety critical system industry, though the use of formal methods is increasingly required by the international standards and guidelines for the development of safety critical computer-controlled systems.

Several approaches to the application of formal methods in the development process have been proposed, differing for the degree of involvement of the method within the development process. They range from the mere use of formal specifications in order to write rigorous specifications, the more or less automated generation of code from the formal specification, to the use of formal verification as an additional validation technique aimed to reach an high level of confidence of the correctness of the system and finally to the derivation of test cases from the formal specifications to ensure that the correctness of implementations.

* This work was partly supported by Progetto Coordinato CNR "Strumenti Automatici per la Verifica Formale nel Progetto di Sistemi Software ".

Industrial acceptance of formal methods is strictly related to the investment needed to introduce them, to the maturity of tool support available, and to the easiness of use of formal methods and tools.

Nowadays, the industrial trend is directed to the adoption of formal verification techniques to validate the design of critical system, integrating them within the existing development process.

Industries are more keen to accept formal verification techniques assessing the quality attributes of their products, obtained by a traditional life cycle, rather than a fully formal life cycle development, due to the lower training and innovation costs of the former.

Formal verification methods and tools can be roughly classified into two categories, the so called proof theoretical approaches and the model theoretical ones. In the proof theoretical approaches, the system state is modelled in terms of set theoretical structures on which invariants are defined, while operations on the state are modelled by specifying their pre and post conditions in terms of the system states. Properties are described by invariants, which must be proved to hold through the system execution, by means of theorem proving.

Proof theoretical approaches require in general a great skill in the use of theorem proving tools, and, therefore, a certain investment, in terms of know-how and training. This is because proofs usually need to be guided by humans and so the theorem proving process is not totally automatic.

Model theoretical approaches work on a finite state representation of the behaviour of the system. Verification is usually carried out by checking the satisfiability of some desired properties over the model of the system by model checking algorithms or equivalence verification algorithms. As an example, safety requirements may be expressed as temporal logic formulae and may be checked on the model of the system. Model theoretical approaches provide a direct automatic verification method of system properties. Unfortunately, this approach suffers of the so called "State Space Explosion" problem: systems composed of several subsystems can be associated to a finite state model with a number of states which is exponential in the number of the component subsystems. Moreover, systems which are highly dependent on data values share the same problem, producing a number of states exponential in the number of data variables. Hence, traditional model checking techniques [6], have shown themselves not to be powerful enough to cope with many "real" systems, when their models are larger than 100000 states.

However, recent advances in model checking techniques, have managed to deal with very large state spaces by the use of symbolic manipulation algorithms inside model checkers: the most notable example is the SMV model checker [4]. In SMV the transition relations are represented implicitly by means of Boolean formulae and are implemented by means of Binary Decision Diagrams (BDDs, [3]). This usually results in a much smaller representation for the systems' transition relations, thus allowing the maximum size of the systems that can be dealt with to be significantly enlarged. This kind of tools have been successfully applied to very large state spaces in the realm of hardware verification.

We believe that safety critical systems are another candidate to share the success of model checking enjoyed by the hardware verification field. This is what we argue in this paper, by first discussing model checking and its application to safety critical systems, and then showing how particular characteristics of this class of systems allow the state explosion problem to be tackled.

2 Model Checking

Model checking is a formal verification technique that, literally, means "Checking (properties on, correctness of) a Model" of a system [6]. Actually, we can find two interpretations of the term Model Checking:

- The first, canonical, interpretation, widely adopted in the formal verification community, denotes as model checking any *algorithmic* and *exhaustive* verification on the model of a system: this includes temporal logic model checking as well as reachability analysis, equivalence or preorder checking and so on: in any case, the whole state space is taken into account. If some optimization which does not actually visit every state, is used it is because some formal reasoning can be applied to show that what is proved on the reached states guarantees the result of the verification on the whole state space.
- The second interpretation is more relaxed, and includes a *non exhaustive* verification techniques such as simulation/animation of the model, in which only some possible executions of the system are analyzed. Some commercial tools, such as the ones working on SDL[21], offer advanced simulation techniques that approximate exhaustive search and provide, together with provisions to cut the state space in depth or width, some techniques to calculate a structural coverage measure of the performed verification with respect to the whole model. These are very similarly to the coverage measures used in software testing. As for testing, a 100% coverage measure does not automatically mean that the verification made is exhaustive; nevertheless the sophisticated techniques used for the generation of the state space (which have been mostly inherited by the research on model checking) may allow exhaustiveness to be reached in many cases.
  Some of these tools are currently used extensively in the telecommunication industry, and are gaining acceptance also in the safety critical systems field.

We advocate however the use of the first interpretation that distinguishes "complete" formal verification (that algorithmically gives an yes/no result) from "partial" verification: a complete verification is what is primarily expected by a safety critical system validation technique.

3 Use of Model Checking Techniques in the Development of Safety Critical Systems

We can observe that railway industries have assessed the introduction of formal verification by model checking in their development processes before other safety critical industries, and with a greater success. We can refer to [2, 5, 16] for some notable examples, but there have been many other published and unpublished work in this direction.

A key-point in the industrial acceptance of model checking is that it relies on models which are essentially (some variants of) finite-state machines, which are commonly used in many industrial activities, especially in the safety critical systems area.

In railway signalling industry, the safety critical part of a control system is the so called interlocking logic, whose main aim is to guarantee, through the typical use of a transition to a safe state (that is, by stopping the trains), that the system does not enter a critical state. Interlocking logic is usually amenable to be formalized through the use of state machines, or of first-order logic predicates, and this does not usually require a large investment in people. Moreover, safety properties to be proved on the system depend on the combination of values of discrete variables, and this makes state explosion problems easier to be attacked.

In medical systems, as well as automotive, avionic or space systems, on the other hand, the safety characteristics of the system are often controlled by sophisticated, numerical algorithms; therefore, the safety properties depend on the combination of values of continuous variables with often more stringent real time requirements. Though specific model checking technologies have been developed to cope with such systems (such as timed and hybrid model checking), continuous variables and time add such a complexity to the state space, that current model checking technologies appear not mature enough for an heavy industrial usage. This difference in the nature of the controlled process is the responsible for the slower acceptance of formal verification techniques in these industries. The same delay has been observed, within the railway industries itself, about the application of model checking to the verification of on-board equipments, since they introduce dependency from real quantities (speed, time, etc...) and hence increasing the complexity and the size of the state space [13].

4 Dealing with State Space Explosion in Safety Critical Systems

As we have said, safety critical industries have difficulties in using formal verification methods because of the limits imposed by the size problem, that even challenges more advanced model checking tools. The use of techniques such as decomposition and abstraction, to overcome the state space explosion problem at the specification level, are only partially successful, and require a great deal of expertise which industries often do not have. Thus current techniques have failed to arrive at the level of usability required by the safety critical industries.

A solution could be the development of domain-specific optimization of model checking algorithms. In particular, some specific features of safety critical systems may be searched that can be used to optimize the verification algorithms. As a first example of possible domain specific optimizations, Eisner [14] has shown how the safety critical characteristics of robustness and locality can be used to avoid difficult fixed-point calculations in symbolic model checking when applied to railway interlocking.

We claim here that the safety critical characteristic of redundancy may provide another chance to limit the state explosion problem.

4.1 Redundancy

Many safety critical systems employ redundancy as a mean to detect or mask faults. A system employing redundancy is composed by a number of identical modules which compute the same results. At the architecture level such modules are, in today's critical systems, independent processors. These modules have to synchronize periodically in order to maintain their consistency, and the synchronizations are usually combined with some comparison or voting operation, aimed to detect or mask errors. A redundant system is therefore a distributed system that uses specialized interaction protocols. Usually such protocols have to ensure some notion of consistency even in presence of faults, and trigger appropriate corrective actions. The formal verification of such protocols is therefore an important step in the establishment of the overall safety of the system even in presence of faults.

The formal modeling of a fault tolerant system can be structured as a network of replicas, each divided in phases of useful work; at the end of each phase the replicas synchronize to maintain consistency through exchange of messages.

A common structure of a fault tolerant system can be represented (in the case of duplication redundancy) as shown in Figure 1, using the graphical formalism employed by the tool ATG [20] to represent networks of synchronizing automata. In this intuitive formalism, which anyway can be directly input to many formal verification tools, a system is specified by drawing a network of automata (or net) making possible a hierarchical approach in specification. In this case the automata, that represent the constituent subsystems, are composed together via synchronization actions allowing also a multiway synchronization operator so that more than two subsystems can synchronize by executing an action.
Using this formalism a duplicated system can be modeled, as we see in Figure 1, as a network of automata; each automaton synchronizes with the other ones at the end of each phase. In general, more than a single synchronization action is involved at the end of a phase. Here we abstract such a complex synchronization protocol with only a single action, without affecting the validity of the following discussion.

The behaviour of the overall system is obtained by the parallel composition of the replicas. Due to the synchronization at the end of each phase, the obtained gloabal automaton appears to be structured in phases as well; each phase of the overall system is actually generated by the interleaving of the corresponding phases of the different replicas, while each phase is terminated by the synchronization of the replicas, leading to a single global state from which the next phase begins (see Figure 2, where $Phase\ i || Phase\ i$ represents the automaton built by interleaving two replicas of $Phase\ i$).

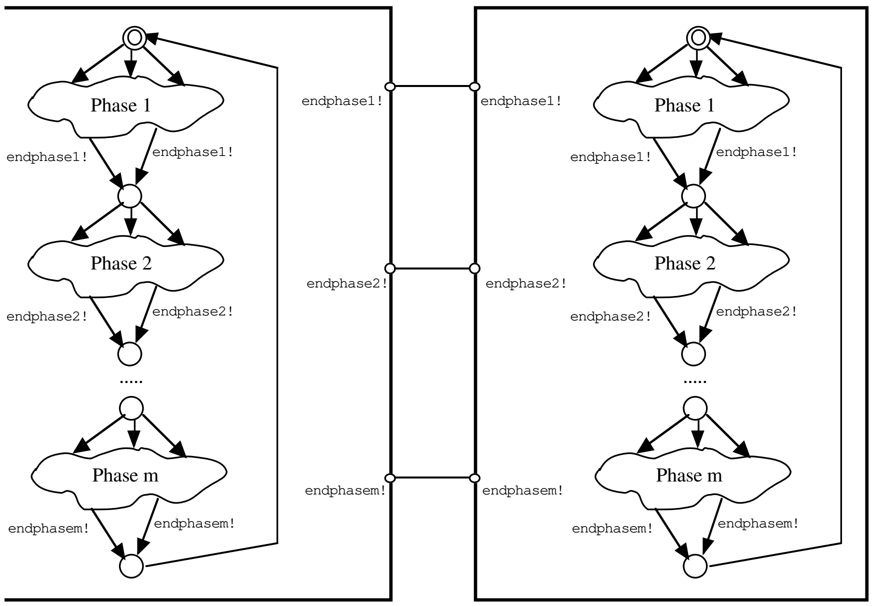

Fig. 1. The phased structure

If we call S the size of the state space of a replica, the cardinality of the state space of the interleaving of n replicas has normally an upper bound of S^n. Due to the phased structure, if we denote by S_i the size of the state space of the i-th phase, the upper bound for S is determined by the size of the interleaving of each phase, that is: ${S_1}^n + {S_2}^n + ... + {S_m}^n$, which is significantly lower.

Modeling a fault tolerant system means also to include in the model a description of the behaviour of the system in case faults occur, in order to be able to prove fault tolerance properties.
The modeling of fault occurrences and of relative corrective actions inside the phases is a major source of complexity for each phase; moreover, they tend to break independency and similarity of replicas.
However, it is important to notice that it is often not interesting to prove the safety critical properties in any general fault scenario, but it is enough to consider only some restricted fault scenarios. Typically, in a redundant system, assumptions are made on the maximum number of admitted faulty replicas. An accurate definition of the fault assumptions, which is usually an important part of the validation of safety critical systems, helps in the containment of state space explosion.

4.2 An example

The reduction in the state space due to the phased structure of redundant systems has been observed in the formal validation of the implementation of inter-

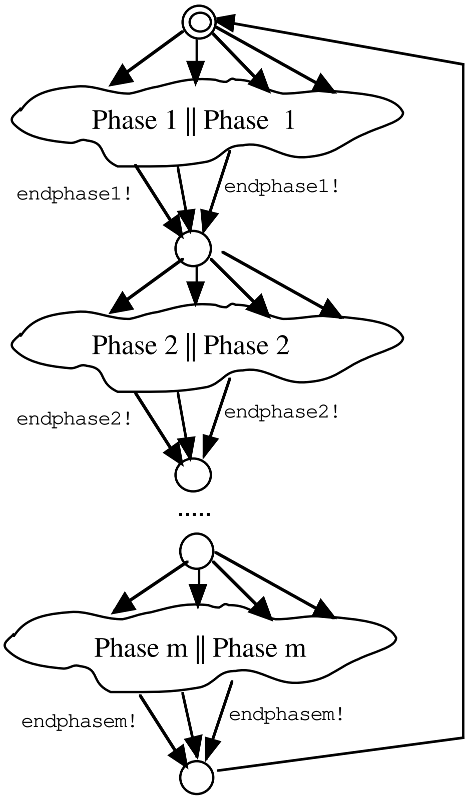

Fig. 2. The phased structure

active consistency mechanisms of a fault tolerant architecture of the GUARDS project [1].

Interactive consistency focuses on the problem of reaching agreement among multiple processors in presence of faults. The principal difficulty to be overcome in achieving interactive consistency is the possibility of conflicting values sent by faulty processors: such a processor may provide one value to a second processor, but a different value to a third, thereby making difficult for the recipients to agree on a common value. Interactive consistency algorithms overcome this problem by using several rounds of messages exchange (*phases* in the following) during which processor P tells processor Q what value it has received from processor R and so on. This problem is also known as the "Byzantine Generals problem" [18] in the literature. Byzantine agreement is extremely expensive to be implemented and classes of protocols have been proposed to optimise different cost factors (number of messages, number of messages, etc.).

Interactive consistency is implemented in GUARDS by the Inter-consistency mechanism, which uses the ZA Byzantine Agreement algorithm described in

[17]. The algorithm is synchronous, and it uses "rounds" of message exchanges during which node P tells node Q what value it received from node Q and so on. Messages are authenticated: the assumption of message authentication requires that faulty processors do not make undetectable modifications to messages as they are relayed from one processor to another.

This algorithm has a phased structure, and has been modeled as a network of four processes, each modeling one of the four replicated nodes; each of this processes is, in its turn, a network of automata, each ones modeling the phases of the algorithm.

Table 1. Number of states for the GUARDS Byzantine Agreement.

Model of:	states
A single non faulty node	428
Network of 4 non faulty nodes	3479
Network with an arbitrarily faulty node and a symmetric faulty node	109613
Network with an arbitrarily faulty node, and authentication violation	122767

Table 1 presents the size of the state space of the single node, and that of the network composed of four nodes under different fault assumptions. The fault assumptions have been modeled by means of specific processes which constraint the occurrences of faults. The table clearly shows:

- the fact that the size of the state space is largely below the fourth power of the size of the state space of a single node.
- the increase of the state space with the generality of the fault assumptions.

5 Conclusions

We believe that model checking can play a major role in the validation of safety critical systems. Model checkers are ready to be introduced in the industrial development process, aside traditional development techniques. Still state explosion represents the main problem for handling large industrial systems in many fields, but the current research efforts in search of powerful algorithms are promising. In particular, we have shown in this paper how the analysis of typical characteristics of fault tolerant systems can be exploited to tackle state explosion problems.
These results are not depending on the particular formalism chosen to model fault tolerant systems and can be applied also if diverse redundancy is employed.

Moreover, the regular structure in a redundant system may be exploited to contain state explosion with the help of other techniques as well, such as *symmetries* and *reduction preorders*. Using symmetries, as proposed by Emerson e. g. in [15], the number of states is reduced by identifying those states which

coincide up to a permutation of the system components. Reduction preorders [19] employ the independency of the property to be checked from the order in which interleaved processes are actually executed, to select just one order and hence only a subset of the state space to check its validity. In the case of reduction preorder applied to phase structured system, for example, the global state space of a phase i will be reduced to be of the order of $n * S_i$, and therefore the global state space of the overall algorithm can be reduced to $n * S$.
The ability of using specific techniques to deal with state explosion problem, like the ones we have shown for redundant systems, provides the opportunity of defining a specific formal verification process for a given application field, thus decreasing the need of general verification expertise in industries.

References

1. C. Bernardeschi, A. Fantechi, S. Gnesi. Formal Validation of Fault Tolerance Mechanisms inside GUARDS, SAFECOMP'99, Toulouse, September 1999, LNCS 1698.
2. C. Bernardeschi, A. Fantechi, S. Gnesi, S. Larosa, G. Mongardi, D. Romano. A Formal Verification Environment for Railway Signaling System Design, in Formal Methods in System Design 12, 139-161, 1998.
3. R.E. Bryant. Graph Based algorithms for boolean function manipulation, *IEEE Transaction on Computers*, C-35(8), 1986.
4. J.R. Burch, E.M.Clarke, K.L. McMillan, D. Dill, J. Hwang. Symbolic Model Checking 10^{20} states and beyond, in Proceedings of Symposium on Logics in Computer Science, 1990.
5. A. Cimatti, F. Giunchiglia, G. Mongardi, D. Romano, F. Torielli, and R. Traverso. Formal Verification of a Railway Interlocking System using Model Checking. Formal Aspects of Computing, Volume 10, Nr. 4, 1998.
6. E.M. Clarke, E.A. Emerson, A.P. Sistla,"Automatic Verification of Finite–State Concurrent Systems Using Temporal Logic Specification," ACM Transaction on Programming Languages and Systems, 8(2):244-263, 1986.
7. E.M. Clarke, E.A. Emerson, A.P. Sistla, S. Jha, Symmetry reduction in model checking. Proceedings of CAV'98, Lecture Notes in Computer Science 1427, pp.147–158.
8. W. Damm, J. Klose. Verification of a Radio-based Signaling System Using the STATEMATE Verfication Environment, To appear in Formal Methods in System Design.
9. C. Eisner. Using Symbolic Model Checking to Verify the Railway Stations of Hoorn-Kersenboogerd and Heerhugowaard. In Proceedings 10th IFIP WG 10.5 Advanced Research Working Conference, CHARME '99, Bad Herrenalb, Germany, September 1999.
10. W. Damm, J. Klose. Verification of a Radio-based Signaling System Using the STATEMATE Verfication Environment, To appear in Formal Methods in System Design.
11. C. Eisner. Using Symbolic Model Checking to Verify the Railway Stations of Hoorn-Kersenboogerd and Heerhugowaard. In Proceedings 10th IFIP WG 10.5 Advanced

Research Working Conference, CHARME '99, Bad Herrenalb, Germany, September 1999.
12. E. A. Emerson and A. P. Sistla, Symmetry and model checking, CAV'93.
13. W. Damm, J. Klose. Verification of a Radio-based Signaling System Using the STATEMATE Verfication Environment, To appear in Formal Methods in System Design.
14. C. Eisner. Using Symbolic Model Checking to Verify the Railway Stations of Hoorn-Kersenboogerd and Heerhugowaard. In Proceedings 10th IFIP WG 10.5 Advanced Research Working Conference, CHARME '99, Bad Herrenalb, Germany, September 1999.
15. E. A. Emerson and A. P. Sistla, Symmetry and model checking, CAV'93.
16. W.J. Fokkink and P.R. Hollingshead, Verification of interlockings: from control tables to ladder logic diagrams. in (J.F. Groote, S.P. Luttik and J.J. van Wamel, eds.) Proc. 3rd Workshop on Formal Methods for Industrial Critical Systems - FMICS'98, Amsterdam, pp. 171-185, Stichting Mathematisch Centrum (May 1998)
17. L. Gong, P. Lincoln, J. Rushby. Byzantine Agreement with Authentication: Observations and Applications in Tolerating Hybrid and Link Faults. Proc. 5th Conference on Dependable Computing for Critical Applications (DCCA-5), Urbana-Champaign, Il, USA, 1995.
18. L. Lamport, R. Shostak, M. Pease. The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems, Vol.4, N. 3, pp.382-401, July 1982
19. G.J. Holzmann and D. Peled. An Improvement in Formal Verification, Proc. FORTE 1994 Conference, Bern, Switzerland.
20. V. Roy, R. De Simone. AUTO and Autograph, in Proceedings of the Workshop on Computer Aided Verification, LNCS 531, 65-75, 1990.
21. Z100: Specification and Description Language SDL. ITU-T, June 1994