What are the Limits of Model Checking Methods for the Verification of Real Life Protocols?[*]

S. Graf, J.-L. Richier, C. Rodríguez, and J. Voiron
LGI, IMAG-CAMPUS, BP 53X
38041 GRENOBLE cedex
FRANCE
e-mail: graf@imag.imag.fr

Abstract
Model checking is a well known method to carry out formal verification of distributed systems. This method needs a model representing the behaviour of the system to be verified. The size of this model depends on the complexity of the system. To be able to verify real life systems, it is necessary to use techniques allowing to take advantage of all the available storage space, to reduce the amount of information needed for the verification and to speed up computation time. But the expressiveness of the languages used to describe the system and its expected behaviours (properties) limit the possible reductions. We present in this paper our choices for an automatic verification tool by using model checking. The preliminary results obtained from its use for the verification of a real life protocol allow us to make some estimations about the limits of model checking methods.

1 Introduction

In different domains, such as design of asynchronous circuits and communication protocols, design validation is an important aspect in order to increase the confidence in the final implementation. Simulation techniques, in which only a part of the possible behaviours is explored, are not sufficient. It is important to apply *formal* verification methods allowing to obtain a more reliable answer on the question of correctness. Given the complexity of the problems in these domains, it is important to use tools allowing to carry out automatically a large part of the verification. Our aim was to implement such a tool based on formal verification methods.

We have chosen communication protocols as application domain. In this domain, the complexity of a whole system can be significantly reduced for the verification purpose

[*]This work has been partially supported by the CNET, the French research programme C-cube (CNRS) and by the ESPRIT programme of the Commission of the European Community in the frame of the project Delta 4.

as it has generally a layered structure. Despite of this, the complexity of many layers remains still too important. Thus, other methods to break down the complexity must be imagined.

We have implemented the tool XESAR [RRSV87a] allowing to verify communicating systems. For the verification of a system by using XESAR, a formal description of this system and its expected service is needed. For the description of the protocol a variant of the ESTELLE language [ISO88a] is used and for the description of the service formulas of a branching time temporal logic. Then, the graph of all reachable states is constructed. On this state graph, the service is verified by model checking. A similar verification tool for the verification of asynchronous circuits has been implemented in [CES83].

The major problem we have encountered is to master the explosion of the number of useful states of the state graph, which induces technical problems concerning the management of space for its representation and the time for the construction of the graph and for the evaluation of formulas. This is inherent to verification by model checking. Here, we make some propositions to push away the limits of model checking due to this explosion of the state graph.

In section 2 of this paper, the principle of our method of automated verification is given, the choices made in XESAR are discussed. In section 3, the principle of the implementation of XESAR is presented as well as some methods improving its performances with respect to the size of the state graphs which can be generated; these methods have been implemented. In section 4, an experiment of the verification of a real life multicasting protocol within the Esprit project Delta 4 [GV89] is presented. In section 5, the question of the limits of model checking techniques for the verification of large systems is discussed.

2 Automatic verification of protocols

Our aim was to implement a tool for automatic verification of the design of the implementation of real life communication protocols. This can be considered as a very ambitious goal as these protocols are in general of very high complexity. However, most protocols respect a layered hierarchical structure such as the recommendation of the basic reference model for the Open System Interconnection ISO 7498; thus, the complexity of the problem can be reduced by proving the protocol in a hierarchical way, layer by layer, under the assumptions that the other layers, constituting the environment, are correct. Each protocol layer has to deliver a given *service* to its environment. Therefore, it is sufficient to prove that each layer delivers its service under the condition that, on the other side, its environment delivers the required service to the layer under study. This environment can be represented by a set of processes delivering the required service. In order to obtain the simplest possible description of the environment, non-deterministic choices are often used.

Several methods have been proposed for the verification of protocols: methods based on formal proof systems as in [OG76], formal transformation methods, where the program is reduced formally modulo an equivalence preserving the validity of the service, as for example observational equivalence [Mil80], methods based on the verification of the containment of the automata representing the program and the specification [Kur86], or methods based on model checking [BMP83].

2.1 Formalisms for the protocol description

For the verification purpose, we need a formalism for the description of the protocol layer and of its environment. It is desirable this formalism to be of high level, in order to avoid description faults. Its semantics should allow a description *as close as possible* to the implemented protocol, since the gap between the verified design to the implementation should be as small as possible. As usually the implementors use variables to encode a part of the control, this should also be possible in the chosen formalism.

This led us to choose the protocol description language ESTELLE/R, which is a variant of the Formal Description Technique ESTELLE [ISO88a] proposed by ISO. ESTELLE is an extension of PASCAL obtained by the addition of communication primitives for the description of a system of communicating processes in a well structured manner.

As PASCAL has no formal semantics, deductive methods and formal transformation methods can not be applied directly on an ESTELLE/R program text. To obtain a model on which automatic verification methods can be applied, we have chosen verification by exhaustive execution of the program.

Thus, the question arises, if for example LOTOS [ISO88b] would not be a better choice as all data aspects are described by abstract data types, and consequently a formal semantics exists. Thus, reductions can be made on a more abstract model than the state graph. We are studying a verification method using LOTOS in which we take advantage of the formal definitions of the data structures [Gar89].

2.2 Formalisms for the service description

The service to be delivered is given by a set of properties characterizing possible or mandatory sequences of message exchanges at the interfaces between the concerned layer and its environment. Often, these properties take also into account the exchanged values. For example, such a property may be of the following form in the framework of the verification of the multicasting protocol described in section 4:

> if a transmitter transmits some message with value i, then its recipient will inevitably receive the same message.

This kind of properties can be easily expressed by using temporal logics. The formulas of these logics express the service properties by using as basic predicates boolean expressions on the variables of the ESTELLE/R program and predicates expressing that a message exchange "has just taken place" or "is possible" in some state of the system.

We have chosen a branching time logic LTAC [Gra84], equivalent to the logic CTL [CES83], as it has a sufficient expressive power for the expression of the service of protocols. Moreover, *efficient* model checking algorithms for the evaluation of its formulas exist.

2.3 Verification by model checking

We have chosen verification by model checking on a model obtained by exhaustive generation: the description of the protocol and its environment is transformed into the graph of all reachable global states of the protocol, where each state contains all the information of control and data of all processes involved in the description. The transitions between states are labelled by the communications between two processes leading to this transition. There are also transitions *internal* to processes, for example a process can non-deterministically choose to send one or another message.

The formulas describing the service properties are evaluated on this graph taken as a model for the logic. These evaluation methods can be found in [QS82,CES83]. The protocol has the property expressed by a formula f if and only if f is true in *all the states* of the model.

As a complete state graph is constructed, this graph must obviously be finite. Thus, the domain of all variables of the protocol must also be finite. This is not really a restriction, as in protocols this is almost always the case.

We have also to point out that each finite graph constructed represents a particular closed system. Protocols describe the communication between *stations* through a network; each closed system to be verified corresponds to a configuration with a *fixed* number of stations. Unfortunately, as far as we know, no general inductive methods exist allowing to deduce the correctness of a net with $n+1$ stations from a net with n stations; therefore, the correctness of the generic protocol with any number of stations can not be proved by this way.

Furthermore, due to the nature of our verification method, we have two problems concerning the expression of properties.

One is the problem of fairness in the case that we want to verify liveness properties. Since a *finite* model of the behaviour is constructed, but the environment is generally described in a very non-deterministic manner and interleaving semantics is used for parallelism, this model contains inevitably "unfair" execution sequences. These sequences can be considered as unrealistic behaviours. For example, in the state graph produced for the protocol presented in section 4, there are many sequences in which the broadcasting of a message never terminates. These sequences must be considered as unrealistic ones, as they contain too many occurrences of faults in the medium or in one of the concerned stations, but they cannot be identified exactly by using the different notions of fairness as defined in [LPS81,Lam80,QS83]. The properties we prove in order to avoid this problem are the following: "if a broadcast terminates, it terminates *properly*" and "termination remains always possible".

Another problem is that some properties assume that *all* messages sent by some process are distinguishable, which requires an infinite set of message identifications if infinite execution sequences are observed. In this case, the properties concerning these messages have to be slightly reformulated so as only finite intervals of each sequence are considered. For example, consider the property:

> Any positively confirmed message will inevitably be indicated to all correct addressed stations. (1)

The problem is the universal quantification of messages. There is no difficulty to express the fact that *all* addressed stations must receive the message, since there is only a finite number of stations for each message emission.

In order to express the property (1) simply by

positive-confirmation(*mess*) $\Rightarrow$ inevitably (indication(*mess*)),

all message identifications *mess* have to be different which is obviously impossible in our model. The solution is to consider a finite set of message identifications, used in a cyclic manner, and to reduce the observation to the interval between the emission of two messages with the same identification. Then, we prove the property

positive-confirmation(*mess*) $\Rightarrow$ (2)
inevitably indication(*mess*) before next-emission(*mess*).

It is clear that if (2) is verified, the original property (1) is verified, but not reciprocally. In order that (2) can be verified, the set of message identifications, determining the size of the interval of observation, must be sufficiently large.

2.4 Practical issues

The major problem we have encountered by using model checking techniques is the explosion of the size of the state graph. If we want to be able to deal with real life protocols, we have to develop techniques allowing to compress informations and to reduce the graph size. Some approaches that could be considered are:

- The use of equivalence relations to merge states during the graph construction. It is clear that two states can be merged if and only if they are equivalent with respect to the properties to be verified. The major drawback of this approach is that, as far as we know, its implementation leads to unacceptable execution time since states may be created and eliminated again too many times.
- By restricting ourself to a subclass of safety properties (those which can be characterized by deterministic Büchi automata where all states are repetitive) very efficient checking algorithms exist, in which each state has to be visited only once. Thus it is possible not to store explicitly the state graph and to evaluate formulas "on the fly" as proposed in [Hol87]. The most efficient versions of this method do not store the states, but use hashing techniques to find out if a state has already been visited. The disavantage of this method is that there is no certitude that the graph has been completly explored and it is difficult to estimate the coverage.
- Reductions at the description level: some protocol layers could be divided into sub-layers; also, it is often possible to identify several independent phases inside a given protocol. As each sub-layer or independent phase can be verified separately, the size of the graphs to be generated could be considerably reduced.

It is clear that all these methods have as a consequence that one has less certainty about the validity of the design of the protocol: each detected error corresponds to an error in the design, but it cannot be assured that all errors are detected.

3 The XESAR tool and its limits

To carry out the automatic verification of protocols, we implemented a tool, called XESAR [RRSV87a,RRSV87b], in which all the verification steps described in section 2 are implemented.

We have already pointed out that to verify service properties we need to store the graph representing all the behaviours of the protocol in order to evaluate the formulas and to ensure the termination.

A first version of XESAR was designed to be very fast and thus uses only the main memory to store the graph. The space limitations reduce the applicability of this version to small systems. With the second version, we pushed away these limitations by keeping most of the graph in the secondary storage devices.

We describe some aspects of the different versions of XESAR and we present estimations of their limits. First, we shortly describe how the verification process is carried out in XESAR.

3.1 The verification process of XESAR

In XESAR the following parts can be distinguished:

- The automatic construction of the state graph. This is done in two phases:
  1. The construction of a set of automata coding the global control. With each transition is associated a PASCAL procedure describing the management of the values of state variables.
  2. The exhaustive execution of the automata to produce the complete state graph. Each state of the generated graph represents global control and data information. The exhaustive generation process may be summarized as follows: first, the initial state of the system is generated; after that, for each state all the possible transitions are executed to generate its successors. The termination is ensured by the fact that it is detected if a state has already been generated.
- The evaluation of formulas characterizing the service on the generated graph used as model. In fact, this verification of a formula may succeed or fail. In the latter case, one or more erroneous states are identified, and then, the faults in the ESTELLE/R program may be detected by using a careful semi-automated analysis [Ras88].

3.2 The implementations of XESAR

Two version of XESAR have been implemented: one designed to verify small systems and the other designed to verify larger ones.

3.2.1 XESAR for small systems

The first implementation of XESAR was intended to carry out the verification process as fast as possible. For this reason, during the graph generation all the informations needed for the exhaustive generation are kept in the main memory. Consequently, with this version only the models corresponding to small systems can be generated. The aim was to generate graphs with about one hundred thousand states and some hundred thousand edges in some minutes on a middle size workstation.

After the termination of the graph generation, the values of variables for each state and the set of edges are saved in secondary storage. During the evaluation phase, the user can evaluate any basic predicate on the state variables, as their values are accessible. The algorithms implemented [Rod88] to evaluate the temporal operators are very efficient: the average response time is less than a second for a formula with less than five temporal modalities on a Sun/3 workstation.

3.2.2 XESAR for larger systems

If we try to verify real life protocols with a tool like XESAR, we are confronted with its main limitation: the size of the model, even after reduction.

The second version of XESAR, mainly intended to verify large size systems, has been designed to use *all the available storage space* and to generate graphs with several million states. For this reason, most of the graph is kept in secondary storage during its generation. As for each generated state it is necessary to determine if it already exists in the state set, new data structures to manage the state set were defined in order to reduce the number of disk accesses.

The state set is organized using the Dynamic Hashing technique [ED88]. This technique, mainly used in data base systems, allows to calculate the page in the secondary storage where a given state must be located by applying some function to it. Then, the state is compared with the states of this page to decide if it already belongs to the state set. The number of comparisons is limited by the page size. It is important to point out that in our implementation neither the number of disk accesses nor the maximal number of comparisons per state depend on the total state number, and that the maximal number of pages allowed by this technique is only limited by the storage capacity of the host system. The main disadvantage of our implementation of the Dynamic Hashing is the space utilisation: to ensure that the page containing a given state can be obtained in one disk access implies an expected utilisation rate of 69% [ED88].

The new organization of the state set is completed by some efforts to reduce the used space. In the previous version of XESAR, a state was the composition of the control information with the values of all the state variables in that state. With this representation simplified data structures can be defined and all the data needed to evaluate the basic predicates during the evaluation phase is accessible. In the new representation, the value spaces of each process type are separated. A state is the encoding of the values of each process in the state. The state size is calculated from a upper bound of the number of values of each process type. Notice that the expected sizes of the value spaces could be automatically computed from the variable definitions.

In this version of XESAR, after the termination of the exhaustive generation, neither the value spaces of the processes nor the state set are stored. This implies that the evaluation of boolean expressions on the state variables must be made during the graph generation phase. During the evaluation phase only the already evaluated expressions may be used in formulas.

3.3 The limits of XESAR

With this new structure XESAR can generate graphs as large as it is allowed by the characteristics of the host system. If we have of a secondary storage device with a capacity of one Giga byte, graphs of some million states can be generated. The exact value depends mainly on the state size and of the output degree of the graph. Duplicate or triplicate the storage capacity will duplicate or triplicate the capability of XESAR. Considering the current storage capacities, we can conclude that the largest graphs that XESAR may generate and store would have some million states. Nevertheless, the time of the global validation process increases significantly, and becomes the major limitation. Notice that changing our organisation of the state set to obtain a better space utilisation does not imply a significant increase of the size of the graph XESAR could generate.

It is important to point out that the improvement of the performances of XESAR has made the verification procedure slightly less automatic: the user must tune some parameters of the graph generation process, such as the size of the pages.

4 Experiment of verification

We are carrying out the verification of a low level broadcast protocol by using XESAR in the framework of the Delta 4 project. The overall aim of this project is to define an open and dependable distributed real-time computing architecture; the approach of dependability is based on the user-transparent replication of software components on distinct host computers. In [PBS*88], an overall framework and the current Delta 4 implementation are presented. In this project, the design of some crucial parts of the architecture of the system under development has to be verified, for example the communication system that provides reliable multi-endpoint communication. The communication in the Delta 4 system is supported by stations connected through a local area network.

Using the OSI model terminology, we are concerned in this experiment with a the medium access control (MAC) sub-layer. In short, the service of "message multicasting" is:

> to deliver the same messages to all correct (fault-free) recipient stations, in the same order to all of them, within a known bounded time. Each delivered message has been transmitted by the sending station, on request of the user. A message is always delivered unless some recipient is inaccessible.

This service on which all facilities of the communication system are built, is provided by the Atomic Multicast protocol (AMp) which belongs to the class of reliable broadcast protocols implementing Byzantine Agreement. The description of the whole service is given in [GV89].

The considered AMp protocol is based on extensions of the standard Token Ring Protocol, IEEE 802.5 [IEE85]. As a centralized commit protocol, it is subject to failures of the coordinator, in this case the sending MAC entity. Failures of the sending MAC entity are recovered by a *recovery procedure*, that effectively blocks all further communications in the affected group of stations until consistent decisions have been taken by all recipients residing on a fault-free station. This recovery procedure causes a large number of states in the model.

The aim of verification is to ensure that the *design* of the AMp protocol is correct, in the sense that from this design an implementation providing the AMp service could be derived.

We have at present interesting results on the verification work on this protocol, but it is clear that even with the second version of XESAR described in the previous section, and by using decomposition techniques taking advantage of the structure of the protocol itself, the design can be verified only with a small number of stations connected through the network. Due to the size of the protocol, we have considered separately the two phases of "normal functioning" (in which no error occurs) and recovery procedure. The graph obtained for a network with four stations when only normal functioning is possible has at least three million states (depending on the number of stations which can send messages). On this graph safety properties have been verified.

5 Conclusion

Our aim was to develop a tool for the verification protocols as the one presented in the preceding section. We wanted to use a formalism for the description of protocols which should be of high level and allow a description as near as possible to the implementation. Furthermore, we wanted a maximum of verification steps to be carried out automatically.

For this reason, we have discarded many verification methods and have chosen a variant of ESTELLE as description language and model checking as verification method, for which we need unfortunately an exhaustive state graph.

By using XESAR, it should be possible to evaluate the limits of this approach to verification since it can generate graphs as large as allowed by the secondary storage of the host system. In the case of the protocol described in the preceding section, in which many data structures and variables are defined, it is possible to generate a graph of some million states.

Using less expressive languages for protocol description (only integer variables and communications without value exchange), simpler generation algorithms can be implemented and much larger state graphs could be traversed. But descriptions of these nature are in general rather far from the actual implementation.

As many protocols may produce state graphs with more than some million states, methods allowing to construct a reduced state graph or "on the fly" techniques for the evaluation of formulas have to be developed. For the first method, we are studying a verification method based on LOTOS, in which data structures are described by formal data types; this can be used to construct reduced state graphs; however, the gap between the protocol description and the implementation will be increased. If we want to consider the second method, the expressiveness of the formalism used to describe the service specifications has to be significantly reduced.

For the verification of the multicasting protocol that we are carrying out at present, we reduce the size of the model by taking advantage of the layered structure of the algorithm and of the independence of some functionalities, but in each of these cases a lot of manual proofs have to be made in order to justify their use. Furthermore, these proofs must not be based on too restrictive hypotheses concerning the protocol.

By using these methods, some significant results have been obtained for the protocol described in the preceding section. But they will not be quite fully satisfactory. In any case, a more constructive approach, with the assistance of verification tools, should be a better solution to cope with such examples that need for the verification of service properties a large amount of information in the associated state graph.

References

[BMP83] M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. *Acta informatica*, 20, 1983.

[CES83] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic. In *10th Annual ACM Symposium on Principles of Programming Languages*, 1983.

[ED88] R. J. Enbody and H. C. Du. Dynamic hashing schemes. *ACM Computing Surveys*, 20, June 1988.

[Gar89] H. Garavel. *Compilation et vérification des programmes Lotos.* Thèse, Université Joseph Fourier, Grenoble, to be published, 1989.

[Gra84] S. Graf. *Logiques du temps arborescent pour la spécification et la preuve de programmes.* Thèse de Troisième Cycle, Institut National Polytechnique de Grenoble, February 1984.

[GV89] S. Graf and J. Voiron. *Using Temporal Logic for Specification and Verification of a Multicast Protocol.* Technical Report SPECTRE RTC12, Laboratoire de Génie Informatique — Institut IMAG, Grenoble, January 1989.

[Hol87] G. Holzmann. On limits and possibilities of automated protocol analysis. In *Proceedings of the 7th International Workshop on Protocol Specification, Testing and Verification*, 1987.

[IEE85] IEEE. *IEEE Standards for Local Area Networks: Token Ring Access Method and Physical Layer Specification.* International Standard, IEEE, 1985.

[ISO88a] ISO. *ESTELLE — A Formal Description Technique Based on an Extended State Transition Model.* International Standard 9074, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, September 1988.

[ISO88b] ISO. *LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour.* International Standard 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, September 1988.

[Kur86] R. P. Kurshan. *Testing Containment of ω-regular Languages.* Technical Report 1121-861010-33-TM, Bell Laboratories, 1986.

[Lam80] L. Lamport. Sometimes is sometimes "not never"- on the temporal logic of programs. In *7th Annual ACM Symp. on Principles of Programming Languages*, pages 174–185, 1980.

[LPS81] D. Lehmann, A. Pnueli, and J. Stavi. *Impartiality, Justice and Fairness: The Ethics of Concurrent Termination*, pages 264–277. Volume 115 of *Lecture Notes in Computer Science*, Springer-Verlag, edition, 1981.

[Mil80] R. Milner. *A calculus of communicating systems.* Volume 92 of *Lecture Notes in Computer Science*, Springer-Verlag, 1980.

[OG76] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs. *Acta informatica*, 6:319–340, 1976.

[PBS*88] D. Powell, G. Bonn, D. Seaton, P. Verissimo, and F. Waeselyck. The Delta 4 approach to dependability in open distributed computing systems. In *Proceedings of FTCS-18*, IEE Computer Society Press, June 1988.

[QS82] J.P. Queille and J. Sifakis. *Specification and Verification of concurrent systems in* CESAR, pages 337–357. Volume 137 of *Lecture Notes in Computer Science*, Springer-Verlag, April 1982.

[QS83] J.P. Queille and J. Sifakis. Fairness and related properties in transition systems. *Acta Informatica*, 19, 1983.

[Ras88] A. Rasse. *CLEO, Interprétation de la non-correction de programmes sur un modèle.* Technical Report SPECTRE RTC10, Laboratoire de Génie Informatique — Institut IMAG, Grenoble, June 1988.

[Rod88] C. Rodríguez. *Spécification et validation de systèmes en* XESAR. Thèse de l'Institut National Polytechnique de Grenoble, May 1988.

[RRSV87a] J.-L. Richier, C. Rodríguez, J. Sifakis, and J. Voiron. XESAR*: A Tool for Protocol Validation. User's Guide.* Laboratoire de Génie Informatique — Institut IMAG, Grenoble, September 1987.

[RRSV87b] J.-L. Richier, C. Rodríguez, J. Sifakis, and J. Voiron. Verification in XESAR of the sliding window protocol. In Harry Rudin and Colin H. West, editors, *Proceedings of the 7th International Symposium on Protocol Specification, Testing and Verification (Zurich)*, IFIP/TC6, May 1987.