Towards Checking Object Systems efficiently

Maik Kollmann

Technical University Braunschweig – Information Systems,
Mühlenpfordtstr. 23, 38106 Braunschweig, Germany.
M.Kollmann@tu-bs.de

Abstract. Verifying object systems using model checking is often limited to checking the global model. This leads to the well known problem of state explosion. We presented a method that avoids the generation of the global state space verifying formulas concerning several objects incrementally. In this work we give an idea towards a systematic creation of checking conditions in multi-object logics. The special focus is on object systems using asynchronous communication.

1 Introduction

The verification of realistic systems is still a hard problem, no matter what kind of systems are under investigation. In [Pin02,EKP02,EKP03] we illustrate an incremental method which enables checking if a certain checking condition that belongs to several objects holds. As this method avoids the creation of the global state space, the checking speed is improved dramatically.

Our method makes use of the multi-object logics D_1 and D_0 in which checking conditions typically consist of two parts. The first part is related to the particular object. Furthermore checking conditions in multi-object logics can be nested – the second part. A formula in multi-object logics concerning objects j and i using synchronous communication then consists of j containing i ($j.(\ldots\ i.(\ldots)\ \ldots)$) or vice versa.

Reasoning about objects that communicate in a synchronous fashion is already in practice. In this case even checking conditions that belong to objects that communicate indirectly is possible. In this work we concentrate on object systems with asynchronous communication and some ideas towards an algorithm for generating multi-object logics formulas addressing this topic.

2 Checking – Model Checking

Automatic verification techniques like model checking [EC81,QS81] are well-established in the verification of hardware protocols and other areas. Not only in research but also in industry tools like SMV [McM96], SPIN [Hol97] etc. are used successfully. Applying model checking to software demands usually higher requirements as to hardware resources in order to be able to handle the typically larger state space. BDDs in various versions and other techniques like partial order reduction to lessen and/or avoid the state explosion [Val90] are often not

that sufficient, in order to cope with these requirements. In addition, the requirements of real time demands increase or the need of supporting continuous processes and/or parameters arises.

Compositional methods often make a substantial reduction of the state space possible. However, they offer only semi-automatic mechanisms and require with increasing efficiency more and more expert knowledge (cf. [Pin02,dR97]). In the age of the Internet or networks in general various advantages in computing have become available (e.g. grid computing or the use of clusters). While computing power is available, methods are still missing, which make automatic verification applicable to real world problems.

In section 3 we give a short introduction to the method that was presented for the verification of synchronous multi-object systems [Pin02,EKP02,EKP03]. This method makes use of the communication among the objects on the one hand and uses on the other hand an intuitive logic, in order to formulate global properties.

3 Multi-Object Logics and Checking

In [EC00], a sound and complete translation $\mathsf{D}_1 \rightarrow 2^{\mathsf{D}_0}$ is presented: working inside-out, every formula $\varphi \Leftrightarrow i.(\ldots j.\psi \ldots)$ with an innermost communication subformula $j.\psi$ is replaced by the D_1 formula $\varphi' \Leftrightarrow i.(\ldots\ q_i\ \ldots)$ and the D_0 formulae $\delta \Leftrightarrow j.(q_j \Leftrightarrow @i \wedge \psi)$, $\alpha \Leftrightarrow i.(q_i \Rightarrow j.q_j)$, and $\beta \Leftrightarrow j.(q_j \Rightarrow i.q_i)$. q_i and q_j are propositional symbols to be matched with existing ones in the signatures P_i and P_j, respectively. The formulae α and β are called *communication requirements*. Since the number of communication subformulae is finite and strictly decreases in each step, the transformation terminates after a finite number of iterations.

Using this transformation global D_1 checking conditions can be broken down into sets of D_0 conditions which can be checked locally, and communication symbols which have to be matched with existing ones according to the communication requirements. This is elaborated in [EP00,PE01,Pin02].

The matching algorithm is based on the translation steps $\varphi \Leftrightarrow i.(\ldots j.\psi \ldots) \longrightarrow \{\varphi', \delta, \alpha, \beta\}$ as given above. Here is a rough sketch. For each step, the following actions are performed:

1. Compute the set S_ψ of states of object j in which ψ holds. This is done by model checking.
2. Retrieve the set R^i_ψ of all communication symbols $r_j \in P_j$ that occur in a subset of S_ψ and establish communication with i (i.e., the D_0 communication formulae $j.(r_j \Rightarrow i.r_i)$ and $i.(r_i \Rightarrow j.r_j)$ hold for some $r_i \in P_i$).
   In other words, R^i_ψ is the set of all communication symbols in j having corresponding 'partner' symbols in i such that α, β, and δ as given above hold true.
3. Let Q^i_ψ be the set of communication symbols $q_i \in P_i$ that are in α-β correspondence with symbols $q_j \in P_j$; let
$$\varphi' \ \equiv\ i.(\ldots \bigvee\nolimits_{q \in Q^i_\psi} q \ldots).$$

φ holds iff φ' does. If φ' has no further communication subformulae, it can be locally model checked. Otherwise, the above step is repeated.

It may happen that one or the other of the above sets of communication symbols is empty. That would be the case if there is no state in j satisfying ψ, or if there is no suitable match of communication symbols. In this case, the disjunction in φ' is empty and evaluates to *false*, indicating that there is no ψ preserving communication to j. If so, detailed warnings may be given to the user helping to find the error in the design.

Further details can be found in [Pin02] where also the correctness of the algorithm is proved. Most interestingly, all checkings necessary for establishing global validity of the checking condition can be done on the local models of the objects. A prototype uses standard model checkers not only for local model checking but also for essential parts of matching the communication symbols and checking the communication requirements.

4 Checking Asynchronous Object Systems

A lot of systems make use of asynchronous communication. Several protocols and applications used in networks are based on asynchronous communication. Asynchronous communication is often a one way communication as to information flow. Only the receiving object gains information iff the message arrives. We do not address the problem of losing messages or additionally introduced messages by some kind of environment or an intruder. In this work we only concentrate on the flow of information, assuming perfect communication without side effects.

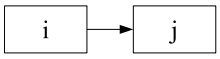

Fig. 1. Object i sending messages to object j. j becomes an observer: $j.(\ldots\ i.(\ldots)\ \ldots)$.

If an object j sends messages to an object i in an asynchronous fashion, checking conditions in multi-object logics regarding j and i can only be expressed by i containing j (cf. figure 1).

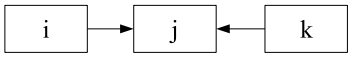

Fig. 2. Objects i and k sending messages to object j. j becomes an observer: $j.(\ldots\ i.(\ldots)\ \ldots\ k.(\ldots)\ \ldots)$

In general a checking condition in multi-object logics addressing several asynchronously communicating objects demands an object that is able to observe

the mentioned objects $o_1, o_2, \ldots o_n$. Suitable observer objects are detected following the flow of information from every single mentioned object. If objects communicate directly, the receiver becomes an observer (cf. figure 2). Otherwise all objects on a directed path from one object $o_i \in o_1, o_2, \ldots o_n$ to an object $o_j \in o_1, o_2, \ldots o_n$ $(o_i \neq o_j)$ become intermediate observers (cf. figure 3).

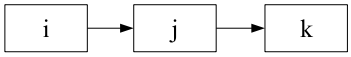

Fig. 3. Objects i and sending messages to object j and j sending messages to k. j becomes an intermediate observer: $k.(\ldots\ j.(\ldots\ i.(\ldots)\ \ldots)\ \ldots)$.

Iff there exist directed paths from all objects in $o_1, o_2, \ldots o_n$ and these paths form a directed tree, a multi-object logics formula can be derived directly. According to figure 2 formulas containing i and k can be checked in j: $j.(\ldots i(\varphi) \ldots k.(\psi) \ldots)$.

Unfortunately some formulas of interest contain more objects than those on a spanning directed tree. In this case there exist no real observer objects in the system (cf. figure 4).

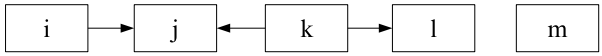

Fig. 4. There exists no observer to evaluate formulas containing objects i and l or m in the object system.

In order to evaluate checking conditions efficently, an appropriate observer object needs to be introduced. This observer object only consumes messages from mentioned objects. In addition, the objects under investigation have to send messages to the observer object(s). These messages depend on the part of the formula that belongs to a particular object. In general, some kind of predicate may be `true` or `false` is under investigation. Under these circumstances the message exchange reduces to `true` or `false`.

The worst case to deal with consists of a scenario in which there is no communication at all among any of the objects in a formula. Then the size of the observer objects grow with the amount of mentioned predicates.

But if there is no observer needed and nesting of formulas becomes applicable, verification can be done in linear time checking one object after the other.

5 Concluding remarks and Future Work

In this contribution we draw a raw sketch towards an efficent method for checking object systems communicating in an asynchronous fashion. The mechanism is

still under investigation but we look forward to be able to incorporate prior work [Pin02,EKP02,EKP03].

By now the automatic creation of D_1-formulas based on CTL- or LTL-formulas and corresponding observer objects is still a way to go.

Acknowledgements

Many thanks are due to Hans-Dieter Ehrich for his valuable hints during several discussions.

References

[dR97] Willem-Paul de Roever. The Need for Compositional Proof Systems: A Survey. In Willem-Paul de Roever, Hans Langmaack, and Amir Pnueli, editors, *Compositionality: The Significant Difference*, volume 1536 of *Lecture Notes in Computer Science*, pages 1–22, September 1997.

[EC81] E. Allen Emerson and Edmund M. Clarke. Characterizing Correctness Properties of Parallel Programs Using Fixpoints. *Lecture Notes in Computer Science*, 85:169–181, 1981.

[EC00] H.-D. Ehrich and C. Caleiro. Specifying communication in distributed information systems. *Acta Informatica*, 36(Fasc. 8):591–616, 2000.

[EKP02] H.-D. Ehrich, M. Kollmann, and R. Pinger. Distributed Model Checking. In D. Haneberg, G. Schellhorn, and W. Reif, editors, *Proc. of FM-TOOLS 2002*, pages 53–58, Augsburg, 2002.

[EKP03] H.-D. Ehrich, M. Kollmann, and R. Pinger. Checking Object System Designs Incrementally. *Journal of Universal Computer Science*, 9(2):106–119, 2003.

[EP00] H.-D. Ehrich and R. Pinger. Checking object systems via multiple observers. In *International ICSC Congress on Intelligent Systems & Applications (ISA'2000)*, volume 1, pages 242–248. University of Wollongong, Australia, International Computer Science Convetions (ICSC), Canada, 2000.

[Hol97] Gerard J. Holzmann. The Model Checker SPIN. In *IEEE Transactions on Software Engineering*, volume 23, pages 279–295, 1997.

[McM96] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, second edition, 1996.

[PE01] R. Pinger and H.-D. Ehrich. Compositional Checking of Communication among Observers. In H. Hussmann, editor, *Fundamental Approaches to Software Engineering (FASE), Part of the Joint European Conferences on Theory and Practice of Software (ETAPS 2001), Genova*, volume 2029 of *Lecture Notes in Computer Science*, pages 32–44, 2001.

[Pin02] R. Pinger. *Kompositionale Verifikation nebenläufiger Softwaremodelle durch Model Checking*. PhD thesis, Institut für Software – Abteilung Datenbanken, TU Braunschweig, 2002.

[QS81] J. P. Quielle and J. Sifakis. Specification and Verification of Concurrent Systems in CESAR. In *Proceedings of the Fifth International Symposium in Programming*, 1981.

[Val90] Antti Valmari. A Stubborn Attack on State Explosion. In *Computer Aided Verification*, volume 531 of *Lecture Notes in Computer Science*, pages 156–165, 1990.