Multi-Object Checking Counterexamples

Maik Kollmann, Yuen Man Hon

Multi-Object Checking Counterexamples

Maik Kollmann[1] Yuen Man Hon[2]

Institute of Information Systems
Technical University Braunschweig
38106 Braunschweig, Germany

Abstract

Model checking has become the most widely used technique for the verification of state based systems. In addition to its automation in checking whether a system model satisfies a specification, model checking provides a counterexample trace if the checking condition does not hold. Such a sequence of states illustrates how the model falsifies the specification. Multi-object checking utilizes model checking as the underlying verification technique for each object so that the verification of a system is only limited by the state space of the largest object. Our contribution proposes an enhancement to multi-object checking – the automated generation of a global system trace, if a specification in multi-object logic does not hold.

Keywords: Verification, Model Checking, Multi-Object Checking, Multi-Object Logic, Counterexample Generation.

1 Introduction

It has been shown in [EKP03] that large state based multi-object systems can be verified by multi-object checking using model checking [CGP00] as the underlying verification technique. Formulas in multi-object logic D_1 [EC00] which are based on well-known temporal logics like `CTL` or `LTL` [HR00] are used to capture the specifications of a multi-object system. By a multi-object system, we mean a couple of sequential objects collaborating concurrently and communicating synchronously in an RPC-like fashion. When checking such systems using automatic verification or testing methods, the complexity tends to grow exponentially with the number of objects: the state space typically "explodes".

Model checking tools like SPIN [Hol97] or SMV/NuSMV [McM96,CCB+02] incorporate the ability to illustrate that a model does not satisfy a checking condition providing a textual, a tabular or a sequence chart-like representation of the involved states. We believe that a graphical representation like message sequence charts is the most convenient one from a user's point of view to illustrate a counterexample in a multi-object system.

[1] Email: M.Kollmann@tu-bs.de
[2] Email: Y.Hon@tu-bs.de

Short presentation at AVoCS 2006
URL: avocs06.loria.fr

In the following section, we give a brief introduction to multi-object logic and multi-object checking in order to provide the basis for our approach. Our approach providing the generation of counterexamples for multi-object checking is depicted in section 3. Finally, we conclude our contribution and have a look at future work.

2 Multi-Object Systems, Logic and Checking

Let I be a finite set of identities representing sequential objects. Misusing terminology slightly, we speak of "object i" when we mean the object with identity i. Each object $i \in I$ has an individual set P_i of atomic state predicate symbols, its *signature*. In applications, these predicate symbols may express the values of attributes, which actions are enabled, which actions have occurred, etc.

We use the multi-object logic D_1 as described in [ECSD98,EC00] with a local logic $i.\mathtt{CTL}$ over signature P_i for each object $i \in I$. D_1 allows to use formulas from $j.\mathtt{CTL}$ for any other object $j \in I$ as subformulas within $i.\mathtt{CTL}$. These constituents are called *communication subformulas*.

[EC00] presents a transformation to break global D_1 checking conditions down into sets of D_0 conditions and communication symbols. These symbols have to be matched with existing ones according to the communication requirements. In addition, the D_0 conditions can be verified locally. Informally, the communication symbols are determined inside out. The D_1 formula holds iff the outermost D_0 formula holds. This is elaborated in [EKP03].

Example 2.1 An object system consists of objects with identifiers i and j. Fig. 1 illustrates the Kripke structures of i and j. The transition from state $s_1; b!i$ to state $s_2; a!i$ in j and the transition from state $s_1; b?j$ to state $s_2; a?j$ in i have to take place simultaneously (i and j synchronize via communication symbols a and b).

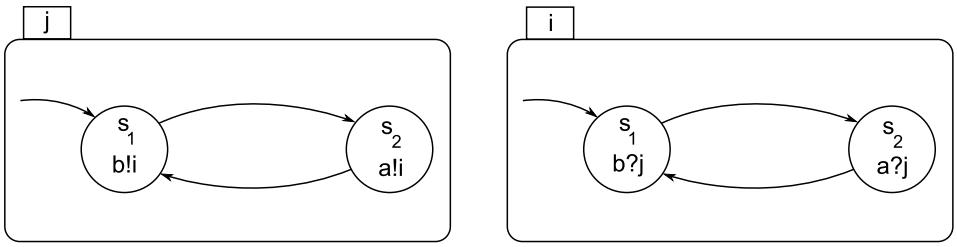

Fig. 1. An object system consisting of objects i and j.

In order to check whether it is possible that i is in state s_2 and that j is in state s_2 the D_1 formula has to be bound to one of these object identifiers. Out of i's perspective we will check: $\psi_{Ex1} \equiv i.(\mathsf{EF}(state = s_2 \wedge j.(state = s_2)))$. According to the transformation in [EC00] ψ_{Ex1} is broken down to:

$$\psi'_{Ex1} \equiv i.(\mathsf{EF}(state = s_2 \wedge q{:}\, j)) \text{ and } \phi_{Ex1} \equiv j.(q{:}\, i \Rightarrow state = s_2).$$

We compute all states in j where ϕ_{Ex1} holds by model checking and determine all communication symbols such that $q{:}\, i$ holds in a subset of these states. For each of these q, a corresponding predicate $q{:}\, j$ in i has to be found. In this example, only $a!i$ is appropriate ($a!i \Rightarrow state = s_2$ in j).

Let Q be the set of these communication symbols; let $\psi''_{Ex1} \equiv i.(\mathsf{EF}(state = s_2 \wedge \bigvee_{q \in Q} q{:}\, j))$. We substitute $\bigvee_{q \in Q} q{:}\, j$ in ψ''_{Ex1} and check whether ψ''_{Ex1} holds by local model checking. ψ_{Ex1} holds because $\psi''_{Ex1} \equiv i.(\mathsf{EF}(state = s_2 \wedge a))$ does.

In the next section, we describe an approach to generate a global counterexample illustrating the detected local error in a global scenario.

3 Generating Multi-Object Checking Counterexamples

When applying multi-object checking the result of the verification is finally determined by evaluating the outermost D_0 formula. If this formula does not hold, up to now the user himself has to analyze the generated trace for the related object and the information provided according to the communication with other objects. If the trace provided here does not encounter communication with further objects the local trace corresponds to a global counterexample. Our approach concentrates on those local traces that contain communication with further objects.

Definition 3.1 A local trace of a communication partner is corresponding to a local trace of an object, iff the order of exchanged messages between these objects is identical and all communication among these objects is covered.

Definition 3.2 We call a set of local traces a global counterexample iff there is a corresponding, local trace for each communication partner.

We have to distinguish between two situations: systems that have a treelike communication structure and those systems that have a more general structure as to communication (cf. fig. 2).

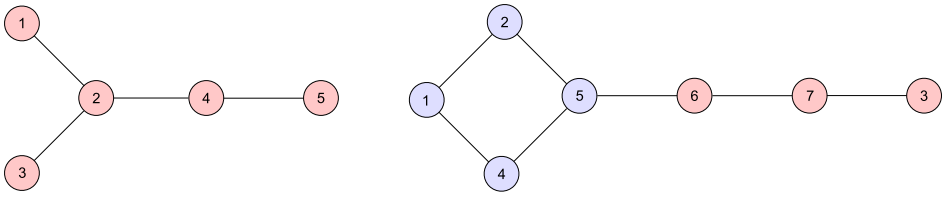

Fig. 2. Left: An object system featuring a treelike communication structure (nodes represent objects and edges illustrate communication among objects). Right: An object system with cyclic communication.

We describe our approach for treelike systems first.

3.1 Treelike Communication in Object Systems

The generation of a global counterexample for a treelike object system regarding communication is nearly straightforward. We developed the following algorithm:

Determine the mentioned communication partners $K_i \subset I$ in a particular local trace of the object with identity $i \in I$ and some checking condition $\psi_i \in D_0^i$. $K_{involved}$, the set of involved object identifiers is initialized: $K_{involved} = \{i\}$.

(i) All communication partners K_i of the object with identity i will have to participate in the global counterexample: $K_{involved} = K_{involved} \cup K_i$.

(ii) Generate an LTL formula ν_k for each $k \in K_i$ that mimics the mentioned communication with the object with identity i.

(iii) Negate ν_k and apply model checking of the model of the object with identity k and checking condition $\neg\nu_k$.

(iv) If the generated local trace for the object with identity k and checking condition $\neg\nu_k$ contains further communication, iterate the process until no new communication partner is discovered.

The set of all these local traces defines the global counterexample immediately.

3.2 Object Systems providing cyclic Communication Structures

If an object system provides a more general communication structure (see fig. 2), the algorithm in 3.1 may fail. A global counterexample for a given D_1 formula may be established by a set of traces for objects 1, 2, 4 and 5. Neither depth-first search nor breadth-first provide a suitable set of traces as to definition 3.2. In contrast, a global counterexample can be synthesized from a set of traces for objects 3, 6 and 7 as the generating subset does not contain any cyclic communication.

In order to generate a proper set of local traces we have to take the dependencies regarding communication into account. Therefore, we modify the algorithm for treelike communicating object systems as follows:

Determine the mentioned communication partners $K_i \subset I$ in a particular local trace of the object with identity $i \in I$ and some checking condition $\psi_i \in D_0^i$. $K_{involved}$, the set of involved object identifiers is initialized: $K_{involved} = \{i\}$.

(i) All communication partners K_i of the object with identity i will have to participate in the global counterexample: $K_{involved} = K_{involved} \cup K_i$.

(ii) Generate an LTL formula ν_k for each $k \in K_i$ such that ν_k mimics the mentioned communication with the object with identity i and all $k' \in K_{involved}$ $(k' \neq k)$ which communicate with k as well.

(iii) Negate ν_k and apply model checking of the model of the object with identity k and checking condition $\neg\nu_k$.

(iv) If the generated local trace for the object with identity k and checking condition $\neg\nu_k$ contains further communication, iterate the process until no new communication partner is discovered.

The algorithms differ in the necessity to take further communication among "neighboring" objects into account. The generation of the temporal logic formula ν_k may offer a number of alternatives as the order of messages can allow several orders. The order of messages of all these objects that have to be taken into account is "shuffled" in such sense that the partial order of all messages is preserved. If $\alpha(k, j)$ denotes the number of messages which objects k and $j \in K_{involved}$ exchange, the number of possible formulas ν_k and therefore the number of traces for each object j is limited by $\prod_{i=1}^{n} \alpha(k_i, j)$ if n determines the maximum number of communication partners of j.

Example 3.3 Example 2.1 depicts a multi-object system with objects i and j. Checking whether $\psi_{Ex2} \equiv i.(\neg\mathsf{EF}(state = s_2 \wedge \mathsf{EF}(state = s_1))$ holds, shows the sequence of states of i in fig. 3. This sequence implies communication with j. The LTL formula $\nu_k \equiv \neg(\mathsf{F}b!i\ \mathsf{U}\ (\mathsf{F}a!i\ \mathsf{U}\ \mathsf{F}b!i))$ generates the corresponding sequence for

j. Each sequence is illustrated by a partial sequence chart. The combination of those parts constitutes the global counterexample.

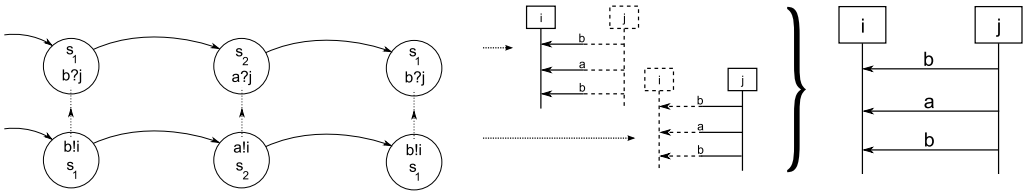

Fig. 3. Left: Sequence of states in the traces of j (bottom) and i (top). Right: Development steps of a counterexample (starting at i and being confirmed by j) illustrated by (partial) sequence charts.

4 Conclusions and Future Work

We presented an approach to generate a global counterexample if multi-object checking determines that a checking condition does not hold for a given multi-object system. Therefore, we identified classes of object systems as to the communication structure and depicted algorithms for generating a global counterexample. They provide linear complexity if a treelike communication structure is given or they provide exponential complexity if cyclic communication structures are involved.

Future work will feature a case study of the development of a UML-based Railway Interlocking System in which multi-object checking is applied to verify the safety of the system. In addition, we will concentrate on a user-friendly presentation of the generated counterexamples and we will develop the idea further in order to generate test cases and test data automatically.

Acknowledgements

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

References

[CCB+02] Roberto Cavada, Alessandro Cimatti, Marco Benedetti, Emanuele Olivetti, Marco Pistore, Marco Roveri, and Roberto Sebastiani. NuSMV: a new symbolic model checker. http://nusmv.itc.it/, 2002.

[CGP00] Edmund M. Clarke, Orna Grumberg, and Doran A. Peled. *Model Checking*. MIT Press, 2000.

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

[ECSD98] H.-D. Ehrich, C. Caleiro, A. Sernadas, and G. Denker. Logics for Specifying Concurrent Information Systems. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 167–198. Kluwer Academic Publishers, 1998.

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

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

[HR00] Michael R. A. Huth and Mark D. Ryan. *Logic in Computer Science - Modelling and reasoning about systems*. Cambridge University Press, 2000.

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