On Model Checking Mechanisms

Federico Mari, Igor Melatti, and Enrico Tronci

Dep. of Computer Science University of Rome "La Sapienza",
Via Salaria 113, 00198 Roma, Italy
{mari,melatti,tronci}@di.uniroma1.it

Abstract. In this position paper we present some preliminary considerations and experimental results about the design and implementation of a model checker for mechanisms, in particular for BAR systems.

1 Introduction

Model checking has been very successful in digital hardware verification (e.g. see [4]), protocol verification (e.g. see [13, 9]) and, more recently, in software verification (e.g. see [6, 2]).

Systems which description involves possibly complicated arithmetical computations have also been analyzed using model checking techniques. Examples are: hybrid systems (e.g. [11, 7, 14, 3]) as well as stochastic (hybrid) systems (e.g. [12, 8]).

Thus, investigating if model checking techniques can be used to verify *mechanism* designs appears to be a quite natural step. More specifically here we are interested in *Byzantine Altruistic Rational* (BAR) systems [1].

In this position paper we will informally discuss the obstructions to be overcome in the design and implementation of a model checker for BAR systems. Moreover we will present some very preliminary results on experiments we carried out with a prototype model checker we built for a class of BAR systems.

We assume from the reader some basic knowledge of game theory (e.g. [10]) and model checking (e.g. see [5]).

2 Model Checking Mechanisms

In this Section we will discuss some of the issues to be considered when designing a model checker for mechanisms.

2.1 Scenario

We are given a population of *nodes* or *agents*. Agents are classified as *rational*, *altruistic* or *byzantine* depending upon their possible behavior. More specifically:

- *Rational* agents behave in such a way as to maximize their gain.
- *Altruistic* agents obey to the given protocol.

– *Byzantine* agents behave arbitrarily.

Typically, it is assumed that each agent knows the *system history*, that is all past actions fired by each agent. On the base of such knowledge all agents *simultaneously* decide the next action.

2.2 As for the past

First of all, model checking technology rests on a notion of state. A *state* just represents the system past history. Thus representing history by itself is not a problem. However, since our systems are nonterminating ones, history will be infinite. This leads to consider systems with an infinite number of states. This is a problem since model checking typically works well for finite state systems. For this reason we only consider finite state systems. This means, among other things, that we restrict ourselves to histories of finite length.

2.3 As for the future

When an agent takes a decision it does so on the basis of the possible *future* consequences of the chosen action. A protocol is a nonterminating (infinite) game. Thus we should consider an infinite horizon, for example considering a discounting schema for gains. Along the same line we may consider that beyond a certain horizon gains become negligible (e.g. because of the discount rate) and thus carry out an *approximate* finite horizon analysis.

2.4 About observability of actions

The fact that an agent knows all past actions of all agents means that each agent knows the state of all other agents (as well as its own, of course). In other words, the system state is *observable* for each agent. This in general may not be true at least for two reasons. First, an agent may not be able to observe other agents actions. Second, our finite length histories may not be *long enough* to reconstruct the state of each agent.

2.5 Synchronous vs asynchronous parallelism

Essentially we can model agents (nodes) behavior in two ways: *synchronous* or *asynchronous*. The *synchronous* model for parallelism is that of synchronous digital hardware. Namely, *all* nodes move together. The *asynchronous* model for parallelism is that of software processes, e.g. UNIX processes. Namely, *exactly one* node move at each turn.

When a node moves it does not know what other nodes will do in the same round. This appears to be easier to model by using a synchronous model for parallelism. Thus we plan to use a synchronous approach in our modeling.

2.6 Communication

Since we plan to use a synchronous model for parallelism it appears to be quite natural to use shared variables in order to model communication between nodes. Of course, typically, this is not what happens in reality. However, buffers, communications channels, etc can be easily modeled by using shared variables.

2.7 The meaning of *rationality*

Each rational agent will select one (or more) actions on the basis of some definition of *rationality* . Well know ones are: *Nash equilibrium* and *Pareto optimality*.

Depending on the situation, one notion of rationality may be better suited than others. Thus it would be nice if a mechanism model checker could be, to same extent, parametric with respect to a (hopefully large) class of definitions of rationality.

3 Experimental Results

Here we show the experimental results we obtained with our approach on the TRB (Terminating Reliable Broadcast [15]) protocol. Our modeling of TRB relies on the following assumptions.

3.1 TRB nodes

Each node involved in TRB may be either *altruistic* (i.e. it follows TRB specification), *rational* (i.e. it strives to maximize its utility) or *byzantine* (i.e. it may deviate arbitrarily from TRB specification).

3.2 Model of parallelism

During execution, all nodes move *simultaneously*. Thus, a *transition* of the whole system consists of a n-tuple of actions, one for each node.

3.3 Communication modeling

Communication between TRB nodes is implemented via shared variables (mailboxes).

3.4 Payoffs

We assume that a *payoff* function is defined for each action $\boldsymbol{a} = \langle a_1, \ldots, a_n \rangle$ in each possible system state $\boldsymbol{s} = \langle s_1, \ldots, s_n \rangle$. The payoff function returns a tuple $\boldsymbol{g} = \langle g_1, \ldots, g_n \rangle$ of real numbers, where g_i is the payoff of node i if action $\boldsymbol{a}$ is fired in state $\boldsymbol{s}$. As usual, a finite sequence $\boldsymbol{a}^{(1)}, \ldots, \boldsymbol{a}^{(k)}$ of transitions, with payoffs $\boldsymbol{g}^{(1)}, \ldots, \boldsymbol{g}^{(k)}$, has payoff $\sum_{i=1}^{k} \boldsymbol{g}^{(i)}$ (where the sum, as usual, is componentwise).

3.5 Rationality

In order to define how *rational* nodes maximize their own utility, we proceed as follows. First, we fix a *rational horizon* k. Then, each rationale node will compute its set of *profitable* actions as follows. Let $\boldsymbol{s}$ be a system state and a be an allowed action for node i in state $\boldsymbol{s}$. Let P $[P_a]$ be the set of all sequences of k consecutive system transitions from state $\boldsymbol{s}$ [with first action a for node i]. To decide if a is profitable, node i takes into account the payoffs of the transition sequences in P_a and P. Namely, a will be discarded (i.e. considered not profitable) iff for each sequence α in P_a there exists a sequence β in P such that the payoff of α is *dominated* by the payoff of β. As usual, a payoff tuple $\boldsymbol{g} = \langle g_1, \ldots, g_n \rangle$ is dominated by a payoff tuple $\boldsymbol{g'} = \langle g'_1, \ldots, g'_n \rangle$ iff for all i, $g_i \leq g'_i$ and there exists j s.t. $g_j \neq g'_j$. In other words, each rational node will consider profitable only actions on the Pareto surface of (the payoffs of) P.

3.6 Properties

We intend to verify the following properties.

Agreement If a non-byzantine node delivers a message m, then all non-byzantine nodes eventually deliver m.

Termination Every non-byzantine process eventually delivers exactly one message.

Integrity If a non-byzantine node delivers m, then the sender sent m.

Non-Triviality In periods of synchrony, if the sender is non-byzantine and sends a message m, then the sender eventually delivers m.

3.7 Some preliminary experimental results

Tables 1, 2 show some preliminary results with horizon k set to 1. These results were obtained on a 2GHz Pentium 4 with 512 MB.

Column **TOT** shows n, i.e. the number of nodes. Column **A** (resp., **B**, **R**) shows the number of altruistic (resp., byzantine, rational) nodes. Column **Send** (resp., **Lead**) shows if the sender (resp., leader) is altruistic (A), byzantine (B) or rational (R). Column **States** gives the number of global states of the TRB system. Column **Time** gives the CPU time (in seconds) to complete the verification task. The remaining two columns deal with the properties to be verified. Namely, column **Expected** shows the expected result for the considered properties. More specifically, "OK" means that all four properties are expected to hold since **B** $\leq \left\lfloor \frac{\texttt{TOT}-2}{3} \right\rfloor$ (see [1]). On the other hand, "Any" means that some property *may* not hold, since **B** $> \left\lfloor \frac{\texttt{TOT}-2}{3} \right\rfloor$. The last column shows the verification result we

obtained with our approach for the considered properties. Namely, "OK" means that all the properties hold, "NO" means that at least one property does not hold.

Parameters								Properties	
TOT	**A**	**R**	**B**	**Send**	**Lead**	**States**	**Time**	**Expected**	**Obtained**
3	1	1	1	A	R	5156	1.96	Any	OK
3	1	1	1	R	B	6660	1.43	Any	NO
3	1	1	1	R	B	55126	5.83	Any	NO
3	1	1	1	B	A	1443	1.11	Any	NO
3	1	1	1	B	A	27248	3.66	Any	NO
5	2	2	1	A	A	16785	8.02	OK	OK
5	2	2	1	A	R	15588	7.36	OK	OK
5	2	2	1	R	R	14634	6.91	OK	OK
5	2	2	1	B	A	16785	8.07	OK	OK
2	0	1	1	R	B	730	1.09	Any	OK
2	0	1	1	B	R	5276	1.31	Any	OK
3	0	2	1	R	R	5156	1.60	Any	OK
3	0	2	1	R	B	21642	3.09	Any	NO
3	0	2	1	R	B	112876	16.04	Any	NO
3	0	2	1	B	R	3931	1.30	Any	NO
3	0	2	1	B	R	6504	1.50	Any	NO
4	0	3	1	R	R	11622	9.05	Any	OK
4	0	3	1	B	R	18273	8.68	Any	NO
4	0	3	1	B	R	18943	9.07	Any	NO
5	0	4	1	R	R	16785	93.92	OK	OK

Table 1. Experimental results for TRB where Byzantine behavior is not constrained

4 Conclusions

We have shown some preliminary considerations and experimental results on model checking mechanisms. We think our preliminary investigation shows the following.

– Mechanism model checking can be made viable for small systems and some suitable hypotheses on the *memory* of rational nodes as well on *observability* of node actions.

Parameters								Properties	
TOT	**A**	**R**	**B**	**Send**	**Lead**	**States**	**Time**	**Expected**	**Obtained**
3	1	1	1	A	R	213	0.10	Any	OK
3	1	1	1	R	B	50	0.10	Any	OK
3	1	1	1	B	A	192	0.10	Any	NO
3	1	1	1	B	A	573	0.24	Any	NO
5	2	1	2	A	A	1665	1.14	Any	OK
5	2	1	2	A	R	1665	1.09	Any	OK
5	2	1	2	R	B	2148	1.42	Any	OK
5	2	1	2	B	B	2578	1.71	Any	OK
5	1	2	2	A	R	1665	1.11	Any	OK
5	1	2	2	R	R	1665	1.12	Any	OK
5	1	2	2	R	B	2148	1.27	Any	OK
5	1	2	2	B	B	8975	2.05	Any	NO
5	2	2	1	A	A	47	0.10	OK	OK
5	2	2	1	A	R	47	0.10	OK	OK
5	2	2	1	R	R	47	0.10	OK	OK
5	2	2	1	R	B	47	0.10	OK	OK
5	2	2	1	B	A	15727	5.18	OK	OK
2	0	1	1	R	B	43	0.10	Any	OK
2	0	1	1	B	R	203	0.10	Any	OK
3	0	2	1	R	R	213	0.10	Any	OK
3	0	2	1	R	B	65	0.10	Any	NO
3	0	2	1	R	B	66	0.10	Any	NO
3	0	2	1	B	R	1066	0.40	Any	NO
3	0	2	1	B	R	1534	0.58	Any	NO
3	0	2	1	B	R	8823	0.98	Any	NO
4	0	3	1	R	R	47	0.10	Any	OK
4	0	3	1	R	B	47	0.10	Any	OK
4	0	3	1	R	B	5803	3.67	Any	NO
4	0	3	1	B	R	68180	46.25	Any	NO
4	0	2	2	R	R	15498	3.87	Any	OK
4	0	2	2	R	B	1798	1.23	Any	NO
4	0	2	2	R	B	2046	1.28	Any	NO
4	0	2	2	B	R	11848	2.35	Any	NO
4	0	2	2	B	R	20826	3.88	Any	NO
5	0	3	2	R	R	1665	2.14	Any	OK
5	0	3	2	R	B	2148	2.47	Any	OK

Table 2. Experimental results for TRB where Byzantine behavior is constrained

- The notion of *rationality* to be used during verification has to be an input to the model checker, unless somehow it is shown that there is only one meaningful notion of rationality for mechanisms.
- We expect that a model checker for mechanisms will mainly be useful to find errors (*bug hunting*) in a mechanism rather than to prove its correctness.

Acknowledgment

We are very grateful to Lorenzo Alvisi, Allen Clement and Harry Li for very helpful discussions about the topics of this paper. Indeed, we are currently working all together towards the realization of an *infinite horizon* mechanism model checker based on a discounting schema for payoffs.

References

1. Amitanand S. Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. Bar fault tolerance for cooperative services. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 45–58, New York, NY, USA, 2005. ACM Press.
2. Thomas Ball and Sriram K. Rajamani. The slam project: debugging system software via static analysis. In *POPL*, pages 1–3, 2002.
3. Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A tutorial on uppaal. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer, 2004.
4. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. Comput.*, 98(2):142–170, 1992.
5. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.
6. Edmund M. Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ansi-c programs. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, volume 2988 of *Lecture Notes in Computer Science*, pages 168–176. Springer, 2004.
7. G. Della Penna, B. Intrigila, I. Melatti, M. Minichino, E. Ciancamerla, A. Parisse, E. Tronci, and M. Venturini Zilli. Automatic verification of a turbogas control system with the murφ verifier. In Oded Maler and Amir Pnueli, editors, *Hybrid Systems: Computation and Control, 6th International Workshop, HSCC 2003 Prague, Czech Republic, April 3-5, 2003, Proceedings*, volume 2623 of *Lecture Notes in Computer Science*, pages 141–155. Springer, 2003.
8. G. Della Penna, B. Intrigila, I. Melatti, E. Tronci, and M. Venturini Zilli. Finite horizon analysis of markov chains with the murφ verifier. *STTT*, 8(4):397–410, 2006.

9. David L. Dill, Andreas J. Drexler, Alan J. Hu, and C. Han Yang. Protocol verification as a hardware design aid. In *Proceedings of the 1991 IEEE International Conference on Computer Design on VLSI in Computer & Processors*, pages 522–525. IEEE Computer Society, 1992.
10. D. Fudenberg and J. Tirole. *Game theory*. MIT Press, aug 1991.
11. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1):110–122, dec 1997.
12. Andrew Hinton, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Prism: A tool for automatic verification of probabilistic systems. In Holger Hermanns and Jens Palsberg, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25 - April 2, 2006, Proceedings*, volume 3920 of *Lecture Notes in Computer Science*, pages 441–444. Springer, 2006.
13. G. J. Holzmann. *The SPIN model checker: Primer and reference manual*. Addison Wesley, 2004.
14. F. Mari and E. Tronci. Cegar based bounded model checking of discrete time hybrid systems. In A. Bemporad, A. Bicchi, and G. Buttazzo, editors, *Hybrid Systems: Computation and Control, 10th International Conference, HSCC 2007, Pisa, Italy, April 3-5, 2007, Proceedings to appear*, Lecture Notes in Computer Science. Springer, 2007.
15. J.-P. Martin, A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, and C. Porth. Bar tolerance for cooperative services. Technical Report TR-05-10, The University of Texas at Austin, 2005.