Verification Methods for Weaker Shared Memory Consistency Models

Rajnish P. Ghughal[1,2] * and Ganesh C. Gopalakrishnan[2]**

[1] Formal Verification Engineer, Intel, Oregon. rajnish.ghughal@intel.com
[2] Department of Computer Science, University of Utah, Salt Lake City, UT 84112-9205. ganesh@cs.utah.edu

Abstract. The problem of verifying finite-state models of shared memory multiprocessor coherence protocols for conformance to weaker memory consistency models is examined. We start with W.W. Collier's architectural testing methods and extend it in several non-trivial ways in order to be able to handle weaker memory models. This, our first contribution, presents the construction of architectural testing programs similar to those constructed by Collier (e.g. the ARCHTEST suite) suited for weaker memory models. Our own primary emphasis has, however, been to adapt these methods to the realm of model-checking. In an earlier effort (joint work with Nalumasu and Mokkedem), we had demonstrated how to adapt Collier's architectural testing methods to model-checking. Our verification approach consisted of abstracting executions that violate memory orderings into a fixed collection of automata (called Test Automata) that depend only on the memory model. The main advantage of this approach, called Test Model-checking, is that the test automata remain fixed during the iterative design cycle when different coherence protocols that (presumably) implement a given memory model are being compared for performance. This facilitates 'push-button' re-verification when each new protocol is being considered. Our second contribution is to extend the methods of constructing test automata to be able to handle architectural tests for weaker memory models. After reviewing prior work, in this paper we mainly focus on architectural tests for weaker memory models and the new abstraction methods thereof to construct test automata for weaker memory models.
An extended version of this paper is available through `www.cs.utah.edu/formal_verification/` *under 'Publications'*

1 Introduction

Virtually all high-end CPUs are designed for multiprocessor operation in systems such as symmetric multiprocessor servers and distributed shared memory systems. As processors are getting faster faster than memories are, modern CPUs

* The author is currently at Intel, Oregon and was at University of Utah during the course of the research work presented here.
** Supported in part by NSF Grant No. CCR-9800928.

employ *shared memory consistency models* that permit more optimizations at the hardware and compiler levels. As weaker memory models (weaker relative to sequential consistency [7]) permit more hardware/compiler optimizations, virtually all modern processors employ a weak memory model such as *total store ordering* (TSO, [13]), *partial store ordering* (PSO, [13]), or the Alpha Shared Memory Model [11]. Most past work in verifying processors for conformance to memory models has, however, focussed on sequential consistency verification. The upshot of these facts is that there is very limited understanding in the formal verification community on verifying conformance to weaker memory models, and to do it in a way that fits in a modern design cycle in which design changes, and hence verification regressions, are very important.

Contribution 1: Architectural tests for Weaker Memory Models

Our first contribution is in formally characterizing several weaker memory models and presenting new architectural tests for them. In our approach, a formal memory model is viewed as a conjunction of elementary ordering "rules" (relations) such as *read ordering* and *write ordering*, as defined by Collier [1] in conjunction with architectural testing methods for multiprocessor machines developed by him. For example, sequential consistency can be viewed as a conjunction of *computational ordering* (CMP), *program ordering* (PO), and *write atomicity* (WA). This is written "SC=(CMP,PO,WA)" where the right-hand side of the equation is called a *compound rule*, with CMP, PO, WA, etc., then called *elementary rules*. Collier's work was largely geared towards strong memory models, as well as certain atypical weaker memory models. For these memory models, it turns out that it is sufficient to verify for *conjunctions* of 'classical' memory ordering rules such as PO, WA, etc. However, weaker memory models relax these classical ordering rules (often PO and WA) in subtle ways. For example, as we show later, TSO relaxes the write-to-read ordering (WR) aspect of PO. TSO also relaxes WA slightly. Therefore, in a memory system that is supposed to implement TSO, a violation of the classical PO rule does not mean that the memory system is erroneous. The memory system is erroneous with respect to PO only if it violates an aspect of PO *other than* WR orderings. Specifically, given that PO is made up of four sub-rules, namely RO (read ordering), WO (write ordering), WR (write-read ordering), and RW (read-write ordering), it means we must be prepared to look for violations of RO, WO, or RW. Generalizing this idea, to extend Collier's method to cover practical weaker memory models, *pure tests* that test for violations of a single elementary architectural rule or limited combinations of elementary rules would be good to have. In this paper, we outline an example pure test. This example presents a test that checks whether (CMP,RO) (the conjunction of CMP and RO) is violated. We have developed several other such pure tests for other rules to faciliate testing for different weak memory models - some of considerably more complexity than the example presented. We will not be presenting all the tests but provide a brief summary of our results at the end of this paper.

In this paper, we explain the technique by which we arrive at pure tests, and examine various aspects of this process, including many non-obvious special cases as well as a few limitations. As one example, we show that sometimes we need to limit the degree to which we leave out rules from a compound rule. For example, we show that the combination (CMP,WO) (WO is "write ordering") is irrelevant in practice; instead, the minimal pure rule worthy of study is (CMP, UPO, WO) where UPO denotes *uniprocessor ordering.* As another example, we show that WO is indistinguishable from WOS if CMP, and a relaxed write atomicity condition WA-S are provided.

The practical implications of these results is that they allow us to explore various tests for a combination of elemental ordering rules and reason about whether an elemental rule is obeyed in presence of other rules. This also enables us to examine a weaker memory model for all aspects of its behavior, come up with different tests to stress these aspects separately, and to correlate the test results. In our work, we have obtained such characterizations for PSO, the Alpha Shared Memory Model, and the IBM 370 memory model. We investigate various pure tests to faciliate verification of conformance to these weaker memory models. In a nutshell, our contribution allows the ARCHTEST methodology to apply to several practical weaker memory models.

Contribution 2: New Abstraction Methods for Architectural Tests

Our second contribution pertains to new abstraction methods in *test model-checking* as explained below. In our earlier work [4, 9], we reported our *test model-checking* approach to verify finite-state models of shared memory systems for conformance to sequential consistency. Test model-checking is basically a reachability analysis technique in which the model of the memory system being verified is closed with test automata playing the roles of the CPUs. The test automata administer a predetermined sequence of write operations involving only a limited range of addresses as well as data values. These writes are interspersed with reads over the same addresses. The test automata were constructed in such a way that when the reads return "unexpected" values, they move to error states, flagging ordering rule violations.

Test model-checking can be carried out in the framework of temporal logic (say, LTL) model-checking by converting each test automaton into a temporal logic formula and checking for the safety property $\Box(\neg inErrorState)$. In a practical setting, however, specialized reachability analysis algorithms may perform better. The fact that the test automata remain the same despite changes in the shared memory system implementation is a significant advantage, as the test model-checking algorithm can be automatically reapplied after each design iteration. In contrast, previous methods required the characterization of the reference specification, namely the desired formal memory model, in terms of very complex temporal logic specifications involving internal details of the memory system under design. This requires the error-prone step of rewriting the temporal logic specification following each design iteration. Many previous efforts also involved manual proofs which are not needed in our approach. For these reasons, test model-checking is eminently suited for use in actual design cycles.

Our earlier reported work on *test model-checking* [4, 9] serves as the background for the work reported here. Our contributions in these works were the following. We demonstrated that test automata can be derived through sound abstractions of architectural tests similar to ARCHTEST. The abstractions were based on *data independence* and *address semi-dependence*. These notions are defined with respect to *executions*, where executions are shared memory programs with reads annotated with the read data values. Under data independence, executions are closed under function applications to the involved data values; in other words, changing the data values does not affect the behavior of the memory system. Under address semi-dependence [5], no other operations may be performed on addresses other than comparison for equality. In our earlier work, we showed that test automata give the effect of running architectural tests for all possible addresses, data values, architectural test-program lengths, and interleavings.

The specific contribution we make with regard to test model-checking is in developing additional abstraction methods that help apply test model-checking for more general varieties of architectural tests. To give a few motivating details, the new *pure tests* we have developed for handling weaker memory models involve architectural tests that examine a finite unbounded history of read values. To handle these situations, we employ data abstraction in conjunction with properties of Boolean operators to derive a finite *summary* of these histories. Details of these abstraction methods and soundness proofs appear in [3].

Another related contribution we make is in handling memory barriers. Given that the test-automata administer a non-deterministic sequence of memory operations, a question that arises in connection with 'membar' instructions is how many membar instructions to consider. We show that under reasonable assumptions – specifically that the memory system does not decode the number of membar instructions it has seen – we need to consider only a limited number of membar instructions. Details appear in [3].

2 Summary of Results

We now summarize our key results in the form of tables and provide an overview (details are in [3]). In Table 1, we summarize the results of test model-checking an operational model of TSO implemented in Berkeley VIS Verilog [12]. This operational model is similar to that used in [2], and usually corresponds to the *reference* specification of TSO. The two 'fail' entries in the table correspond to program ordering, (CMP,PO), and write-to-read orderings, (CMP,WR). Since these orderings are not obeyed in TSO, we obtain 'fail' correctly. The other architectural tests in the tables indicate 'pass' which means that TSO obeys them. These pass/fail results provide added assurance (a 'sanity check') that our characterization of weaker memory models is consistent with the popular understanding of weaker memory models.

Table 2 shows various architecture rules and their *transition templates.* The idea of transition templates introduced in [1] specified a summary of the ordering rule. Many of the entries in this table were specified in [1]. We have defined new architectural rules ($MB - RR$ through $WA - S_{intra}$), defined tests and test automata for them, as well as provided more complete tests for many of the previously existing rules.

Table 3 shows the architecture rules in our discussion and the subrules each of them consists of. In particular, note $WA - S$, which is a relaxed write-atomicity rule that is one of the central sub-rules of TSO. Briefly, write events become visible to the processor issuing the write first, and *then* the events become atomically visible to all other processors. In contrast, in sequential consistency, each write becomes atomically visible (at the same time) to all the processors.

Table 4 shows the memory models in our discussion and their specification in the ARCHTEST framework. These results provide, to the best of our knowledge, the first formal characterization, in one consistent framework, of several practical weaker memory ordering rules. For example, by contrasting TSO and the Alpha Shared Memory Model, it becomes clear that the later is much weaker than the former in terms of read/write orderings, but provides more safety-net operations to recover these orderings.

The Alpha architecture manual [11] describes a number of executions called *Litmus tests* to illustrate which shared memory behavior is allowed and not allowed by the Alpha Shared Memory Model. In [3], we show that all these litmus tests are (often trivially) covered by our characterization of the Alpha Shared Memory architectural compound rule. In addition to sanity-checking our results, these results indicate that a developer of a modern memory system can use our architectural rules to debug the memory system focusing on each *facet* (sub-rule) at a time.

3 Conclusions and Future Work

We formally characterize the problem of verifying finite-state models of shared memory multiprocessor coherence protocols for conformance to weaker memory consistency models in terms of Collier's architectural testing methods. We extend Collier's framework in several non-trivial ways in order to be able to handle

Table 1. Verification results on an operational model of TSO using VIS

test automata	#states	#bdd nodes	runtime (mn:sec)	status
CMP, RO, WO	3819	4872	< 1s	pass
CMP, PO	6.50875e+06	50051	2:38	fail
CMP, WR	6.50875e+06	50051	1:25	fail
CMP,RW	6.50875e+06	50051	3:02	pass
CMP, RO	10187	2463	0:37	pass

Table 2. Architecture rules and their transition templates

Architecture rule	Transition template
SRW	$(P,L,R,V,O,S) <_{SRW} (=,=,W,-,-,-)$
CRW	$(P,L,R,V,O,S) <_{CRW} (-,-,W,-,=,=)$
CWR	$(P,L,W,V,O,S) <_{CWR} (-,-,R,=,=,=)$
CWW	$(P,L,W,V,O,S) <_{CWW} (-,-,W,-,=,=)$
URW	$(P,L,R,V,O,S) <_{URW} (=,-,W,-,=,=)$
UWR	$(P,L,W,V,O,S) <_{UWR} (=,-,R,-,=,=)$
UWW	$(P,L,W,V,O,S) <_{UWW} (=,-,W,-,=,=)$
RW	$(P,L,R,V,O,S) <_{RW} (=,-,W,-,-,-)$
WR	$(P,L,W,V,O,S) <_{WR} (=,-,R,-,-,-)$
RO	$(P,L,R,V,O,S) <_{RO} (=,-,R,-,-,=)$
WO	$(P,L,W,V,O,S) <_{WO} (=,-,W,-,-,-)$
$MB{-}RR$	$(P,L,R,V,O,S) <_{MB\text{-}RR} (=,-,MB-RR,-,-,-)$ $(P,L,MB-RR,-,-,-) <_{MB-RR} (=,-,R,-,-,-)$
$MB{-}RW$	$(P,L,R,V,O,S) <_{MB\text{-}RW} (=,-,MB-RW,-,-,-)$ $(P,L,MB-RW,-,-,-) <_{MB-RW} (=,-,W,-,-,-)$
$MB{-}WR$	$(P,L,W,V,O,S) <_{MB\text{-}WR} (=,-,MB-WR,-,-,-)$ $(P,L,MB-WR,-,-,-) <_{MB-WR} (=,-,R,-,-,-)$
$MB{-}WW$	$(P,L,W,V,O,S) <_{MB\text{-}WW} (=,-,MB-WW,-,-,-)$ $(P,L,MB-WW,-,-,-) <_{MB-WW} (=,-,W,-,-,-)$
$=_{WA\text{-}S}$	$(P,L,W,V,O,\neq P) =_{WA\text{-}S} (=,=,W,=,=,\neq P)$
$WA{-}S_{intra}$	$(P,L,W,V,O,=P) <_{WA\text{-}S_{intra}} (=,=,W,=,=,\neq P)$
CON	$(P,L,W,V,O,S) <_{CON} (-,-,W,-,=,=)$
WA	$(P,L,A,V,O,S) <_{WA} (-,-,-,-,-,-)$

Table 3. Architecture rules

Architecture rule	Subrules
CMP	SRW, CRW, CWW, CWR
PO	WR, WO, RO, RW
$WA-S$	$=_{WA\text{-}S}$, $WA{-}S_{intra}$, CON
MB	$MB{-}WR$, $MB{-}WO$, $MB{-}RO$, $MB{-}RW$

weaker memory models. This permits the construction of architectural testing programs similar to those constructed by Collier (e.g. the ARCHTEST suite). We then adapt these tests to the realm of model-checking, to permit early life-cycle formal verification of design descriptions. Our approach consists of abstracting executions that violate memory orderings into a fixed collection of automata (called Test Automata) that depend only on the memory model. The main advantage of our test model-checking approach is that the test automata remain fixed during the iterative design cycle when different coherence protocols that (presumably) implement a given memory model are being compared for performance. This facilitates 'push-button' re-verification when each new protocol is being considered. Here, we report our new results that extend the methods of constructing test automata to be able to handle architectural tests for weaker memory models. We achieve this as follows: (i) we define new abstraction techniques for summarizing execution histories; (ii) we prove the soundness of these abstractions; and (iii) we provide practical means to reduce the number of memory barrier instructions used in test automata.

We provide a formal characterization, in one consistent framework, of several practical weaker memory ordering rules, including the TSO and PSO models [13], the IBM 370 memory model, and the Alpha Shared Memory Model [11]. We show that 'Litmus tests' that practitioners employ for the Alpha memory model are covered by our formalism. We define a suite of architectural tests and corresponding test model-checking automata to facilitate verification of weaker memory models. We report on VIS based model-checking results that clearly show how developers of modern memory systems can use our architectural rules to debug the memory system focusing on each *facet* (sub-rule) at a time. This helps cut down verification complexity and helps pinpoint errors.

We believe that test automata can provide leverage in attacking the shared memory system verification problem, as they avoid many redundant test cases that would otherwise have been used in a simulation framework. We are working on overcoming a limitation of the present work, namely that we do not yet have complete test automata for the weaker memory models examined here.

Table 4. Memory models specification in the ARCHTEST framework

Memory Model	ARCHTEST specification
Sequential consistency	$A(CMP, PO, WA)$
IBM 370	$A(CMP, UPO, RO, WO,$ $RW, WA, MB{-}WR)$
Total Sorted Orer (TSO)	$A(CMP, UPO, RO, WO,$ $RW, WA{-}S, MB{-}WR)$
Partial Sorted Order (PSO)	$A(CMP, UPO, RO,$ $RW, WA{-}S, MB{-}WR, MB{-}WW)$
Alpha Shared Memory Model	$A(CMP, UPO, ROO,$ $WA{-}S, MB, MB{-}WW)$

Promising completeness results obtained in another work [8] may provide the necessary directions to pursue.

We are also investigating ways to mitigate state explosion. Since test model-checking can be cast as *finite-state reachability analysis*, efficient techniques under development by many groups for exact- as well as approximate reachability analysis are believed to be one promising direction to pursue. Once we get a handle on state explosion through reachability analysis based test model-checking, we plan to work on overcoming many other sources of inefficiency, including symmetries that, as yet, stand unexploited. Works on symmetry exploitation (e.g., [6]) and the use of symbolic state descriptors (e.g., [10]) will be examined for possible answers. The integration of these ideas back into the realm of reachability analysis is expected to be a long-term direction.

References

[1] Collier, W. W. *Reasoning About Parallel Architectures.* Prentice-Hall, Englewood Cliffs, NJ, 1992.

[2] Dill, D. L., Park, S., and Nowatzyk, A. Formal specification of abstract memory models. In *Research on Integrated Systems* (1993), G. Borriello and C. Ebeling, Eds., MIT Press, pp. 38–52.

[3] Ghughal, R. Test model-checking approach to verification of formal memory models, 1999. Also available from `http://www.cs.utah.edu/formal_verification`.

[4] Ghughal, R., Nalumasu, R., Mokkedem, A., and Gopalakrishnan, G. Using "test model-checking" to verify the Runway-PA8000 memory model. In *Tenth Annual ACM Symposium on Parallel Algorithms and Architectures* (Puerto Vallarta, Mexico, June 1998).

[5] Hojati, R., and Brayton, R. Automatic datapath abstraction of hardware systems. In *Conference on Computer-Aided Verification* (1995).

[6] Ip, C. N., and Dill, D. L. Better verification through symmetry. In *Int'l Conference on Computer Hardware Description Language* (1993).

[7] Lamport, L. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers 9*, 29 (1979), 690–691.

[8] Nalumasu, R. *Formal design and verification methods for shared memory systems.* PhD thesis, University of Utah, Salt Lake City, UT, USA, Dec. 1998.

[9] Nalumasu, R., Ghughal, R., Mokkedem, A., and Gopalakrishnan, G. The 'test model-checking' approach to the verification of formal memory models of multiprocessors. In *Computer Aided Verification* (Vancouver, BC, Canada, June 1998), A. J. Hu and M. Y. Vardi, Eds., vol. 1427 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 464–476.

[10] Pong, F., and Dubois, M. New approach for the verification of cache coherence protocols. *IEEE Transactions on Parallel and Distributed Systems 6*, 8 (Aug. 1995), 773–787.

[11] Sites, R. L. *Alpha Architecture Reference Manual.* Digital Press and Prentice-Hall, 1992.

[12] Vis-1.2 release. http://www-cad.eecs.berkeley.edu/Respep/Research/vis/.

[13] Weaver, D. L., and Germond, T. *The SPARC Architecture Manual – Version 9.* P T R Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1994.