

# Position Paper: Formal Verification of Transactional Memories

Ariel Cohen<sup>1</sup>, Amir Pnueli<sup>1</sup>, and Lenore D. Zuck<sup>2</sup>

<sup>1</sup> New York University, {arielc, amir}@cs.nyu.edu

<sup>2</sup> University of Illinois at Chicago, lenore@cs.uic.edu

## 1 Introduction

Transactional Memory [HM93] is a simple solution for coordinating and synchronizing concurrent threads that access the same memory locations. It transfers the burden of concurrency management from the programmers to the system designers and enables a safe composition of scalable applications. Multicore, which requires concurrent programs in order to gain a full advantage of the multiple number of processors, has become the mainstream architecture for microprocessor chips and thus many new transactional memory implementations have been proposed recently (see [LR07] for an excellent survey).

A transactional memory (TM) receives requests from clients and issues responses. The requests are usually part of a *transaction* that is a sequence of operations starting with a request to *open* a transaction, followed by a sequence of read/write requests, followed by a request to *commit* (or *abort*). When a transaction requests a successful “commit,” all of its effects are stored in the memory. If a transaction is aborted (by either issuing an abort request or when TM detects that it should be aborted) all of its effects are removed. Thus, a transaction is a sequence of *atomic* operations, either all complete successfully and all its write operations update the memory, or none completes and its write operations do not alter the memory. In addition, committed transaction should be serializable – the sequence of operations belonging to successful transactions should be such that it can be reordered (preserving the order of operations in each transaction) so that the operation of each transaction appear consecutive, and a “read” from any memory location returns the value of the last “write” to that memory location.

Transactional memories implementations differ in numerous aspects, including their support for nested transactions, non-transactional accesses, conflict detection, conflict arbitration, and the granularity of the memory.

We believe that, in order to allow for safe development of software on top a transactional memory, one should be given the specification of the transactional memory, and consider it as a “black box” while developing software on top of it. That is, we advocate a compositional approach for the development of such software. Casting it in the assume-guarantee paradigm, suppose one wishes to prove that a property  $\Phi$  is satisfied by a software module  $M$  running on top of a transactional memory  $T$ . Our suggestions it to is to obtain the transactional memory specification, say  $T\text{-spec}$ , and to show that:

1.  $T \models T\text{-spec}$ ; our work focuses on obtaining mechanical proofs of this part.
2.  $T\text{-Spec} \parallel M \models \Phi$

For the past couple of years we have worked on developing methodologies and tools towards obtaining simple specifications for transactional memories ( $T\text{-spec}$  above) mechanical proofs of (1). Our accomplishments so far are:

- A parameterized state-machine model for transactional memory, whose parameters describe the particular features of the memory and that, for each particular set of parameters, generates *all* “legal” transactional sequences for such parameters. We refer to this model as the “specification.”
- A methodology, supported by tools, to formally verify that a particular transactional memory implementations (e.g., UTM, TCC, etc.) satisfy their specifications. That is, that each transactional sequence that can be obtained by the implementation is allowed by the specification.

So far, we focused on conflict arbitration and detection mechanisms, as well as on non-transactional accesses. Attaining actual formal proofs of correctness (of implementation against specifications) requires a significant effort and a strong theorem prover, our experience so far makes us believe that our approach is a promising, and will assist in developing correct software over transactional memories.

In [COP<sup>+</sup>07] we introduced the general framework to deal with transactional memories, where the focus was the conflict detection and arbitration mechanisms. The specifications differ from one another by “admissible interchanges” that describe when two consecutive events in a trace of transactional accesses can be interchanged. Each of [Sco06]’s detection/arbitration policy can be described by some such admissible interchanges. We also presented a proof rule to verify that transactional memory implementations satisfy their specifications as described in our framework. Small instantiations of several known protocols were model checked (using TLA) and shown to satisfy their specifications. In [CPZ08b] and [CPZ08a] we extended the model to handle non-transactional accesses, that are explicitly defined as such, which required some modifications of the specifications given in [COP<sup>+</sup>07]. We extended the theorem prover TLPVS of [PA03] to obtain a mechanical proofs of correctness. This was successfully applied to a variant of TCC of [HWC<sup>+</sup>04] augmented with non-transactional access.

We strongly believe that our models can be extended to handle other aspects of transactional memories, that is, in addition to the conflict detection and arbitration and the non-transactional access.

## References

- [COP<sup>+</sup>07] A. Cohen, J. W. O’Leary, A. Pnueli, M. R. Tuttle, and L. D. Zuck. Verifying correctness of transactional memories. In *Proceedings of the 7th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 37–44, November 2007.
- [CPZ08a] A. Cohen, A. Pnueli, and L. D. Zuck. Mechanical verification of transactional memories with non-transactional memory accesses, July 2008. To appear in CAV 2008.
- [CPZ08b] A. Cohen, A. Pnueli, and L. D. Zuck. Verification of transactional memories that support non-transactional memory accesses, February 2008. TRANSACT 2008.
- [HM93] M. Herlihy and J. E. B. Moss. Transactional memory: architectural support for lock-free data structures. In *ISCA ’93: Proceedings of the 20th annual international symposium on Computer architecture*, pages 289–300, New York, NY, USA, 1993. ACM Press.
- [HWC<sup>+</sup>04] L. Hammond, W. Wong, M. Chen, B. D. Carlstrom, J. D. Davis, B. Herzberg, M. K. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun. Transactional memory coherence and consistency. In *Proc. 31<sup>st</sup> annu. Int. Symp. on Computer Architecture*, page 102. IEEE Computer Society, June 2004.
- [LR07] J. R. Larus and R. Rajwar. *Transactional Memory*. Morgan & Claypool Publishers, 2007.
- [PA03] A. Pnueli and T. Arons. TLPVS: A PVS-based LTL verification system. In *Verification—Theory and Practice: Proceedings of an International Symposium in*

*Honor of Zohar Manna's 64th Birthday*, Lect. Notes in Comp. Sci., pages 84–98. Springer-Verlag, 2003.

- [Sco06] M.L. Scott. Sequential specification of transactional memory semantics. In *Proc. TRANSACT the First ACM SIGPLAN Workshop on Languages, Compiler, and Hardware Support for Transactional Computing*, Ottawa, 2006.