Towards a Unified Verification Theory for Various Memory Consistency Models

Tatsuya Abe Toshiyuki Maeda

RIKEN AICS, 7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo, 650-0047, Japan
{abet,tosh}@riken.jp

1 Introduction

Memory consistency models (e.g., Total Store Ordering (TSO), Partial Store Ordering (PSO), and Relaxed Memory Ordering [7]) specify behaviors of shared memories that are simultaneously accessed by multiple threads. For example, consider the program `(x = 1; y = 1)` || `(r0 = y; r1 = x)` that consists of two threads where || denotes a parallel composition, `r0` and `r1` are thread-local variables, `x` and `y` are shared variables, and all variables are initialized to `0`. When the program finishes, TSO ensures `r0 <= r1`. On the other hand, PSO allows `r1 < r0` since the write operation to `x` may be reordered with the write operation to `y` under PSO. Thus, program verification cannot ignore memory consistency models.

In recent years, several verification theories have been proposed [13, 10, 8, 5, 6, 9, 12, 11] to verify programs under various memory consistency models. However, most of them are specific to fixed memory consistency models except for a few works [13, 11]. This means that they are not suitable for practical relaxed memory consistency models that may vary from language to language, and CPU to CPU.

Our goal is to construct a unified verification theory for various memory consistency models. In the verification theory, a memory consistency model is formalized as an *input* of verification as with a program. That is, we do not have to modify the verification theory even when we handle a new memory consistency model. This can be useful to design and implement programming languages and CPU architectures.

In the paper, we describe our former, on-going, and future works.

2 A General Model Checking Framework

Model checking is a promising method for program verification based on exhaustive searches of execution traces of programs. Given a program and a property that we check, it detects *unsafety* of the program, which is ensured by existence of *counterexamples* for the property.

Ebnenasir developed a model checker for UPC memory model [8]. We also developed model checkers for CAF and XMP memory models [5, 6]. However, these model checkers are specific to their memory consistency models.

To make a model checker independent of fixed memory consistency models, we designed a low-level language that contains instructions for memory operations, and developed a general model checking framework for various memory consistency models [3]. In the framework, we can formalize a memory consistency model as a set of formulas consisting of relations between instructions, and we succeeded in giving sets of formulas that represent Itanium, CAF, and UPC memory models, respectively. Differently from the unified model checking approach of Yang et al. [13], we explicitly handle low-level jump instructions for loops. This enables reordering of instructions and their effects across loop iterations.

We also implemented a model checker generator, called McSPIN [1], following the framework. McSPIN takes a program, a property, and a (formalized) memory consistency model as inputs, and generates a model checker to verify whether the program has the property under the memory consistency model. To avoid the state explosion problem, McSPIN can perform some optimizations in exploring execution traces [4]. One interesting optimization is to prune execution traces by utilizing relations that occur in a given memory consistency model formalized as a set of formulas consisting of relations.

3 Concurrent Program Graph Logic

Theorem proving ensures safety of programs by existence of *proofs*. *Concurrent program logic* is a logic for concurrent programs. Ridge and Vafeiadis et al. gave concurrent program logics [10, 12], however, they are specific to x86-TSO and (restricted) C11 memory models, respectively.

We proposed new representations of programs under memory consistency models, called *program graphs* [2]. Program

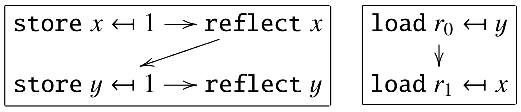
store $x \hookleftarrow 1$ → reflect x
store $y \hookleftarrow 1$ → reflect y
load $r_0 \hookleftarrow y$
load $r_1 \hookleftarrow x$

Figure 1: A program graph under TSO

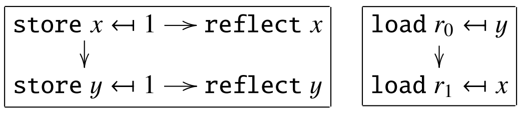
store $x \hookleftarrow 1$ → reflect x
store $y \hookleftarrow 1$ → reflect y
load $r_0 \hookleftarrow y$
load $r_1 \hookleftarrow x$

Figure 2: A program graph under PSO

graphs are directed acyclic graphs consisting of instructions and effects (as nodes) and dependences between instructions (as edges). The edges are transitive. Operational semantics of a program graph is defined as a state transition system with graph rewriting that removes its root nodes. We define a memory consistency model as a translation from programs to program graphs. In this paper, we intuitively explain TSO and PSO translations by examples. TSO *prohibits* a write operation to overtake an effect of another write operation that has been issued before. This is represented by the edge between `reflect` x and `store` $y \hookleftarrow 1$ in the program graph of Fig. 1. On the other hand, PSO allows a write operation to overtake an effect of another write operation that has been issued before. This is represented by no edge between `reflect` x and `store` $y \hookleftarrow 1$ in the program graph of Fig. 2. Thus, program graphs naturally represent dependences between instructions/effects specified by memory consistency models.

We gave a concurrent program logic for program graphs, called *concurrent program graph logic*, which supports memory consistency models that can be defined as a translation from programs to program graphs. To the best of our knowledge, the concurrent program graph logic is the first relatively complete logic for relaxed memory consistency models.

4 On-going and Future Works

Compared to general model checking framework in Sec. 2, the expressiveness of the concurrent program logic in Sec. 3 is limited. To relax the limitation, we are extending the definition of program graphs to handle memory consistency models that explicitly refer to memory hierarchy and/or do not assume *global time*.

In both of our approaches, programs are written in simple imperative low-level languages that do not support pointers, arrays, or functions. We are enhancing the general model checking framework and the concurrent program logic to support them.

There are two directions for future work.

One problem is that the general model checking framework of Sec. 2 and the concurrent program logic of Sec. 3 handle only safety properties. One direction is to improve our approach to handle *liveness* properties. Especially, construction of concurrent program logic for liveness under relaxed memory consistency models is challenging.

Another problem is that McSPIN [1] is a *bounded model checker* by restricting the numbers of loop iterations. Another direction is to relax the restriction.

References

[1] McSPIN. `https://bitbucket.org/abet/mcspin/`.

[2] T. Abe and T. Maeda. Concurrent program logic for relaxed memory consistency models with dependences across loop iterations. Submitted.

[3] T. Abe and T. Maeda. A general model checking framework for various memory consistency models. In *Proc. of HIPS2014*, pages 332–341, 2014.

[4] T. Abe and T. Maeda. Optimization of a general model checking framework for various memory consistency models. In *Proc. of PGAS2014*, 2014.

[5] T. Abe, T. Maeda, and M. Sato. Model Checking with User-Definable Abstraction for Partitioned Global Address Space Languages. In *Proc. of PGAS2012*, 2012.

[6] T. Abe, T. Maeda, and M. Sato. Model Checking Stencil Computations Written in a Partitioned Global Address Space Language. In *Proc. of HIPS2013*, pages 365–374, 2013.

[7] S. Adve and K. Gharachorloo. Shared memory consistency models: a tutorial. *Computer*, 29(12):66–76, 1996.

[8] A. Ebnenasir. UPC-SPIN: A Framework for the Model Checking of UPC Programs. In *Proc. of PGAS2011*. ACM, 2011.

[9] M. Gligoric, P. C. Mehlitz, and D. Marinov. X10X: Model Checking a New Programming Language with an "Old" Model Checker. In *Proc. of ICST2012*, pages 11–20, 2012.

[10] T. Ridge. A rely-guarantee proof system for x86-TSO. In *Proc. of VSTTE2010*, pages 55–70, 2010.

[11] A. Turon, V. Vafeiadis, and D. Dreyer. GPS: Navigating weak memory with ghosts, protocols, and separation. In *Proc. of OOPSLA2014*, pages 691–707. ACM, 2014.

[12] V. Vafeiadis and C. Narayan. Relaxed separation logic: A program logic for C11 concurrency. In *Proc. of OOPSLA2013*, pages 867–884. ACM, 2013.

[13] Y. Yang, G. Gopalakrishnan, and G. Lindstrom. UMM: an operational memory model specification framework with integrated model checking capability. *Concurr. Comput.: Pract. Exper.*, 17(5-6):465–487, 2005.