Verification of a lazy cache coherence protocol against a weak memory model Proof of TSO-LB satisfies TSO

Christopher J. Banks[1], Marco Elver[1], Ruth Hoffmann[2], Susmit Sarkar[2], Paul Jackson[1], Vijay Nagarajan[1]
[1]University of Edinburgh, [2]University of St Andrews

I. TSO-LB SATISFIES TSO

We prove that the operational TSO-LB model exhibits only TSO behaviour. Since TSO-LB is defined as a LTS, behaviour of TSO-LB is defined with respect to an arbitrary trace of this LTS. We show (Theorem 1), by means of an interpretation of logical and physical time over these traces, that the behaviour satisfies the `herd` axiomatic characterisation of TSO [1]. We also show via a counterexample (Appendix II-D) that TSO-LB does not permit all allowable behaviours of TSO, i.e. TSO-LB is in fact stricter than TSO.

Theorem 1 (TSO-LB satisfies TSO): The read and write events of traces of the TSO-LB LTS satisfy the TSO axiomatic memory consistency model (as formalised in Alglave et al. [1], and recalled in Appendix II).

Our proof strategy starts with defining a trace P of TSO-LB (Definition 1). The trace order might be seen as the physical-time representation of events, which contains writes, reads, and propagates. We will then construct a *strict linear order* L from P which contains the same writes and reads (with the same values). We will then show how to instantiate the required ordering relations from the `herd` framework of Alglave et al. [1] (see Table I in SectionII) from L, and show all those orders are contained in L. This will then allow us to show that the `herd` axiomatic constraints of TSO hold over the write and read events. Note that we assume a simplified TSO model excluding fences, as TSO-LB does not model fences by definition.

Definition 1 (Trace): A *trace* of an LTS is a sequence (finite or infinite) of labels that results from a path of transitions starting at the initial state. Let us call this trace order P.

Definition 2 (Logical-time L): We define L to be an order on the read and write events in the trace P. All writes $\mathsf{Write}(p, a, v)$ appear in L in the same order as in the physical-time trace P. A read $\mathsf{Read}(p, a, v)$ is pulled backwards in the trace to just after the event in P which made the processor p get the value v for a. Such an event is either a write from the same processor p, or a propagate to the processor p (if $\mathsf{Read}(p, a, v)$ reads from a write on another processor).

Note that several reads from the same processor can be pulled back to the same point in this scheme, if the same address is read by multiple reads, or if the propagated values for different addresses for the same propagate event are read from by different reads. In such a case, we order these multiple reads in L (which have to be from the same processor) according to program order.

Definition 3 (co in TSO-LB): The order co is defined in TSO-LB as $\mathsf{Write}(p, a, v) \overset{\mathsf{co}}{\to} \mathsf{Write}(q, b, w)$ if and only if $\mathsf{Write}(p, a, v)$ occurs before $\mathsf{Write}(q, b, w)$ in the physical-time trace P, and the addresses a and b are the same.
Note that p and q may be the same or different processors, and v and w the same or different values.

Definition 4 (rf in TSO-LB): The order rf is defined as $\mathsf{Write}(p, a, v) \overset{\mathsf{rf}}{\to} \mathsf{Read}(q, a, v)$ where p and q may be the same or different processors, and the read gets its value from the write.

We can now show that co , rf , and all the derived relations of the `herd` TSO formalisation are sub-orders of L. Then all axioms state the acyclicity and irreflexivities of various order relations, which are satisfied by any sub-orders of a strict linear order L.

II. SUMMARY OF AXIOMATIC FRAMEWORK

This section summarises the framework proposed by Alglave et al. [1] to specify the axiomatic semantics of MCMs. We have chosen to use this framework, as it is succinct while eliminating ambiguity (especially for the purpose of implementing decision procedures) and is flexible enough to describe a variety of consistency models while relying on a library of common reusable definitions (hence a framework).

Definition 5 (Events): An *event* captures the thread ($\mathsf{proc}(e)$), address ($\mathsf{addr}(e)$) and action. Events are unique, referred to with a lower case letter (e.g. e). For all events in a program, WR, WW, RR, $\mathsf{RW}$are the relations capturing all write-read, write-write, read-read and read-write pairs respectively.

Definition 6 (Candidate executions): A *candidate execution* is a tuple $(\mathbb{E}, \mathsf{po}, \mathsf{rf}, \mathsf{co})$, with the set of events $\mathbb{E}$ in the program, and the relations program-order po (via control-flow semantics), reads-from rf and coherence (or write-serialisation) order co (via data-flow semantics); see Table I for details.

A. Architecture Definition

An *architecture* defines the architecture specific details of a particular memory consistency model, and is used by the constraint specification to decide if a particular execution is valid or not. An architecture is defined by the tuple of relations

TABLE I
DEFINITION OF RELATIONS USED TO SPECIFY MEMORY CONSISTENCY MODELS IN THE AXIOMATIC FRAMEWORK.

Relation	*Name*	*Source*	*Subset of*	*Definition*
po	program-order	execution	$\mathbb{E} \times \mathbb{E}$	instruction order lifted to events
rf	read-from	execution	WR	links a write w to a read r taking its value from w
co	coherence	execution	WW	total order over writes to the same memory location
ppo	preserved program order	architecture	po	program order maintained by the architecture
fences	fences	architecture	po	events ordered by fences
prop	propagation	architecture	WW	order in which writes propagate
po-loc	program order subset of same address events	derived	po	po-loc $\triangleq$ $\{(x, y) \mid x \overset{\mathsf{po}}{\rightarrow} y \wedge \mathsf{addr}(x) = \mathsf{addr}(y)\}$
com	communications or conflict orders	derived	$\mathbb{E} \times \mathbb{E}$	com $\triangleq$ co $\cup$ rf $\cup$ fr
rfe	read-from external	derived	rf	rfe $\triangleq$ $\{(w, r) \mid w \overset{\mathsf{rf}}{\rightarrow} r \wedge \mathsf{proc}(w) \neq \mathsf{proc}(r)\}$
fr	from-read	derived	RW	links reads to writes based on observed rf and co : fr $\triangleq$ $(\mathsf{rf}^{-1}; \mathsf{co})$ $\triangleq$ $\{(r, w) \mid \exists w'.w' \overset{\mathsf{rf}}{\rightarrow} r \wedge w' \overset{\mathsf{co}}{\rightarrow} w\}$
fre	from-read external	derived	fr	fre $\triangleq$ $\{(r, w) \mid r \overset{\mathsf{fr}}{\rightarrow} w \wedge \mathsf{proc}(r) \neq \mathsf{proc}(w)\}$
hb	happens before	derived	$\mathbb{E} \times \mathbb{E}$	hb $\triangleq$ ppo$\cup$fences$\cup$rfe

(ppo, fences, prop), preserved program-order ppo , fences and propagation prop ; see Table I for details.

Definition 7 (Total Store Order): Only the write to read ordering is relaxed. Reads to the same address as a preceding writes w by the same thread must observe either w or a write by another thread that happened after w in co ; this, however, has no effect on the required visibility by other threads, which implies that prop only includes rfe (and not rf). The relation mfence captures write-read pairs separated by a fence instruction.

$$\begin{aligned}
\mathsf{ppo} &\triangleq \mathsf{po} \setminus \mathsf{WR} \\
\mathsf{fences} &\triangleq \mathsf{mfence} \\
\mathsf{prop} &\triangleq \mathsf{ppo} \cup \mathsf{fences} \cup \mathsf{rfe} \cup \mathsf{fr}
\end{aligned}$$

B. Constraint Specifications

Given a candidate execution and an architecture specification, the constraints (or axioms) decide if the execution is valid under the complete model.

*Definition 8 (*SC PER LOCATION): SC PER LOCATION ensures that communications/conflict orders com cannot contradict program-order per memory location po-loc . Formally SC PER LOCATION is satisfied if

$$\mathtt{acyclic}(\mathsf{po\text{-}loc} \cup \mathsf{com}).$$

*Definition 9 (*NO THIN AIR): This constraint ensures that the happens-before order hb , which captures preserved program-order ppo , fenced instructions fences , but also reads from other threads rfe is not contradictory. Formally NO THIN AIR is satisfied if

$$\mathtt{acyclic}(\mathsf{hb}).$$

Effectively, this prevents reads from observing values "out of thin air", i.e. before they appear to have been written by some other thread.

*Definition 10 (*OBSERVATION): OBSERVATION constrains the values a read may observe. Assume a write w_a to address a, a write w_b to address b, which are ordered in prop ($w_a \overset{\mathsf{prop}}{\rightarrow} w_b$), and a read r_b reading from w_b ($w_b \overset{\mathsf{rf}}{\rightarrow} r_b$), then any read r_a that happens after r_b ($r_b \overset{\mathsf{hb}}{\rightarrow} r_a$) cannot read from a write before w_a. OBSERVATION is satisfied if

$$\mathtt{irreflexive}(\mathsf{fre}; \mathsf{prop}; \mathsf{hb}^*).$$

*Definition 11 (*PROPAGATION): This constraint imposes restrictions on the order in which writes are propagated to other threads, i.e. ensuring that the propagation order prop does not contradict coherence (write-serialisation) order co . This constraint and depending on the prop order defines if the consistency model is *write/multi-copy atomic*, i.e. if all threads observe writes in the same order or not. PROPAGATION is satisfied if

$$\mathtt{acyclic}(\mathsf{co} \cup \mathsf{prop})$$

.

C. Proof that TSO-LB satisfies TSO

Proposition 1: The co order defined in Definition 3 is a sub-order of the logical time order L from Definition 2.

Proof: Since L has writes in the same order as P, if $\mathsf{Write}(p, a, v)$ is before $\mathsf{Write}(q, b, w)$ in P, it is before in L as well. ■

Proposition 2: The rf order defined in Definition 4 is a sub-order of the logical time order L from Definition 2.

Proof: There are two cases, where the read and the write are from the same or different processors.

If the read and the write are from the same processor, the read has been pulled to be just after the write in L, so the write is before the read.

If the read and the write are from different processor, in P there must have been a propagate to the processor of the read before the read. This propagate must necessarily be after the write in P (to get its value). In constructing L, we have pulled the read to just after that propagate, so the write is still before the read. ■

Proposition 3: By defining fr $\triangleq$ $(\mathsf{rf}^{-1}; \mathsf{co})$ as usual, fr is a sub-order of the logical time order L from Definition 2.

Proof: Consider a $\mathsf{Read}(p, a, v) \overset{\mathsf{fr}}{\rightarrow} \mathsf{Write}(q, a, w)$. There are two cases, where the read and the write are from the same or different processors.

If the read and the write are from the same processor, the read must come before the write in program order, and therefore before in P. Since reads are only pulled back in constructing L, the read is still before the write in L.

If the read and the write are from different processors, the write *may* be before the read in P. However, in constructing L, the read is pulled backwards to just after the last propagate on the reading thread. This propagate must be before the write (or else the propagate would have got the value of the write). Therefore in L the read is before the write, as required. ■

Proposition 4: The order ppo , defined as $\mathsf{ppo} \triangleq \mathsf{po} \setminus \mathsf{WR}$, is a sub-order of the logical time order L from Definition 2.

Proof: Note that all such events occur in the order ppo in P. Since writes are not moved, the order between write-write pairs is preserved in L. Since reads are moved, but are moved in program order, order between read-read pairs is preserved in L. Since reads are only moved backwards, order between read-before-write pairs is preserved in L. ■

Proposition 5: The order $\mathsf{po\text{-}loc}$, defined as relating events from the same processor and touching the same address in program order, is a sub-order of the logical time order L from Definition 2.

Proof: Note that all such events occur in the order $\mathsf{po\text{-}loc}$ in P. Since the pairs in $\mathsf{po\text{-}loc}$ are a subset of those in ppo (except for write-read pairs), the same proof cases as for Proposition 4 apply. For the new case of write-read pairs to the same address, note that reads are never pulled backwards over a write to the same address in constructing L, so the order from P is preserved in L. ■

We can now prove Theorem 1.

Proof: We require two facts about order relations: if orders R_1 and R_2 are sub-orders of L, then so is their union $R_1 \cup R_2$; and if an order R is a sub-order of L, then so is any sub-order of R. Using the second fact, fre and rfe are sub-orders of fr and rf respectively, and so (using Propositions 3 and 4) are sub-orders of L. Using the first fact, $\mathsf{prop} \triangleq \mathsf{ppo} \cup \mathsf{rfe} \cup \mathsf{fr}$, $\mathsf{com} \triangleq \mathsf{co} \cup \mathsf{rf} \cup \mathsf{fr}$, and $\mathsf{hb} \triangleq \mathsf{ppo} \cup \mathsf{rfe}$ are all sub-orders of L. This then gives us the axioms from Alglave et al. [1]; SC PER LOCATION, NO THIN AIR, OBSERVATION and PROPAGATION (Definitions 8, 9, 10, 11 respectively), as the acyclicity and irreflexivities for the axioms are satisfied by any sub-orders of a strict linear order L. ■

Using the framework of Alglave et al. [2] (Appendix II) we have shown that all valid traces of TSO-LB are permitted by the TSO axiomatic model.

D. TSO-LB is stronger than TSO

We note, however, that TSO-LB is in fact subtly stricter than TSO, which we demonstrate with the litmus test in Figure 1. The result is allowed by TSO; specifically, considering the store-buffering model of TSO, it is possible for both loads of a and b to read 0, since it is possible for the corresponding writes of a and b to still be held in the the local store buffers of $P1$ and $P2$ respectively, but writes of x and y have already propagated to the global memory. The behaviour cannot be reproduced in TSO-LB, as writes do not enter a queue and are propagated from global to local memory in "batches." Indeed, the absence of queues in TSO-LB is crucial to enable finite-state model checking in our approach.

Init: $x = y = a = b = 0$	
$P1$	$P2$
$x \leftarrow 1$	$y \leftarrow 1$
$a \leftarrow 1$	$b \leftarrow 1$
$r_1 \leftarrow y$	$r_4 \leftarrow x$
$r_2 \leftarrow y$	$r_5 \leftarrow x$
$r_3 \leftarrow b$	$r_6 \leftarrow a$
Observable?: $r_1 = 0 \land r_2 = 1 \land r_3 = 0$ $\land\ r_4 = 0 \land r_5 = 1 \land r_6 = 0$	

Fig. 1. Litmus test result allowed by TSO, but not observable with TSO-LB.

References

[1] J. Alglave, L. Maranget, and M. Tautschnig, "Herding Cats," *ACM TOPLAS*, vol. 36, no. 2, pp. 1–74, jul 2014.

[2] J. Alglave, L. Maranget, S. Sarkar, and P. Sewell, "Litmus: Running tests against hardware," *Lecture Notes in Computer Science*, vol. 6605 LNCS, pp. 41–44, 2011.