Explaining Relaxed Memory Models with Program Transformations

Ori Lahav and Viktor Vafeiadis

Presented by
Akshay Gopalakrishnan

September 26, 2021

Introduction

- Uniprocessor semantics respects sequential consistency.
- Mutliprocessor has more behaviors that programs can exhibit.
- They are either justified by hardware features such as Load/Store buffers, speculation, etc.
- Or they can be justified by the program begin optimized in various passes by a compiler.
- Semantics of such behaviors however, are not specified using such intuition, rather, per-execution or event-structure based axiomatic models, or more non-trivial operational models.
- This paper investigates upto what extent such memory consistency models can be specified using program transformations.

Contributions

- TSO can be explained by W-R reordering and Read-after-Write elimination over SC.
- RA cannot be specified in the same manner - counter example to show this.
- A substantial subset of POWER can be specified in a similar way over a stronger model of power - SPOWER
- ARM however, cannot be done the same way.
- Advantage of using this to simplify compilation correctness proofs.

Axiomatic Model Definitions

Axiomatic events:

- Read event $\Re = Ld \cup U$.
- Write event $W = St \cup U$.
- Fence - Can be of various types depending on the architecture / language's memory model.

Binary relations:

- po - Program order per thread.
- rf - Between a read and the write from which its read value comes.
- co - Order between writes/updates to same location.
- mo - Order between memory writes/updates.

Sequential Consistency (SC)

Additional binary relations:

- hb - Happens before $hb = (po \cup rf)^+$
- fr - Reads before (from reads) $fr = (rf^{-1}; mo_{|loc})$

SC is defined using the following irreflexivity relations that must hold for any program execution:

a) mo total order on W/U.

b) hb irreflexive.

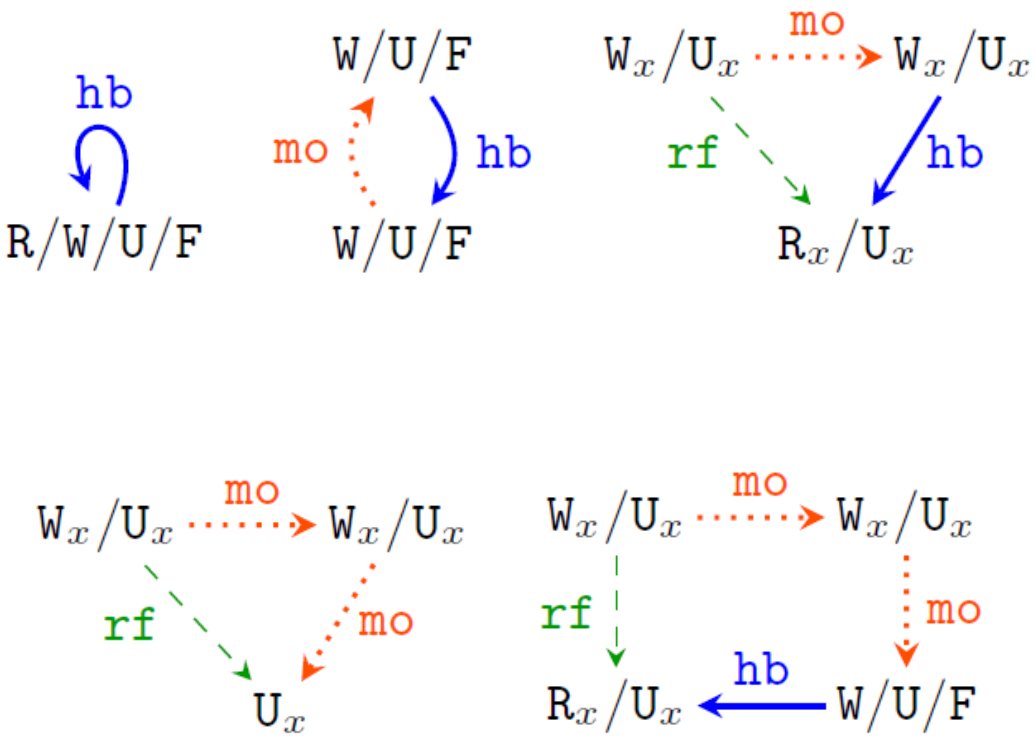
c) mo;hb irreflexive.

d) fr;hb irreflexive.

e) fr;mo irreflexive.

f) fr;mo;hb irreflexive.

Figures to explain above axioms of irreflexivity



Total Store Order (TSO)

Additional binary relations:

- xhb - Happens before $xhb = (po \cup rf)^+$
- fr - Reads before (from reads) $fr = (rf^{-1}; mo_{|loc})$
- rfe - Reads from external $rf \setminus po$

x86 TSO is defined using the following irreflexivity relations that must hold for any program execution:

a) mo is a strict total order on W/U/F.

b) xhb irreflexive.

c) mo;xhb irreflexive.

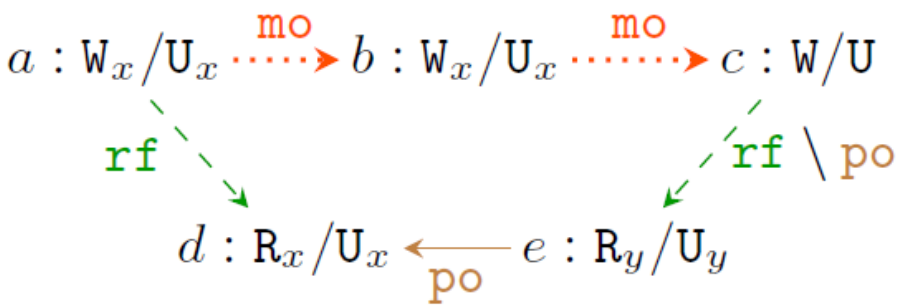
d) fr;xhb irreflexive.

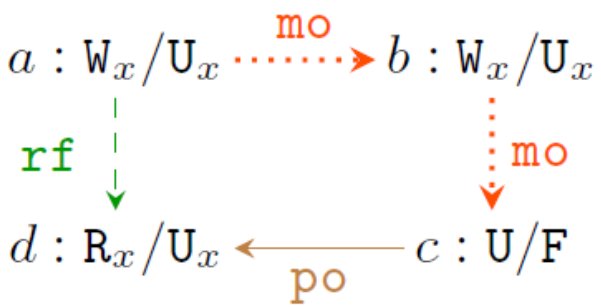
e) fr;mo irreflexive.

f) fr;mo;rfe;po irreflexive.

g) fr;mo;$[U \cup F]$;po irreflexive.

Figures to explain the last two axioms of irreflexivity.





TSO Equivalence

TSO is precisely categorized by write-read reordering and read-after-write elimination over SC.

Theorem 1

A plain execution G is TSO-consistent iff $G \longmapsto^*_{TSO} G'$ for some SC-consistent execution G'.

where $\longmapsto^*_{TSO}$ implies we either do write-read reordering or read-after-write elimination.

Proof elements

Proposition 1

If $G \longmapsto^*_{TSO} G'$ *and* G' *is TSO consistent, then so is* G.

Proposition 2

If G *is TSO-consistent execution, then so is* $RemoveWR(G, a, b)$.

Proposition 3

If G *is TSO-consistent execution, then so is* $ReorderWR(G, a, b)$ *if* $\langle a, b\rangle \notin mo; rf$.

Proof elements cntd

Proposition 4

Suppose G is TSO-consistent but not SC-consistent, then $G \longmapsto^*_{TSO} G'$ for some TSO-consistent execution G'.

Release Acquire (RA)

This model exhibits non-multi-copy-atomicity, meaning, threads may observe updates/writes in different orders. Here mo is defined only between events W/U accessing the same memory. The axioms then are as follows:

a) mo disjoint union of relations mo_x each of which are strict total orders.

b) hb irreflexive.

c) mo;hb irreflexive.

d) fr;hb irreflexive.

e) fr;mo irreflexive.

where hb, fr are as that of Sequential consistency.

Why RA cannot be described the same way

$$\begin{array}{l} a := x; \ /\!/ 1 \\ b := y; \ /\!/ 0 \end{array} \Big\| \ x := 1; \ \Big\| \ y := 1; \ \Big\| \ \begin{array}{l} c := y; \ /\!/ 1 \\ d := x; \ /\!/ 0 \end{array}$$

$$\begin{array}{l} x := 1; \\ a := x; \ /\!/ 1 \\ b := y; \ /\!/ 0 \end{array} \Big\| \ y := 1; \ \Big\| \ \begin{array}{l} c := y; \ /\!/ 1 \\ d := x; \ /\!/ 0 \end{array} \rightsquigarrow \begin{array}{l} b := y; \ /\!/ 0 \\ x := 1; \\ a := 1; \ /\!/ 1 \end{array} \Big\| \ y := 1; \ \Big\| \ \begin{array}{l} c := y; \ /\!/ 1 \\ d := x; \ /\!/ 0 \end{array}$$

Why RA cannot be described the same way

$$
\begin{array}{l}
y := 1; \\
x := 1; \\
a := x; \quad /\!/ 3 \\
b := z; \quad /\!/ 0
\end{array}
\;\Big\|\;
x := 3;
\;\Big\|\;
\begin{array}{l}
z := 1; \\
x := 2; \\
c := x; \quad /\!/ 3 \\
d := y; \quad /\!/ 0
\end{array}
$$

Similar Analysis for POWER and ARMv7/v8

- POWER can be specified using SPOWER, whose only difference is it disallows $po \cup rf$ cycles.
- Then POWER is equivalent to reordering over SPOWER (you keep reordering until only dependency order = program order).
- ARM cannot be specified in a similar way (perhaps refer to figure in notes section that cannot be explained by program transformations)

Advantage of Transformational Models: Compiler Correctness

Consider $[[P]]_M$ to be behaviors of program P under memory model M. Assume we have a compilation scheme from source C to target A (code gen phase: direct mapping of instructions). Let M_C and M_A be their respective memory models. Then compiler correctness requires:

$$\forall P_C.[[compile(P_C)]]_{M_A} \subseteq [[P_C]]_{M_C}.$$

Compiler Correctness simplified using results in paper

Theorem 1 and 2 provide us with the following property:

$$\forall P_C.[[compile(P_C)]]_{M_A} \subseteq \cup\{[[P'_A]]_{SM_A} | P'_A.compile(P_C) \longmapsto^*_{M_A} P'_A\}.$$

where SM_A is the stronger memory model than M_A (like SC for TSO in Theorem 1).

Simplified proof requirements

Using above info, compiler correctness boils down to two conditions.

- Compilation should be correct for the strong model SM_A.
- There should be a set of source program transformations sound for source model M_C and it should capture all target transformations from a compiled program.

Thank you

Questions ?