Fences in Weak Memory Models: Appendix

Jade Alglave[1], Luc Maranget[1], Susmit Sarkar[2], and Peter Sewell[2]

[1] INRIA [2] University of Cambridge

Abstract. We present an axiomatic framework, implemented in Coq, to define weak memory models $w.r.t.$ several parameters: local reorderings of reads and writes, and visibility of inter and intra processor communications through memory, including full store atomicity relaxation. Thereby, we give a formal hierarchy of weak memory models, in which we provide a formal study of what should be the action and placement of fences to restore a given model such as SC from a weaker one. Finally, we provide a tool, diy, that tests a given machine to determine the architecture it exhibits. We detail the results of our experiments on Power and the model we extract from it. This identified an implementation error in Power 5 memory barriers (for which IBM is providing a software workaround); our results also suggest that Power 6 does not suffer from this problem.

We present here additional data and explanations, as well as the proof sketches of the results that appear in the paper. Moreover, our results are entirely implemented in the Coq proof assistant: the development is available at `http://moscova.inria.fr/~alglave/wmm/`.

1 Tables of notations

Communication relaxations

diy relaxation	Relation	Source	Target	Processor	Location
Rfe	$\overset{\text{rfe}}{\longrightarrow}$	W	R	Different	Same
Rfi	$\overset{\text{rfi}}{\longrightarrow}$	W	R	Same	Same
Wse	$\overset{\text{wse}}{\longrightarrow}$	W	W	Different	Same
Wsi	$\overset{\text{wsi}}{\longrightarrow}$	W	W	Same	Same
Fre	$\overset{\text{fre}}{\longrightarrow}$	R	W	Different	Same
Fri	$\overset{\text{fri}}{\longrightarrow}$	R	W	Same	Same

Program order relaxations

Program order proper

diy relaxation	Relation	Source	Target	Processor	Location
PosRR	$\bigcup_\ell \overset{\text{po}}{\rightarrow} \cap(\mathbb{R}_\ell \times \mathbb{R}_\ell)$	R	R	Same	Same
PodRR	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{po}}{\rightarrow} \cap(\mathbb{R}_{\ell_1} \times \mathbb{R}_{\ell_2})$	R	R	Same	Diff
PosRW	$\bigcup_\ell \overset{\text{po}}{\rightarrow} \cap(\mathbb{R}_\ell \times \mathbb{W}_\ell)$	R	W	Same	Same
PodRW	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{po}}{\rightarrow} \cap(\mathbb{R}_{\ell_1} \times \mathbb{W}_{\ell_2})$	R	W	Same	Diff
PosWW	$\bigcup_\ell \overset{\text{po}}{\rightarrow} \cap(\mathbb{W}_\ell \times \mathbb{W}_\ell)$	W	W	Same	Same
PodWW	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{po}}{\rightarrow} \cap(\mathbb{W}_{\ell_1} \times \mathbb{W}_{\ell_2})$	W	W	Same	Diff
PosWR	$\bigcup_\ell \overset{\text{po}}{\rightarrow} \cap(\mathbb{W}_\ell \times \mathbb{R}_\ell)$	W	R	Same	Same
PodWR	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{po}}{\rightarrow} \cap(\mathbb{W}_{\ell_1} \times \mathbb{R}_{\ell_2})$	W	R	Same	Diff

Dependencies

diy relaxation	Relation	Source	Target	Processor	Location
DpsR	$\bigcup_\ell \overset{\text{dp}}{\rightarrow} \cap(\mathbb{R}_\ell \times \mathbb{R}_\ell)$	R	R	Same	Same
DpdR	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{dp}}{\rightarrow} \cap(\mathbb{R}_{\ell_1} \times \mathbb{R}_{\ell_2})$	R	R	Same	Diff
DpsW	$\bigcup_\ell \overset{\text{dp}}{\rightarrow} \cap(\mathbb{R}_\ell \times \mathbb{W}_\ell)$	R	W	Same	Same
DpdW	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{dp}}{\rightarrow} \cap(\mathbb{R}_{\ell_1} \times \mathbb{W}_{\ell_2})$	R	W	Same	Diff

Barriers relaxations: base cases

Sync

diy relaxation	Relation	Source	Target	Processor	Location
SyncsRR	$\bigcup_\ell \overset{\text{sync}}{\rightarrow} \cap(\mathbb{R}_\ell \times \mathbb{R}_\ell)$	R	R	Same	Same
SyncdRR	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{sync}}{\rightarrow} \cap(\mathbb{R}_\ell \times \mathbb{R}_\ell)$	R	R	Same	Diff
SyncsRW	$\bigcup_\ell \overset{\text{sync}}{\rightarrow} \cap(\mathbb{R}_\ell \times \mathbb{W}_\ell)$	R	W	Same	Same
SyncdRW	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{sync}}{\rightarrow} \cap(\mathbb{R}_\ell \times \mathbb{W}_\ell)$	R	W	Same	Diff
SyncsWR	$\bigcup_\ell \overset{\text{sync}}{\rightarrow} \cap(\mathbb{W}_\ell \times \mathbb{R}_\ell)$	W	R	Same	Same
SyncdWR	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{sync}}{\rightarrow} \cap(\mathbb{W}_\ell \times \mathbb{R}_\ell)$	W	R	Same	Diff
SyncsWW	$\bigcup_\ell \overset{\text{sync}}{\rightarrow} \cap(\mathbb{W}_\ell \times \mathbb{W}_\ell)$	W	W	Same	Same
SyncdWW	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{sync}}{\rightarrow} \cap(\mathbb{W}_\ell \times \mathbb{W}_\ell)$	W	W	Same	Diff

LwSync

diy relaxation	Relation	Source	Target	Processor	Location
LwSyncsRR	$\bigcup_\ell \overset{\text{lwsync}}{\rightarrow} \cap(\mathbb{R}_\ell \times \mathbb{R}_\ell)$	R	R	Same	Same
LwSyncdRR	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{lwsync}}{\rightarrow} \cap(\mathbb{R}_\ell \times \mathbb{R}_\ell)$	R	R	Same	Diff
LwSyncsRW	$\bigcup_\ell \overset{\text{lwsync}}{\rightarrow} \cap(\mathbb{R}_\ell \times \mathbb{W}_\ell)$	R	W	Same	Same
LwSyncdRW	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{lwsync}}{\rightarrow} \cap(\mathbb{R}_\ell \times \mathbb{W}_\ell)$	R	W	Same	Diff
LwSyncsWR	$\bigcup_\ell \overset{\text{lwsync}}{\rightarrow} \cap(\mathbb{W}_\ell \times \mathbb{R}_\ell)$	W	R	Same	Same
LwSyncdWR	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{lwsync}}{\rightarrow} \cap(\mathbb{W}_\ell \times \mathbb{R}_\ell)$	W	R	Same	Diff
LwSyncsWW	$\bigcup_\ell \overset{\text{lwsync}}{\rightarrow} \cap(\mathbb{W}_\ell \times \mathbb{W}_\ell)$	W	W	Same	Same
LwSyncdWW	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{lwsync}}{\rightarrow} \cap(\mathbb{W}_\ell \times \mathbb{W}_\ell)$	W	W	Same	Diff

Barriers relaxations: A-cumulativity

Sync

diy relaxation	Relation	Source	Target	Processor	Location
ACSyncsRR	$\bigcup_\ell \overset{\text{rfe}}{\rightarrow}; (\overset{\text{sync}}{\rightarrow} \cap (\mathbb{R}_\ell \times \mathbb{R}_\ell))$	R	R	Same	Same
ACSyncdRR	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{rfe}}{\rightarrow}; (\overset{\text{sync}}{\rightarrow} \cap (\mathbb{R}_\ell \times \mathbb{R}_\ell))$	R	R	Same	Diff
ACSyncsRW	$\bigcup_\ell \overset{\text{rfe}}{\rightarrow}; (\overset{\text{sync}}{\rightarrow} \cap (\mathbb{R}_\ell \times \mathbb{W}_\ell))$	R	W	Same	Same
ACSyncdRW	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{rfe}}{\rightarrow}; (\overset{\text{sync}}{\rightarrow} \cap (\mathbb{R}_\ell \times \mathbb{W}_\ell))$	R	W	Same	Diff

LwSync

diy relaxation	Relation	Source	Target	Processor	Location
ACLwSyncsRR	$\bigcup_\ell \overset{\text{rfe}}{\rightarrow}; (\overset{\text{lwsync}}{\rightarrow} \cap (\mathbb{R}_\ell \times \mathbb{R}_\ell))$	R	R	Same	Same
ACLwSyncdRR	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{rfe}}{\rightarrow}; (\overset{\text{lwsync}}{\rightarrow} \cap (\mathbb{R}_\ell \times \mathbb{R}_\ell))$	R	R	Same	Diff
ACLwSyncsRW	$\bigcup_\ell \overset{\text{rfe}}{\rightarrow}; (\overset{\text{lwsync}}{\rightarrow} \cap (\mathbb{R}_\ell \times \mathbb{W}_\ell))$	R	W	Same	Same
ACLwSyncdRW	$\bigcup_{\ell_1 \neq \ell_2} \overset{\text{rfe}}{\rightarrow}; (\overset{\text{lwsync}}{\rightarrow} \cap (\mathbb{R}_\ell \times \mathbb{W}_\ell))$	R	W	Same	Diff

Barriers relaxations: B-cumulativity

Sync

diy relaxation	Relation	Source	Target	Processor	Location
BCSyncsRW	$\bigcup_\ell (\overset{\text{sync}}{\rightarrow} \cap (\mathbb{R}_\ell \times \mathbb{W}_\ell)); \overset{\text{rfe}}{\rightarrow}$	R	W	Same	Same
BCSyncdRW	$\bigcup_{\ell_1 \neq \ell_2} (\overset{\text{sync}}{\rightarrow} \cap (\mathbb{R}_\ell \times \mathbb{W}_\ell)); \overset{\text{rfe}}{\rightarrow}$	R	W	Same	Diff
BCSyncsWW	$\bigcup_\ell (\overset{\text{sync}}{\rightarrow} \cap (\mathbb{W}_\ell \times \mathbb{W}_\ell)); rfe$	W	W	Same	Same
BCSyncdWW	$\bigcup_{\ell_1 \neq \ell_2} (\overset{\text{sync}}{\rightarrow} \cap (\mathbb{W}_\ell \times \mathbb{W}_\ell)); \overset{\text{rfe}}{\rightarrow}$	W	W	Same	Diff

LwSync

diy relaxation	Relation	Source	Target	Processor	Location
BCLwSyncsRW	$\bigcup_\ell (\overset{\text{lwsync}}{\rightarrow} \cap (\mathbb{R}_\ell \times \mathbb{W}_\ell)); \overset{\text{rfe}}{\rightarrow}$	R	W	Same	Same
BCLwSyncdRW	$\bigcup_{\ell_1 \neq \ell_2} (\overset{\text{lwsync}}{\rightarrow} \cap (\mathbb{R}_\ell \times \mathbb{W}_\ell)); \overset{\text{rfe}}{\rightarrow}$	R	W	Same	Diff
BCLwSyncsWW	$\bigcup_\ell (\overset{\text{lwsync}}{\rightarrow} \cap (\mathbb{W}_\ell \times \mathbb{W}_\ell)); \overset{\text{rfe}}{\rightarrow}$	W	W	Same	Same
BCLwSyncdWW	$\bigcup_{\ell_1 \neq \ell_2} (\overset{\text{lwsync}}{\rightarrow} \cap (\mathbb{W}_\ell \times \mathbb{W}_\ell)); \overset{\text{rfe}}{\rightarrow}$	W	W	Same	Diff

2 Properties of validity

From our definition of validity arises a simple notion of comparison among architectures. $A_1 \leq A_2$ means that A_1 is *weaker* than A_2:

Definition 1 (Weaker).

$$A_1 \leq A_2 \triangleq (\overset{ppo_1}{\rightarrow} \subseteq \overset{ppo_2}{\rightarrow}) \wedge (\overset{grf_1}{\rightarrow} \subseteq \overset{grf_2}{\rightarrow})$$

2.1 Validity is decreasing

We prove validity of an execution to be decreasing w.r.t the weaker predicate; thus, a weak architecture exhibits at least all the behaviours authorised by a stronger one:

Theorem 1 (Validity is decreasing).

$$\forall A_1 A_2,\ (A_1 \leq A_2) \Rightarrow (\forall X, A_2^{\epsilon}\,.\,\mathrm{valid}(X) \Rightarrow A_1^{\epsilon}\,.\,\mathrm{valid}(X))$$

Proof (in Coq). From $A_1 \leq A_2$, we immediately have $A_1.ghb \subseteq A_2.ghb$, thus if $A_2.ghb$ is acyclic, so is $A_1.ghb$. □

2.2 Monotonicity of validity

Programs running on an architecture A_1 exhibit executions that would be valid on a stronger architecture A_2; we characterise all such executions as follows:

$$A_1.\,\mathrm{check}_{\mathrm{A}_2}(X) \ \triangleq\ \mathrm{acyclic}(\overset{\mathrm{grf}_2}{\rightarrow} \cup \overset{\mathrm{ws}}{\rightarrow} \cup \overset{\mathrm{fr}}{\rightarrow} \cup \overset{\mathrm{ppo}_2}{\rightarrow})$$

Theorem 2 (Characterisation).

$$\forall A_1 A_2,\ (A_1 \leq A_2) \Rightarrow (\forall X, (A_1^{\epsilon}\,.\,\mathrm{valid}(X) \wedge A_1\,.\,\mathrm{check}_{\mathrm{A}_2}(X)) \Leftrightarrow A_2^{\epsilon}\,.\,\mathrm{valid}(X))$$

Proof (in Coq).

- ⇒ X being valid on A_1, we have all requirements – well formedness and uniproc – to guarantee it is valid on A_2, except the last predicate, which holds by the hypothesis $\mathrm{check}_{\mathrm{A}_2}$.
- ⇐ X being valid on A_2 gives us all requirements – well formedness and uniproc – to guarantee its validity on A_1 except the last one. As $A_1 \leq A_2$, we know that $A_1\,.\,ghb \subseteq A_2\,.\,ghb$ (lemma ghb_incl), thus the acyclicity requirement for $A_1\,.\,ghb$ holds if $A_2\,.\,ghb$ is acyclic. □

3 Examples of models in our framework

3.1 *SC*

Definitions

$$Sc \ \triangleq\ (MM, \overset{\mathrm{rf}}{\rightarrow}, \emptyset)$$

$$\overset{\text{hb-seq}}{\rightarrow} \ \triangleq\ \overset{\mathrm{ws}}{\rightarrow} \cup \overset{\mathrm{fr}}{\rightarrow} \cup \overset{\mathrm{rf}}{\rightarrow}$$

Sc characterisation

Corollary 1 (*Sc* characterisation).

$$\forall A, (A \leq Sc) \Rightarrow (\forall X, \mathrm{valid}\, AX \wedge A\,.\,\mathrm{check}_{Sc}\ X \Leftrightarrow Sc.\,\mathrm{valid}\, X)$$

Proof (in Coq).

- ⇒ As $\overset{\mathrm{po}}{\rightarrow} \cup \overset{\text{hb-seq}}{\rightarrow} = Sc\,.\,ghb$, this is a direct consequence of thm. 2.
- ⇐ as $A \leq Sc$, this is a direct consequence of thm. 1. □

Sc is SC *SC* has been defined in [1] as follows:

> *[. . .] the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.*

Thus an *SC* execution is a total order $\overset{\text{ex}}{\rightarrow}$ consistent with the program order:

$$\text{seq}(\overset{\text{ex}}{\rightarrow}) \triangleq \text{total-order}(\overset{\text{ex}}{\rightarrow}, \mathbb{E}) \wedge \overset{\text{po}}{\rightarrow} \subseteq \overset{\text{ex}}{\rightarrow}$$

The implicit execution model of [1] states that a read r takes its value from the most recent write that precedes it in $\overset{\text{ex}}{\rightarrow}$. Writing $\text{pw}(o, r)$ for the set of writes to the same location that precede r according to partial order o, we extract our $\overset{\text{rf}}{\rightarrow}$ relation from $\overset{\text{ex}}{\rightarrow}$ as follows:

$$\text{pw}(\overset{\text{ex}}{\rightarrow}, r) \triangleq \{w \mid \text{loc}(w) = \text{loc}(r) \wedge w \overset{\text{ex}}{\rightarrow} r\}$$
$$SC\,.\,\text{rf}(\overset{\text{ex}}{\rightarrow}) \triangleq \{(w, r) \mid w = \max_{\overset{\text{ex}}{\rightarrow}} \left(\text{pw}(\overset{\text{ex}}{\rightarrow}, r)\right)\}$$

We extract write serialisation as well, and one of our execution witnesses from $\overset{\text{ex}}{\rightarrow}$, which we use to show that the *SC* definition from [1] is equivalent to ours:

$$SC\,.\,\text{ws}(\overset{\text{ex}}{\rightarrow}) \triangleq \bigcup_{\ell} (\mathbb{W}_\ell \times \mathbb{W}_\ell) \cap \overset{\text{ex}}{\rightarrow}$$
$$SC\,.\,\text{wit}(\overset{\text{ex}}{\rightarrow}) \triangleq (\mathbb{E}, \overset{\text{po}}{\rightarrow}, SC\,.\,\text{rf}(\overset{\text{ex}}{\rightarrow}), SC\,.\,\text{ws}(\overset{\text{ex}}{\rightarrow}))$$

Theorem 3 (*Sc* is *SC*).

$$\forall X,\ Sc\,.\,\text{valid}(X) \Leftrightarrow \exists \overset{ex}{\rightarrow},\ \text{seq}(\overset{ex}{\rightarrow}) \wedge SC\,.\,\text{wit}(\overset{ex}{\rightarrow}) = X$$

Proof (in Coq).

- $\Rightarrow$ from X being valid on *Sc*, we have $\text{acyclic}(\overset{\text{ghb}}{\rightarrow})$, that is $\text{acyclic}(\overset{\text{hb-seq}}{\rightarrow} \cup \overset{\text{po}}{\rightarrow})$ on *Sc*, which gives us an equivalent *SC* execution by Cor. 1.
- $\Leftarrow$ from $\overset{\text{ex}}{\rightarrow}$, we produce a $SC\,.\,\text{wit}$ which is valid by Thm. 1.

□

3.2 *TSO*

Definitions

$$\overset{\text{ppo-tso}}{\rightarrow} \triangleq RM \cup WW$$

$$Tso^{\epsilon} \triangleq (\overset{\text{ppo-tso}}{\rightarrow}, \overset{\text{rfe}}{\rightarrow}, \emptyset)$$

$$\overset{\text{hb-tso}}{\rightarrow} \triangleq \overset{\text{ws}}{\rightarrow} \cup \overset{\text{fr}}{\rightarrow} \cup \overset{\text{rfe}}{\rightarrow}$$

Tso characterisation By Thm. 2 we show the following criterion – where $\overset{\text{hb-tso}}{\rightarrow}$ is $\overset{\text{ws}}{\rightarrow} \cup \overset{\text{fr}}{\rightarrow} \cup \overset{\text{rfe}}{\rightarrow}$ – characterises valid executions that are Tso^* on any $A \leq Tso$:

$$A\,.\,\text{check}_{Tso}(X) \triangleq \text{acyclic}(\overset{\text{ppo-tso}}{\rightarrow} \cup \overset{\text{hb-tso}}{\rightarrow})$$

Corollary 2 (Tso characterisation).

$$\forall A, (A \leq Tso) \Rightarrow (\forall X, \text{valid}\, AX \wedge A\,.\,\text{check}_{Tso}\, X \Leftrightarrow \text{valid}\, Tso(X))$$

Proof (in Coq).

$\Rightarrow$ As $\overset{\text{ppo-tso}}{\rightarrow} \cup \overset{\text{hb-tso}}{\rightarrow} = Tso\,.\,\overset{\text{ghb}}{\rightarrow}$, this is a direct consequence of thm. 2.
$\Leftarrow$ as $A \leq Tso$, this is a direct consequence of thm. 1. □

Tso is TSO

Equivalence with the native model Sparc [3, V. 8, Appendix K] formally defines a *TSO* execution as a partial order $\overset{\text{ex}}{\rightarrow}$ on memory events constrained by some axioms that constrain $\overset{\text{ex}}{\rightarrow}$. We formulate[1] those as follows:

$$\text{ptso}(\overset{\text{ex}}{\rightarrow}) \triangleq \text{partial-order}(\overset{\text{ex}}{\rightarrow}, \mathbb{E}) \wedge RM \subseteq \overset{\text{ex}}{\rightarrow} \wedge\ WW \subseteq \overset{\text{ex}}{\rightarrow} \wedge$$
$$\exists \overset{\text{tso}}{\rightarrow},\ \overset{\text{tso}}{\rightarrow} \subseteq \overset{\text{ex}}{\rightarrow} \wedge \text{total-order}(\overset{\text{tso}}{\rightarrow}, \mathbb{W})$$

Note that $\overset{\text{tso}}{\rightarrow}$ above is the postulated total order on stores that justify the name *TSO*. An additional axiom, *Value*, where L_a (resp. S_a) is the notation of [3] for a load (resp. a store) with location a:

$$Val(L_a) = Val(\max_{\overset{\text{ex}}{\rightarrow}}\{S_a \mid S_a \overset{\text{ex}}{\rightarrow} L_a \vee S_a \overset{\text{po}}{\rightarrow} L_a\})$$

> *[...] states that the value of a data load is the value written by the most recent store to that location. Two terms combine to define the most recent store. The first corresponds to stores by other processors, while the second corresponds to stores by the processor that issued the load.*

We relate Sparc *TSO* and our *Tso* by interpreting the *Value* axiom as specifying $\overset{\text{rf}}{\rightarrow}$, and extract $\overset{\text{ws}}{\rightarrow}$ from $\overset{\text{ex}}{\rightarrow}$, as we did in the *Sc* case:

$$TSO\,.\,\text{rf}(\overset{\text{ex}}{\rightarrow}) \triangleq \{(w, r) \mid w = \max_{\overset{\text{ex}}{\rightarrow}}\left(\text{pw}(\overset{\text{ex}}{\rightarrow} \cup \overset{\text{po}}{\rightarrow}, r)\right)\}$$
$$TSO\,.\,\text{ws}(\overset{\text{ex}}{\rightarrow}) \triangleq \{(w_1, w_2) \mid \exists l, (w_1, w_2) \in (\mathbb{W}_l \times \mathbb{W}_l) \wedge w_1 \overset{\text{ex}}{\rightarrow} w_2\}$$
$$TSO\,.\,\text{wit}(\overset{\text{ex}}{\rightarrow}) \triangleq (\mathbb{E}, \overset{\text{po}}{\rightarrow}, TSO\,.\,\text{rf}(\overset{\text{ex}}{\rightarrow}), SC\,.\,\text{ws}(\overset{\text{ex}}{\rightarrow}))$$

Theorem 4 (Tso is TSO).

$$\forall X, Tso^*\,.\,\text{valid}(X) \Leftrightarrow \exists \overset{ex}{\rightarrow}, \text{ptso}(\overset{ex}{\rightarrow}) \wedge TSO\,.\,\text{wit}(\overset{ex}{\rightarrow}) = X$$

[1] We omit the axioms *Atomicity* and *Termination*.

Proof (in Coq).

⇒ from X being valid on Tso, we have acyclic($\overset{\text{ghb}}{\rightarrow}$), that is acyclic($\overset{\text{hb-tso}}{\rightarrow} \cup \overset{\text{po-tso}}{\rightarrow}$) on Tso, which gives us an equivalent TSO execution by Cor. 2.
⇐ from $\overset{\text{ex}}{\rightarrow}$, we produce a TSO. wit which is valid by Thm. 1.

□

4 Barriers

From now on, we note $A.\ \overset{\text{hb-seq}\prime}{\rightarrow}$ for $\overset{?\text{rfe}}{\rightarrow} \cup (\overset{\text{ws}}{\rightarrow}; \overset{?\text{rfe}}{\rightarrow}) \cup (\overset{\text{fr}}{\rightarrow}; \overset{?\text{rfe}}{\rightarrow} \cup \overset{\text{ws}}{\rightarrow} \cup \overset{\text{fr}}{\rightarrow}$. We trivially have $A.\ \overset{\text{hb-seq}}{\rightarrow} \subseteq A.\ \overset{\text{hb-seq}\prime}{\rightarrow}$, and that $\overset{\text{hb-seq}\prime}{\rightarrow}$ is transitive.

A key lemma is that forall relation $\overset{\text{r}}{\rightarrow}$ over events, such that there is a cycle in $\overset{\text{hb-seq}}{\rightarrow} \cup \overset{\text{r}}{\rightarrow}$, then there is a cycle in $\overset{\text{hb-seq}\prime}{\rightarrow}; (\overset{+}{\rightarrow} \overset{\text{r}}{\rightarrow})$.

4.1 Barriers guarantee

Consider two architectures $A_1 \leq A_2$. We define the predicate fb (*fully barriered*) on $A_1 \leq A_2$, where $\overset{\text{r}_2 \setminus 1}{\rightarrow} \triangleq \overset{\text{r}_2}{\rightarrow} \setminus \overset{\text{r}_1}{\rightarrow}$ is the set difference, and $x \overset{\text{r}_1}{\rightarrow}; \overset{\text{r}_2}{\rightarrow} y \triangleq \exists z, x \overset{\text{r}_1}{\rightarrow} z \wedge z \overset{\text{r}_2}{\rightarrow} y$ stands for the sequence:

$$A_1 \,.\, \text{fb}_{A_2}(X) \triangleq \left((\overset{\text{ppo}_2 \setminus 1}{\rightarrow}) \cup (\overset{\text{grf}_2 \setminus 1}{\rightarrow}; \overset{\text{ppo}_2}{\rightarrow}) \right) \subseteq \overset{\text{ab}_1}{\rightarrow}$$

We prove that the above condition on $\overset{\text{ab}_1}{\rightarrow}$ suffices to restore A_2 from A_1:

Theorem 5 (Barriers guarantee).

$$\forall A_1 A_2, (A_1 \leq A_2) \Rightarrow (\forall X, A_1 \,.\, \text{valid}(X) \wedge A_1 \,.\, \text{fb}_{A_2}(X) \Rightarrow A_2^{\epsilon} \,.\, \text{valid}(X))$$

Proof (in Coq).

Let relation $\overset{\text{hb'}_2}{\rightarrow}$ be $\overset{?\text{rfe}_2}{\rightarrow} \cup (\overset{\text{ws}}{\rightarrow}; \overset{?\text{rfe}_2}{\rightarrow}) \cup (\overset{\text{fr}}{\rightarrow}; \overset{?\text{rfe}_2}{\rightarrow}) \cup \overset{\text{ws}}{\rightarrow} \cup \overset{\text{fr}}{\rightarrow}$, and relation $\overset{\text{hb"}_2}{\rightarrow}$ be $(\overset{?\text{rfi}_2}{\rightarrow} \cup \overset{\text{ppo}_2}{\rightarrow})^+$. We prove that $\overset{?\text{rf}_2}{\rightarrow} \cup \overset{\text{ws}}{\rightarrow} \cup \overset{\text{fr}}{\rightarrow} \cup \overset{\text{ppo}_2}{\rightarrow}$ is acyclic if and only if $R_2 = \overset{\text{hb'}_2}{\rightarrow}; \overset{\text{hb"}_2}{\rightarrow}$ is. Then, to prove the theorem it suffices to prove that $\overset{\text{ghb}_1}{\rightarrow}$ is cyclic whenever R_2 is.

Let us consider a cycle in R_2. Because $m_1 \overset{\text{rf}}{\rightarrow} m_2 \overset{\text{hb'}_2}{\rightarrow} m_3$ implies $m_1 \overset{\text{hb'}_2}{\rightarrow} m_3$, we can safely assume that no $\overset{\text{hb"}_2}{\rightarrow}$ step ends by a $\overset{?\text{rfi}_1}{\rightarrow}$ step. Then, as no two $\overset{\text{rfi}}{\rightarrow}$ steps can follow one another, condition $A_1 \,.\, \text{fb}_{A_2}(X)$ suffices to guarantee the inclusion $R_2 \subseteq (\overset{\text{ghb}_1}{\rightarrow})^+$ and conclude. □

4.2 Considering a weaker barriers guarantee

When two architectures A_2 and A_1 have the same policy *w.r.t.* the store atomicity and store buffer relaxations, which we model by $\overset{\text{grf}_1}{\rightarrow} = \overset{\text{grf}_2}{\rightarrow}$, there is no need for a barrier as powerful as above to restore A_2 from A_1: a barrier that only orders the events that surround it statically—that is, a non cumulative barrier, which action we model by non-cumul(X, $\overset{\text{fenced}}{\rightarrow}$) $\triangleq$ $\overset{\text{fenced}}{\rightarrow}$—is enough. Consider

the wfb predicate, which states that the barriers provided by A_1 maintain the pairs that are preserved in the program order on A_2 but not on A_1.:

$$A_1 \,.\, \mathrm{wfb}_{A_2}(X) \triangleq {}^{\mathrm{ppo}_2 \setminus 1}\!\!\rightarrow \subseteq {}^{\mathrm{ab}_1}\!\!\rightarrow$$

The same guarantee applies if A_2 hinders the store buffer relaxation by its preserved program order, *i.e.* when ${}^{\mathrm{rfi}}\!\!\rightarrow \subseteq {}^{\mathrm{ppo}_2}\!\!\rightarrow$—which is particular to *Sc*:

Theorem 6 (Non cumulative barriers guarantee).

$$\forall A_1 A_2, ((A_1 \leq A_2) \wedge ({}^{grf_1}\!\!\rightarrow = {}^{grf_2}\!\!\rightarrow \vee ({}^{rfi}\!\!\rightarrow \subseteq {}^{ppo_2}\!\!\rightarrow))) \Rightarrow$$

$$(\forall X, A_1 \,.\, \mathrm{valid}(X) \wedge A_1 \,.\, \mathrm{wfb}_{A_2}(X) \Rightarrow A_2^{\epsilon} \,.\, \mathrm{valid}(X))$$

Proof (in Coq). As for Thm. 5 we need prove $R_2 = ({}^{\mathrm{hb'}_2}\!\!\rightarrow; {}^{\mathrm{hb"}_2}\!\!\rightarrow) \subseteq ({}^{\mathrm{ghb}_1}\!\!\rightarrow)^+$. First, by hypothesis $ext_1 = ext_2$, we get ${}^{\mathrm{hb'}_2}\!\!\rightarrow = {}^{\mathrm{hb'}_1}\!\!\rightarrow \subseteq {}^{\mathrm{ghb}_1}\!\!\rightarrow$. Second, we prove ${}^{\mathrm{hb"}_2}\!\!\rightarrow \subseteq ({}^{\mathrm{ghb}_1}\!\!\rightarrow)^+$, by considering each step in ${}^{\mathrm{hb"}_2}\!\!\rightarrow$. For a particular step $m_1 \, {}^{\mathrm{ppo}_2}\!\!\rightarrow \, m_2$, we have either $m_1 \, {}^{\mathrm{ppo}_1}\!\!\rightarrow \, m_2 \Rightarrow m_1 \, {}^{\mathrm{ghb}_1}\!\!\rightarrow m_2$, or $m_1 \, {}^{\mathrm{ppo}_2 \setminus 1}\!\!\rightarrow m_2 \Rightarrow m_1 \, {}^{\mathrm{ab}_1}\!\!\rightarrow m_2 \Rightarrow m_1 \, {}^{\mathrm{ghb}_1}\!\!\rightarrow m_2$. While for all ${}^{\mathrm{rfi}}\!\!\rightarrow$ steps, we conclude either by ${}^{\mathrm{rfi}}\!\!\rightarrow \subseteq {}^{\mathrm{ghb}_1}\!\!\rightarrow$ — when $int_2 = int_1 = true$, or by ${}^{\mathrm{rfi}}\!\!\rightarrow \subseteq {}^{\mathrm{ppo}_2}\!\!\rightarrow$. □

From Tso to Sc As ${}^{\mathrm{rfe}}\!\!\rightarrow$ are global in both *Tso* and *Sc*, and *Sc* hinders the store buffering relaxation by its ${}^{\mathrm{ppo}}\!\!\rightarrow$ definition, it suffices by Thm. 6 to fence all pairs in ${}^{\text{ppo-sc}}\!\!\rightarrow \setminus {}^{\text{ppo-tso}}\!\!\rightarrow = WR$ (where $WR \triangleq (\mathbb{W} \times \mathbb{R}) \cap {}^{\mathrm{po}}\!\!\rightarrow$) to restore *Sc* from *Tso*:

Corollary 3 (Barriers restoring *Sc* from *Tso*).

$$\forall X, (Tso \,.\, \mathrm{valid}(X) \wedge \text{non-cumul}(X, WR) \subseteq {}^{ab_{Tso}}\!\!\rightarrow) \Rightarrow Sc \,.\, \mathrm{valid}(X))$$

From Pso to Tso$^{\epsilon}$ We comment here on the two definitions of *PSO* given in Sparc documentations [3]. We adapt the definition of [3, V8] to our framework:

$$Pso^{\epsilon} \,.\, Arch \triangleq (\lambda X.RM, {}^{\mathrm{rfe}}\!\!\rightarrow, \lambda X.\emptyset)$$

Tso and *Pso* agree on both the store atomicity and the store buffering relaxations, which allows us to apply Thm. 6: *Tso$^{\epsilon}$* is restored from *Pso* by inserting non cumulative barriers between all ${}^{\text{ppo-tso}}\!\!\rightarrow \setminus {}^{\text{ppo-pso}}\!\!\rightarrow = WW$ pairs. Indeed, *TSO* is obtained from *PSO* by adding *StoreStore* barriers after each write [3, V9].

5 diy

5.1 Cycles generation

A type *edge* represents the arrows of an execution witness. It can be in ${}^{\mathrm{rf}}\!\!\rightarrow$, ${}^{\mathrm{ws}}\!\!\rightarrow$, ${}^{\mathrm{fr}}\!\!\rightarrow$: their source and target share the same location and have the appropriate

directions (*e.g.* $\overset{\text{rf}}{\rightarrow}$ has a write as source and a read as target). Similarly to $\overset{\text{rfi}}{\rightarrow}$ and $\overset{\text{rfe}}{\rightarrow}$, we define $\overset{\text{wsi}}{\rightarrow}$ and $\overset{\text{wse}}{\rightarrow}$, $\overset{\text{fri}}{\rightarrow}$ and $\overset{\text{fre}}{\rightarrow}$. An edge can also be in $\overset{\text{po}}{\rightarrow}$, $\overset{\text{dp}}{\rightarrow}$ or $\overset{\text{fenced}}{\rightarrow}$, where we need to specify the source and target directions (except for $\overset{\text{dp}}{\rightarrow}$ whose source is a read by definition), and if they share the same location.

We specify a concrete syntax for the edges: thus Rfe represents a $\overset{\text{rfe}}{\rightarrow}$ arrow, Fre a $\overset{\text{fre}}{\rightarrow}$ arrow, PosRW specifies a $\overset{\text{po}}{\rightarrow}$ arrow with a read (R) as source, a write (W) as target, knowing these events share the same location (s), and DpdR a $\overset{\text{dp}}{\rightarrow}$ arrow with a read (R) as target, with different (d) source and target locations.

We define two relations $\overset{\text{relax}}{\rightarrow}$ and $\overset{\text{safe}}{\rightarrow}$, whose default values are respectively $\overset{\text{po}}{\rightarrow} \cup \overset{\text{rf}}{\rightarrow}$ and $\overset{\text{ws}}{\rightarrow} \cup \overset{\text{fr}}{\rightarrow} \cup \overset{\text{dp}}{\rightarrow} \cup \overset{\text{fenced}}{\rightarrow}$. They can be modified by an input of the user: *e.g.* on a x86 machine, which has a *Tso* model [2], $\overset{\text{rfe}}{\rightarrow}$ can be specified as safe.

We show a cycle in $\overset{\text{hb-seq}}{\rightarrow} \cup \overset{\text{po}}{\rightarrow}$ is a cycle in $\overset{(\text{relax})+}{\rightarrow} \cup \overset{(\text{safe})+}{\rightarrow} \cup \overset{(\text{relax})+}{\rightarrow} ; \overset{(\text{safe})+}{\rightarrow}$. Thus, a test violating *Sc* corresponds to a cycle in this second relation. But we do not generate cycles with more than one relaxation: if a test exhibits a relaxation, we cannot decide which one is responsible. A cycle Rfe; DpsW; Fre determines whether $\overset{\text{rfe}}{\rightarrow}$ is global or not since $\overset{\text{dp}}{\rightarrow}$ and $\overset{\text{fr}}{\rightarrow}$ are safe, whereas Rfe; DpsW; Rfi; Fre is inconclusive as it exercises both $\overset{\text{rfe}}{\rightarrow}$ and $\overset{\text{rfi}}{\rightarrow}$. We also generate the cycles exhibiting no relaxation (in $\overset{(\text{safe})+}{\rightarrow}$), to test our assumption $\overset{\text{safe}}{\rightarrow}$ is global.

5.2 Code generation

diy interprets a sequence of edges as a cycle from which it computes a litmus test or fails. One of its execution witnesses includes a cycle compliant with the input edges sequence. The final condition is a conjunction of equalities over the final state of registers and memory locations characterising this cycle.

Test generation performs the following successive steps.

1. We map the edges sequence to a circular double-linked list, whose cells represent memory events, with direction, location, and value fields. An additional field records the edge starting from the event. This list represents the *input cycle* and appear in at least one of the execution witnesses of the produced test.
2. A linear scan sets the events directions, by comparing each target direction with the following source direction. When equal, the in-between cell direction is set to the common value; otherwise (*e.g.* Rfe; Rfe), the generation fails.
3. If no event e has an incoming edge specifying a location change, generation fails. Otherwise, a linear scan starting from e sets the locations. We reject the cycles where e and its predecessor have different locations (*e.g.* Rfe; PodRW).
4. We cut the input cycle into maximal sequences of events with the same location, each being scanned $w.r.t.$ the cycle order: we give the value 1 to the first write in this sequence, 2 to the second one, *etc.* Thus the values also reflect the write serialisation order for the specified location.

5. *Significant reads* are the sources of $\overset{\text{fr}}{\rightarrow}$ edges and the targets of $\overset{\text{rf}}{\rightarrow}$ edges. They are associated with the write on the other side of the edge. In the $\overset{\text{rf}}{\rightarrow}$ case, the value of the read is the one of its associated write. In the $\overset{\text{fr}}{\rightarrow}$ case, the value of the read is the value of the predecessor of its associated write in $\overset{\text{ws}}{\rightarrow}$, *i.e.* by construction the value of its associated write minus 1. Non significant reads do not appear in the test condition.
6. We cut the cycle into maximal sequences of events from the same processor, each being scanned, generating load instructions to (resp. stores from) fresh registers for reads (resp. writes). We insert code implementing a dependency in front of events targeting $\overset{\text{dp}}{\rightarrow}$ and the appropriate barrier instruction for events targeting $\overset{\text{fenced}}{\rightarrow}$ edges. We built initial state at this step: stores and loads take their addresses from fresh registers whose content (a memory location) is defined in the initial state. Part of the final condition is also built: for any significant read with value v resulting in a load instruction to register r, we add the equality $r = v$.
7. We complete the final condition to characterise write serialisations. The write serialisation for a given location x is defined by the sequence of values 0 (initial value of x), ..., n, where n is the last value allocated for location x at step 4. If n is 0 or 1 then no addition to the final condition needs to be performed, because the write serialisation is either a singleton or a pair. If n is 2, we add the equality $x = 2$. Otherwise ($n > 2$), we add an *observer* to the program, *i.e.* we add a thread performing n loads from x to r_1, ..., r_n and add the equalities $r_1 = 1 \wedge \ldots \wedge r_n = n$ to the final condition.

References

1. L. Lamport. How to Make a Correct Multiprocess Program Execute Correctly on a Multiprocessor. *IEEE Trans. Comput.*, 46(7):779–782, 1979.
2. S. Owens, S. Sarkar, and P. Sewell. A Better x86 Memory Model: x86-TSO. In *TPHOL 2009*.
3. Sparc Architecture Manual Versions 8 and 9, 1992 and 1994.