A Framework for the Investigation of Shared Memory Systems[1]

Bart Van Assche **Erik H. D'Hollander**
Dept. of Electrical Engineering
University of Ghent
B-9000 Gent, Belgium

Abstract

The behavior of programs running on a shared memory computer system is defined by the **memory consistency model** of its architecture. Since the consistency model does not only define behavior but inherently limits the efficiency of the memory system, research on memory models is still ongoing. There are several difficulties with the formulation of existing memory models. Different models use a different notation for operations with the same semantics, and similar synchronization primitives have different semantics. Also, there is no distinction between properties that hold for all memory models and properties that are model-specific. We propose a formalism that uses the same notation for all memory models, that has a unified set of synchronization primitives and that clearly separates general and model-specific properties. This facilitates the classification and comparison of different consistency models.

1 Introduction

A shared memory system has a set of locations that can be used by all connected processors to store values into. Every location in the shared memory has an address that is known to all participating processors, and every processor has its own interface to the shared memory. This interface is used to issue load, store and synchronization operations to the shared memory. The memory model of a shared memory system determines the relative order of shared memory operations and hence how a parallel program will be executed.

A shared memory can be either centralized or distributed, and in both cases caches may be used. When a shared memory system is physically distributed, the memory model defines how to keep the cache memories consistent. The generic structure of a shared memory is shown in figure 1.

When a memory model is defined in terms of actions known to the programmer, it is called programmer-centric. If the memory model is defined in terms of implementation-specific actions, it is called hardware-centric. In this paper we use the programmer-centric approach.

The designer of a shared memory system faces two opposite requirements: make every processor run as fast as possible, and make the exchange of data between processors also as fast as possible. A partial solution for this problem is to allow reordering of load and store operations, and by asking the programmer to supply explicit reordering information. This information is supplied as a label on every load- and store-operation.

Memory models have more applications than only their use in multiprocessors. They are also applicable to multithreaded programs, programs using distributed shared memory (DSM-) systems on a network of workstations, distributed databases and network filesystems. All these systems store data, and have more than one interface along which it is possible to modify the data.

2 Framework

Every processor runs one thread, or actually a uniprocessor program associated with that thread. The different uniprocessor programs together are the explicitly parallel program. While running a thread, a processor issues read, write and synchronization operations to the shared memory. These read, write and synchronization operations are the only interaction of the processor with the shared memory. The sequence of operations that results from running a thread is called the **execution** of that thread. Since the order of the operations in the execution is derived from the order of instructions in the uniprocessor program running, we call this order the **program order** of the operations issued by the corresponding processor.

In a shared memory system without local memories, every processor has the same view of the shared memory. When duplicating or caching the contents of the shared memory in local memories, however, every processor potentially can have a different view of the shared memory. We model a processor's own view of the shared memory by specifying the order in which changes have been applied to that view. These changes are the memory operations of *all* processors, and their observed order is called the **memory order** relation. This relation is a partial order, and hence able to model concurrent operations.

In order to allow the programmer to control reordering of load and store operations, we will define the fol-

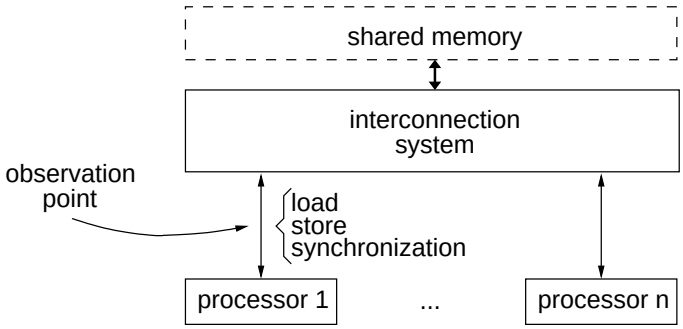

Figure 1: Generic structure of a shared memory system: n processors, a shared memory and in-between an interconnection system. The interconnection system can reorder requests and can cause arbitrary delays when propagating requests between a processor and the memory. The shared memory itself is either physical or virtual. The point where the order of load, store and synchronization operations for a given processor is observed is indicated on the figure.

lowing four labels: weak or ordinary, acquire, release and strong. Where applicable, we will use the following kinds of synchronization operations: lock, unlock and barrier operations. The precise semantics of these operations will be defined in sections 4 and 5.

3 Notation and Definitions

An operation is of one of the following kinds: a load, store, load-modify-store, lock, unlock or barrier operation. These types are abbreviated as resp. $l, s, f, \mathit{lock}, \mathit{unlock}$ or bar. Store, load and load-modify-store operations have an label λ that specifies allowed reordering: one of weak_L, acq_L, rel_L or strong_L.

The symbol $n \in \mathbb{N}$ stands for the **number of processors** connected to the memory system, and $p \in P$ for a **processor number**. $i \in \mathbb{N}$ is the per processor **operation number**, instructions executed later in program order having a higher operation number. $j \in \mathbb{N}$ is an **identification number** for synchronization operations, indicating which operations are associated. An entity in the memory that can be addressed is called a **location**, and *Mem* is the set of all locations. $m \in \mathit{Mem}$ is a **single memory location**, and $M \subset \mathit{Mem}$ is a **set of memory locations**, indicating which set of locations is involved in an operation. For loads, stores and load-modify-stores this is typically a singleton, and for synchronization instructions this set M is a non-empty subset of *Mem*.

The load-modify-store operation models e.g. fetch-and-increment and test-and-set with the operations $f(m, v_l, v_l + 1)$ and $f(m, v_l, 1)$ respectively. v_l is the value that was stored in location m before the operation started.

The functions $\mathit{lbl}()$, $\mathit{num}()$, $\mathit{proc}()$, $\mathit{mem}()$, $\mathit{val}_l()$ and $\mathit{val}_s()$ return the values λ, i, p, M, v_l and v_s respectively. The sets of operations of types $s, l, f, \mathit{lock}, \mathit{unlock}$ and *bar* are called respectively S, L, F, *Lock*, *Unlock*, and *Bar*. We further define the following derived sets: synchronization operations $\mathit{Sync} = \mathit{Lock} \cup \mathit{Unlock} \cup \mathit{Bar}$, operations on processor p $\mathit{Op}_p = \{op \in \mathit{Op} | \mathit{proc}(op) = p\}$, ordinary operations $\mathit{Op}_{\mathit{ord}} = \{op \in \mathit{Op} | \mathit{lbl}(op) \in \{\text{weak}_\text{L}\}\}$, special operations $\mathit{Op}_s = \{op \in \mathit{Op} | \mathit{lbl}(op) \in \{\mathit{acq}_L, \mathit{rel}_L, \text{strong}_\text{L}\}\}$, and operations specific to location m: $\mathit{Op}_m = \{op \in \mathit{Op} | m \in \mathit{mem}(op)\}$, $L_m = \mathit{Op}_m \cap L$, $S_m = \mathit{Op}_m \cap S$, $F_m = \mathit{Op}_m \cap F$.

The notation $\xrightarrow{\text{po}\,p}$ stands for the program order relation of processor p, and $\xrightarrow{\text{po}}$ for the union of these relations. The per-processor **program order** is a total order relation and orders the operations of that processor by increasing operation sequence-number. Every processor p observes the operations of all processors in the order specified by the **memory order** or $\xrightarrow{\text{mo}\,p}$.

4 Common properties

There are five properties common to all memory models: uniprocessor correctness, order of special accesses, value condition, liveness condition and location consistency.

If the data-dependent operations executed on processor p are observed by that processor in program-order, that processor obeys the **uniprocessor correctness** property. Formally:

$$\forall\, m \in \mathit{Mem} :\ \forall\, (op_1, op_2) \in (\mathit{Op}_m^2 \setminus L_m^2) :\ \forall\, p \in P :\\ op_1 \xrightarrow{\text{po}\,p} op_2 \implies op_1 \xrightarrow{\text{mo}\,p} op_2 \tag{1}$$

Labels further specify which part of the program order must be observed by all processors: the order of operations executed after an acquire, before a release, or before or after an operation labeled strong must be preserved.

We call this condition the **order of special accesses**:

$$\begin{array}{l}\forall\,(op_1, op_2) \in \mathit{Op}^2 :\ \forall\, p \in P :\\ \quad op_1 \xrightarrow{\text{po}\,p} op_2 \wedge \text{in-order}(op_1, op_2) \Longrightarrow op_1 \xrightarrow{\text{mo}\,p} op_2,\\ \text{with}\\ \quad \text{in-order}(op_1, op_2) = (op_1 \in \mathit{Sync} \vee op_2 \in \mathit{Sync}\\ \vee lbl(op_1) \in \{acq_L, \text{strong}_\text{L}\} \vee lbl(op_2) \in \{rel_L, \text{strong}_\text{L}\}).\end{array} \tag{2}$$

This condition is trivially fulfilled when all accesses are ordinary accesses.

The result of a load operation is the value written to the same location by the store operation immediately preceding that load, where preceding is defined by the memory order relation $\xrightarrow{\text{mo}\,p}$ on the processor p where the load is performed. There are two degenerate cases: no store operations or more than one store operation immediately precede a load. The last case is also called a data race. In both cases the result of the load is undefined. This condition is called the **value condition**, and can be formalized as follows, with $l \in L \cup F$:

$$\begin{array}{rl} p & \triangleq\ proc(l)\\ \mathit{hist}(l) & \triangleq\ \{s \in S_{mem(l)} \cup F_{mem(l)} | s \neq l \wedge s \xrightarrow{\text{mo}\,p} l\}\\ prev(l) & \triangleq\ \{s \in hist(l) | \forall\, s' \in hist(l) :\\ & \qquad\qquad s = s' \vee \neg(s \xrightarrow{\text{mo}\,p} s')\}\\ \forall\, s_1 \in prev(l) :\ & (\forall\, s_2 \in prev(l) :\ s_1 = s_2)\\ & \Longrightarrow val_l(l) = val_s(s_1)\end{array} \tag{3}$$

We assume further that every value that is written to the memory, eventually will be observed by every other processor. We call this the **liveness condition**:

$$\forall\, p, q \in P :\ \forall\, s \in S_q :\ \{op | op \xrightarrow{\text{mo}\,p} s\} \text{ is a finite set.} \tag{4}$$

While the view of operations of different processors on different memory locations can vary from processor to processor, all processors must have the same view of the order of operations on one single memory location. We call this condition the **location consistency** condition:

$$\forall\, m \in \mathit{Mem} :\ \xrightarrow{\text{mo}\,1} \ldots \xrightarrow{\text{mo}\,n} \text{ are consistent over } \mathit{Op}_m \tag{5}$$

For the definition of relation consistency, see the appendix.

A designer of a memory model can choose not to follow conditions 1, 3, 4 and 5. We only know of one memory model that does not obey condition 5: the PRAM memory model [12, 8]. There are several software implementations of distributed shared memory that do not satisfy the liveness condition, especially those who implement lazy release consistency [5].

5 Synchronization

There are three types of synchronizing operations: barrier, lock and unlock.

A barrier operation b on processor p orders operations executed on the same processor in the same way in other memory order relations. The set of barrier operations with the same identification number $id(b)$ is a single **barrier**.

A **critical section** has an identification number j and protects the locations of the set M by sequentializing other critical sections to any of these locations M. A critical section on processor p starts with an *lock* operation $(\mathit{lock}, i_1, p, M, j)$ and ends with a *unlock* operation $(\mathit{unlock}, i_2, p, M, j)$. Critical sections sharing at least one memory location cannot overlap. Operations executed inside the critical section are for every memory order relation ordered between the lock and the unlock.

Before we start the formalization of barrier and critical section synchronization, we define the equivalence relations N and N'. These relations partition the barrier operations and the lock/unlock pairs, based on the identification number $id()$. The relation N is extended to a reflexive relation over the set *Op*, and the relation N' has been extended with unpaired lock operations.

$$\begin{array}{rl} N & \triangleq\ \{(op_1, op_2) \in \mathit{Bar}^2 | id(op_1) = id(op_2)\}\\ & \quad \cup \{(op, op) | op \in \mathit{Op}\}\\ N' & \triangleq\ \{(op_1, op_2) \in (\mathit{Lock} \cup \mathit{Unlock})^2 |\\ & \qquad id(op_1) = id(op_2)\}\end{array}$$

The expression below formalizes that barriers are totally ordered per memory location, and that all synchronization operations on a common memory location have the same ordering in all memory order relations. Also, we require that lock-unlock pairs are totally ordered per memory location and neither overlap nor nest. For a definition of the quotient-operator /, see the appendix.

$$\begin{array}{l}\forall\, m \in \mathit{Mem} :\ \xrightarrow{\text{mo}\,1} / N \ldots \xrightarrow{\text{mo}\,n} / N \text{ s.c. in } \mathit{Sync}_m / N\\ \wedge\, \forall\, m \in \mathit{Mem} :\ \xrightarrow{\text{mo}\,1} / N' \ldots \xrightarrow{\text{mo}\,n} / N' \text{ s.c. in } \mathit{Sync}_m / N'\end{array} \tag{6}$$

Lock and unlock Every unlock requires exactly one matching lock, but a lock is allowed to have no matching unlock. In that case, the unlock is considered to be past the end of the execution of the thread on the processor of the lock. Further, matching lock and unlock operations have to reference the same set of memory locations and the lock must be ordered before the unlock in the program order relation. Also, any operation ordered by program order between the lock and the unlock is for every processor also ordered by memory order between the lock and the unlock – see also the formalization in equation 7.

The lock and unlock operations have similar functionality to the corresponding operations available with

$$
\begin{array}{l}
\forall\, l \in \mathit{Lock} :\ \forall\, u_1, u_2 \in \mathit{Unlock} :\ lN'u_1 \wedge lN'u_2 \Longrightarrow u_1 = u_2 \\
\wedge\ \forall\, u \in \mathit{Unlock} :\ \exists!\, l \in \mathit{Lock} :\ lN'u \\
\wedge\ \forall\, l \in \mathit{Lock} :\ \forall\, u \in \mathit{Unlock} :\ lN'u \Longrightarrow mem(l) = mem(u) \\
\wedge\ \forall\, l \in \mathit{Lock} :\ \forall\, u \in \mathit{Unlock} :\ lN'u \Longrightarrow l \xrightarrow{\text{po}} u \\
\wedge\ \forall\, l \in \mathit{Lock} :\ \forall\, op \in \mathit{Op}_{mem(l)} :\ (l \xrightarrow{\text{po}} op \Longrightarrow \forall\, p \in P :\ l \xrightarrow{\text{mo}\,p} op) \\
\wedge\ \forall\, u \in \mathit{Unlock} :\ \forall\, op \in \mathit{Op}_{mem(u)} :\ (op \xrightarrow{\text{po}} u \Longrightarrow \forall\, p \in P :\ op \xrightarrow{\text{mo}\,p} u)
\end{array}
\tag{7}
$$

Table 1: Conditions on lock- and unlock-operations.

POSIX threads. Typically these operations are not primitive operations, but implemented using load and store operations or with message communication.

Barriers There are two conditions specific for barriers: any operation of a barrier has to reference the same set of memory locations, and all instructions executed before a barrier are observed before that barrier by any processor, while all instructions executed after a barrier are observed after that barrier.

$$
\begin{array}{l}
\forall\, b_1, b_2 \in \mathit{Bar} :\ b_1 N b_2 \Longrightarrow mem(b_1) = mem(b_2) \\
\wedge\ \forall\, b_1, b_2 \in \mathit{Bar} :\ \forall\, p \in P :\ \forall\, op \in \mathit{Op}_{mem(b_1)} \setminus \{b_1\} : \\
\quad b_1 N b_2 \Longrightarrow (op \xrightarrow{\text{po}} b_1 \Longrightarrow op \xrightarrow{\text{mo}\,p} b_2) \\
\qquad\qquad \wedge (b_1 \xrightarrow{\text{po}} op \Longrightarrow b_2 \xrightarrow{\text{mo}\,p} op)
\end{array}
\tag{8}
$$

There are two main ways uses for this barrier operation. The first way is to implement a fence instruction, e.g. STBAR in the Sparc architecture model [15]. This instruction is equivalent with the single operation $(\mathit{bar}, i, \mathit{Mem}, j, p)$ where j is a different integer for every invocation of the STBAR instruction. The second way for using barrier operations is to implement the behavior of e.g. TreadMark's Tmk_barrier: replace the invocation on processor p with $(\mathit{bar}, i_p, Mem, j, p)$. In this case j is an integer that identifies the barrier synchronization point over all processes. These examples illustrate that a single barrier operation can correspond either to one instruction or even to a function call.

We call the conditions 1 to 8 the **general memory model conditions**.

6 Memory Models

A **memory model** *Mod* constraints a set of executions $E = (\mathit{Op}, \xrightarrow{\text{po}})$ in terms of partial order relations $\xrightarrow{\text{mo}\,1} \ldots \xrightarrow{\text{mo}\,n}$, such that Op, $\xrightarrow{\text{po}}$, $\xrightarrow{\text{mo}\,1} \ldots \xrightarrow{\text{mo}\,n}$ satisfy the general conditions 1 – 8 and also the model-specific conditions.

Two memory models Mod_1 and Mod_2 are **equivalent** if the corresponding sets of executions are equal. A memory model Mod_1 is **stronger** than a memory model Mod_2 if Mod_1 and Mod_2 are not equivalent and all executions possible under the first model are possible under the second model.

7 Examples of Memory Model Definitions

In this section we will define several existing memory models formally. Every definition will consist of two parts: first a definition that is equivalent to the original model, and next a definition that extends the original model with the synchronization operations introduced in this paper. The extended definitions will allow easy comparison of the different memory models.

7.1 Sequential consistency

The sequential consistency memory model was defined by Lamport in 1979. It was the first accurate description of how a multiprocessor with shared memory should behave with respect to load and store operations. Lamport defined sequential consistency as follows:

> *... the result of an 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 [13].*

See table 2 for a formalization.

Sequential consistency can be extended with synchronization operations. When introducing barriers, it is necessary to group the separate barrier operations to barriers via the relation N (see section 5). Instead of the relations $\xrightarrow{\text{mo}\,p}$ and the set Op, we now use the relations $\xrightarrow{\text{mo}\,p}/N$ and the set Op/N. Extended sequential consistency, or **SC*** , is the memory model where the general conditions hold, and also
$\forall\, p \in P :\ \xrightarrow{\text{po}\,p} \subset \xrightarrow{\text{mo}\,p}$
and $\xrightarrow{\text{mo}\,1}/N, \ldots, \xrightarrow{\text{mo}\,n}/N$ s.c. in Op/N.

7.2 Processor Consistency

There exist at least two different memory models that both have the name processor consistency [3, 9]. We

use Ahamad's definition, based on Goodman's earlier definition:

A multiprocessor is said to be processor consistent if the result of any execution is the same as if the operations of each individual processor appear in the sequential order specified by its program. Thus the order in which writes from two processors occur, as observed by themselves or a third processor, need not be identical, but writes issuing from any processor may not be observed in any order other than in which they are issued [11]. Also, there must be a unique view of the writes to each location, and each processor's writes must be perceived in the order in which it invokes them. Furthermore, these two requirements must be mutually consistent [3].

The formalization of the processor consistency or **PC** memory model is in table 3.

Again it is meaningful to extend the original definition of PC with synchronization operations. Extended processor consistency **PC*** is the memory model where the general conditions hold, and also $\forall\, p \in P:\ \xrightarrow{\text{po}\,p} \subset \xrightarrow{\text{mo}\,p}$, $\forall\, m \in \mathit{Mem}:\ \xrightarrow{\text{mo}\,1}/N, \ldots, \xrightarrow{\text{mo}\,n}/N$ s.c. in $(S_m \cup F_m)/N$, and $\forall\, p \in P:\ \left(\xrightarrow{\text{po}} \cap\, (S \cup F)^2\right) \subset \xrightarrow{\text{mo}\,p}$.

7.3 Partial Store Ordering

PSO is the memory model with load, store, atomic load-modify store and memory barrier operations such that the following of table 4 hold.

We can extend the PSO memory model to the model PSO* with more general synchronization instructions: **PSO*** is the memory model where conditions 1 to 8 hold, and also for all $m \in \mathit{Mem}$ and $p \in P$, one has $\xrightarrow{\text{mo}\,1}/N, \ldots, \xrightarrow{\text{mo}\,n}/N$ s.c. in $(\mathit{Op} \setminus L)/N$, $\xrightarrow{\text{mo}\,1}/N \ldots \xrightarrow{\text{mo}\,n}/N$ s.c. in Op_m/N, $\xrightarrow{\text{po}} \cap\, \mathit{Op}_m^2 \subset \xrightarrow{\text{mo}\,p}$, and $\left(\xrightarrow{\text{po}} \cap\, ((L \cup F) \times \mathit{Op})\right) \subset \xrightarrow{\text{mo}\,p}$. For a proof of the equivalence of the PSO and PSO*, see the appendix.

7.4 Release Consistency with sequentially consistent special instructions

The original definition of RCsc considers load and store operations with all four label types, but no synchronization operations:

> *Before an ordinary load or store access is allowed to perform with respect to any processor, all previous acquire accesses must be performed. Before a release access is allowed to perform with respect to any other processor, all previous ordinary load and store accesses must be performed. Special accesses [nsync, acquire and release] are sequentially consistent with respect to one another [9].*

In this definition, every acquire and release operates on one memory location. Gharachorloo's ordinary and nsync labels correspond to our weak_L and strong_L labels respectively – see also table 5.

We can now extend the definition of RCsc into **RCsc*** : extended RCsc is the memory model where conditions 1 to 8 hold, and also $\xrightarrow{\text{mo}\,1}/N, \ldots, \xrightarrow{\text{mo}\,n}/N$ s.c. in Op_s/N.

8 Related Work

Several authors have formalized memory models before. Collier uses architecture rules to define memory models for load and store operations, where every store operation is divided into one store operation for every participating processor [6]. Memory models or architectures are built with rules ordering pairs of memory operations. Gharachorloo's approach [10] divides every store operation into $n + 1$ sub-operations: locally a s_{init} operation and n s_i operations, with $1 \leq i \leq n$. Condon [7] introduced two new ideas for hardware-centric models: instead of dividing stores into n sub-operations, a store is divided into a private and a public store operation. Further, Lamport clocks are used to enforce partial order relations on memory operations in an efficient way. The formalisms of Collier, Gharachorloo and Condon are hardware-centric, and suited for hardware designers.

Attiya [4] defines a framework for defining memory models starting from a program instead of starting from a program execution. This has the advantage that control dependencies are visible inside the framework. Using this framework memory models for sequential consistency, weak ordering and hybrid consistency are defined. These definitions support both weak and strong load and store operations. In contrast with our framework, Attiya defines no synchronization operations.

The framework that has been developed in this paper is well suited for treatment with an automatic theorem prover. For a translation to the specification language VDM, the Vienna Development Method, see [14].

For an overview of the many existing memory models, see the survey papers by Adve e.a. [1, 2].

9 Conclusion

In the design of a multiprocessor with or without physically shared memory, much effort is spent in optimizing the memory subsystem. This involves specifying the memory model of the designed system, and implementing the consistency protocol as efficiently as possible. We succeeded in defining a set of operations, including cooperative synchronization, yielding a framework for specifying programmer-centric memory models. In these specifications general and memory-model specific properties are clearly separated. The framework enables

$Op = Op_{ord}$	Only ordinary load and store operations are allowed.
$\forall\, op \in Op :\ \#mem(op) = 1$	Load and store operations refer only one location.
$\forall\, p \in P :\ \xrightarrow{\text{po}\,p} \subset \xrightarrow{\text{mo}\,p}$	Operations of an individual processor appear in program order.
$\xrightarrow{\text{mo}\,1} = \ldots = \xrightarrow{\text{mo}\,n}$	All memory order relations are identical,
$\xrightarrow{\text{mo}\,1}$ is a total order in Op	and they are totally ordered.
cond. 1	Uniprocessor correctness holds.
cond. 2, 3, 4, 5	Other conditions hold (implicit in textual definition).

Table 2: Lamport's definition of sequential consistency reformulated.

$Op \subset L_{ord} \cup S_{ord}$	Only ordinary load and store operations are allowed.
$\forall\, op \in Op :\ \#mem(op) = 1$	Load and store operations refer only one location.
$\forall\, p \in P :\ \xrightarrow{\text{po}\,p} \subset \xrightarrow{\text{mo}\,p}$	Operations of an individual processor appear in program order.
$\forall\, p \in P :\ \left(\xrightarrow{\text{po}} \cap\, (S \cup F)^2\right) \subset \xrightarrow{\text{mo}\,p}$	Same observation order of writes executed in p.o.
$\xrightarrow{\text{mo}\,1}, \ldots, \xrightarrow{\text{mo}\,n}$ s.c. in $S_m \cup F_m$	Same view of writes per location.
cond. 1	Required by textual definition of PC.
cond. 2, 3, 4, 5	Other conditions hold (implicit in textual definition).

Table 3: Definition of processor consistency reformulated.

Program Order: $;$ is a partial order in Op, such that $;$ is a valid program order relation.
Memory Order: $\leq$ is a partial order in Op, and $\leq$ is a total order in $Op \setminus L$.
Atomicity: a load-modify-store operation cannot be interrupted by an operation on another processor to the same location.
Termination: when a processor issues an infinite sequence of load operations to a location, and another processor issues a store to that location, then there exists a load of that sequence that observes the store.
Value: $hist_{SPARC,\,\leq}(l) \triangleq \{s \in S_{mem(l)} \cup F_{mem(l)} | s \neq l \wedge (s \leq l \vee s\,;\,l)\}$
$max_{SPARC,\,\leq}(l) \triangleq \{s \in hist_{SPARC,\,\leq}(l) | \forall s' \in hist_{SPARC,\,\leq}(l) : s = s' \vee \neg(s \leq s')\}$
$\forall\, s_1 \in \max_{\leq} :\ (l)(\forall\, s_2 \in \max_{\leq} :\ (l) s_1 = s_2) \Rightarrow val_l(l) = val_s(s_1)$.
LoadOp: $\forall\, l \in (L \cup F) :\ \forall\, op \in Op :\ l\,;\,op \Rightarrow l \leq op$.
StoreStore: $\forall\, op_1, op_2 \in Op \setminus L :\ \forall\, s \in Bar :\ op_1\,;\,s\,;\,op_2 \Rightarrow op_1 \leq op_2$.
StoreStoreEq: $\forall\, m \in Mem :\ \forall\, op_1, op_2 \in Op_m \setminus L_m :\ op_1\,;\,op_2 \Rightarrow op_1 \leq op_2$.

Table 4: Original definition of the PSO memory model [15].

$Op = L \cup S$	All kinds of load and store operations are allowed.
$\forall\, op \in Op :\ \#mem(op) = 1$	Load and store operations refer only one location.
$\xrightarrow{\text{mo}\,1}/N, \ldots, \xrightarrow{\text{mo}\,n}/N$ are s.c. in Op_s/N	Special accesses are sequentially consistent.
cond. 1, 2, 3, 4, 5	Other conditions specified in [9].

Table 5: Gharachorloo's definition of release consistency with sequentially consistent special accesses reformulated.

an precise specification and eases comparison of memory models.

While our framework is oriented towards the programmer of a shared-memory system, it does not limit the hardware-designer in implementing optimizations.

References

[1] Sarita V. Adve and Kourosh Gharachorloo. Shared memory consistency models: A tutorial. *IEEE Computer*, 29(12):66–76, February 1996.

[2] Sarita V. Adve, S. V. Pai, and P. Ranganathan. Recent advances in memory consistency models for hardware shared-memory systems. *Proceedings of the IEEE*, 87(3):445–455, March 1999.

[3] Mustaque Ahamad, Rida A. Bazzi, Ranjit John, and Gil Neiger. The power of processor consistency. In *Proc. of the 5th ACM Symp. on Parallel Algorithms and Architectures (SPAA'93)*, 1993.

[4] H. Attiya, S. Chaudhuri, R. Friedman, and J. L. Welch. Shared memory consistency conditions for non-sequential execution: Definitions and programming strategies. *SIAM Journal on Computing*, 27(1):65–89, February 1998.

[5] John B. Carter, John K. Bennett, and Willy Zwaenepoel. Implementation and performance of Munin. In *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pages 152–164, October 1991.

[6] William W. Collier. *Reasoning about Parallel Architectures*. Prentice-Hall, 1992.

[7] A. Condon, M. Hill, M. Plakal, and D. Sorin. Using lamport clocks to reason about relaxed memory models. In *Proceedings of the 5th IEEE Symposium on High-Performance Computer Architecture (HPCA-5)*, January 1999.

[8] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proceedings of the 10th ACM Symposium on the Theory of Computing*, pages 114–118, May 1978.

[9] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. L. Henessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 15–26, June 1990.

[10] Kourosh Gharachorloo. *Memory Consistency Models for Shared-Memory Multiprocessors*. PhD thesis, Computer Systems Laboratory, Stanford University, December 1995.

[11] James R. Goodman. Cache consistency and sequential consistency. Technical Report 61, SCI Committee, March 1989.

[12] David J. Kuck. A survey of parallel machine organization and programming. *ACM Computing Surveys*, 9(1):29–59, March 1977.

[13] Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, 9:690–691, September 1979.

[14] Noemie Slaats, Bart Van Assche, and Albert Hoogewijs. *Shared Memory Synchronization*, chapter 4, pages 123–156. FACIT – Formal Approaches to Computing and Information Technology. Springer-Verlag, March 1998.

[15] SPARC International Inc. *The SPARC Architecture Manual: Version 8*, 1992.

Appendix

A Mathematical relations

A relation R between the sets A and B holds between elements $a \in A$ and $b \in B$, notation aRb, or does not hold, notation $\neg(aRb)$. With each relation R, a set $\{(a,b)|aRb\}$ is associated, also with the notation R.

A **partial order** (p.o.) relation is a relation that is reflexive, antisymmetric and transitive. A relation R is **total** in A if $\forall\, a_1, a_2 \in A : \; a_1 R a_2 \vee a_2 R a_1 \vee a_1 = a_2$. A **total order** (t.o.) relation is a partial order that is also total. In this paper all partial and total order relations are reflexive, unless stated otherwise.

An equivalence relation Q in a set A partitions that set. The **partition** will be written as A/Q, with $A/Q = \{S[a]|a \in A\}$, and where $S[a]$ is the **equivalence class** of $a \in A$, defined by $S[a] = \{a' \in A|aQa'\}$.

The **quotient relation** R/Q is defined by $R/Q = \{(S_1[a_1], S_2[a_2])|a_1 R a_2\}$.

The relations $R_1 \ldots R_n$ are **consistent** over the set A if and only if the relations $R_1 \ldots R_n$ are identical when restricted to A: $R_1 \cap A^2 = \ldots = R_n \cap A^2$.

Relations $R_1 \ldots R_n$ are **sequentially consistent (s.c.) relations** over the set A if and only if $R_1 \ldots R_n$ are consistent and $R_1 \ldots R_n$ are total order relations over A.

B Equivalence proof of PSO and PSO*

See http://www.elis.rug.ac.be/~bvassche.

Proofs

Relations

Further definitions The *domain* of a relation R is the set of elements for which there exists at least one other element such that R is true:

$$dom(R) \triangleq \{a | \exists b : \ aRb \vee bRa\}.$$

Two relations R_1 and R_2 *conflict* if and only if

$$\exists a, b : \ aR_1b \wedge bR_2a.$$

Two relations R_1 and R_2 are *disjoint* if and only if there domains are disjoint.

The *composition* $(R_1; R_2)$ of two relations R_1 and R_2 is defined by

$$\forall a_1, a_3 : \ a_1(R_1; R_2)a_3 \iff \exists a_2 : \ a_1R_1a_2 \wedge a_2R_2a_3.$$

A relation R raised to the *power* $n \in \mathbb{N}$ is defined recursively as follows:

$$\begin{array}{l} R^0 = \{(a, a) | a \in dom(R)\} \\ \forall\, n \in \mathbb{N} : \ R^{n+1} = (R^n; R). \end{array}$$

The *reflexive and transitive closure* $R^\star$ of the relation R is defined by

$$R^\star \triangleq \bigcup_{n=0}^{\infty} R^n.$$

If the relation R^- is a *transitive reduction* of the relation R, it satisfies $\{R^-\}^\star = R^\star$. If R is reflexive and transitive, then $\{R^-\}^\star = R$ holds.

Lemma B.1 *If R is a p.o. and if relation A is a subrelation of R, then $(R \cap A)^\star$ is a p.o. and it also is a subrelation of R.*

Proof

With the definition $R_1 \triangleq R \cap A$, it follows that $\forall\, n \in \mathbb{N} : \ R_1^n \subset R^n$. Using the definition of transitive closure, one has that that $R_1^\star \subset R^\star$. Since R is a p.o., $R^\star = R$, which implies that $R_1^\star \subset R$. $R_1^\star$ is a p.o. by definition.

Lemma B.2 *If R is a p.o., A and B are subrelations of R, then $((R \cap A) \cup (R \cap B))^\star$ is a p.o. and a subrelation of R.*

Proof

Apply lemma B.1 to relation R and to the set $A \cup B$.

Lemma B.3 *If R_1 and R_2 are disjoint p.o. relations, then $R = R_1 \cup R_2$ is also a p.o.*

Proof

With the definitions $A_1 \triangleq dom(R_1)$, and $A_2 \triangleq dom(R_2)$, sets A_1 and A_2 are disjoint. This implies that $R_1; R_2 = \{\}$ and also that $R_2; R_1 = \{\}$, which leads to

$$\begin{array}{l} R^2 \\ = (R_1 \cup R_2; R_1 \cup R_2) \\ = (R_1; R_1) \cup (R_1; R_2) \cup (R_2; R_1) \cup (R_2; R_2) \\ = R_1 \cup \{\} \cup \{\} \cup R_2 \\ = R. \end{array}$$

It follows that $R^\star = R$ and hence that R is transitive. Since R_1 and R_2 are reflexive, R also is. R is antisymmetric since:

$$\begin{array}{l} op_1Rop_2 \wedge op_2Rop_1 \\ \Longrightarrow (op_1R_1op_2 \vee op_1R_2op_2) \\ \quad \wedge (op_2R_1op_1 \vee op_2R_2op_1) \\ \quad \wedge (op_1, op_2) \in (A_1^2 \cup A_2^2) \\ \Longrightarrow op_1R_1op_2 \wedge op_2R_1op_1 \vee op_1R_2op_2 \wedge op_2R_2op_1 \\ \Longrightarrow op_1 = op_2. \end{array}$$

Since R is reflexive, antisymmetric and transitive, R is a p.o.

Unless otherwise specified, in the remainder of this section the relation R_1 is a total order with domain A_1, the relation R_2 is a partial order with domain A_2, the set A is the union of the sets A_1 and A_2, and the relations R_1 and R_2 do not conflict. Further, R is defined as $R_1 \cup R_2$.

Lemma B.4 $\forall\, a_1, a_2, a_3, a_4 \in A :$
$a_1R_1a_2 \wedge a_2R_2a_3 \wedge a_3R_1a_4 \Longrightarrow a_1R_1a_4$

Proof

Using the totalness of R_1 and the transitivity of R_1 and R_2, one has:

$$\begin{array}{l} a_1R_1a_2 \wedge a_2R_2a_3 \wedge a_3R_1a_4 \\ \iff a_1R_1a_2 \wedge a_2R_2a_3 \\ \quad \wedge a_3R_1a_4 \wedge (a_2R_1a_3 \vee a_3R_1a_2) \\ \iff a_1R_1a_2 \wedge a_3R_1a_4 \\ \quad \wedge ((a_2R_2a_3 \wedge a_2R_1a_3) \vee (a_2R_2a_3 \wedge a_3R_1a_2)) \\ \iff a_1R_1a_2 \wedge a_3R_1a_4 \\ \quad \wedge (a_2 = a_3 \vee (a_2R_2a_3 \wedge a_3R_1a_2)) \\ \iff a_1R_1a_2 \wedge a_3R_1a_4 \\ \quad \wedge a_2R_2a_3 \wedge a_2R_1a_3 \\ \Longrightarrow a_1R_1a_4 \end{array}$$

Lemma B.5 $\exists\, a_2, a_3, a_4 \in A : \ a_1R_2a_2 \wedge a_2R_1a_3$
$\wedge a_3R_2a_4 \wedge a_4R_1a_5 \iff \exists\, a_2 \in A : \ a_1R_2a_2 \wedge a_2R_1a_5$

Proof

From left to right:

$$\begin{array}{l} \exists\, a_2, a_3, a_4 \in A : \ a_1R_2a_2 \wedge a_2R_1a_3 \\ \qquad\qquad\qquad\quad \wedge a_3R_2a_4 \wedge a_4R_1a_5 \\ \Longrightarrow \exists\, a_2 \in A : \ a_1R_2a_2 \wedge a_2R_1a_5 \end{array}$$

From right to left:

$$\begin{array}{l} \exists\, a_2 \in A : \ a_1R_2a_2 \wedge a_2R_1a_5 \\ \Longrightarrow \exists\, a_2 \in A : \ a_1R_2a_2 \wedge a_2R_1a_2 \\ \qquad\qquad\quad \wedge a_2R_2a_2 \wedge a_2R_1a_5 \\ \Longrightarrow \exists\, a_2, a_3, a_4, a_5 \in A : \ a_1R_2a_2 \wedge a_2R_1a_2 \\ \qquad\qquad\qquad\qquad\quad \wedge a_2R_2a_2 \wedge a_2R_1a_5 \end{array}$$

Lemma B.6 $(\exists\, n \in \mathbb{N} : \ a_1R^na_5) \iff \exists\, a_2, a_3 \in A :$
$a_1R_2a_2 \wedge a_2R_1a_3 \wedge a_3R_2a_5.$

Proof

If the left-hand side holds for $n \in \mathbb{N}$, one has

$$
\begin{array}{l}
a_1 R^{n+1} a_5 \\
\Longleftrightarrow \exists\, a_4 \in A :\ a_1 R^n a_4 \wedge a_4 R a_5 \\
\Longleftrightarrow \exists\, a_2, a_3, a_4 \in A :\ a_1 R_2 a_2 \wedge a_2 R_1 a_3 \wedge a_3 R_2 a_4 \\
\qquad\qquad \wedge (a_4 R_1 a_5 \vee a_4 R_2 a_5) \\
\Longleftrightarrow \exists\, a_2, a_3, a_4 \in A :\ (a_1 R_2 a_2 \wedge a_2 R_1 a_3 \\
\qquad\qquad \wedge a_3 R_2 a_4 \wedge a_4 R_1 a_5) \\
\qquad\qquad \vee (a_1 R_2 a_2 \wedge a_2 R_1 a_3 \\
\qquad\qquad \wedge a_3 R_2 a_4 \wedge a_4 R_2 a_5) \\
\Longleftrightarrow \exists\, a_2, a_3, a_4 \in A :\ (a_1 R_2 a_2 \wedge a_2 R_1 a_3 \\
\qquad\qquad \wedge a_3 R_2 a_4 \wedge a_4 R_1 a_5) \\
\qquad \vee \exists\, a_2, a_3, a_4 \in A :\ (a_1 R_2 a_2 \wedge a_2 R_1 a_3 \\
\qquad\qquad \wedge a_3 R_2 a_4 \wedge a_4 R_2 a_5) \\
\Longrightarrow \exists\, a_2 \in A :\ a_1 R_2 a_2 \wedge a_2 R_1 a_5 \\
\qquad \vee \exists\, a_2, a_3 \in A :\ a_1 R_2 a_2 \wedge a_2 R_1 a_3 \wedge a_3 R_2 a_5 \\
\Longrightarrow \exists\, a_2, a_3 \in A :\ a_1 R_2 a_2 \wedge a_2 R_1 a_3 \wedge a_3 R_2 a_5
\end{array}
$$

From right to left:

$$
\begin{array}{l}
\exists\, a_2, a_3 \in A :\ a_1 R_2 a_2 \wedge a_2 R_1 a_3 \wedge a_3 R_2 a_5 \\
\Longrightarrow \exists\, a_2, a_3 \in A :\ a_1 R a_2 \wedge a_2 R a_3 \wedge a_3 R a_5 \\
\Longrightarrow a_1 R^3 a_5 \\
\Longrightarrow \exists\, n \in \mathbb{N} :\ a_1 R^n a_5
\end{array}
$$

Lemma B.7 *R is antisymmetric and reflexive*

Proof

Since R_1 and R_2 are reflexive, one has that R is also reflexive. Antisymmetry follows from:

$$
\begin{array}{l}
a_1 R a_2 \wedge a_2 R a_1 \\
\Longrightarrow a_1 (R_1 \cup R_2) a_2 \wedge a_2 (R_1 \cup R_2) a_1 \\
\Longrightarrow (a_1 R_1 a_2 \vee a_1 R_2 a_2) \wedge (a_2 R_1 a_1 \vee a_2 R_2 a_1) \\
\Longrightarrow (a_1 R_1 a_2 \wedge a_2 R_1 a_1) \vee (a_1 R_1 a_2 \wedge a_2 R_2 a_1) \\
\qquad \vee (a_1 R_2 a_2 \wedge a_2 R_1 a_1) \vee (a_1 R_2 a_2 \wedge a_2 R_2 a_1) \\
\Longrightarrow a_1 = a_2 \vee a_1 = a_2 \vee a_1 = a_2 \vee a_1 = a_2 \\
\Longrightarrow a_1 = a_2
\end{array}
$$

Lemma B.8 *R^n is antisymmetric and reflexive*

Proof

R^n is reflexive because R is reflexive. Antisymmetry follows from $a_1 R^n a_4 \wedge a_4 R^n a_1 \Longrightarrow a_1 = a_4$:

$$
\begin{array}{l}
a_1 R^n a_4 \wedge a_4 R^n a_1 \\
\Longrightarrow \exists\, a_2, a_3, a_5, a_6 \in A :\ a_1 R_2 a_2 \wedge a_2 R_1 a_3 \\
\qquad \wedge a_3 R_2 a_4 \wedge a_4 R_2 a_5 \wedge a_5 R_1 a_6 \wedge a_6 R_2 a_1 \\
\Longrightarrow \exists\, a_2, a_3, a_5, a_6 \in A :\ a_1 R_2 a_2 \wedge a_2 R_1 a_3 \\
\qquad \wedge a_3 R_2 a_4 \wedge a_4 R_2 a_5 \wedge a_5 R_1 a_6 \wedge a_6 R_2 a_1 \\
\qquad \wedge a_3 R_2 a_5 \wedge a_6 R_2 a_2 \\
\Longrightarrow \exists\, a_2, a_3, a_5, a_6 \in A :\ a_1 R_2 a_2 \wedge a_2 R_1 a_3 \\
\qquad \wedge a_3 R_2 a_4 \wedge a_4 R_2 a_5 \wedge a_5 R_1 a_6 \wedge a_6 R_2 a_1 \\
\qquad \wedge a_3 R_2 a_5 \wedge a_6 R_2 a_2 \\
\qquad \wedge a_2 (R_1; R_2) a_5 \wedge a_5 (R_1; R_2) a_2 \\
\Longrightarrow \exists\, a_2, a_3, a_5, a_6 \in A :\ a_1 R_2 a_2 \wedge a_2 R_1 a_3 \\
\qquad \wedge a_3 R_2 a_4 \wedge a_4 R_2 a_2 \wedge a_2 R_1 a_6 \wedge a_6 R_2 a_1 \\
\qquad \wedge a_3 R_2 a_2 \wedge a_6 R_2 a_2 \wedge a_2 = a_5 \\
\Longrightarrow \exists\, a_2, a_3, a_5, a_6 \in A :\ a_1 R_2 a_2 \wedge a_2 R_2 a_4 \\
\qquad \wedge a_4 R_2 a_2 \wedge a_2 R_2 a_1 \\
\qquad \wedge a_2 = a_5 = a_3 = a_6 = a_1 = a_4 \\
\Longrightarrow \exists\, a_2, a_3, a_5, a_6 \in A : \\
\qquad a_2 = a_5 = a_3 = a_6 = a_1 = a_4 \\
\Longrightarrow a_1 = a_4
\end{array}
$$

Lemma B.9 *$R^\star$ is a partial order.*

Proof

Due to lemma B.8, $R^\star$ is antisymmetric. By definition of transitive closure, $R^\star$ is reflexive and transitive.

Lemma B.10 *If relations $R_1 \ldots R_n$ are pairwise disjunct t.o. relations, if Q is a p.o. and if none of the R_j relations conflict with Q, then the relation $R \triangleq (Q \cup \bigcup_{1 \le j \le n} R_j)^\star$ is a p.o.*

Proof

With the definitions ($1 \le k \le n$, $0 \le l \le n$)

$$
\begin{array}{l}
U_k \triangleq (Q \cup R_k)^\star \\
S_l \triangleq \cup_{1 \le j \le l} U_j
\end{array}
$$

the following properties hold: U_k is a p.o. due to lemma B.9, $Q \subset U_k$ by definition of U_k, with $1 \le j \le k$, one has $U_j \subset S_k$ by definition of S_k and with $0 \le k < n$, one also has $S_k \subset S_{k+1}$, S_k does not conflict with R_{k+1}, S_k is a p.o. with $0 \le k \le n$. If $0 \le k < n$, one has $(S_k; U_{k+1}) \subset S_{k+1}$:

$$
\begin{array}{l}
a_1 (S_k; U_{k+1}) a_3 \\
\Longrightarrow \exists a_2 :\ a_1 S_k a_2 \wedge a_2 U_{k+1} a_3 \\
\Longrightarrow \exists a_2 :\ (\exists j :\ 1 \le j \le k \wedge a_1 U_j a_2) \wedge a_2 U_{k+1} a_3 \\
\Longrightarrow \exists a_2 :\ (\exists j :\ 1 \le j \le k \wedge a_1 U_j a_2 \wedge a_2 U_{k+1} a_3) \\
\Longrightarrow \exists a_2 :\ (\exists j :\ 1 \le j \le k \wedge a_1 S_{k+1} a_2 \wedge a_2 S_{k+1} a_3) \\
\Longrightarrow \exists a_2 :\ (\exists j :\ 1 \le j \le k \wedge a_1 S_{k+1} a_3) \\
\Longrightarrow a_1 S_{k+1} a_3
\end{array}
$$

The proof of $(U_{k+1}; S_k) \subset S_{k+1}$ is similar. S_k is transitive since $S_k = S_k^2$ holds and since this implies that $S_k = S_k^\star$:

• Induction Base

S_0 is transitive.

• Induction Hypothesis
$0 \leq k < n \wedge S_k^2 = S_k \Longrightarrow S_{k+1}^2 = S_{k+1}$
• Proof
Since $S_0 = \{\}$, the induction base holds. The proof of the induction hypothesis is as follows:

$$\begin{aligned}
&S_{k+1}^2 \\
&= (S_{k+1}; S_{k+1}) \\
&= (S_k \cup U_{k+1}); (S_k \cup U_{k+1}) \\
&= (S_k; S_k) \cup (S_k; U_{k+1}) \cup (U_{k+1}; S_k) \cup (U_{k+1}; U_{k+1}) \\
&= S_k \cup S_{k+1} \cup S_{k+1} \cup U_{k+1} \\
&= S_{k+1}.
\end{aligned}$$

□
This proves that S_k is transitive. Below follows the proof of the antisymmetry of S_k:
• Induction Base
S_0 is antisymmetric
• Induction Hypothesis
S_k is antisymmetric $\Longrightarrow$ S_{k+1} is antisymmetric, with $0 \leq k < n$.
• Proof

$$\begin{aligned}
&a_1 S_{k+1} a_2 \wedge a_2 S_{k+1} a_1 \\
&\Longrightarrow a_1 (S_k \cup U_{k+1}) a_2 \wedge a_2 (S_k \cup U_{k+1}) a_1 \\
&\Longrightarrow (a_1 S_k a_2 \vee a_1 U_{k+1} a_2) \wedge (a_2 S_k a_1 \vee a_2 U_{k+1} a_1) \\
&\Longrightarrow (a_1 S_k a_2 \wedge a_2 S_k a_1) \vee (a_1 S_k a_2 \wedge a_2 U_{k+1} a_1) \\
&\quad \vee (a_1 U_{k+1} a_2 \wedge a_2 S_k a_1) \vee (a_1 U_{k+1} a_2 \wedge a_2 U_{k+1} a_1)) \\
&\Longrightarrow a_1 = a_2 \\
&\quad \vee (a_1 S_k a_2 \wedge (a_2 Q a_1 \vee a_2 R_{k+1} a_1)) \\
&\quad \vee ((a_1 Q a_2 \vee a_1 R_{k+1} a_2) \wedge a_2 S_k a_1) \\
&\quad \vee ((a_1 Q a_2 \vee a_1 R_{k+1} a_2) \wedge (a_2 Q a_1 \vee a_2 R_{k+1} a_1)) \\
&\Longrightarrow a_1 = a_2.
\end{aligned}$$

□
Since S_k is reflexive by definition, and since it is antisymmetric and transitive, S_k is a p.o. From the definition of R one has $R = S_n$, hence R is also a p.o.

Lemma B.11 *If relation R_1 is a partial order in A and total in $B \subset A$, and if R_2 is a relation over $(A \setminus B) \times B$ such that $\forall\, a_1, a_2 \in A :\ a_1 R_2 a_2 \iff \neg a_1 R_1 a_2 \wedge \neg a_2 R_1 a_1$, then $(R_1 \cup R_2)^\star$ is a partial order in A and a total order in $A^2 \setminus (A \setminus B)^2$.*
Proof

1. $(R_1; R_2) \subset R_2$.
2. $(R_2; R_1; R_2) \subset (R_2; R_1)$.
3. $(R_1; R_2) \subset (R_1 \cup R_2)$.
4. $\forall\, n \in \mathbb{N} :\ n \geq 1 \Longrightarrow (R_1 \cup R_2)^n \subset R_1 \cup (R_2; R_1)$.
5. $R = R_1 \cup (R_2; R_1)$.
6. R is total in $A^2 \setminus (A \setminus B)^2$.
7. R is reflexive.
8. R is antisymmetric.
9. R is transitive.

1. With a_1 and $a_3 \in A$, the proof of $(R_1; R_2) \subset R_2$ is as follows:

$$\begin{aligned}
&a_1 (R_1; R_2) a_3 \\
&\Longrightarrow \exists\, a_2 \in A :\ a_1 R_1 a_2 \wedge a_2 R_2 a_3 \\
&\Longrightarrow \exists\, a_2 \in A :\ a_1 R_1 a_2 \wedge a_2 \notin B \wedge a_3 \in B \\
&\wedge \neg a_2 R_1 a_3 \wedge \neg a_3 R_1 a_2 \\
&\Longrightarrow \exists\, a_2 \in A :\ a_1 R_1 a_2 \wedge a_2 \notin B \wedge a_3 \in B \\
&\wedge \neg a_2 R_1 a_3 \wedge \neg a_3 R_1 a_2 \\
&\wedge (a_1 \notin B \vee a_1 R_1 a_3 \vee a_3 R_1 a_1) \\
&\Longrightarrow \exists\, a_2 \in A :\ a_1 R_1 a_2 \wedge a_2 \notin B \wedge a_3 \in B \\
&\wedge \neg a_2 R_1 a_3 \wedge \neg a_3 R_1 a_2 \\
&\wedge a_1 \notin B \\
&\Longrightarrow \exists\, a_2 \in A :\ a_1 R_1 a_2 \wedge a_2 \notin B \wedge a_3 \in B \\
&\wedge \neg a_2 R_1 a_3 \wedge \neg a_3 R_1 a_2 \\
&\wedge a_1 \notin B \wedge a_1 R_2 a_3 \\
&\Longrightarrow a_1 R_2 a_3
\end{aligned}$$

2. Proof of $(R_2; R_1; R_2) \subset (R_2; R_1)$:

$$\begin{aligned}
&a_1 (R_2; R_1; R_2) a_4 \\
&\Longrightarrow \exists\, a_2, a_3 \in A :\ a_1 R_2 a_2 \\
&\wedge a_2 R_1 a_3 \wedge a_3 R_2 a_4 \\
&\Longrightarrow \exists\, a_2, a_3 \in A :\ a_1 R_2 a_2 \\
&\wedge a_2 R_1 a_3 \wedge a_3 R_2 a_4 \\
&\wedge (a_2 R_1 a_4 \vee a_4 R_1 a_2) \\
&\Longrightarrow \exists\, a_2 \in A :\ a_1 R_2 a_2 \wedge a_2 R_1 a_4 \\
&\Longrightarrow a_1 (R_2; R_1) a_4
\end{aligned}$$

3. Proof of $(R_1; R_2) \subset (R_1 \cup R_2)$:

$$\begin{aligned}
&a_1 (R_1; R_2) a_3 \\
&\Longrightarrow \exists\, a_2 \in A :\ a_1 R_1 a_2 \wedge a_2 R_2 a_3 \\
&\Longrightarrow \exists\, a_2 \in A :\ a_1 R_1 a_2 \wedge a_2 R_2 a_3 \\
&\wedge (a_1 \notin B \wedge \neg a_1 R_1 a_3 \wedge \neg a_3 R_1 a_1 \\
&\vee a_1 R_1 a_3 \vee a_3 R_1 a_1) \\
&\Longrightarrow a_1 R_1 a_3 \vee a_1 R_2 a_3
\end{aligned}$$

4. Proof of $\forall\, n \in \mathbb{N} :\ n \geq 1 \Longrightarrow (R_1 \cup R_2)^n \subset R_1 \cup (R_2; R_1)$:
• Induction Base
$(R_1 \cup R_2)^1 = R_1 \cup R_2 = R_1 \cup R_2; R_1$
• Induction Hypothesis
$(R_1 \cup R_2)^n \subset R_1 \cup (R_2; R_1) \Longrightarrow (R_1 \cup R_2)^{n+1} \subset R_1 \cup (R_2; R_1)$
• Proof

$$\begin{aligned}
&(R_1 \cup R_2)^{n+1} \\
&= (R_1 \cup R_2)^n; (R_1 \cup R_2) \\
&\subset (R_1 \cup (R_2; R_1)); (R_1 \cup R_2) \\
&\subset (R_1; R_1) \cup (R_2; R_1; R_1) \\
&\cup (R_1; R_2) \cup (R_2; R_1; R_2) \\
&\subset R_1 \cup (R_2; R_1) \cup (R_1 \cup R_2) \cup (R_2; R_1) \\
&\subset R_1 \cup (R_2; R_1)
\end{aligned}$$

□

5. From the previous step and from the definition of transitive closure, one has $R \subset R_1 \cup (R_2; R_1)$. Since also $R_1 \cup (R_2; R_1) \subset (R_1 \cup R_2)^2 \subset R$, it follows that $R = R_1 \cup (R_2; R_1)$.

6. For every $a_1, a_3 \in A$ one has:

$$
\begin{array}{l}
a_1 R a_3 \vee a_3 R a_1 \\
\Longrightarrow (a_1 R_1 a_3 \vee \exists\, a_2 \in B :\ a_1 R_2 a_2 \wedge a_2 R_1 a_3) \\
\vee (a_3 R_1 a_1 \vee \exists\, a_4 \in B :\ a_3 R_2 a_4 \wedge a_4 R_1 a_1) \\
\Longrightarrow \exists\, a_2, a_4 \in B :\ a_1 R_1 a_3 \vee a_3 R_1 a_1 \\
\vee (a_3 R_2 a_4 \wedge a_4 R_1 a_1) \vee (a_1 R_2 a_2 \wedge a_2 R_1 a_3) \\
\Longrightarrow \exists\, a_2, a_4 \in B :\ (a_1, a_3) \in B^2 a_1 R_1 a_3 \vee a_3 R_1 a_1 \\
\vee ((a_3 R_2 a_4 \wedge a_4 R_1 a_1) \vee (a_1 R_2 a_2 \wedge a_2 R_1 a_3)) \\
\wedge (a_1 R_2 a_3 \vee a_3 R_2 a_1 \vee (a_1, a_3) \in (A \setminus B)^2) \\
\Longrightarrow \exists\, a_2, a_4 \in B :\ (a_1, a_3) \in B^2 a_1 R_1 a_3 \vee a_3 R_1 a_1 \\
\vee (a_3 R_2 a_4 \wedge a_4 R_1 a_1 \wedge a_3 R_2 a_1) \vee (a_1 R_2 a_2 \wedge a_2 R_1 a_3 \wedge a_1 R_2 a_3) \\
\Longrightarrow (a_1, a_3) \in B^2 \vee a_1 R_1 a_3 \vee a_3 R_1 a_1 \\
\vee a_3 R_2 a_1 \vee a_1 R_2 a_3 \\
\Longrightarrow (a_1, a_3) \in A^2 \setminus (A \setminus B)^2
\end{array}
$$

   This proves that R is total in $A^2 \setminus (A \setminus B)^2$.

7. R is reflexive by definition.

8. R is antisymmetric:

$$
\begin{array}{l}
a_1 R a_3 \wedge a_3 R a_1 \\
\Longrightarrow \exists\, a_2, a_4 \in A :\ (a_1 R_1 a_3 \\
\vee a_1 R_2 a_2 \wedge a_2 R_2 a_3) \\
\wedge (a_3 R_1 a_1 \vee a_3 R_2 a_4 \wedge a_4 R_1 a_1) \\
\Longrightarrow \exists\, a_2, a_4 \in A : \\
(a_1 R_1 a_3 \wedge a_3 R_2 a_4 \wedge a_4 R_1 a_1) \\
\vee (a_1 R_2 a_2 \wedge a_2 R_2 a_3 \wedge a_3 R_1 a_1) \\
\Longrightarrow \mathit{false}.
\end{array}
$$

9. R is transitive by definition.

Conclusion: $(R_1 \cup R_2)^\star$ is a partial order in A, and total in $A^2 \setminus (A \setminus B)^2$.

Equivalence proof of PSO and PSO*

The PSO memory model has the following restrictions: $\mathit{Op} \subset \mathit{L_{ord}} \cup \mathit{S_{ord}} \cup \mathit{F_{ord}} \cup \mathit{Bar}$, $\mathit{op} \in (L \cup S \cup F) \Longrightarrow \#mem(op) = 1$, $\mathit{op} \in \mathit{Bar} \Longrightarrow mem(op) = \mathit{Mem}$ and $\forall\, b_1, b_2 \in \mathit{Bar} :\ id(b_1) = id(b_2) \Longrightarrow b_1 = b_2$. The two proofs below are valid under the above assertions. Equivalence of the two memory models will be proven by mutual inclusion. In addition to the notation in the main text, the symbols LF_m and SF_m are synonyms for the sets $L_m \cup F_m$ and $S_m \cup F_m$ resp.

Inclusion of PSO in PSO* Suppose that the trace $T = (\mathit{Op}, \mathbf{;}, \leq)$ is a valid PSO trace. The relation $\mathbf{;}$ is by definition a valid program order relation $\xrightarrow{\text{po}}$. We further define the following relations:

$$
\begin{array}{rcl}
R_{1,m,p} & \triangleq & \leq \cap\, (\mathit{Op}_m^2 \setminus (L_{m,p} \times S_{m,p})) \\
R_{2,m} & \triangleq & \{(l, s) \in (L_m \times \mathit{SF}_m) \mid \neg l \leq s \wedge \neg s \leq l\} \\
R_{3,m,p} & \triangleq & (R_{1,m,p} \cup R_{2,m})^\star \\
R_4 & \triangleq & \leq \cap\, \mathit{SF}^2 \\
R_{5,m,p} & \triangleq & \mathbf{;} \cap\, \mathit{Op}_{m,p}^2 \\
R_{6,m,p} & \triangleq & (R_4 \cup R_{3,m,p} \cup \bigcup_{q \in P} R_{5,m,q})^\star \\
R_{7,m,p} & \triangleq & \mathrm{linext}_{\mathit{LSF}_m}\, R_{6,m,p} \\
R_8 & \triangleq & \mathbf{;} \cap\, (\mathit{LF} \times \mathit{Op}) \\
R_{9,p} & \triangleq & (R_8 \cup \bigcup_{m \in \mathit{Mem}} R_{7,m,p})^\star
\end{array}
$$

The proof that $R_{9,p}$ is a p.o. is as follows:

1. $R_{1,m,p}, R_4, R_{5,m,p}$ and R_8 are by definition partial order relations over Op.

2. $R_{2,m}$ is antisymmetric but neither reflexive nor transitive.

3. Due to lemma B.11, $R_{3,m,p}$ is a p.o. in Op_m.

4. $R_{1,m,p}$ and $R_{5,m,p}$ do not conflict due to LoadOp and StoreStoreEq:

$$
\begin{array}{l}
op_1 R_{1,m,p} op_2 \wedge op_2 R_{5,m,p} op_1 \\
\Longrightarrow op_1 \leq op_2 \wedge op_2 \mathbf{;} op_1 \\
\quad \wedge (op_2, op_1) \in (\mathit{Op}_{m,p}^2 \setminus (S_{m,p} \times L_{m,p})) \\
\Longrightarrow (op_1 = op_2 \vee \neg op_2 \leq op_1) \\
\quad \wedge op_2 \mathbf{;} op_1 \\
\quad \wedge (op_2, op_1) \in (\mathit{Op}_{m,p}^2 \setminus (S_{m,p} \times L_{m,p})) \\
\Longrightarrow op_1 = op_2 \\
\quad \vee \neg op_2 \leq op_1 \wedge op_2 \mathbf{;} op_1 \\
\qquad \wedge (op_2, op_1) \in (\mathit{Op}_{m,p}^2 \setminus (S_{m,p} \times L_{m,p})) \\
\Longrightarrow op_1 = op_2 \\
\quad \vee \neg op_2 \leq op_1 \wedge op_2 \mathbf{;} op_1 \\
\qquad \wedge (op_2, op_1) \in (\mathit{LF}_{m,p} \times \mathit{LSF}_{m,p} \cup \mathit{SF}_{m,p}^2) \\
\Longrightarrow op_1 = op_2
\end{array}
$$

5. $R_{2,m} \cap \mathit{Op}_{m,p}^2 = \{\}$ due to LoadOp.

6. R_4 and $R_{1,m,p}$ do not conflict since both are subrelations of $\leq$.

7. R_4 does not conflict with $R_{3,m,p}$ for $op_1, op_2 \in \mathit{Op}_m$:

$$
\begin{array}{l}
op_1 R_4 op_2 \wedge op_2 R_{3,m,p} op_1 \\
\Longrightarrow op_1 R_4 op_2 \wedge (op_1 = op_2 \vee \neg op_1 R_{3,m,p} op_2) \\
\Longrightarrow op_1 R_4 op_2 \\
\quad \wedge (op_1 = op_2 \\
\qquad \vee \neg op_1 R_{2,m} op_2 \wedge \neg op_1 R_{1,m,p} op_2) \\
\Longrightarrow op_1 R_4 op_2 \wedge (op_1 = op_2 \vee \neg op_1 R_{1,m,p} op_2) \\
\Longrightarrow op_1 = op_2.
\end{array}
$$

8. R_4 and $R_{5,m,p}$ do not conflict due to StoreStoreEq.

9. $R_{3,m,p}$ and $R_{5,m,p}$ do not conflict for $op_1, op_2 \in \mathit{Op}_m$:

$$\begin{array}{l} op_1 R_{3,m,p} op_2 \wedge op_2 R_{5,m,p} op_1 \\ \Longrightarrow (op_1 = op_2 \vee \neg op_2 R_{2,m} op_1 \\ \wedge \neg op_2 R_{1,m,p} op_1) \wedge (op_2 R_{5,m,p} op_1) \\ \Longrightarrow (op_1 = op_2 \\ \vee \neg op_2 R_{1,m,p} op_1) \wedge (op_2 R_{5,m,p} op_1) \\ \Longrightarrow op_1 = op_2. \end{array}$$

10. From lemma B.10 one has that $R_{6,m,p}$ is a p.o.

11. R_8 and $R_{7,m,p}$ do not conflict:

$$\begin{array}{l} op_1 R_8 op_2 \wedge op_2 R_{7,m,p} op_1 \\ \Longrightarrow op_1 R_8 op_2 \wedge (op_1 = op_2 \vee \neg op_1 R_{7,m,p} op_2) \\ \Longrightarrow op_1 = op_2 \vee (op_1 R_8 op_2 \wedge \neg op_1 R_{6,m,p} op_2) \\ \Longrightarrow op_1 = op_2 \vee (op_1 R_8 op_2 \wedge \neg op_1 R_4 op_2 \\ \wedge \neg op_1 R_{3,m,p} op_2 \wedge \bigwedge_{q \in P} \neg op_1 R_{5,m,q} op_2) \\ \Longrightarrow op_1 = op_2 \vee (op_1 \,;\, op_2 \wedge \neg op_1 R_4 op_2 \\ \wedge \neg op_1 R_{3,m,p} op_2 \wedge proc(op_1) \neq proc(op_2)) \\ \Longrightarrow op_1 = op_2. \end{array}$$

12. From lemma B.9 one has that $R_{9,p}$ is a p.o.

Lemma B.12 $\forall l \in \mathit{LF}_m : \forall s \in \mathit{SF}_m : proc(l) = proc(s) \wedge s \leq l \Longrightarrow s \,;\, l$.

Proof

The proof is based on the LoadOp property:

$$\begin{array}{l} s \leq l \\ \Longrightarrow s = l \vee \neg l \leq s \\ \Longrightarrow s = l \vee \neg l \,;\, s \\ \Longrightarrow s \,;\, l \vee proc(s) \neq proc(l) \end{array}$$

Lemma B.13 *$R_{3,m,p}$ is total over* $(L_m \times \mathit{SF}_m \cup \mathit{SF}_m \times L_m) \setminus \cup_{p \in P} \mathit{Op}^2_{m,p}$.

Proof

Using lemma B.12, with $l \in L_{m,p}$ and $s \in \mathit{SF}_m$, one has

$$\begin{array}{l} l \leq s \vee s \leq l \vee \neg l \leq s \wedge \neg s \leq l \\ \Longrightarrow l R_{1,m,p} s \vee s \leq l \wedge proc(s) = proc(l) = p \\ \vee s \leq l \wedge proc(s) \neq proc(l) \vee l R_{2,m} s \\ \Longrightarrow l R_{1,m,p} s \vee s \,;\, l \vee s R_{1,m,p} l \vee l R_{2,m} s \\ \Longrightarrow l R_{3,m,p} s \vee s R_{3,m,p} l \vee s R_{5,m,p} l \\ \Longrightarrow l R_{3,m,p} s \vee s R_{3,m,p} l \vee s \,;\, l \end{array}$$

Lemma B.14 *$R_{6,m,p}$ is total over* $(\mathit{LSF}^2_m \setminus L^2_m) \cup \cup_{p \in P} \mathit{Op}^2_{m,p})$.

Proof

$$\begin{array}{l} op_1 R_{6,m,p} op_2 \vee op_2 R_{6,m,p} op_1 \\ \Longrightarrow (op_1 R_4 op_2 \vee op_1 R_{3,m,p} op_2 \vee \bigvee_{q \in P} op_1 R_{5,m,q} op_2) \\ \vee (op_2 R_4 op_1 \vee op_2 R_{3,m,p} op_1 \vee \bigvee_{r \in P} op_2 R_{5,m,r} op_1) \\ \Longrightarrow (op_1 R_4 op_2 \vee op_2 R_4 op_1) \\ \vee op_1 R_{3,m,p} op_2 \vee op_2 R_{3,m,p} op_1 \\ \vee \bigvee_{q \in P} op_1 R_{5,m,q} op_2 \vee \bigvee_{r \in P} op_2 R_{5,m,r} op_1 \\ \Longrightarrow op_1 = op_2 \\ \vee (op_1, op_2) \in \mathit{SF}^2_m \\ \vee (op_1, op_2) \in ((L_m \times \mathit{SF}_m \cup \mathit{SF}_m \times L_m) \setminus \cup_{p \in P} \mathit{Op}^2_{m,p}) \\ \vee (op_1, op_2) \in \cup_{q \in P} \mathit{Op}^2_{m,q} \\ \vee (op_1, op_2) \in \cup_{r \in P} \mathit{Op}^2_{m,r} \\ \Longrightarrow op_1 = op_2 \vee (op_1, op_2) \in ((\mathit{LSF}^2_m \setminus L^2_m) \cup \cup_{p \in P} \mathit{Op}^2_{m,p}) \end{array}$$

Lemma B.15 $R_{7,m,p} \cap (\mathit{LSF}^2_m \setminus L^2_m) \subset R_{6,m,p}$.

Proof

This lemma follows directly from lemma B.14: $R_{6,m,p}$ is also total over $\mathit{LSF}^2_m \setminus L^2_m$.

Lemma B.16 $\forall l \in \mathit{LF}_m : p = proc(l) \Longrightarrow \mathit{hist}_{R_{9,p}}(l) = \mathit{hist}_{\mathit{SPARC},\leq}(l)$

Proof

Using LoadOp and lemma B.15, one has

$$\begin{array}{l} \mathit{hist}_{R_{9,p}}(l) \\ \quad = \{s \in \mathit{SF}_m | s \neq l \wedge s R_{9,p} l\} \\ \quad = \{s \in \mathit{SF}_m | s \neq l \wedge (s R_8 l \vee s R_{7,m,p} l)\} \\ \quad = \{s \in \mathit{SF}_m | s \neq l \wedge (s R_8 l \vee s R_{6,m,p} l)\} \\ \quad = \{s \in \mathit{SF}_m | s \neq l \wedge (s \,;\, l \vee s R_4 l \vee s R_{y,m} l \\ \qquad \vee s R_{1,m,p} l \vee s R_{5,m,p} l)\} \\ \quad = \{s \in \mathit{SF}_m | s \neq l \\ \qquad \wedge (s R_8 l \vee s \leq l \vee s \leq l \vee s \leq l \vee s \leq l)\} \\ \quad \subset \{s \in \mathit{SF}_m | s \neq l \wedge (s \,;\, l \vee s \leq l)\} \\ \quad = \mathit{hist}_{\mathit{SPARC},\leq}(l) \\ \mathit{hist}_{\mathit{SPARC},\leq}(l) \\ \quad = \{s \in \mathit{SF}_m | s \neq l \wedge (s \,;\, l \vee s \leq l)\} \\ \quad = \{s \in \mathit{SF}_m | s \neq l \wedge (s \,;\, l \vee s \leq l \wedge \neg(s \,;\, l))\} \\ \quad = \{s \in \mathit{SF}_m | s \neq l \\ \qquad \wedge (s \,;\, l \vee s \leq l \wedge (l = s \vee l \,;\, s \vee proc(s) \neq p))\} \\ \quad = \{s \in \mathit{SF}_m | s \neq l \\ \qquad \wedge (s \,;\, l \vee s \leq l \wedge l \,;\, s \vee s \leq l \wedge proc(s) \neq p)\} \\ \quad \subset \{s \in \mathit{SF}_m | s \neq l \\ \qquad \wedge (s \,;\, l \vee s \leq l \wedge l \leq s \vee s \leq l \wedge proc(s) \neq p)\} \\ \quad = \{s \in \mathit{SF}_m | s \neq l \wedge (s \,;\, l \vee s \leq l \wedge proc(s) \neq p)\} \\ \quad = \{s \in \mathit{SF}_m | s \neq l \wedge (s R_{5,m,p} l \vee s R_{1,m,p} l)\} \\ \quad \subset \{s \in \mathit{SF}_m | s \neq l \wedge s R_{9,p} l\} \\ \quad \subset \mathit{hist}_{R_{9,p}}(l) \end{array}$$

which proves the equality of both sets.

Conclusion – for all p, one has: $R_8 \subset R_{9,p}$, $R_4 \subset R_{9,p}$, hence $R_{9,p}$ s.c. in $(S \cup F)$, $R_{7,m,p} \subset R_{9,p}$, hence $R_{9,p}$ s.c. in Op_m, $R_{5,m,p} \subset R_{9,p}$.

Define the execution $T' = (\mathit{Op}, \xrightarrow{\text{po}}, \xrightarrow{\text{mo}\,1} \ldots \xrightarrow{\text{mo}\,n})$ with $\xrightarrow{\text{po}} = \,;$ and $\forall p \in P : \xrightarrow{\text{mo}\,p} = R_{9,p}$. T' is a valid PSO* trace because:

1. Uniprocessor correctness (1) holds because of $\xrightarrow{\text{po}\,p} = R_{2,p} \subset R_{9,p} = \xrightarrow{\text{mo}\,p}$.

2. Order of special accesses (2) holds because of StoreStore.

3. The value condition (3) holds because of Value, LoadOp and lemma B.16.

4. Liveness (4) holds because of Termination.

5. Location consistency (5) holds because $\forall\, m \in \mathit{Mem} : \forall\, p \in P :\ R_{7,m} \subset R_{9,p} = \xrightarrow{\text{mo}\,p}$.

6. Conditions (6), (7) and (8) hold because of StoreStore and because PSO does not define lock/unlock operations.

7. Since $\xrightarrow{\text{po}} \cap\, (\mathit{LF} \times \mathit{Op}) = R_8 \subset R_{6,p}$, also $(\mathit{LF} \times \mathit{Op} \subset \xrightarrow{\text{mo}\,p}$ holds.

8. Since $\xrightarrow{\text{po}\,p} = R_{2,p}$, also $\xrightarrow{\text{po}\,p} \subset \xrightarrow{\text{mo}\,p}$ holds.

9. $\forall\, p \in P :\ \ ; \cap\, (\mathit{Op} \setminus L)^2 \subset R_{9,p}$, hence $\xrightarrow{\text{mo}\,1}, \ldots, \xrightarrow{\text{mo}\,n}$ s.c. in $\mathit{Op} \setminus L$.

Inclusion of PSO* in PSO Starting from the PSO*-trace $T = (\mathit{Op}, \xrightarrow{\text{po}}, \xrightarrow{\text{mo}\,1} \ldots \xrightarrow{\text{mo}\,n})$, we define the following relations:

$$
\begin{aligned}
R_{8,m} &\triangleq \xrightarrow{\text{po}} \cap\, \mathit{Op}_m^2 \\
R_{4,m} &\triangleq \xrightarrow{\text{mo}\,1} \cap\, \mathit{Op}_m^2 \\
R_5 &\triangleq (\cup_{m \in \mathit{Mem}} R_{4,m})^\star \\
R_2 &\triangleq \xrightarrow{\text{po}} \cap\, (\mathit{LF} \times \mathit{Op}) \\
R_6 &\triangleq (R_2 \cup R_5)^\star \\
R_1 &\triangleq \xrightarrow{\text{mo}\,1} \cap\, \mathit{SF}^2 \\
R_7 &\triangleq (R_1 \cup R_6)^\star
\end{aligned}
$$

Since the execution $(\mathit{Op}, \xrightarrow{\text{po}}, \xrightarrow{\text{mo}\,1} \ldots \xrightarrow{\text{mo}\,n})$ is a valid PSO* execution, from the definition of PSO* one has $R_{8,m} \subset R_{4,m}$, $R_{4,m}$ t.o. in Op_m, $\forall\, p \in P :\ R_{4,m} \subset \xrightarrow{\text{mo}\,p}$, $\forall\, p \in P :\ R_2 \subset \xrightarrow{\text{mo}\,p}$ and R_1 t.o. in SF.

From lemma B.2 it follows that R_5 p.o. and $R_5 \subset \xrightarrow{\text{mo}\,p}$, R_6 p.o. and $R_6 \subset \xrightarrow{\text{mo}\,p}$, R_7 p.o. and $R_7 \subset \xrightarrow{\text{mo}\,p}$. Combining $R_{8,m} \cap \mathit{Op}_{m,p}^2$ is by definition a t.o. in $\mathit{Op}_{p,m}$, $R_{8,m} \subset R_{4,m}$, $R_{8,m} \cap \mathit{Op}_{m,p}^2 \subset R4, m \cap \mathit{Op}_{m,p}^2$, $R_{4,m}$ is a p.o. in $\mathit{Op}_{m,p}$, it follows that $R_{8,m} \cap \mathit{Op}_{m,p}^2 = R4, m \cap \mathit{Op}_{m,p}^2$

By definition $\forall\, p \in P :\ \forall\, m \in \mathit{Mem} :\ R_2 \subset \xrightarrow{\text{mo}\,p}$ holds and also $R_{4,m} \subset \xrightarrow{\text{mo}\,p}$, which leads to $R_{5,\mathit{Mem}} \subset \xrightarrow{\text{mo}\,p}$. It follows that $R_6 \subset \xrightarrow{\text{mo}\,p}$ and hence R_6 is a p.o. over Op. From lemma B.9 one also has that R_7 is a p.o. over Op.

Relation R_7 results in the same value-history as $\xrightarrow{\text{mo}\,p}$, with $l \in L \cup F$, $\mathit{proc}(l) = p$ and $\mathit{mem}(l) = \{m\}$:

$$
\begin{aligned}
&\mathit{hist}_{\xrightarrow{\text{mo}\,p}}(l) = \{s \in \mathit{SF}_m | s \neq l \wedge s \xrightarrow{\text{mo}\,p} l)\} \\
&= \{s \in \mathit{SF}_m | s \neq l \wedge (s \xrightarrow{\text{mo}\,p} l \wedge \mathit{proc}(s) = \mathit{proc}(l) \\
&\qquad \vee s \xrightarrow{\text{mo}\,p} l)\} \\
&= \{s \in \mathit{SF}_m | s \neq l \wedge (s R_{4,m} l \wedge \mathit{proc}(s) = \mathit{proc}(l) \\
&\qquad \vee s R_{4,m} l)\} \\
&= \{s \in \mathit{SF}_m | s \neq l \wedge (s R_{8,m} l \\
&\qquad \vee s R_{4,m} l)\} \\
&= \{s \in \mathit{SF}_m | s \neq l \wedge (s \xrightarrow{\text{po}\,p} l \vee s R_{4,m} l)\} \\
&= \{s \in \mathit{SF}_m | s \neq l \wedge (s \xrightarrow{\text{po}\,p} l \vee s R_7 l)\} \\
&= \mathit{hist}_{\mathit{SPARC}, R_7}(l)
\end{aligned}
$$

Define the trace $T' = (\mathit{Op}, \ ;, \leq)$ with $\ ; \ = \xrightarrow{\text{po}}$ and with $\leq \ = R_7$. This trace T' is a valid PSO trace because

1. $\ ;$ and $\leq$ are by definition p.o. relations over Op.

2. Program Order holds by definition of $\xrightarrow{\text{po}}$.

3. Memory Order holds because $R_1 \subset R_7 = \leq$.

4. Atomicity holds because of condition 3.

5. Termination holds because of condition 4.

6. Value holds because $\mathit{hist}_{\mathit{SPARC}, R_7}(l) = \mathit{hist}_{\xrightarrow{\text{mo}\,p}}(l)$ and because $R_7 \cap \mathit{SF}^2 = R_1 = \xrightarrow{\text{mo}\,p} \cap\, \mathit{SF}^2$.

7. LoadOp holds because $R_2 \subset R_7 = \leq$.

8. StoreStore holds because of condition 8.

9. StoreStoreEq holds because $\forall\, m \in \mathit{Mem} :\ R_{4,m} \subset R_7 = \leq$.