A Framework for the Investigation of Shared Memory Systems

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.

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. \( l, s, f, \text{lock}, \text{unlock} \) or \( \text{bar} \). Store, load and load-modify-store operations have a label \( \lambda \) that specifies allowed reordering: one of weak, acquire, release or strong.

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 \( \text{Mem} \) is the set of all locations. \( m \in \text{Mem} \) is a single memory location, and \( M \subseteq \text{Mem} \) 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 \( \text{Mem} \).

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

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

The notation \( \xrightarrow{p \text{op}} \) stands for the program order relation of processor \( p \), and \( \xrightarrow{\text{mem}} \) 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 \( \xrightarrow{\text{mem}} \).

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:

\[
\forall m \in \text{Mem} : \forall (\text{op}_1, \text{op}_2) \in (\text{Op}_m^2 \setminus L_m^2) : \forall p \in P : 
\text{op}_1 \xrightarrow{p \text{op}} \text{op}_2 \implies \text{op}_1 \xrightarrow{\text{mem}} \text{op}_2
\] (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:**

\[
\forall (op_1, op_2) \in Op^2 : \forall p \in P : 
\quad op_1 \xrightarrow{\text{op}} op_2 \land \text{in-order}(op_1, op_2) \implies op_1 \xrightarrow{\text{op}} op_2,
\]

with

\[
\text{in-order}(op_1, op_2) = (op_1 \in \text{Sync} \lor op_2 \in \text{Sync} \\
\lor \text{lb}(op_1) \in \{\text{acq}_i, \text{strong}_i\} \lor \text{lb}(op_2) \in \{\text{rel}_i, \text{strong}_i\})
\]

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 \(\xrightarrow{\text{op}}\) 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 \(l \in L \cup F\):

\[
p \triangleq \text{proc}(l)
\]

\[
\text{hist}(l) \triangleq \{s \in S_{\text{mem}(l)} \cup F_{\text{mem}(l)} | s \neq l \land s \xrightarrow{nop} l\}
\]

\[
\text{prev}(l) \triangleq \{s \in \text{hist}(l) | \forall s' \in \text{hist}(l) : s = s' \lor \neg(s \xrightarrow{nop} s')\}
\]

\[
\forall s_1 \in \text{prev}(l) : \forall s_2 \in \text{prev}(l) : s_1 = s_2 \implies \text{val}_l(l) = \text{val}_l(s_1)
\]

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:**

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

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:

\[
\forall m \in \text{Mem} : \xrightarrow{m_1} \ldots \xrightarrow{m_n} \text{ are consistent over } Op_m \tag{5}
\]

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 \((\text{lock}, i_1, p, M, j)\) and ends with a **unlock** operation \((\text{unlock}, i_2, p, M, j)\). 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.

\[
N \triangleq \{(op_1, op_2) \in \text{Bar}^2 | id(op_1) = id(op_2)\} \tag{2}
\]

\[
N' \triangleq \{(op_1, op_2) \in (\text{Lock} \cup \text{Unlock})^2 | id(op_1) = id(op_2)\}
\]

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.

\[
\forall m \in \text{Mem} : \xrightarrow{n_1}/ \ldots \xrightarrow{n_m}/ N \text{ s.c. in } \text{Sync}_m/N \tag{6}
\]

\[
\forall m \in \text{Mem} : \xrightarrow{n_1}/ \ldots \xrightarrow{n_m}/ N' \text{ s.c. in } \text{Sync}_m/N'
\]

**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 order 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
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 refer to 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.

\[
\forall b_1, b_2 \in \text{Bar} : b_1 N b_2 \implies \text{mem}(b_1) = \text{mem}(b_2)
\]

(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 \((\text{bar}, i, \text{Mem}, j, p)\) 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 \(\text{Tmk\_barrier}\): replace the invocation on processor \(p\) with \((\text{bar}, i_p, \text{Mem}, j, p)\). 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** \(\text{Mod}\) constrains a set of executions \(E = (\text{Op}, \overset{m_p}{\rightarrow})\) in terms of partial order relations \(\overset{m_1}{\rightarrow} \ldots \overset{m_n}{\rightarrow}\), such that \(\text{Op}, \overset{m_p}{\rightarrow}, \overset{m_1}{\rightarrow} \ldots \overset{m_n}{\rightarrow}\) satisfy the general conditions 1 – 8 and also the model-specific conditions.

Two memory models \(\text{Mod}_1\) and \(\text{Mod}_2\) are **equivalent** if the corresponding sets of executions are equal. A memory model \(\text{Mod}_1\) is **stronger** than a memory model \(\text{Mod}_2\) if \(\text{Mod}_1\) and \(\text{Mod}_2\) 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 \(\overset{m_p}{\rightarrow}\) and the set \(\text{Op}\), we now use the relations \(\overset{m_p}{\rightarrow}/N\) and the set \(\text{Op}/N\). Extended sequential consistency, or SC*, is the memory model where the general conditions hold, and also

\[
\forall p \in P : \overset{m_p}{\rightarrow}/N \subseteq \overset{m_p}{\rightarrow}/N \text{ s.c. in } \text{Op}/N.
\]

### 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 \( \forall p \in P : \frac{m_p}{m_p} \subseteq \frac{m_p}{m_p} \), \( \forall m \in \text{Mem} : \frac{m_1}{m_1} / N, \ldots, \frac{m_n}{m_n} / N \) s.c. in \( (S_m \cup F_m) / N \), and \( \forall p \in P : (\frac{m_p}{m_p} \cap (S \cup F)^2) \subseteq \frac{m_p}{m_p} \).

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 \( m \in \text{Mem} \) and \( p \in P \), one has \( \frac{m_1}{m_1} / N, \ldots, \frac{m_n}{m_n} / N \) s.c. in \( (Op \setminus L) / N \), \( \frac{m_1}{m_1} / N, \ldots, \frac{m_n}{m_n} / N \) s.c. in \( Op_m / N \), \( \frac{m_1}{m_1} / N, \ldots, \frac{m_n}{m_n} / N \) s.c. in \( Op^2 / N \), and \( (\frac{m_1}{m_1} / (L \cup F) \times Op) \) \( \subseteq \frac{m_p}{m_p} \). 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 weakL and strongL 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 \( \frac{m_1}{m_1} / N, \ldots, \frac{m_n}{m_n} / N \) s.c. in \( Op_m / N \).

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 \( s_{\text{init}} \) 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
Only ordinary load and store operations are allowed. Load and store operations refer only one location. Operations of an individual processor appear in program order. All memory order relations are identical, and they are totally ordered. Uniprocessor correctness holds. Other conditions hold (implicit in textual definition).

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

Only ordinary load and store operations are allowed. Load and store operations refer only one location. Operations of an individual processor appear in program order. Same observation order of writes executed in p.o. Same view of writes per location. Required by textual definition of PC. 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: ≤ is a partial order in Op, and ≤ is a total order in Op \ 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: histSPARC ≤ (l) = {s ∈ S_{mem(l)} \cup F_{mem(l)} | s ≠ l ∧ (s ≤ l ∨ s ; l)}
maxSPARC, ≤ (l) = {s ∈ histSPARC, ≤ (l) | ∀s' ∈ histSPARC, ≤ (l) : s = s' ∨ ¬(s ≤ s')} ∀s_1 ∈ max ≤ : (l) ∀s_2 ∈ max ≤ : (l)s_1 = s_2 ⇒ val(l) = val_s(s_1).

LoadOp: ∀l ∈ (L \cup F) : ∀op ∈ Op : l \ op ⇒ l ≤ op.
StoreStore: ∀op_1, op_2 ∈ Op \ L : ∀s ∈ Bar : op_1 ; s ; op_2 ⇒ op_1 ≤ op_2.
StoreStoreEq: ∀m ∈ Mem : ∀op_1, op_2 ∈ Op_{m \setminus L_{m}} : op_1 ; op_2 ⇒ op_1 ≤ op_2.

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

All kinds of load and store operations are allowed. Load and store operations refer only one location. Special accesses are sequentially consistent. 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


Appendix

A Mathematical relations

A relation $R$ between the sets $A$ and $B$ holds between elements $a \in A$ and $b \in B$, if $aRb$, 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 $\forall a_1, a_2 \in A : a_1Ra_2 \lor a_2Ra_1 \lor a_1 = a_2$.

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 $S[a] = \{a'|a \in aQua'\}$. The quotient relation $R/Q$ is defined by $R/Q = \{(S_1[a_1], S_2[a_2])|a_1Ra_2\}$.

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 $A$: $R_1 \cap A^2 \ldots = \ldots = R_n \cap A^2$.

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*

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:

$$\text{dom}(R) \triangleq \{ a \mid \exists b : a R b \lor b R a \}.$$

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

$$\exists a, b : a R_1 b \land b R_2 a.$$

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

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

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

$$R^0 = \{(a, a) \mid a \in \text{dom}(R)\}$$

$$\forall n \in \mathbb{N} : R^{n+1} = (R^n; R).$$

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

$$R^* \triangleq \bigcap_{n=0}^{\infty} R^n.$$

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

Lemma B.1 If $R$ is a p.o. and if relation $A$ is a subrelation of $R$, then $(R \cap A)^*$ 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 $R_1^\ast \subset R^\ast$. Since $R$ is a p.o., $R^\ast = R$, which implies that $R_1^\ast \subset R$. $R_1^\ast$ is a p.o. by definition.

Lemma B.2 If $R$ is a p.o., $A$ and $B$ are subrelations of $R$, then $((R \cap A) \cup (R \cap B))^\ast$ 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 $A_1 \triangleq \text{dom}(R_1)$, and $A_2 \triangleq \text{dom}(R_2)$, 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

$$R^2 = (R_1 \cup R_2; R_1 \cup R_2)$$

$$= (R_1; R_1) \cup (R_1; R_2) \cup (R_2; R_1) \cup (R_2; R_2)$$

$$= R_1 \cup \{\} \cup \{\} \cup R_2$$

$$= R.$$

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

$$a R_1 op_1 \land a R_2 op_2 \implies (a R_1 op_1 \lor a R_2 op_2)$$

$$\land (a R_1 op_1 \lor a R_2 op_2)$$

$$\land (a R_1 op_1 \lor a R_2 op_2)$$

$$\implies a R_1 op_2 \land a R_2 op_1 \land a R_2 op_1 \land a R_2 op_1$$

$$\implies a R_1 op_2.$$

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 $\forall a_1, a_2, a_3, a_4 \in A : a_1 R_1 a_2 \land a_2 R_2 a_3 \land a_3 R_1 a_4 \implies a_1 R_1 a_4$

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

$$a_1 R_1 a_2 \land a_2 R_2 a_3 \land a_3 R_1 a_4$$

$$\iff a_1 R_1 a_2 \land a_2 R_2 a_3 \land a_3 R_1 a_4$$

$$\land (a_2 R_2 a_3 \land a_3 R_1 a_4)$$

$$\iff a_1 R_1 a_2 \land a_3 R_1 a_4$$

$$\land (a_2 = a_3 \ifv (a_2 R_2 a_3 \land a_3 R_1 a_4))$$

$$\iff a_1 R_1 a_2 \land a_3 R_1 a_4$$

$$\land a_2 R_2 a_3 \land a_2 R_2 a_3$$

$$\iff a_1 R_1 a_4$$

Lemma B.5 $\exists a_2, a_3, a_4 \in A : a_1 R_2 a_2 \land a_2 R_2 a_3 \land a_3 R_1 a_4 \implies a_2 \in A : a_1 R_2 a_2 \land a_2 R_2 a_3$

Proof From left to right:

$$\exists a_2, a_3, a_4 \in A : a_1 R_2 a_2 \land a_2 R_2 a_3$$

$$\land a_3 R_1 a_4$$

$$\iff a_2 \in A : a_1 R_2 a_2 \land a_2 R_2 a_3$$

From right to left:

$$\exists a_2 \in A : a_1 R_2 a_2 \land a_2 R_2 a_3$$

$$\iff a_2 \in A : a_1 R_2 a_2 \land a_2 R_2 a_3$$

$$\land a_3 R_1 a_4$$

$$\iff a_2, a_3, a_4, a_5 \in A : a_1 R_2 a_2 \land a_2 R_2 a_3$$

$$\land a_3 R_1 a_4$$

$$\land a_2 R_2 a_3 \land a_2 R_2 a_3$$

$$\iff a_2, a_3, a_4, a_5 \in A : a_1 R_2 a_2 \land a_2 R_2 a_3 \land a_3 R_1 a_4.$$
If the left-hand side holds for \( n \in \mathbb{N} \), one has

\[
a_1 R^{n+1} a_5 \\
\iff 3a_4 \in A : a_1 R^n a_4 \land a_4 Ra_5 \\
\iff 3a_2, a_3, a_4 \in A : a_1 R_2 a_2 \land a_2 R_1 a_3 \land a_3 R_2 a_4 \\
\land (a_4 R_3 a_5 \lor a_5 R_2 a_4)
\]

\[
\iff 3a_2, a_3, a_4 \in A : (a_1 R_2 a_2 \land a_2 R_1 a_3 \\
\land a_3 R_2 a_4 \land a_4 R_1 a_5) \\
\lor (a_1 R_2 a_2 \land a_2 R_1 a_3) \\
\land a_3 R_2 a_4 \land a_4 R_2 a_5)
\]

\[
\iff 3a_2, a_3, a_4, a_5 \in A : a_1 R_2 a_2 \land a_2 R_1 a_3 \\
\land a_3 R_2 a_4 \land a_4 R_1 a_5) \\
\lor 3a_2, a_3, a_4 \in A : (a_1 R_2 a_2 \land a_2 R_1 a_3 \\
\land a_3 R_2 a_4 \land a_4 R_2 a_5)
\]

\[
\iff 3a_2, a_3, a_5 \in A : a_1 R_2 a_2 \land a_2 R_1 a_3 \\
\land a_3 R_2 a_4 \land a_4 R_2 a_5)
\]

From right to left:

\[
3a_2, a_3, a_5 \in A : a_1 R_2 a_2 \land a_2 R_1 a_3 \\
\land a_3 R_2 a_4 \land a_4 R_2 a_5)
\]

\[
\iff 3a_2, a_3 \in A : a_1 R_2 a_2 \land a_2 R_1 a_3 \\
\land a_3 R_2 a_4 \land a_4 R_2 a_5)
\]

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:

\[
a_1 R a_2 \land a_2 R a_1 \\
\implies a_1 (R_1 \cup R_2) a_2 \land a_2 (R_1 \cup R_2) a_1 \\
\implies (a_1 R_1 a_2 \lor a_1 R_2 a_2) \land (a_2 R_1 a_1 \lor a_2 R_2 a_1) \\
\implies (a_1 R_1 a_2 \land a_2 R_1 a_1) \lor (a_1 R_2 a_2 \land a_2 R_2 a_1) \\
\implies a_1 = a_2 \lor a_1 = a_2 \lor a_2 = a_1 = a_2 \\
\implies a_1 = a_2
\]

Lemma B.8 \( R^n \) is antisymmetric and reflexive

Proof

\( R^n \) is reflexive because \( R \) is reflexive. Antisymmetry follows from \( a_1 R^n a_4 \land a_4 R^n a_1 \implies a_1 = a_4: \)

\[
a_1 R^n a_4 \land a_4 R^n a_1 \\
\implies 3a_2, a_3, a_5, a_6 \in A : a_1 R_2 a_2 \land a_2 R_1 a_3 \\
\land a_3 R_2 a_4 \land a_4 R_2 a_5) \\
\implies 3a_2, a_3, a_5, a_6 \in A : a_1 R_2 a_2 \land a_2 R_1 a_3 \\
\land a_3 R_2 a_4 \land a_4 R_2 a_5) \\
\implies 3a_2, a_3, a_5, a_6 \in A : a_1 R_2 a_2 \land a_2 R_1 a_3 \\
\land a_3 R_2 a_4 \land a_4 R_2 a_5) \\
\implies 3a_2, a_3, a_5, a_6 \in A : a_1 R_2 a_2 \land a_2 R_1 a_3 \\
\land a_3 R_2 a_4 \land a_4 R_2 a_5) \\
\implies 3a_2, a_3, a_5, a_6 \in A : a_1 R_2 a_2 \land a_2 R_1 a_3 \\
\land a_3 R_2 a_4 \land a_4 R_2 a_5) \\
\implies 3a_2, a_3, a_5, a_6 \in A : a_1 R_2 a_2 \land a_2 R_1 a_3 \\
\land a_3 R_2 a_4 \land a_4 R_2 a_5) \\
\implies 3a_2, a_3, a_5, a_6 \in A : a_1 R_2 a_2 \land a_2 R_1 a_3 \\
\land a_3 R_2 a_4 \land a_4 R_2 a_5)
\]

Lemma B.9 \( R^* \) is a partial order.

Proof

Due to lemma B.8, \( R^* \) is antisymmetric. By definition of transitive closure, \( R^* \) is reflexive and transitive.

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

Proof

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

\[
U_k \triangleq (Q \cup R_k)^* \\
S_l \triangleq \bigcup_{1 \leq j \leq l} U_j
\]

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

\[
a_1 (S_k; U_{k+1}) a_3 \\
\implies 3a_2 : a_1 S_k a_2 \land a_2 U_{k+1} a_3 \\
\implies 3a_2 : (\exists j : 1 \leq j \leq k \land a_1 U_j a_2) \land a_2 U_{k+1} a_3 \\
\implies 3a_2 : (\exists j : 1 \leq j \leq k \land a_1 U_j a_2) \land a_2 S_{k+1} a_3 \\
\implies 3a_2 : (\exists j : 1 \leq j \leq k \land a_1 S_{k+1} a_3) \\
\implies a_1 S_{k+1} a_3
\]

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

- **Induction Base**: \( S_0 \) is transitive.

- **Induction**
• Induction Hypothesis
0 ≤ k < n ∧ S_k^2 = S_k = S_k^2 + 1
• Proof
Since S_0 = \{ \}, the induction base holds. The proof of the induction hypothesis is as follows:

\[
S_{k+1}^2 = (S_{k+1}; S_{k+1})
= (S_k ∪ U_{k+1}; S_k ∪ U_{k+1})
= (S_k; S_k) ∪ (S_k; U_{k+1}) ∪ (U_{k+1}; S_k) ∪ (U_{k+1}; U_{k+1})
= S_k ∪ S_{k+1} ∪ S_{k+1} ∪ U_{k+1}
= S_{k+1}.
\]

☐

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 ⇒ S_{k+1} is antisymmetric, with 0 ≤ k < n.
• Proof

\[
a_1 S_{k+1}^2 a_2 \land a_2 S_{k+1}^2 a_1
\implies a_1 (S_k ∪ U_{k+1}) a_2 \land a_2 (S_k ∪ U_{k+1}) a_1
\implies (a_1 S_k a_2 \lor a_1 U_{k+1} a_2) \land (a_2 S_k a_1 \lor a_2 U_{k+1} a_1)
\implies (a_1 S_k a_2 \land a_2 S_k a_1) \lor (a_1 S_k a_2 \land a_2 U_{k+1} a_1)
\lor (a_1 U_{k+1} a_2 \land a_2 S_k a_1) \lor (a_1 U_{k+1} a_2 \land a_2 U_{k+1} a_1)
\implies a_1 = a_2
\lor (a_1 S_k a_2 \lor a_2 Q a_1 \lor a_2 R_{k+1} a_1)
\lor (a_1 Q a_2 \lor a_1 R_{k+1} a_2 \lor a_2 S_k a_1)
\lor (a_1 Q a_2 \lor a_1 R_{k+1} a_2 \lor a_2 Q a_1 \lor a_2 R_{k+1} a_1)
\implies a_1 = a_2.
\]

☐

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 ⊆ A, and if R_2 is a relation over (A \ B) × B such that ∀ a, a' ∈ A : a_1 R_2 a_2 ⇔ ¬a_1 R_1 a_2 ∧ ¬a_2 R_1 a_1, then (R_1 ∪ R_2) is a partial order in A and a total order in A^2 \ (A \ B)^2.

Proof

1. (R_1; R_2) ⊆ R_2.

2. (R_2; R_1) ⊆ (R_2; R_1).

3. (R_1; R_2) ⊆ (R_1 ∪ R_2).

4. ∀ n ∈ N : n ≥ 1 =⇒ (R_1 ∪ R_2)^n ⊆ R_1 ∪ (R_2; R_1).

5. R = R_1 ∪ (R_2; R_1).

6. R is total in A^2 \ (A \ B)^2.

7. R is reflexive.

8. R is antisymmetric.

9. R is transitive.

1. With a_1 and a_3 ∈ A, the proof of (R_1; R_2) ⊆ R_2 is as follows:

\[
a_1(R_1; R_2)a_3
\implies \exists a_2 ∈ A : a_1 R_1 a_2 \land a_2 R_2 a_3
\implies \exists a_2 ∈ A : a_1 R_1 a_2 \land a_2 \notin B \land a_3 ∈ B
\land ¬a_2 R_1 a_3 \land ¬a_3 R_1 a_2
\implies \exists a_2 ∈ A : a_1 R_1 a_2 \land a_2 \notin B \land a_3 ∈ B
\land ¬a_2 R_1 a_3 \land ¬a_3 R_1 a_2
\land a_1 \notin B \land a_1 R_1 a_3
\implies a_1 R_1 a_3.
\]

2. Proof of (R_2; R_1) ⊆ (R_2; R_1):

\[
a_1(R_2; R_1)a_4
\implies \exists a_2, a_3 ∈ A : a_1 R_2 a_2
\land a_2 R_1 a_3 \land a_3 R_2 a_4
\implies a_2, a_3 ∈ A : a_1 R_2 a_2
\land a_2 R_1 a_3 \land a_3 R_2 a_4
\implies a_2 R_2 a_4
\land a_2 R_1 a_4
\implies \exists a_2 ∈ A : a_1 R_2 a_2 \land a_2 R_2 a_4
\implies a_1 (R_2; R_1) a_4.
\]

3. Proof of (R_1; R_2) ⊆ (R_1 ∪ R_2):

\[
a_1(R_1; R_2)a_3
\implies \exists a_2 ∈ A : a_1 R_1 a_2 \land a_2 R_2 a_3
\land a_2 ∈ A : a_1 R_1 a_2 \land a_2 R_2 a_3
\land (a_1 \notin B \land ¬a_1 R_1 a_3 \land ¬a_3 R_1 a_1)
\land a_1 R_1 a_3 \land a_1 R_2 a_3
\implies a_1 R_1 a_3 \land a_1 R_2 a_3.
\]

4. Proof of ∀ n ∈ N : n ≥ 1 =⇒ (R_1 ∪ R_2)^n ⊆ R_1 ∪ (R_2; R_1):

• Induction Base

(\ R_1 ∪ R_2)^1 = R_1 ∪ R_2 = R_1 ∪ R_2; R_1

• Induction Hypothesis

(R_1 ∪ R_2)^n ⊆ R_1 ∪ (R_2; R_1) =⇒ (R_1 ∪ R_2)^{n+1} ⊆ R_1 ∪ (R_2; R_1)

• Proof

\[
(R_1 ∪ R_2)^{n+1}
= (R_1 ∪ R_2)^n \cap (R_1 ∪ R_2)
\subseteq (R_1 ∪ (R_2; R_1)) \cap (R_1 ∪ R_2)
\subseteq (R_1; R_1) \cap (R_2; R_1)
\subseteq (R_1; R_2) \cup (R_2; R_1)
\subseteq R_1 \cup (R_2; R_1)
\subseteq R_1 ∪ (R_2; R_1).
\]

10
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:
\[
\begin{align*}
    a_1 R a_3 \land a_3 R a_1 & \iff (a_1 R a_3 \land \exists a_2 \in B : a_1 R_2 a_2 \land a_2 R_1 a_3) \\
    \lor (a_3 R_1 a_1 \land \exists a_4 \in B : a_3 R_2 a_4 \land a_4 R_1 a_1) \\
    \lor \exists a_2, a_4 \in B : a_1 R_1 a_3 \lor a_3 R_1 a_1 \\
    \lor (a_3 R_2 a_4 \land a_4 R_1 a_1) \lor (a_3 R_2 a_2 \land a_2 R_1 a_3) \\
    \lor \exists a_2, a_4 \in B : (a_1, a_3) \in B^2 a_1 R_1 a_3 \land a_3 R_1 a_1 \\
        \land ((a_3 R_2 a_4 \land a_4 R_1 a_1) \lor (a_3 R_2 a_2 \land a_2 R_1 a_3)) \\
        \land (a_1, a_3) \in B^2 \land a_1 R_1 a_3 \land a_3 R_1 a_1 \\
        \lor a_3 R_2 a_4 \land a_4 R_1 a_1 \\
        \land (a_1, a_3) \in A^2 \land (A \setminus B)^2.
\end{align*}
\]
This proves that \( R \) is total in \( A^2 \setminus (A \setminus B)^2 \).

7. \( R \) is reflexive by definition.

8. \( R \) is antisymmetric:
\[
\begin{align*}
    a_1 R a_3 \land a_3 R a_1 & \iff \exists a_2, a_4 \in A : (a_1 R a_3) \\
        \lor \exists a_2, a_4 \in A : (a_1 R a_3) \\
        \land (a_3 R_1 a_1 \land a_3 R_2 a_4 \land a_4 R_1 a_1) \\
        \lor \exists a_2, a_4 \in A : (a_1 R a_3) \\
        \land (a_3 R_2 a_2 \land a_2 R_3 a_3 \land a_3 R_1 a_1) \\
        \iff \text{false}.
\end{align*}
\]

9. \( R \) is transitive by definition.

Conclusion: \((R_1 \cup R_2)^*\) 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:
\[
\begin{align*}
    Op \subset L_{ord} \cup S_{ord} \cup F_{ord} \cup Bar, \quad op \in (L \cup S \cup F) \implies \#mem(op) = 1, \quad op \in Bar \implies mem(op) = Mem \\
    \land \forall b_1, b_2 \in Bar : \text{id}(b_1) = \text{id}(b_2) \implies b_1 = b_2.
\end{align*}
\]
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 \( T = (Op, \leq) \) is a valid PSO trace. The relation \( \leq \) is by definition a valid program order relation. We further define the following relations:
\[
\begin{align*}
    R_{1,m,p} & \triangleq \leq \cap (Op^2_m \setminus (L_{m,p} \times S_{m,p})) \\
    R_{2,m} & \triangleq \{(l, s) \in (L_m \times S_m) | \nexists \leq s \land \neg s \leq l\} \\
    R_{3,m,p} & \triangleq (R_{1,m,p} \cup R_{2,m})^* \\
    R_4 & \triangleq \leq \cap SF^2 \\
    R_{5,m,p} & \triangleq \cap Op^2_m \\
    R_{6,m,p} & \triangleq \text{Lin(}LF \times Op\text{)} \\
    R_{7,m,p} & \triangleq \text{Lin}(L_{SF,m} \times R_{6,m,p}) \\
    R_8 & \triangleq \text{Lin}(LF \times Op) \\
    R_{9,p} & \triangleq (R_8 \cup \bigcup_{m \in Mem} R_{7,m,p})^*
\end{align*}
\]
The proof that \( R_{9,p} \) is a p.o. is as follows:

1. \( R_{1,m,p}, R_{2,m} \), and \( R_{7,m,p} \) 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:
\[
\begin{align*}
    op_1 R_{1,m,p} op_2 \land op_2 R_{5,m,p} op_1 & \implies op_1 \leq op_2 \land op_2 \leq op_1 \\
    \land (op_2, op_1) \in (Op^2_m \setminus (S_{m,p} \times L_{m,p})) & \implies (op_1, op_2) \in (Op^2_m \setminus (L_{m,p} \times S_{m,p})) \\
        \implies op_1 = op_2 \\
        \land \neg op_2 \leq op_1 \land op_1 \leq op_2 \\
        \land (op_2, op_1) \in (Op^2_m \setminus (S_{m,p} \times L_{m,p})) & \implies op_1 = op_2 \\
        \land \neg op_2 \leq op_1 \land op_1 \leq op_2 \\
        \land (op_2, op_1) \in (Op^2_m \setminus (S_{m,p} \times L_{m,p})) & \implies op_1 = op_2
\end{align*}
\]

5. \( R_{2,m} \cap Op^2_m = \{\} \) 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 \( op_1, op_2 \in Op_m \):
\[
\begin{align*}
    op_1 R_{4,op_2} \land op_2 R_{3,m,p} op_1 & \implies op_1 R_{4,op_2} \land (op_1 = op_2 \land \neg op_1 R_{3,m,p} op_2) \\
        \lor (op_1, op_2) \in (Op^2_m \setminus (L_{m,p} \times SF^2_{m,p})) & \implies op_1 = op_2 \\
        \lor (op_1, op_2) \in (Op^2_m \setminus (L_{m,p} \times SF^2_{m,p})) & \implies op_1 = op_2
\end{align*}
\]
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 $op_1, op_2 \in Op_m$:

\[
\begin{align*}
op_1 R_{3,m,p} op_2 \land R_{5,m,p} op_1 & \implies (op_1 = op_2 \lor \neg op_2 R_{5,m,p} op_1) \\
\land \neg op_2 R_{1,m,p} op_1 & \land (op_2 R_{5,m,p} op_1) \\
& \implies (op_1 = op_2) \\
\lor \neg op_2 R_{1,m,p} op_1 & \land (op_2 R_{5,m,p} op_1) \\
& \implies op_1 = op_2.
\end{align*}
\]

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

11. $R_5$ and $R_{7,m,p}$ do not conflict:

\[
\begin{align*}
op_1 R_{6,m,p} op_2 & \land R_{7,m,p} op_1 \\
& \implies (op_1 = op_2 \lor \neg op_1 R_{7,m,p} op_2) \\
\land \neg op_1 R_{4,m,p} op_2 & \land R_{7,m,p} op_2 \\
& \implies (op_1 = op_2) \\
\lor \neg op_1 R_{4,m,p} op_2 & \land (op_1 R_{7,m,p} op_2) \\
& \implies op_1 = op_2.
\end{align*}
\]

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

Lemma B.12 \( \forall l \in LF_m : \forall s \in SF_m : \text{proc}(l) = \text{proc}(s) \land s \leq l \implies s \leq l \).

**Proof**

The proof is based on the LoadOp property:

\[ s \leq l \implies s = s \land \neg l \leq s \land \neg s \leq l \]

\[ s = s \land \neg l \leq s \land \neg s \leq l \implies s \lor \text{proc}(s) \neq \text{proc}(l) \]

Lemma B.13 $R_{3,m,p}$ is total over \((L_m \times SF_m \cup SF_m \times L_m) \setminus \cup p \in P Op_{m,p}^2\).

**Proof**

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

\[ l \leq s \lor s \leq l \lor \neg l \leq s \land \neg s \leq l \implies \text{proc}(l) = \text{proc}(s) \neq \text{proc}(l) \]

which proves the equality of both sets.

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

Define the execution $T' = (Op, m \mapsto m_0, m_1, \ldots, m_n)$ with $m_0 = \emptyset$ and $\forall p \in P: m_p \mapsto R_{9,p}$. $T'$ is a valid PSO* trace because:

\[ \text{hist}_{R_{9,p}}(l) = \text{hist}_{\text{SPARC}}(l) \]

**Proof**

Using LoadOp and lemma B.15, one has

\[ \text{hist}_{R_{9,p}}(l) = \{ s \in SF_m | s \neq l \land s \cap R_{9,p} \} \]

Lemma B.16 \( \forall l \in LF_m : p = \text{proc}(l) \implies \text{hist}_{R_{9,p}}(l) = \text{hist}_{\text{SPARC}}(l) \).

**Proof**

Using LoadOp and lemma B.15, one has

\[ \text{hist}_{R_{9,p}}(l) = \{ s \in SF_m | s \neq l \land s \cap R_{9,p} \} \]

Lemma B.14 $R_{6,m,p}$ is total over \((LSF_m \setminus L_m^2) \cup \cup p \in P Op_{m,p}^2\).

**Proof**

Using lemma B.12, with $l \in L_{m,p}$, one has

\[ l \leq s \lor s \leq l \lor \neg l \leq s \land \neg s \leq l \implies \text{proc}(l) = \text{proc}(s) \neq \text{proc}(l) \]

which proves the equality of both sets.
1. Uniprocessor correctness (1) holds because of \(\frac{m_p}{p} \in R_{2,p} \subseteq R_{9,p} = \frac{m_p}{p}\).

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

3. The value condition (3) holds because of Value.

4. Liveness (4) holds because of Termination.

5. Location consistency (5) holds because for all \(m \in \text{Mem} : p \in P : R_{7,m} \subset R_{9,p} = \frac{m_p}{p}\).

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

7. Since \(\frac{m_p}{p} \cap (LF \times Op) = R_8 \subseteq R_{6,p}\), also \((LF \times Op) \subset \frac{m_p}{p}\) holds.

8. Since \(\frac{m_p}{p} = R_{2,p}\), also \(\frac{m_p}{p} \subset \frac{m_p}{p}\) holds.

9. \(\forall p \in P : \; \cap (Op \setminus L) \subset R_{9,p},\) hence \(\frac{m_1}{1}, \ldots, \frac{m_n}{n}\) s.c. in \(Op \setminus L\).

**Inclusion of PSO** in PSO Starting from the PSO*-trace \(T = (Op, \frac{m_p}{p}, \frac{m_1}{1}, \ldots, \frac{m_n}{n})\), we define the following relations:

\[
\begin{align*}
R_{8,m} &\triangleq \frac{m_p}{p} \cap Op_{m}^2 \\
R_{4,m} &\triangleq \frac{m_1}{1} \cap Op_{m}^2 \\
R_5 &\triangleq (\cup_{m \in \text{Mem}} R_{4,m})^* \\
R_7 &\triangleq \frac{m_p}{p} \cap (LF \times Op) \\
R_0 &\triangleq (R_2 \cup R_5)^* \\
R_1 &\triangleq \frac{m_1}{1} \cap SF^2 \\
R_7 &\triangleq (R_1 \cup R_0)^*
\end{align*}
\]

Since the execution \((Op, \frac{m_p}{p}, \frac{m_1}{1}, \ldots, \frac{m_n}{n})\) is a valid PSO*-execution, from the definition of PSO* one has \(R_{8,m} \subset R_{4,m} \subset R_{4,m} \text{ t.o. in } Op_{m}, \forall p \in P : R_{4,m} \subset \frac{m_p}{p}, \forall p \in P : R_2 \subset \frac{m_p}{p} \text{ and } R_1 \text{ t.o. in } SF^2\).

From lemma B.2 it follows that \(R_5 \text{ p.o. and } R_0 \subset \frac{m_p}{p}, R_6 \text{ p.o. and } R_0 \subset \frac{m_p}{p}, R_7 \text{ p.o. and } R_7 \subset \frac{m_p}{p}\), combining \(R_{8,m} \cap Op_{m,p}^2\) is by definition a t.o. in \(Op_{p,m}\). 
\(R_{8,m} \cap R_{4,m} \cap Op_{m,p}^2 \subset R_{4,m} \cap Op_{m,p}^2\). 
\(R_{4,m}\) is a p.o. in \(Op_{m,p}\), it follows that \(R_{8,m} \cap Op_{m,p}^2 = R_{4,m} \cap Op_{m,p}^2\).

By definition \(\forall p \in P : \forall m \in \text{Mem} : R_2 \subset \frac{m_p}{p}\) holds and also \(R_{4,m} \subset \frac{m_p}{p}\), which leads to \(R_{5,\text{Mem}} \subset \frac{m_p}{p}\).

It follows that \(R_5 \subset \frac{m_p}{p}\) and hence \(R_5\) is a p.o. over \(Op\). 
From lemma B.9 one also has that \(R_7\) is a p.o. over \(Op\).