Sufficient Conditions for Implementing the Data-Race-Free-1 Memory Model

Sarita V. Adve and Mark D. Hill

Technical Report #1107

September 1992

Computer Sciences Technical Report #1107, September 1992

Sufficient Conditions for Implementing the Data-Race-Free-1 Memory Model[†],[*]

Sarita V. Adve
Mark D. Hill

Computer Sciences Department
University of Wisconsin
Madison, Wisconsin 53706

ABSTRACT

The paper, *A Unified Formalization of Four Shared-Memory Models* [AdH92], defines the data-race-free-1 memory model and informally discusses sufficient conditions for implementing the data-race-free-1 memory model. This note formalizes and gives correctness proofs for the sufficient conditions.

This note does not stand alone. It should only be read as a supplement to the main paper [AdH92]. That paper and this note subsume the original technical report [AdH91].

1. Introduction

This note is a companion to a paper that defines the data-race-free-1 memory model [AdH92]. This note does not stand alone because it assumes the reader is familiar with the companion paper.

This note formalizes and gives correctness proofs for sufficient conditions, discussed informally in the companion paper, for implementing the data-race-free-1 memory model. Section 2 develops a formalism to describe shared-memory hardware based on work by Collier [Col92]. Section 3 uses this formalism to develop and prove the correctness of the sufficient conditions.

2. A Formalism for Describing Shared-Memory Hardware

To formalize conditions for implementing memory models, a formalism to describe non-atomic shared-memory operations and interactions between such operations is necessary. We use a formalism based on an abstraction of shared-memory systems developed by Collier [Col92] as follows.

[†] This work was supported in part by a National Science Foundation Presidential Young Investigator Award (MIPS-8957278) with matching funds from A.T. & T. Bell Laboratories, Cray Research Foundation, and Digital Equipment Corporation. Sarita Adve is also supported by an IBM graduate fellowship.

[*] An earlier version of the formalism, conditions, and proof of this note appeared in the first version of the technical report, *A Unified Formalization of Four Shared-Memory Models* [AdH91]. That technical report has since been updated to the version of the paper that will be published [AdH92]. The altered conditions and proof in this note reflect the changes advised by the referees of the above paper.

© 1992 Sarita V. Adve and Mark D. Hill.

Definition 1.1: A shared-memory system with n processors, $P_1, P_2, ..., P_n$ is represented as follows.

(1) Each processor has a copy of the shared memory.

(2) A write operation W, on address x, is comprised of sub-operations $W(1)$, $W(2)$, ..., $W(n)$, where the sub-operation $W(i)$ atomically updates the address x in the memory copy of P_i to the specified value.

(3) A read operation R, by processor P_i, on address x, is comprised of a single atomic sub-operation $R(i)$, that results in returning the value of x in the memory copy of P_i.

Although real systems do not usually provide physical copies of the entire memory to any processor, a logical copy of memory can be assumed to be associated with every processor. For example, in a cache-based system, the logical copy of memory for a processor may be the union of the processor's cache and all the lines from the main memory that are not in the cache.

Also, in a real system, some sub-operations may not be distinct physical entities. However, logically distinct sub-operations can be associated with every operation and a memory copy. For example, an update of main memory on a write constitutes the sub-operations of the write in the memory copies of the processors that do not have the line in their cache.

Finally, in real systems, sub-operations may not actually execute atomically; i.e., one-at-a-time and instantaneously. However, in most systems, sub-operations *appear* to execute atomically. For example, the sub-operation involving an update of a processor cache may be spread over an interval of time and may occur in parallel with other sub-operations. However, one can identify a single instant of time at which the update takes effect such that other sub-operations take effect either before or after this time.

The notion of sub-operations is similar to that of memory operations performing with respect to a processor defined by Dubois et al. [DSB86]. A write sub-operation, $W(i)$, corresponds to the write W performing with respect to processor P_i. A read sub-operation, $R(i)$, corresponds to the read R performing with respect to all processors. We find Collier's representation conceptually and notationally simpler.

With the above abstraction, an *execution* of a program is a set of sub-operations along with the values read and written by the sub-operations during a run of the program. A multiprocessor execution must satisfy two constraints: (i) the sub-operations of a single processor in the multiprocessor execution (including the values written by the write sub-operations and the addresses accessed by all sub-operations) must comprise a correct uniprocessor execution of the processor's code, assuming the reads are made to return the values of the multiprocessor execution, and (ii) a total order called the *execution order* can be defined on the sub-operations of the execution such that a read sub-operation returns the value of the write sub-operation ordered last before it (by the execution order) that is to the same address and executes in the same memory copy as the read. (The execution order is possible because of the assumption of atomicity for sub-operations and the definition of a read.) There may be more than one execution order corresponding to an execution.

Sufficient conditions for implementing memory models can be specified in terms of constraints that have to be satisfied by *some* execution order of every execution allowed by the implementation. Hardware designers can implement such a specification by ensuring that the real time ordering of sub-operations in the implementation satisfies the specified constraints on execution order. However, if useful, hardware designers are free to violate the ordering constraints in real time as long as there is *some* total order on the sub-operations that is an execution order and that obeys the specified constraints; i.e., sub-operations *appear* to execute in the correct order. This is in contrast to many earlier specifications [DSB86, GLL90, ScD87] which impose constraints on the real time ordering of events.

3. Sufficient Conditions for Implementing Data-Race-Free-1

The definition of data-race-free-1 [AdH92] gives the necessary and sufficient condition for implementing data-race-free-1; i.e., hardware is data-race-free-1 if it appears sequentially consistent to data-race-free programs. However, it is difficult to translate this condition directly into an implementation or to check if an implementation obeys this condition. This section formalizes two sets of sufficient conditions for hardware that satisfy data-race-free-1 and that are simpler than the definition of data-race-free-1 to translate into an implementation. The advantage of the first condition is that it encompasses all data-race-free-1 hardware that we can currently envisage; the disadvantage is that it is still not very simple to translate into an implementation. The second condition places greater restrictions on the hardware allowed; however, it is easier to translate into implementations including ones that do not obey weak ordering [DSB86], release consistency (with sequentially consistent special operations) [GLL90], and data-race-free-0 [AdH90] (e.g., the implementation proposal in the companion paper [AdH92]).

The rest of this note uses the following terminology. The following abbreviations denote the various relations introduced in the companion paper [AdH92] and this note: $\xrightarrow{po}$ denotes program order, $\xrightarrow{so1}$ denotes synchronization-order-1, $\xrightarrow{hb1}$ denotes happens-before-1, and $\xrightarrow{xo}$ denotes execution order. A condition such as "$X(i) \xrightarrow{xo} Y(j)$ for all i, j" implicitly refers to pairs of values for i and j for which both $X(i)$ and $Y(j)$ are defined. Similarly, "$X(i) \xrightarrow{xo} Y(i)$ for all i" implicitly refers to values of i for which both $X(i)$ and $Y(i)$ are defined. When considering two executions E_1 and E_2, we say a read or a write operation from execution E_1 is in execution E_2 if the dynamic instruction from which the operation was generated in E_1 is issued in E_2, and if the generated operation accesses the same location in E_1 and E_2. Finally, two sub-operations *conflict* with each other if their corresponding operations conflict with each other.

Section 3.1 describes the first sufficient condition for implementing data-race-free-1 and Section 3.2 describes the second condition.

3.1. First Sufficient Condition for Implementing the Data-Race-Free-1 Memory Model

The first sufficient condition for implementing the data-race-free-1 memory model consists of the data, synchronization, and control requirements, informally discussed in the companion paper [AdH92]. The formalization of the data and control requirements results directly from the discussion in the companion paper [AdH92]. The following discussion motivates the formalization of the synchronization requirement.

Informally, the synchronization requirement states that for a data-race-free program, conflicting synchronization operations should be seen by a processor in the same order as on a sequentially consistent machine. This is analogous to the informal statement of the data requirement which requires that a conflicting pair of operations, where at least one is a data operation, should be seen by a processor in the same order as on a sequentially consistent machine. The formalization of the data requirement considers the happens-before-1 relation [AdH92] since on a sequentially consistent machine, happens-before-1 orders all pairs of conflicting operations such that at least one is a data operation. The following formalization of the synchronization requirement considers the happens-before-0 relation (called happens-before in earlier work [AdH90]), which orders all pairs of conflicting synchronization operations.

Happens-before-0 (denoted $\xrightarrow{hb0}$) is the irreflexive, transitive closure of program order and synchronization-order-0 (denoted $\xrightarrow{so0}$), where $X \xrightarrow{so0} Y$ if X and Y are conflicting synchronization operations and $X(i) \xrightarrow{xo} Y(i)$ for some i. Note that $\xrightarrow{hb0}$ depends on the specific execution order being considered while $\xrightarrow{hb1}$ does not.

The formalization below also uses the notion of a well-formed execution order for a sequentially consistent execution as follows. For a sequentially consistent execution, a *well-formed* execution order is one that preserves program order and that makes all operations appear atomic. More formally, $\xrightarrow{xo}$ is well-formed iff it obeys the following: (a) if $X \xrightarrow{po} Y$, then $X(i) \xrightarrow{xo} Y(j)$ for all i,j, and (b) either $X(i) \xrightarrow{xo} Y(j)$ for all i, j or $Y(i) \xrightarrow{xo} X(j)$ for all i, j. By the definition of sequential consistency, a well-formed execution order must exist for every sequentially consistent execution.

The $\xrightarrow{hb0}$ relation always orders two conflicting synchronization operations. Further, for every well-formed $\xrightarrow{xo}$ of a sequentially consistent execution, if X and Y are conflicting synchronization operations and $X \xrightarrow{hb0} Y$, then $X(i) \xrightarrow{xo} Y(i)$ for all i. Thus, conflicting synchronization operations are seen by a processor in the same order as on a sequentially consistent machine if $\xrightarrow{xo}$ orders their sub-operations in the processor's memory copy in the same order as the corresponding $\xrightarrow{hb0}$. The first sufficient condition for data-race-free-1 follows next.

Condition 2.1: Hardware obeys the data-race-free-1 memory model if for every execution, E_{drf}, of a program, *Prog*, on the hardware, there is an $\xrightarrow{xo}$ (and a corresponding $\xrightarrow{hb0}$) that satisfies the following conditions.

(1) *Data* - If X and Y are conflicting operations, at least one of X or Y is a data operation, and $X \xrightarrow{hb1} Y$, then $X(i) \xrightarrow{xo} Y(i)$ for all i.[1]

(2) *Synchronization* - If X and Y are conflicting synchronization operations, and $X \xrightarrow{hb0} Y$, then $X(i) \xrightarrow{xo} Y(i)$ for all i.

(3) *Control*[2] - If *Prog* is data-race-free, then there exists a sequentially consistent execution, E_{sc}, with a well-formed $\xrightarrow{xo}$ and a corresponding $\xrightarrow{hb0}$ such that (i) an operation is in E_{drf} iff it is in E_{sc}, (ii) for two conflicting operations X and Y, such that at least one of them is a data operation, if $X \xrightarrow{hb1} Y$ in E_{sc}, then $X \xrightarrow{hb1} Y$ in E_{drf}, and (iii) for two conflicting synchronization operations X and Y, if $X \xrightarrow{hb0} Y$ in E_{sc}, then $X \xrightarrow{hb0} Y$ in E_{drf}.

The proof of correctness of condition 2.1 follows from the discussion in the companion paper, which shows that the data, synchronization, and control requirements together obey the data-race-free conditions [AdH92].[3] Specifically, the control requirement ensures that for an execution, E_{drf}, of a data-race-free program, *Prog*, on hardware that obeys condition 2.1, the operations, the $\xrightarrow{hb1}$ relation, and the $\xrightarrow{hb0}$ relation are the same as those for a sequentially consistent execution, E_{sc}, of *Prog* and its well-formed $\xrightarrow{xo}$. We know that the well-formed $\xrightarrow{xo}$ of E_{sc} orders all conflicting sub-operations in a given memory copy in the same way as the corresponding $\xrightarrow{hb0}$ or $\xrightarrow{hb1}$ orders the corresponding operations. The data and synchronization requirements ensure that there

1. The data requirement in the companion paper [AdH92] mentions that all processors should *see* X before Y (if X and Y satisfy the pre-conditions of the data requirement in condition 2.1 of this note). For most cases, the informal event of a processor P_i seeing an operation is equivalent to the more formal event of the sub-operation of the operation executing in P_i's memory copy. However, the paper [AdH92] allows processor P_i to also *see* a read operation by another processor P_j (when the read returns its value). This is an extra event for which there is no equivalent sub-operation in the memory copy of P_j. Due to this extra event, the paper [AdH92] effectively imposes an unnecessary ordering constraint between the read sub-operation in P_i and other conflicting write sub-operations in the memory copy of P_j. The formalism of this note does not define the extra event and eliminates this unnecessary constraint.

2. The need for an explicit control requirement may not be obvious. It is needed only if there are reads whose values determine if an operation will be executed (as in the presence of branches), or which address an operation will access (as in the presence of indirect addressing). An example that gives incorrect results if only the data and synchronization requirements hold follows. With recent proposals for superscalar processors [FrS92], this example could be practically possible.

```
              Initially x = flag1 = flag2 = 0
              P0                      P1
if (x == 0) {flag1 = 1;}     while (flag2 != 1) {;}
flag2 = 1;                   if (flag1 == 0) {x = 1;}
```

Let the operations on flag1 and flag2 be unpaired synchronization operations and those on x be data operations. In any sequentially consistent execution of the program, P_1's read on flag1 would always return the value 1, and therefore P_1 would never issue the write on x. Thus, there cannot be a data race, the program is data-race-free, and a data-race-free-1 implementation should appear sequentially consistent to the program. In the absence of the control requirement, an aggressive implementation [FrS92] could allow P_0 to write flag2 before its read of x returned a value. This could result in the following sequence of events which makes P_1's read on flag1 return 0, and violates sequential consistency without violating the data or synchronization requirements: (a) P_0 writes flag2, (b) P_1's read on flag2 returns 1, (c) P_1's read on flag1 returns 0, (d) P_1 issues its write on x, (e) P_0's read on x returns 1, (f) P_0 does not issue its write on flag1.

3. In the data-race-free conditions [AdH92], the informal notion that "a processor should see two conflicting operations in the same order as a sequentially consistent execution" can now be interpreted as "the execution order should order conflicting sub-operations in a given memory copy in the same order as an execution order of a sequentially consistent execution."

is an $\xrightarrow{xo}$ of E_{drf} that does the same. Thus, E_{drf} and E_{sc} have the same sub-operations and $\xrightarrow{xo}$'s that order conflicting sub-operations in a given memory copy similarly. Therefore, all read sub-operations in E_{drf} return the value of the same write sub-operation as in E_{sc}. This implies that all write sub-operations write the same value in E_{drf} and E_{sc}. Thus, all reads return the same value in E_{drf} and E_{sc} and so the result of E_{drf} is the same as that of E_{sc}. Thus, hardware that obeys condition 2.1 obeys the data-race-free-1 memory model.

A slightly less restrictive form of the control requirement (Condition 2.1(3)) is possible. We have not stated that form above because its proof of correctness is more complex. Specifically, part (i) of the control requirement can be modified to require only that a write operation that is in E_{drf} should also be in E_{sc}. Parts (ii) and (iii) of the control requirement can be modified to apply only if X and Y meet the following additional restrictions: X is a read that is in both E_{sc} and E_{drf} or X is a write in E_{sc}, and Y is any operation that is in both E_{sc} and E_{drf}. Briefly, the proof with the modified form of the control requirement involves recognizing that it is sufficient to prove that if all reads that execute in both E_{drf} and the E_{sc} mentioned in the control requirement return the same value in E_{drf} and E_{sc}. Then the proof involves analyzing the first read in E_{sc} (as ordered by the $\xrightarrow{xo}$ of E_{sc} considered in the control requirement) for which the above is not true, and showing that there must be some sub-operation in E_{drf} that violates one of the three parts of the control requirement.

3.2. Second Sufficient Condition for Implementing the Data-Race-Free-1 Memory Model

The second sufficient condition for implementing the data-race-free-1 memory model is given as condition 2.2 below. Condition 2.2 obeys condition 2.1 and is also made of three sub-conditions respectively corresponding to the data, synchronization and control requirements of condition 2.1. The condition for the data requirement is similar to, but slightly less restrictive than the data requirement conditions in the companion paper [AdH92]. Further, it consists of only the actual ordering constraints between sub-operations and also includes the case for two conflicting operations from the same processor. The condition for the synchronization requirement is based on a sufficient condition for sequential consistency proposed by Dubois et al. [ScD87].

Condition 2.1(1) and 2.1(2) are symmetric, and we could have given similar implementations for each in condition 2.2(1) and 2.2(2). However, since we expect synchronization operations to be less frequent than data operations, we give a more restrictive, but easier to implement, condition for the synchronization requirement.

Below, a read R *controls* memory operation X if (a) both R and X are by the same processor, and (b) the value that R returned determined if the dynamic instruction that generated X would be executed, or determined the address accessed by X. For example, X may be in only one path of a branch whose outcome was decided by R, or X may access an address in an array whose index was returned by R.

Condition 2.2: Hardware satisfies condition 2.1 and therefore obeys the data-race-free-1 memory model if for every execution, E_{drf}, of a program, *Prog*, on the hardware, there is an $\xrightarrow{xo}$ that satisfies the following conditions.

(1) *Data* - Let *Rel* and *Rel′* be release operations and *Acq* and *Acq′* be acquire operations. Let Z be any operation. Let X and Y be conflicting operations such that at least one of X or Y is a data operation.

(a) *Release-Acquire* - (i) If $Rel \xrightarrow{sol} Acq$, then $Rel(i) \xrightarrow{xo} Acq(j)$ for all i,j. (ii) If $Z \xrightarrow{po} Rel' \xrightarrow{sol} Acq' \xrightarrow{po} Rel \xrightarrow{sol} Acq$, then $Z(i) \xrightarrow{xo} Acq(j)$ for all i,j.

(b) *Post-Acquire* - (i) If $Acq \xrightarrow{po} Z$, then $Acq(i) \xrightarrow{xo} Z(j)$ for all i,j. (ii) If $X \xrightarrow{po} Rel \xrightarrow{sol} Acq \xrightarrow{po} Y$, then $X(i) \xrightarrow{xo} Y(i)$ for all i.

(c) *Intra-processor* - If $X \xrightarrow{po} Y$, then $X(i) \xrightarrow{xo} Y(i)$ for all i.

(2) *Synchronization*- Let X, Y, and Z be synchronization operations.

(a) If $Y \xrightarrow{po} Z$, then $Y(i) \xrightarrow{xo} Z(j)$ for all i,j.

(b) If X is a write operation, Y is a read operation that conflicts with X and $X \xrightarrow{so0} Y \xrightarrow{po} Z$, then $X(i) \xrightarrow{xo} Z(j)$ for all i,j.

(c) If X and Y are conflicting write operations, then either $X(i) \xrightarrow{xo} Y(i)$ for all i or $Y(i) \xrightarrow{xo} X(i)$ for all i.

(3) *Control* -

(a) Let read R control an operation X or determine the value that X writes (if X is a write).[4] Then $R(i) \xrightarrow{xo} X(j)$ for all i, j.

(b) Consider any sequentially consistent execution, E_{sc}, of *Prog* and operations X and Y such that $X \xrightarrow{po} Y$ and either X and Y conflict, or X is an acquire, or Y is a release, or X and Y are synchronization operations in E_{sc}. Let operation X not be executed in E_{drf} and operation Y be executed in E_{drf}. Let read R control operation X in E_{sc} and let R be one of the reads in E_{drf} whose value determined that X would not be executed in E_{drf}.[5] Then $R(i) \xrightarrow{xo} Y(j)$ for all i, j.

(c) If $X \xrightarrow{po} Y$ and X is an acquire, then $X(i) \xrightarrow{po} Y(j)$ for all i, j.

(d) Let X and Y be synchronization operations. If $X \xrightarrow{po} Y$, then $X(i) \xrightarrow{xo} Y(j)$ for all i,j.[6]

The apparent complexity of condition 2.2 is because it covers a wide range of implementations. Nevertheless, converting the individual requirements into an implementation or verifying if an implementation satisfies the individual requirements is much simpler than with condition 2.1 or with the definition of data-race-free-1 as illustrated below.

4. Kourosh Gharachorloo pointed out that a read that determines the value written by a write should be considered for condition 2.2(3a).

5. Richard Zucker pointed out that for condition 2.2(3b), of the reads that control X, only those reads that determine that X is not executed in E_{drf} need be considered. This also implies that reads that determine the values written by writes need not be considered.

6. Conditions 2.2(3c) and 2.2(3d) are needed to meet the data and synchronization requirements respectively as well. They are duplicated in the control requirement for completeness since the data and synchronization requirements could potentially be satisfied in other ways.

For the data requirement, all implementations of the models of weak ordering [DSB86] and release consistency (with sequentially consistent synchronization operations) [GLL90] can be easily seen to satisfy the requirement (with an appropriate interpretation of the constraint of sequentially consistent synchronization operations imposed by these models). A more aggressive implementation of the data requirement appears in the companion paper [AdH92].

For the synchronization requirement, part (a) is satisfied if a processor does not issue a synchronization operation until its previous (by $\xrightarrow{po}$) synchronization operation completes (i.e., all sub-operations are executed). With this, parts (b) and (c) are automatically satisfied in systems that do not keep multiple copies of any memory location. For other systems (such as those with caches), part (b) is satisfied if a processor does not issue a synchronization operation until the completion of a write synchronization operation whose value the processor's previous (by $\xrightarrow{po}$) read synchronization operation may have returned. Part (c) is satisfied by implementing a cache coherence protocol [ASH88].

For the control requirement, parts (c) and (d) are already satisfied by the data and synchronization requirements respectively. Part (a) is satisfied if an operation is not issued until it is known that it will be (committed) in the execution and it is known what value it will write (if it is a write). Part (b) is satisfied if an operation is not issued until the following is known about the previous (by $\xrightarrow{po}$) operations that are not yet issued. First, a previous unissued operation cannot conflict with the current operation and cannot be an acquire. Second, if the current operation is a release operation, then all the previous operations that will be in the execution are known. Finally, if the current operation is any synchronization operation, then it is known that a previous unissued operation cannot be a synchronization operation. Two alternative, but more conservative, ways of satisfying part (d) are for a processor to block on all reads that could possibly control an operation, or to stall the issue of a memory operation until it is known which previous (by $\xrightarrow{po}$) memory operations will be (committed) in this execution.

A slightly less restrictive form of the control requirement of condition 2.2 is possible. This form obeys condition 2.1 with its less restrictive control requirement discussed earlier. Specifically, the restriction in part (a) of the control requirement of condition 2.2 need be imposed only if X is a write. The proof given below, with a few changes, applies with this modification as well.

The proof of correctness of condition 2.2 follows next.

Proof of Correctness of Condition 2.2.

The proof must show that hardware that obeys condition 2.2 also obeys condition 2.1; i.e., an execution, E_{drf}, of a program, *Prog*, on hardware that obeys condition 2.2 has an $\xrightarrow{xo}$ (and a corresponding $\xrightarrow{hb0}$) that obeys the data, synchronization, and control requirements of condition 2.1. We show (in three steps) that the $\xrightarrow{xo}$

of E_{drf} guaranteed by condition 2.2 is the required $\xrightarrow{xo}$. Step I shows that any $\xrightarrow{xo}$ that satisfies the data requirement of condition 2.2 satisfies the data requirement of condition 2.1. Step II shows that any $\xrightarrow{xo}$ that satisfies the synchronization requirement of condition 2.2 satisfies the synchronization requirement of condition 2.1. Step III shows that any $\xrightarrow{xo}$ that satisfies the data and synchronization requirements of condition 2.1 and the control requirement of condition 2.2 satisfies the control requirement of condition 2.1. All the steps use the following definitions.

Definition: A *base* $\xrightarrow{hb0}$ *path* is a path in $\xrightarrow{hb0}$ such that each arc on the path is also a $\xrightarrow{po}$ arc or an $\xrightarrow{so0}$ arc, and no two consecutive arcs on the path are $\xrightarrow{po}$ arcs.

Definition: A *base* $\xrightarrow{hb1}$ *path* is a path in $\xrightarrow{hb1}$ such that each arc on the path is also a $\xrightarrow{po}$ arc or an $\xrightarrow{so1}$ arc, and no two consecutive arcs on the path are $\xrightarrow{po}$ arcs.

The proofs for Steps I and II are fairly straightforward, involving a simple case analysis of all possible base $\xrightarrow{hb0}$ and $\xrightarrow{hb1}$ paths. We include them here for completeness, but the reader may wish to skip them.

Step I: *To prove that an* $\xrightarrow{xo}$ *that satisfies condition 2.2(1) also satisfies condition 2.1(1).*

Proof: Consider an $\xrightarrow{xo}$ (and the corresponding $\xrightarrow{hb1}$) that satisfies condition 2.2(1). We have to prove that if X and Y are conflicting operations, at least one of X or Y is a data operation, and $X \xrightarrow{hb1} Y$, then $X(i) \xrightarrow{xo} Y(i)$ for all i. If X and Y are from the same processor, then condition 2.2(1c) implies that $X(i) \xrightarrow{xo} Y(i)$ for all i, thereby proving the proposition. The rest of the proof assumes that X and Y are from different processors.

A base $\xrightarrow{hb1}$ path from X to Y must contain X, (possibly) alternating release and acquire operations, and Y. Specifically, there must be release operations, Rel_p, and acquire operations, Acq_p, where $1 \le p \le n$ such that either $X \xrightarrow{po} Rel_1$ or $X = Rel_1$, $Rel_k \xrightarrow{so1} Acq_k$, $Acq_k \xrightarrow{po} Rel_{k+1}$, and either $Acq_n \xrightarrow{po} Y$ or $Acq_n = Y$.

We first show the following observation. Let op represent any Rel_p or Acq_p above, but $op \neq Y$. We show that $op(i) \xrightarrow{xo} Y(j)$ for all i, j. We then use this observation to show that all cases for the base $\xrightarrow{hb1}$ path imply $X(i) \xrightarrow{xo} Y(i)$ for all i.

Condition 2.2(1a) implies $Rel_k(i) \xrightarrow{xo} Acq_k(j)$ for all i, j. Condition 2.2(1b) implies $Acq_k(i) \xrightarrow{xo} Rel_{k+1}(j)$ for all i, j. Thus, for any operation op mentioned above and such that $op \neq Acq_n$, $op(i) \xrightarrow{xo} Acq_n(j)$ for all i, j. Further, if $Y = Acq_n$, then $op(i) \xrightarrow{xo} Y(j)$ for all i, j. If $Acq_n \xrightarrow{po} Y$, then condition 2.2(1b) implies $Acq_n(i) \xrightarrow{xo} Y(j)$ for all i, j and so again $op(i) \xrightarrow{xo} Y(j)$ for all i, j. Thus, always $op(i) \xrightarrow{xo} Y(j)$ for all i, j. We use this observation for most of the cases discussed below.

The following three cases cover all possibilities for the above base $\xrightarrow{hb1}$ path from X to Y.

Case 1: $X = Rel_1$: If $X = Rel_1$, then the observation above implies $X(i) \xrightarrow{xo} Y(j)$ for all i, j.

Case 2: $X \xrightarrow{po} Rel_1$ and n above ≥ 2: Condition 2.2(1a) implies $X(i) \xrightarrow{xo} Acq_2(j)$ for all i, j, and so by the above observation, $X(i) \xrightarrow{xo} Y(j)$ for all i, j.

Case 3: $X \xrightarrow{po} Rel_1$ and n above $= 1$: Either $X \xrightarrow{po} Rel_1 \xrightarrow{so1} Acq_1 \xrightarrow{po} Y$ or $X \xrightarrow{po} Rel_1 \xrightarrow{so1} Acq_1 = Y$. In the former case, condition 2.2(1b) implies $X(i) \xrightarrow{xo} Y(i)$ for all i. In the latter case, X and Rel_1 must be to the same address and so conflict. Therefore, by condition 2.2(1c), $X(i) \xrightarrow{xo} Rel_1(i)$ for all i and so $X(i) \xrightarrow{xo} Y(i)$ for all i.

This completes the proof for Step I. □

Step II: *To prove that an $\xrightarrow{xo}$ that satisfies condition 2.2(2) also satisfies condition 2.1(2).*

Proof: Consider an $\xrightarrow{xo}$ (and the corresponding $\xrightarrow{hb0}$) that satisfies condition 2.2(2). We have to prove that if X and Y are conflicting synchronization operations and $X \xrightarrow{hb0} Y$, then $X(i) \xrightarrow{xo} Y(i)$ for all i. There are three cases.

Case 1: There is a base $\xrightarrow{hb0}$ path from X to Y that has only $\xrightarrow{so0}$ arcs.

This case uses the following lemma.

Lemma 1: Consider an $\xrightarrow{xo}$ (and the corresponding $\xrightarrow{hb0}$) that satisfies condition 2.2(2). If A and B are conflicting synchronization operations with a base $\xrightarrow{hb0}$ path from A to B that contains only $\xrightarrow{so0}$ arcs, then $A \xrightarrow{so0} B$.

Proof: The proof proceeds by induction on the number of $\xrightarrow{so0}$ arcs in the path.

Base Case: The proposition trivially holds if the number of $\xrightarrow{so0}$ arcs is one.

Induction: Let the number of $\xrightarrow{so0}$ arcs in the base $\xrightarrow{hb0}$ path be $n > 1$. Then there is an operation C such that $A \xrightarrow{so0} C \xrightarrow{hb0} B$ and a base $\xrightarrow{hb0}$ path from C to B has $n-1$ arcs and consists only of $\xrightarrow{so0}$ arcs. There are two sub-cases.

Case 1a: C and B conflict: By the induction hypothesis, $C \xrightarrow{so0} B$. By the definition of $\xrightarrow{so0}$, $A(m) \xrightarrow{xo} C(m)$ for some m, and $C(n) \xrightarrow{xo} B(n)$ for some n. Condition 2.2(2c) implies that $A(i) \xrightarrow{so0} C(i)$ for all i and $C(i) \xrightarrow{xo} B(i)$ for all i. Since A, B, and C all conflict with each other, there must be some k for which

$A(k)$, $C(k)$ and $B(k)$ are all defined. Then $A(k) \xrightarrow{xo} B(k)$ and so $A \xrightarrow{so0} B$.

Case 1b: C and B do not conflict: There must be a write, W, on the base $\xrightarrow{hb0}$ path from C to B. By the induction hypothesis, $A \xrightarrow{so0} C \xrightarrow{so0} W \xrightarrow{so0} B$. Applying the induction hypothesis twice again, $A \xrightarrow{so0} W$ and then $A \xrightarrow{so0} B$. □

Lemma 1 implies $X \xrightarrow{so0} Y$; therefore, $X(i) \xrightarrow{xo} Y(i)$ for some i. By condition 2.2(c), $X(i) \xrightarrow{xo} Y(i)$ for all i.

Case 2: There is a base $\xrightarrow{hb0}$ path from X to Y that ends in a $\xrightarrow{po}$ arc.

This case follows directly from the following, more general, lemma.

Lemma 2: Consider an $\xrightarrow{xo}$ (and the corresponding $\xrightarrow{hb0}$) that satisfies condition 2.2(2). If A and B are synchronization operations with a base $\xrightarrow{hb0}$ path from A to B that ends in a $\xrightarrow{po}$ arc, then $A(i) \xrightarrow{xo} B(j)$ for all i, j.

Proof: The proof proceeds by induction on the number of arcs on a base $\xrightarrow{hb0}$ path.

Base Case: The lemma holds for a base $\xrightarrow{hb0}$ path of one arc because such a path must be $A \xrightarrow{po} B$. Condition 2.2(2a) implies that $A(i) \xrightarrow{xo} B(j)$ for all i, j.

Induction: Let the number of arcs in the base $\xrightarrow{hb0}$ path be $n > 1$. Then there must be an operation C such that either $A \xrightarrow{po} C \xrightarrow{hb0} B$ or $A \xrightarrow{so0} C \xrightarrow{hb0} B$ and a base $\xrightarrow{hb0}$ path from C to B ends in a $\xrightarrow{po}$ arc and has $< n$ arcs. By the induction hypothesis, $C(i) \xrightarrow{xo} B(j)$ for all i, j. If $A \xrightarrow{po} C$, then Condition 2.2(2a) implies $A(i) \xrightarrow{xo} C(j)$ for all i,j and so $A(i) \xrightarrow{xo} B(j)$ for all i, j, proving the proposition. Therefore, the rest of the proof assumes $A \xrightarrow{so0} C$. There are three sub-cases.

Case 2a: A is a read: By the definition of $\xrightarrow{so0}$, $A(m) \xrightarrow{xo} C(m)$ for some m. Since A has only one sub-operation, the induction hypothesis implies $A(i) \xrightarrow{xo} B(j)$ for all i, j.

Case 2b: A and C are both writes: Condition 2.2(2c) implies $A(i) \xrightarrow{xo} C(i)$ for all i. The induction hypothesis implies $A(i) \xrightarrow{xo} B(j)$ for all i, j.

Case 2c: A is a write and C is a read: Let the first $\xrightarrow{po}$ arc on the base $\xrightarrow{hb0}$ path from C to B be $E \xrightarrow{po} F$. By the induction hypothesis, $E(i) \xrightarrow{xo} B(j)$ for all i, j and $F(i) \xrightarrow{xo} B(j)$ for all i, j. If E is the same as C, then by condition 2.2(2b) $A(i) \xrightarrow{xo} F(j)$ for all i, j and so $A(i) \xrightarrow{xo} B(j)$ for all i, j. If E is not the same as C, then by lemma 1, $A \xrightarrow{so0} E$. This results in a base $\xrightarrow{hb0}$ path from A to B of length $< n$. Therefore, by the induction hypothesis, $A(i) \xrightarrow{xo} B(j)$ for all i, j. □

Case 3: All base $\xrightarrow{hb0}$ paths from X to Y contain a $\xrightarrow{po}$ arc and end in an $\xrightarrow{so0}$ arc.

Consider any base $\xrightarrow{hb0}$ path from X to Y. Let the first $\xrightarrow{so0}$ arc after the last $\xrightarrow{po}$ arc on the above base path be $C \xrightarrow{so0} D$. Then either $X \xrightarrow{hb0} C \xrightarrow{so0} D \xrightarrow{hb0} Y$ or $X \xrightarrow{hb0} C \xrightarrow{so0} D = Y$. For a contradiction, assume $Y(m) \xrightarrow{xo} X(m)$ for some m. This implies that $Y \xrightarrow{so0} X$. Thus, either $D \xrightarrow{hb0} Y \xrightarrow{so0} X \xrightarrow{hb0} C$ or $D = Y \xrightarrow{so0} X \xrightarrow{hb0} C$. There is a base $\xrightarrow{hb0}$ path from X to C that ends in a $\xrightarrow{po}$ arc. Therefore, by lemma 2, $D(i) \xrightarrow{xo} C(j)$ for all i, j. Therefore, it cannot be that $C \xrightarrow{so0} D$, a contradiction.

This completes the proof for Step II. □

Step III: *To prove that an $\xrightarrow{xo}$ that satisfies conditions 2.1(1,2) and condition 2.2(3) also satisfies condition 2.1(3).*

Proof: Consider an $\xrightarrow{xo}$ of an execution E_{drf} that satisfies condition 2.1(1,2) and condition 2.2(3). We have to prove that this $\xrightarrow{xo}$ also satisfies condition 2.1(3). Condition 2.1(3) only applies to executions of data-race-free programs. We assume that E_{drf} is an execution of a data-race-free program, *Prog*.

The proof proceeds by contradiction. Below, E_{sc} denotes a sequentially consistent execution of the program *Prog*. Suppose the considered $\xrightarrow{xo}$ (and corresponding $\xrightarrow{hb0}$) of E_{drf} does not obey condition 2.1(3). Then for every well-formed $\xrightarrow{xo}$ (and corresponding $\xrightarrow{hb0}$) of every E_{sc}, there is at least one sub-operation $Y(m)$ that satisfies the following properties:

(P1) Y is in E_{sc} but Y is not in E_{drf}, or
(P2) Y is in E_{drf} and either
 (i) Y is not in E_{sc}, or
 (ii) there is an operation X such that X and Y conflict, at least one of X or Y is a data operation, $X \xrightarrow{hb1} Y$ in E_{sc}, but not in E_{drf}, or
 (iii) there is an operation X such that $X(m)$ exists, X and Y are conflicting synchronization operations, $X \xrightarrow{hb0} Y$ in E_{sc}, but not in E_{drf}.

In the following, terms such as *first*, *before*, *after*, etc. applied to sub-operations of an execution refer to the ordering of the sub-operations by the considered $\xrightarrow{xo}$ of the execution. Also since (P1) and (P2) above apply to only well-formed $\xrightarrow{xo}$'s of E_{sc}, below we will implicitly consider only well-formed $\xrightarrow{xo}$'s for any E_{sc}.

The proof consists of four sub-steps as follows. Sub-step I shows that for any E_{sc} and its well-formed $\xrightarrow{xo}$, if there is a sub-operation that satisfies (P1), then there is a sub-operation in E_{drf} that satisfies (P2). Let the first sub-operation in E_{drf} that satisfies (P2) for E_{sc} and its considered $\xrightarrow{xo}$ be called the *violator* for E_{sc} and its $\xrightarrow{xo}$. Sub-step 2 shows that the violator for any E_{sc} cannot satisfy (P2(i)). Sub-step 3 shows that if the violator for any

E_{sc} satisfies (P2(ii)), it also satisfies (P2(iii)). Sub-step 4 shows that the violator of at least one E_{sc} does not satisfy (P2(iii)), thus proving a contradiction.

Sub-steps 2, 3, and 4 use two observations that we first prove next. Below, the set of sub-operations in E_{drf} before the violator for E_{sc} and its considered $\xrightarrow{xo}$ is called the *prefix* for E_{sc} and its $\xrightarrow{xo}$.

The Prefix Observation: For every sub-operation $Y(k)$ in the prefix of any E_{sc} and any of its $\xrightarrow{xo}$'s

(a) $Y(k)$ executes in E_{sc},

(b) for X that conflicts with Y, $X(k) \xrightarrow{xo} Y(k)$ in E_{sc} iff $X(k) \xrightarrow{xo} Y(k)$ in E_{drf},

(c) if $Y(k)$ is a read, then it returns the value of the same write in E_{sc} and E_{drf}; if $Y(k)$ is a write, then it writes the same value in E_{sc} and E_{drf} (this implies that a read in the prefix returns the same value in E_{sc} and E_{drf}),

(d) if $X \xrightarrow{so1} Y$ in E_{sc}, then $X \xrightarrow{so1} Y$ in E_{drf},

(e) if $X \xrightarrow{so0} Y$ in E_{sc} and $X(k)$ exists, then $X \xrightarrow{so0} Y$ in E_{drf},

Proof: The proof uses the observation that no sub-operation in the prefix can satisfy (P2).

Part (a): If $Y(k)$ does not execute in E_{sc}, then $Y(k)$ satisfies (P2(i)) and therefore cannot be in the prefix.

Part (b): Suppose part (b) does not hold. There are two cases. For the first case, $X(k) \xrightarrow{xo} Y(k)$ in E_{sc}, but not in E_{drf}. Since E_{sc} is an execution of a data-race-free program, either $X \xrightarrow{hb0} Y$ and both X and Y are synchronization operations, or $X \xrightarrow{hb1} Y$ and at least one of X or Y is a data operation. Since $Y(k)$ is in the prefix, it does not satisfy (P2(ii)) or (P2(iii)). Therefore, $X \xrightarrow{hb0} Y$ or $X \xrightarrow{hb1} Y$ in E_{drf} also. In either case, by conditions 2.1(1,2), $X(i) \xrightarrow{xo} Y(i)$ in E_{drf} for all i, a contradiction. For the second case, $X(k) \xrightarrow{xo} Y(k)$ in E_{drf} but not in E_{sc}. Y must be in E_{sc} because it does not satisfy (P2(i)). Since $X(k)$ is also in the prefix, X is in E_{sc} too. Therefore, $Y(k) \xrightarrow{xo} X(k)$ in E_{sc}. But then $Y \xrightarrow{hb0} X$ or $Y \xrightarrow{hb1} X$ in E_{sc} but not in E_{drf}. This implies $X(k)$ satisfies (P2(ii)) or (P2(iii)), a contradiction.

Part (c): Part (b) implies that if Y is a read, then the last write $W(k)$ before $Y(k)$ that conflicts with $Y(k)$ is the same in E_{drf} and E_{sc}. Therefore, $Y(k)$ must return the value of the same write in E_{drf} and E_{sc}, proving the first part of part (c). For the second part, consider the first write $W(m)$ in the prefix that does not write the same value in E_{sc} and E_{drf}. Then a read $R(n)$ that determines the value of this write must have returned a different value in E_{sc} and E_{drf}. Condition 2.2(3a) implies $R(n) \xrightarrow{xo} W(m)$; i.e., $R(n)$ is in the prefix. Therefore, by the above argument, $R(n)$ must have read the value of the same write in E_{sc} and E_{drf}. This write must be before $R(n)$ and so before $W(m)$ and so must have written the same value in E_{sc} and E_{drf}. Thus, $R(n)$ cannot have read a different value in E_{sc} and E_{drf}, a contradiction. Thus, all writes in the prefix write the same value in E_{sc} and E_{drf}.

Part (d): If $X \xrightarrow{so1} Y$ in E_{sc}, then Y is an acquire that reads the value of X in E_{sc}. From part (c), Y should do the same in E_{drf} also. Therefore, $X \xrightarrow{so1} Y$ in E_{drf}.

Part (e): If $X \xrightarrow{so0} Y$ in E_{sc}, then since $Y(k)$ does not satisfy (P2(iii)), $X \xrightarrow{hb0} Y$ in E_{drf}. Therefore, by condition 2.1(2), $X(i) \xrightarrow{xo} Y(i)$ for all i in E_{drf}. Therefore, $X \xrightarrow{so0} Y$ in E_{drf}. □

The $\xrightarrow{po}$ *Path Observation:* Consider an E_{sc} and its $\xrightarrow{xo}$. Let $A \xrightarrow{po} B$ in E_{sc} such that $B(k)$ is in the prefix for some k or $B(k)$ is the violator of E_{sc} and its $\xrightarrow{xo}$, and either (a) A and B conflict in E_{sc}, or (b) $A \xrightarrow{po} B$ could be part of a bigger base $\xrightarrow{hb1}$ path, or (c) A and B are both synchronization operations in E_{sc}. Then A is executed in E_{drf} and $A \xrightarrow{po} B$ in E_{drf}.

Proof: First note that for case (b) to be true, either A is an acquire or B is a release. Now assume for contradiction that A is not executed in E_{drf}. Then there must be a read R in E_{sc} and E_{drf} that controls A in E_{sc}, and returns a different value in E_{sc} and E_{drf} which determines that A will not be executed in E_{drf}. By condition 2.2(3b), $R(i) \xrightarrow{xo} B(j)$ for all i,j. Thus, $R(m)$ is in the prefix for some m and it violates part (c) of the prefix observation, a contradiction. Thus, A is executed in E_{drf} and $A \xrightarrow{po} B$ in E_{drf}. □

Sub-step 1: To prove that if there is a sub-operation in E_{sc} that satisfies (P1), then there is a sub-operation in E_{drf} that satisfies (P2).

Proof: Consider the first sub-operation $Y(m)$ in E_{sc} that satisfies (P1); i.e., Y is not in E_{drf}. Then there must be a read in E_{sc} and E_{drf} such that the read controls Y in E_{sc} and returns a different value in E_{sc} and E_{drf}. Consider the first read $R(k)$ in E_{sc} that is also in E_{drf} and that returns a different value in E_{sc} and E_{drf}. ($R(k) \xrightarrow{xo} Y(m)$ in E_{sc}.) Let the last write that is before $R(k)$ in E_{sc} and that conflicts with $R(k)$ be $W(k)$. $W(k)$ must be in E_{drf} since $Y(m)$ is the first sub-operation in E_{sc} that is not in E_{drf} and $W(k) \xrightarrow{xo} Y(m)$ in E_{sc}. Since $R(k)$ returns a different value in E_{sc} and E_{drf}, either $W(k)$ writes a different value in E_{sc} and E_{drf} or there is a write $W'(k)$ in E_{drf} that conflicts with $R(k)$ and $W(k) \xrightarrow{xo} W'(k) \xrightarrow{xo} R(k)$ in E_{drf}.

First consider the former possibility where $W(k)$ writes a different value in E_{sc} and E_{drf}. Then there must be a read before $W(k)$ in E_{sc} (that determined the value written by W) that is also in E_{drf} but returns a different value in E_{sc} and E_{drf}. This read must be before $R(k)$ in E_{sc}. But $R(k)$ is the first read that returns a different value in E_{drf} and E_{sc}, a contradiction.

Now consider the latter possibility which involves a $W'(k)$ in E_{drf}. There are three cases possible.

Case 1: W' is not in E_{sc}: $W'(k)$ satisfies (P2(i)), proving the proposition.

Case 2: W' is in E_{sc} and $W'(k) \xrightarrow{xo} W(k)$ in E_{sc}: Since E_{sc} is an execution of a data-race-free program, then in E_{sc}, either $W' \xrightarrow{hb1} W$ and at least one of W or W' is a data operation, or $W' \xrightarrow{hb0} W$ and both W and W' are synchronization operations. This cannot be true in E_{drf} because then conditions 2.1(1,2) require that $W'(k) \xrightarrow{xo} W(k)$ in E_{drf}, a contradiction. Therefore, $W(k)$ is a sub-operation in E_{drf} that satisfies (P2(ii)) or (P2(iii)), proving the proposition.

Case 3: W' is in E_{sc} and $R(k) \xrightarrow{xo} W'(k)$ in E_{sc}: As for case 2, $R \xrightarrow{hb0} W'$ or $R \xrightarrow{hb1} W'$ in E_{sc}. This cannot be true in E_{drf}; therefore, $W'(k)$ is a sub-operation in E_{drf} that satisfies (P2(ii)) or (P2(iii)), proving the proposition. □

Sub-step 2: To prove that the violator for any E_{sc} cannot satisfy (P2(i)).

Proof: Suppose the violator, $V(v)$, for some E_{sc} and its $\xrightarrow{xo}$ satisfies (P2(i)); i.e., V is not in E_{sc}. Then there must be a read $R(k)$ in E_{sc} and E_{drf} that controls V in E_{sc} and that returns a different value in E_{sc} and E_{drf}. By condition 2.2(3a), $R(k) \xrightarrow{xo} V(v)$ in E_{drf}; i.e., $R(k)$ is in prefix. This contradicts part (c) of the prefix observation. □

Sub-step 3: To prove that if the violator for any E_{sc} and its $\xrightarrow{xo}$ satisfies (P2(ii)), then it also satisfies (P2(iii)).

Proof: Suppose the violator, $V(v)$, for some E_{sc} and its $\xrightarrow{xo}$ satisfies (P2(ii)). Then there is an operation U that conflicts with V such that at least one of U or V is a data operation, and $U \xrightarrow{hb1} V$ in E_{sc} but not in E_{drf}. The following (more general) lemma shows that in this case, $V(v)$ must satisfy (P2(iii)).

Lemma 3: For any E_{sc} and its $\xrightarrow{xo}$, let $B(k)$ be in the prefix or let $B(k)$ be the violator, and let $A \xrightarrow{hb1} B$ in E_{sc}. Consider a base $\xrightarrow{hb1}$ path from A to B in E_{sc}. If A and B conflict, or if the base $\xrightarrow{hb1}$ path could be part of a bigger base $\xrightarrow{hb1}$ path, then either the base $\xrightarrow{hb1}$ path also exists in E_{drf} and so $A \xrightarrow{hb1} B$ in E_{drf}, or $B(k) = V(v)$ and $V(v)$ satisfies (P2(iii)). Further, if the base $\xrightarrow{hb1}$ path could end another base $\xrightarrow{hb1}$ path, then either $A(m) \xrightarrow{xo} B(k)$ in E_{drf} for some m, or $B(k) = V(v)$ and $V(v)$ satisfies (P2(iii)).

Proof: We prove the lemma by induction on the number of arcs on the base $\xrightarrow{hb1}$ path.

Base Case: We show that the lemma holds for a base $\xrightarrow{hb1}$ path from A to B in E_{sc} consisting of one arc. There are four cases.

Case 1: $A \xrightarrow{po} B$ in E_{sc}.

Let A and B conflict, or let $A \xrightarrow{po} B$ be such that it could be part of a bigger base $\xrightarrow{hb1}$ path. Then since $B(k)$ is in the prefix or $B(k)$ is the violator, the $\xrightarrow{po}$ path observation implies that $A \xrightarrow{po} B$ in E_{drf}. Therefore, the base path exists in E_{drf} and $A \xrightarrow{hb1} B$ in E_{drf}. Further, if the base path could end another base path, then A must be an acquire. Then condition 2.2(3c) ensures $A(i) \xrightarrow{xo} B(k)$ for all i.

Case 2: $A \xrightarrow{sol} B$ in E_{sc} and $B(k)$ is in the prefix.

Part (d) of the prefix observation ensures that $A \xrightarrow{sol} B$ in E_{drf}. Further, by condition 2.1(2), $A(k) \xrightarrow{xo} B(k)$ in E_{drf}.

Case 3: $A \xrightarrow{sol} B$ in E_{sc}, $B(k)$ is the same as $V(v)$, and $V(v)$ satisfies (P2(iii)).

Since $B(k) = V(v)$ and $V(v)$ satisfies (P2(iii)), the proposition is trivially satisfied.

Case 4: $A \xrightarrow{sol} B$ in E_{sc}, $B(k)$ is the same as $V(v)$, and $V(v)$ does not satisfy (P2(iii)).

$A \xrightarrow{hb0} B$ in E_{sc} and $B(k) = V(v)$ does not satisfy (P2(iii)); therefore, $A \xrightarrow{hb0} B$ in E_{drf}. By condition 2.1(2), $A(k) \xrightarrow{xo} B(k)$ in E_{drf}. If $A \not\xrightarrow{sol} B$[7] in E_{drf}, then there must be another write, $W(k)$, in E_{drf} whose value $B(k)$ returns in E_{drf}. Then $W(k)$ is in the prefix and so $W(k)$ is in E_{sc} as well (by part (a) of the prefix observation). $A(k) \xrightarrow{xo} W(k)$ in E_{drf} and therefore in E_{sc} (by part (b) of the prefix observation). Further, since $A \xrightarrow{sol} B$ in E_{sc}, $B(k) \xrightarrow{xo} W(k)$ in E_{sc}. By part (b) of the prefix observation, $B(k) \xrightarrow{xo} W(k)$ in E_{drf} as well, a contradiction. Therefore, $A \xrightarrow{sol} B$ in E_{drf} and by condition 2.1(2), $A(k) \xrightarrow{xo} B(k)$ in E_{drf}.

Induction: We show that the lemma holds for a base $\xrightarrow{hb1}$ path from A to B in E_{sc} consisting of $n > 1$ arcs. There must be a C such that either $A \xrightarrow{po} C \xrightarrow{hb1} B$ or $A \xrightarrow{sol} C \xrightarrow{hb1} B$ in E_{sc}, and there is a base $\xrightarrow{hb1}$ path from C to B of $< n$ arcs. By the induction hypothesis, either $B(k) = V(v)$ and $V(v)$ satisfies (P2(iii)), or $C \xrightarrow{hb1} B$ in E_{drf} and $C(m) \xrightarrow{xo} B(k)$ for some m in E_{drf}. If $B(k) = V(v)$ and $V(v)$ satisfies (P2(iii)), the proposition is proved. Therefore, assume $V(v)$ does not satisfy (P2(iii)). This implies $C(m)$ is in the prefix.

Consider the first case where $A \xrightarrow{po} C$ in E_{sc}. By the $\xrightarrow{po}$ path observation, $A \xrightarrow{po} C$ in E_{drf}. Therefore, the base $\xrightarrow{hb1}$ path exists in E_{drf} and $A \xrightarrow{hb1} B$ in E_{drf}. Further, if the above $\xrightarrow{hb1}$ path is to end another base $\xrightarrow{hb1}$ path, then A must be an acquire. By condition 2.2(3c), $A(i) \xrightarrow{xo} C(j)$ in E_{drf} for all i,j. There-

7. An arrow with a slash such as in $A \not\xrightarrow{sol} B$ implies that the corresponding relation (in this case $A \xrightarrow{sol} B$) is not true.

fore, $A(l) \xrightarrow{xo} B(k)$ for some l in E_{drf}.

Now consider the second case where $A \xrightarrow{so1} C$. By part (d) of the prefix observation, $A \xrightarrow{so1} C$ in E_{drf} and so the above $\xrightarrow{hb1}$ path exists in E_{drf} and $A \xrightarrow{hb1} B$ in E_{drf}. Further, by condition 2.1(2), $A(m) \xrightarrow{xo} C(m)$ in E_{drf} and therefore $A(m) \xrightarrow{xo} B(k)$ in E_{drf}. □

Sub-step 4: To prove that there is at least one E_{sc} and its $\xrightarrow{xo}$ for which the violator cannot satisfy (P2(iii)).

Proof: Suppose for every E_{sc} and its $\xrightarrow{xo}$ (and corresponding $\xrightarrow{hbo}$), the violator, $V(v)$, satisfies (P2(iii)). Thus, for every such E_{sc} and $\xrightarrow{xo}$, there is an operation U such that $U(v)$ exists, U and V are conflicting synchronization operations, and $U \xrightarrow{hb0} V$ in E_{sc} but not in E_{drf}.

Choose an E_{sc} and $\xrightarrow{xo}$ such that they have the longest prefix. We will show a contradiction by proving the existence of another sequentially consistent execution and $\xrightarrow{xo}$ which has a longer prefix. We use the following three lemmas for this.

Lemma 4: For an E_{sc} and its $\xrightarrow{xo}$, let $B(k)$ be in the prefix or let $B(k)$ be the violator, and let $A \xrightarrow{hb0} B$ in E_{sc} where A and B are both synchronization operations. Consider a base $\xrightarrow{hb0}$ path from A to B in E_{sc} that ends in a $\xrightarrow{po}$ arc. Then the base $\xrightarrow{hb0}$ path also exists in E_{drf} and so $A \xrightarrow{hb0} B$ in E_{drf}. Further, $A(i) \xrightarrow{xo} B(j)$ in E_{drf} for all i, j.

Proof: The proof for this lemma parallels that for lemma 3 and proceeds by induction on the number of arcs on a base $\xrightarrow{hb0}$ path.

Base Case: We show that the lemma holds for a base $\xrightarrow{hb1}$ path from A to B in E_{sc} consisting of one arc. A base $\xrightarrow{hb0}$ path from A to B that ends in a $\xrightarrow{po}$ arc and with only one arc is $A \xrightarrow{po} B$. Since $B(k)$ is in the prefix or it is the violator, the $\xrightarrow{po}$ path observation implies that $A \xrightarrow{po} B$ in E_{drf}. Therefore, the base path exists in E_{drf} and $A \xrightarrow{hb0} B$ in E_{drf}. Further, condition 2.2(3d) ensures $A(i) \xrightarrow{xo} B(j)$ for all i, j.

Induction: We show that the lemma holds for a base $\xrightarrow{hb0}$ path from A to B in E_{sc} consisting of $n > 1$ arcs. There must be an operation C such that either $A \xrightarrow{po} C \xrightarrow{hb0} B$ or $A \xrightarrow{so0} C \xrightarrow{hb0} B$ in E_{sc}, and there is a base $\xrightarrow{hb0}$ path from C to B of $< n$ arcs that ends in a $\xrightarrow{po}$ arc. By the induction hypothesis, $C \xrightarrow{hb0} B$ in E_{drf} and $C(i) \xrightarrow{xo} B(j)$ for all i, j in E_{drf}. Thus, $C(i)$ is in the prefix for all i.

Consider the first case where $A \xrightarrow{po} C$. By the $\xrightarrow{po}$ path observation, $A \xrightarrow{po} C$ in E_{drf}. Therefore, the base $\xrightarrow{hb1}$ path exists in E_{drf} and $A \xrightarrow{hb1} B$ in E_{drf}. Further, by condition 2.2(3d), $A(i) \xrightarrow{xo} C(j)$ for all i,

j in E_{drf}. Therefore, $A(i) \xrightarrow{xo} B(j)$ in E_{drf} for all i, j.

Now consider the second case where $A \xrightarrow{so0} C$ in E_{sc}. There is some m for which both $A(m)$ and $C(m)$ exist. For this m, $C(m)$ is in the prefix. Therefore, by part (e) of the prefix observation, $A \xrightarrow{so0} C$ in E_{drf} and so the above $\xrightarrow{hb0}$ path exists in E_{drf} and $A \xrightarrow{hb0} B$ in E_{drf}. Further, by condition 2.1(2), $A(k) \xrightarrow{xo} C(k)$ in E_{drf} for all k. Therefore, $A(i) \xrightarrow{xo} B(j)$ in E_{drf} for all i, j. □

Lemmas 5 and 6 below exploit the following property. Recall that we are considering only well-formed $\xrightarrow{xo}$'s for any E_{sc}. Such an $\xrightarrow{xo}$ orders all sub-operations of any operation either after or before all sub-operations of any other operation. Therefore, for a well-formed $\xrightarrow{xo}$, we can overload the definition of $\xrightarrow{xo}$ to apply to operations as well as sub-operations: for operations A and B, $A \xrightarrow{xo} B$ if $A(i) \xrightarrow{xo} B(j)$ for all i,j.

Lemma 5: Consider an E_{sc} and its $\xrightarrow{xo}$ (and the corresponding $\xrightarrow{hb0}$). Let A and B be non-conflicting operations that are not ordered by $\xrightarrow{hb0}$ in E_{sc}. Consider the total order on the operations of E_{sc} formed by modifying the above $\xrightarrow{xo}$ as follows. Move all operations X that are between A and B and such that $X \xrightarrow{hb0} B$ in E_{sc} to above A. Retain the original relative order of the moved operations. Then the above total order orders all conflicting operations just as the original $\xrightarrow{xo}$, and the above total order is also a well-formed execution order for E_{sc}.

Proof: In the new total order, the pairs of operations, C, D, whose relative ordering is changed from that of the original $\xrightarrow{xo}$ must satisfy the following restrictions with the original $\xrightarrow{xo}$ and $\xrightarrow{hb0}$. (R1) $A \xrightarrow{xo} C \xrightarrow{xo} D = B$ or $A \xrightarrow{xo} C \xrightarrow{xo} D \xrightarrow{xo} B$, (R2) $D \xrightarrow{hb0} B$, and (R3) $C \not\xrightarrow{hb0} B$.

C and D cannot conflict because of the following. If C and D conflict, then originally $C \xrightarrow{hb0} D$ since E_{sc} is a sequentially consistent execution of a data-race-free program. Thus, by (R2), $C \xrightarrow{hb0} B$, a contradiction to (R3). Therefore, the new total order preserves the relative ordering of all conflicting operations, proving the first part of the lemma.

The new total order is an execution order because a read returns the value of the write ordered last before it by the total order (since the relative order of conflicting operations is the same as before). Finally, we need to show that the execution order is also well-formed; i.e., it preserves program order and atomicity of operations. It preserves atomicity because it only moves all sub-operations of an operation before (or after) all sub-operations of other operations. It also preserves program order because otherwise there are operations C and D such that $C \xrightarrow{po} D$, but the new order puts D before C, changing the relative order of C and D. However, by (R2), $C \xrightarrow{po} D$ implies that $C \xrightarrow{hb0} B$, a contradiction to (R3). □

The following discussion uses the notion of critical operations defined below.

Definition: For any E_{sc} and its $\xrightarrow{xo}$, an operation is called *critical* iff it is a synchronization operation that has a sub-operation in the prefix or that has a sub-operation that is the violator of E_{sc} and its $\xrightarrow{xo}$.

Lemma 6: Consider any E_{sc} and its $\xrightarrow{xo}$ with violator $V(v)$. Let U be the first operation in E_{sc} for which $V(v)$ satisfies (P2(iii)). Then there is another sequentially consistent execution (of program *Prog*) and its $\xrightarrow{xo}$ such that (a) $V(v)$ is the violator of the new execution and $\xrightarrow{xo}$, (b) U is the first operation in the new execution for which $V(v)$ satisfies (P2(iii)), (c) all operations between U and the last critical operation in the new execution are critical operations, (d) for every operation C after U and up to (and including) the last critical operation, $U \xrightarrow{hb0} C$, and all base $\xrightarrow{hb0}$ paths from U to C are made only of $\xrightarrow{so0}$ arcs.

Proof: The proof proceeds by contradiction. Suppose for some E_{sc} and its $\xrightarrow{xo}$, there is no sequentially consistent execution (of *Prog*) and $\xrightarrow{xo}$ that satisfies properties (a)-(d) specified in lemma 6. Then consider a sequentially consistent execution E_{sc}' of *Prog* and its $\xrightarrow{xo}$ such that they satisfy the following restrictions: (R1) $V(v)$ is the violator for E_{sc}' and its considered $\xrightarrow{xo}$, (R2) U is the first operation in E_{sc}' for which $V(v)$ satisfies (P2(iii)), and (R3) the number of operations between U and the last critical operation in E_{sc}' is the minimal of all sequentially consistent executions and $\xrightarrow{xo}$'s that satisfy (R1) and (R2).

We first prove that E_{sc}' and its considered $\xrightarrow{xo}$ satisfy part (c) of lemma 6 and then prove that they also satisfy part (d) of lemma 6. Let the last critical operation in E_{sc}' be C_{last}.

For part (c), we have to prove that all operations between U and C_{last} are critical operations. For a contradiction, assume there is some non-critical operation between U and C_{last} in E_{sc}'. Let the last non-critical operation before C be N. There are four cases.

Case 1: There is no critical operation C such that $N \xrightarrow{po} C$ in E_{sc}', and N is a write that conflicts with some critical operation, O, that is after N in E_{sc}' and that has a sub-operation in the prefix.

Let $O(k)$ be a sub-operation of O that is in the prefix. There are two cases depending on whether N is a synchronization or a data operation.

First, suppose N is a synchronization operation. Since $N(k)$ exists, by part (b) of the prefix observation, $N(k)$ is in the prefix. Therefore, N is a critical operation, a contradiction.

Now suppose N is a data operation. Since E_{sc}' is an execution of a data-race-free program, there is a release operation Rel such that $N \xrightarrow{po} Rel \xrightarrow{hb1} O$ in E_{sc}'. Lemma 3 implies that $Rel(m) \xrightarrow{xo} O(k)$ for some m in E_{drf}. Therefore, Rel is a critical operation and $N \xrightarrow{po} Rel$ is a contradiction.

Case 2: There is no critical operation C such that N $\xrightarrow{po}$ *C in* E_{sc}'*, and either N is a read or N does not conflict with any critical operation after N in* E_{sc}' *that has a sub-operation in the prefix.*

Consider a total order that is the same as the considered $\xrightarrow{xo}$ of E_{sc}' until before N, followed by all the critical operations in E_{sc}' after N, followed by N. This order retains the original order of all conflicting pairs of operations until C_{last} in E_{sc}', except that between N and V. If V is a write operation or a read operation that does not control any other operation in E_{drf} and does not determine the value written by any write in E_{drf}, then the above total order is the beginning of some well-formed execution order of some sequentially consistent execution of the program *Prog*. Even if V is a read that controls an operation in E_{drf} or determines the value of a write in E_{drf}, we know that none of the critical operations after V in E_{sc} are controlled by V (by condition 2.2(3a)). Thus, again, the new total order above is the beginning of some well-formed execution order of some sequentially consistent execution of the program *Prog*. Consider an execution order that begins with the new total order. This new $\xrightarrow{xo}$ orders all the critical operations of E_{sc}' in the same order as the originally chosen $\xrightarrow{xo}$ of E_{sc}'. Therefore, a sub-operation from the original prefix does not satisfy (P2(iii)) even with the new $\xrightarrow{xo}$. Thus, $V(v)$ is the violator even for the new $\xrightarrow{xo}$. Further, since no operations were moved before U, U is still the first operation in E_{sc}' for which $V(v)$ satisfies (P2(iii)). However, the number of operations between U and the last critical operation in the new $\xrightarrow{xo}$ is less than that for the original $\xrightarrow{xo}$, a contradiction to restriction (R3) of the original $\xrightarrow{xo}$.

Case 3: There is a critical operation C such that N $\xrightarrow{po}$ *C in* E_{sc}'*, and U* $\xrightarrow{hb0}$ *N in* E_{sc}'*.*

It follows that $U \xrightarrow{hb0} C$ in E_{sc}' and a base $\xrightarrow{hb0}$ path from U to C ends in a $\xrightarrow{po}$ arc. Therefore, by lemma 4, $U(i) \xrightarrow{xo} C(j)$ for all i, j in E_{drf}. Therefore, $U \xrightarrow{so0} V$ in E_{drf}, a contradiction since V satisfies (P2(iii)).

Case 4: There is a critical operation C such that N $\xrightarrow{po}$ *C in* E_{sc}'*, and U* $\not\xrightarrow{hb0}$ *N in* E_{sc}'*.*

Since $N \xrightarrow{po} C$ in E_{sc}', N cannot be a synchronization operation because otherwise the $\xrightarrow{po}$ path observation implies that $N \xrightarrow{po} C$ in E_{drf}, condition 2.3(3d) implies that $N(i)$ is in the prefix for all i, and therefore N is a critical operation, a contradiction.

By lemma 5, N and all operations ordered before it by $\xrightarrow{hb0}$ can be moved to before U to give a new well-formed execution order of E_{sc}'. By lemma 5, the resulting execution order has the same order of conflicting operations as before; therefore, it has the same violator as before. Further, all the synchronization operations moved before U were ordered before C by a path in the original $\xrightarrow{hb0}$ that ends in $\xrightarrow{po}$. Therefore, by lemma 4, those synchronization operations are critical operations too and all their sub-operations are in the prefix. Therefore, U is still the first operation in the new execution order for which V satisfies (P2(iii)). Thus, the new $\xrightarrow{xo}$ satisfies restrictions (R1) and (R2). However, the number of operations between U and the last critical operation in the new $\xrightarrow{xo}$ is less than for the original $\xrightarrow{xo}$, a contradiction to restriction (R3) of the original $\xrightarrow{xo}$.

This completes the four cases that prove that E_{sc}' and its considered $\xrightarrow{xo}$ satisfy part (c) of lemma 6; i.e., all operations in E_{sc}' between U and C_{last} are critical operations. We now prove that E_{sc}' and its considered $\xrightarrow{xo}$ also satisfy part (d) of lemma 6; i.e., for every operation C after U and up to (and including) C_{last}, U $\xrightarrow{hb0}$ C and all $\xrightarrow{hb0}$ paths from U to C are made only of $\xrightarrow{so0}$ arcs. For a contradiction, consider the first operation C in E_{sc}' that does not obey the above.

If $U \not\xrightarrow{hb0} C$, then there is no other operation O after U such that $O \xrightarrow{hb0} C$ because then $U \not\xrightarrow{hb0} O$, contradicting the choice of C as the first operation in E_{sc}' that does not obey part (d) of the lemma. By lemma 5, there is another well-formed execution order of E_{sc}' that is the same as the considered $\xrightarrow{xo}$ except that it moves C before U. By lemma 5, the resulting execution order orders conflicting operations as the originally chosen $\xrightarrow{xo}$; therefore, the new $\xrightarrow{xo}$ has the same violator as the old $\xrightarrow{xo}$, and therefore the new $\xrightarrow{xo}$ satisfies restriction (R1). Since $U \not\xrightarrow{hb0} C$, U and C do not conflict; therefore, either C accesses a different address from U or both U and C are reads to the same address. In the former case, clearly $V(v)$ does not satisfy (P2(iii)) with C. In the latter case, if $C(v)$ exists, then it must be in the prefix and therefore again $V(v)$ does not satisfy (P2(iii)) with C. Thus, U is the first operation in the new $\xrightarrow{xo}$ for which $V(v)$ satisfies (P2(iii)). Thus, the new $\xrightarrow{xo}$ satisfies restriction (R2). In addition, the new $\xrightarrow{xo}$ has fewer operations between U and C_{last} than the old $\xrightarrow{xo}$, a contradiction to restriction (R3) for the old $\xrightarrow{xo}$.

If there is a $\xrightarrow{hb0}$ path from U to C with a $\xrightarrow{po}$ arc, then C must be the operation on the $\xrightarrow{po}$ arc. By lemma 4, $U(i) \xrightarrow{xo} C(j)$ for all i, j; i.e., $U(v)$ is in prefix. Therefore, $V(v)$ cannot satisfy (P2(iii)) for U, the final contradiction. □

We use lemma 6 to show the contradiction discussed earlier. Recall that we have to prove that there is some E_{sc} and its $\xrightarrow{xo}$ for which its violator does not satisfy (P2(iii)). Consider an E_{sc} and its $\xrightarrow{xo}$ (with violator called $V(v)$ and the first operation in E_{sc} for which $V(v)$ satisfies (P2(iii)) called U) such that (a) E_{sc} and $\xrightarrow{xo}$ have the longest prefix possible, (b) all operations between U and the last critical operation in E_{sc} are critical operations, and (c) for every operation C after U and up to (and including) the last critical operation in E_{sc}, U $\xrightarrow{hb0}$ C and all base $\xrightarrow{hb0}$ paths from U to C are made of only $\xrightarrow{so0}$ arcs. Lemma 6 ensures that such an execution is possible.

We now show that there must be another sequentially consistent execution and $\xrightarrow{xo}$ with a longer prefix than for the chosen E_{sc} and $\xrightarrow{xo}$, a contradiction to (a) above. Let the last critical operation in E_{sc} be C_{last}. Note that by (c) above, there is no critical operation C such that $U \xrightarrow{po} C$ in E_{sc}. Further, again by (c), all critical operations after U access the same location as U. There are three cases.

Case 1: U and V are both writes.

$U(k)$ cannot be in the prefix for E_{sc} and its considered $\xrightarrow{xo}$ for any k. (Otherwise, by part (b) of the prefix observation, $U(k) \xrightarrow{xo} V(k)$ in E_{drf} and therefore, $U \xrightarrow{xo0} V$ in E_{drf}. Thus, $V(v)$ does not satisfy (P2(iii)) with U, a contradiction.) Therefore, no critical operation after U in E_{sc} other than V itself can conflict with U (otherwise, its sub-operation in the prefix will satisfy (P2(iii))). Since U is a write, this implies (due to (c) above) that the only critical operation after U is V. Further, $U \not\xrightarrow{po} V$. Consider a total order of operations that is the same as $\xrightarrow{xo}$ until U except that V is moved before U. This total order must be the beginning of some well-formed execution order of some sequentially consistent execution of the program $Prog$. Further, this total order preserves the ordering of all conflicting critical operations except that between U and V. Since U was the first operation for which V satisfied (P2(iii)) in the original E_{sc}, it follows that the violator for the new execution order is after $V(v)$ in E_{drf}. Thus, the new execution order has a longer prefix than the original E_{sc}, a contradiction.

Case 2: U is a read by processor P_v and V is a write.

$U(v)$ cannot be in the prefix; otherwise, $V(v)$ does not satisfy (P2(iii)). Suppose there is a critical write operation C (other than V) after U in E_{sc}. Then $C(v)$ is not in the prefix because otherwise $C(v)$ would satisfy (P2(iii)) with U. Thus, $V(v) \xrightarrow{xo} C(v)$ in E_{drf} and so by condition 2.1(2) $V(i) \xrightarrow{xo} C(i)$ in E_{drf} for all i. Since C is critical, $C(k)$ must be in the prefix for some k. Therefore, $V \xrightarrow{xo} C$ in E_{sc}. Thus, all critical writes after U are also after V in E_{sc}; i.e., there are only critical reads between U and V in E_{sc}. Consider a total order of operations up to C_{last} that is the same as $\xrightarrow{xo}$ until before U except that all critical operations after U and up to (and including V) are moved before U. (The relative ordering of the moved operations is retained.) The argument applied to the total order generated in case 1 applies here as well and so again we have a new E_{sc} and $\xrightarrow{xo}$ with a longer prefix than the chosen one, a contradiction.

Case 3: U is a write and V is a read.

$U(v)$ cannot be in the prefix; otherwise, $V(v)$ does not satisfy (P2(iii)). Therefore, for all writes, W, between U and V in E_{sc}, $W(v)$ cannot be in the prefix; otherwise, $W(v)$ will satisfy (P2(iii)). Therefore, for all writes between U and V in E_{sc}, $V(v)$ must satisfy (P2(iii)). Consider a total order of operations up to C_{last} that is the same as $\xrightarrow{xo}$ until C_{last} except that V is moved before U. Again, this total order must be the beginning of some well-formed execution order of some sequentially consistent execution of the program *Prog*. Further, this total order preserves the ordering of all conflicting critical operations except that between V and all the operations after and including U in E_{sc} for which V satisfied (P2(iii)). Since U was the first operation for which V satisfied (P2(iii)) in the original E_{sc}, it follows that the violator for the new execution order is after $V(v)$ in E_{drf}. Thus, the new execution order has a longer prefix than the original E_{sc}, a contradiction.

This completes all the cases for sub-step III and completes the entire proof. □

Acknowledgements

We thank the anonymous referees of the companion paper [AdH92], Kourosh Gharachorloo, and Richard Zucker for helpful comments on previous versions of this work.

References

[AdH90] S. V. ADVE and M. D. HILL, Weak Ordering - A New Definition, *Proc. 17th Annual Intl. Symp. on Computer Architecture*, May 1990, 2-14.

[AdH91] S. V. ADVE and M. D. HILL, A Unified Formalization of Four Shared-Memory Models, Computer Sciences Technical Report #1051, University of Wisconsin, Madison, September 1991.

[AdH92] S. V. ADVE and M. D. HILL, A Unified Formalization of Four Shared-Memory Models, *To appear in the IEEE Transactions on Parallel and Distributed Systems*, Accepted August 1992. Also available as Computer Sciences Technical Report #1051, University of Wisconsin, Madison, September 1991, Revised September 1992.

[ASH88] A. AGARWAL, R. SIMONI, M. HOROWITZ and J. HENNESSY, An Evaluation of Directory Schemes for Cache Coherence, *Proc. 15th Annual Intl. Symp. on Computer Architecture*, Honolulu, Hawaii, June 1988, 280-289.

[Col92] W. W. COLLIER, *Reasoning about Parallel Architectures*, Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

[DSB86] M. DUBOIS, C. SCHEURICH and F. A. BRIGGS, Memory Access Buffering in Multiprocessors, *Proc. 13th Annual Intl. Symp. on Computer Architecture 14*, 2 (June 1986), 434-442.

[FrS92] M. FRANKLIN and G. S. SOHI, The Expandable Split Window Paradigm for Exploiting Fine-Grain Parallelism, *Proc. 19th Annual Intl. Symp. on Computer Architecture*, May 1992, 58-67.

[GLL90] K. GHARACHORLOO, D. LENOSKI, J. LAUDON, P. GIBBONS, A. GUPTA and J. HENNESSY, Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors, *Proc. 17th Annual Intl. Symp. on Computer Architecture*, May 1990, 15-26.

[ScD87] C. SCHEURICH and M. DUBOIS, Correct Memory Operation of Cache-Based Multiprocessors, *Proc. Fourteenth Annual Intl. Symp. on Computer Architecture*, Pittsburgh, PA, June 1987, 234-243.