Consistency Conditions for Multi-Object Distributed Operations

Neeraj Mittal and Vijay K. Garg

TR-PDS-1998-005 June 1998

Parallel & Distributed Systems group

Department of Electrical & Computer Engineering

University of Texas at Austin

Austin, Texas 78712

Consistency Conditions for Multi-Object Distributed Operations

Neeraj Mittal *
neerajm@cs.utexas.edu
Dept. of Computer Sciences

Vijay K. Garg †
garg@ece.utexas.edu
Dept. of Electrical and Computer Engg.

Parallel and Distributed Systems Laboratory
http://maple.ece.utexas.edu
The University of Texas at Austin, Austin, TX 78712

Abstract

The traditional Distributed Shared Memory (DSM) model provides atomicity at levels of read and write on single objects. Therefore, multi-object operations such as double compare and swap, and atomic m-register assignment cannot be efficiently expressed in this model. We extend the traditional DSM model to allow operations to span multiple objects. We show that memory consistency conditions such as sequential consistency and linearizability can be extended to this general model. We also provide algorithms to implement these consistency conditions in a distributed system.

1 Introduction

Applications such as distributed file systems, transaction systems and cache coherence for multiprocessors require concurrent accesses to shared data. The underlying system must provide certain guarantees about the values returned by data accesses, possibly to distinct copies of a single logical data object. A consistency condition specifies what guarantees are provided by the system. The consistency conditions should be strong enough to enable easy programming. Sequential consistency and linearizability are two well-known consistency conditions defined in the literature.

Sequential consistency was proposed by Lamport [15] to formulate a correctness criterion for a multiprocessor shared-memory system. It requires that all data operations appear to have executed atomically, in some sequential order that is consistent with the order seen by individual processes.

Linearizability was introduced by Herlihy and Wing [12] to exploit the semantics of abstract data types. It provides the illusion that each operation applied by concurrent processes takes effect instantaneously at some point between its invocation and response. Linearizability is stronger than sequential consistency and has two advantages over it. First, it is more convenient to use because it preserves real-time ordering of operations, and hence corresponds more naturally to the intuitive notion of atomic execution of operations. Consequently, it is easier to develop programs assuming a linearizable implementation of shared objects. Second, linearizability satisfies the local property, that is the system as a whole is linearizable whenever the implementation of each object is linearizable.

These and other consistency conditions [16, 3, 8, 6, 14, 9] are based on the model in which an operation is invoked on a single object. In fact, the traditional Distributed Shared Memory (DSM) provides

*supported in part by the MCD Fellowship

†supported in part by the NSF Grants ECS-9414780, CCR-9520540, Texas Higher Education Board grant ARP-320, a General Motors Fellowship, and an IBM grant

atomicity only at levels of read and write on single objects. While this may be appropriate for models at the level of hardware, they do not provide an expressive [11] and convenient abstraction for concurrent programming. Herlihy [12] extended the model to arbitrary operations on single objects. That allows the representation of more powerful concurrent objects, for example test and set, fetch and add, FIFO queues and stacks. However, the model assumes that all operations are unary, that is, they are invoked on a single object. There are many applications in which operations are more naturally expressed as encompassing multiple objects. For example, operations like double compare and swap (DCAS)[1] [10] cannot be efficiently expressed in that model. DCAS reduces the allocation and copy cost thereby permitting a more efficient implementation of concurrent objects. As another application, if a transaction in a database is viewed as an atomic operation then it is clear that it operates, in general, on multiple data items.

In this paper, we develop a framework for consistency conditions for distributed objects with multi-object operations or multi-methods. We introduce a formal model for execution of operations that span multiple objects, denoted by m-operations. In this model, each process executes multiple m-operations and each m-operation consists of multiple operations (possibly on different objects). We extend the definitions of sequential consistency and linearizability to give m-sequential consistency and m-linearizability respectively. With the increasing popularity of distributed objects it is important to understand the conditions for their consistency in presence of replication and caches. Independently, Raynal *et al* [22] also generalized the Herlihy's model to transactions on multiple objects but they focussed on weaker consistency conditions, namely causal consistency and causal serializability.

Besides practical implications, our model has nice theoretical consequences. It serves to unify results from two areas. By restricting the number of m-operations per process to one, the model reduces to that of database transactions. Similarly, if we restrict each m-operation to execute operations on a single object then the model reduces to that of distributed shared memory [1] on concurrent objects [12]. Thus with our model, one set of consistency conditions, their implementation, and complexity results are applicable to both the areas.

It has been shown that determining whether a given execution is sequentially consistent is an NP-complete problem [23]. We show that the problem of checking whether a given history is m-linearizable is also NP-complete. This is true even when the reads-from relation (defined later) is known. Note that when the reads-from relation is known, the linearizability can be checked in polynomial time [19].

We show that execution constraints proposed by Mizuno *et al* [20] to ensure efficient implementation for sequential consistency can also be used for operations that span multiple objects. Specifically, under these execution constraints, it is necessary and sufficient to ensure legality of reads to guarantee m-sequential consistency (and m-linearizability).

Finally, we provide algorithms for ensuring proposed consistency conditions in a distributed system. Several papers [2, 4, 18, 20] have proposed sequentially consistent implementations for read/write objects. Attiya and Welch [4] provide sequentially consistent and linearizable implementations for read/write objects, FIFO queues and stacks. In addition, they also give an analysis of the response time of their implementations. But their implementation for linearizability assumes that clocks are perfectly synchronized and there is an upper bound on the delay of the message. Our algorithm for m-sequential consistency is an extension of the algorithm proposed by them. We show that their algorithm also works for multi-object operations. More importantly, we provide an algorithm for implementation of m-linearizability in an asynchronous distributed system which does not make any assumptions about clock synchronization or the message delay.

It should be noted that there may be a temptation to model multi-methods by defining an aggregate

[1]DCAS atomically updates locations $addr_1$ and $addr_2$ to values new_1 and new_2 respectively if $addr_1$ holds value old_1 and $addr_2$ holds old_2 when the operation is invoked.

object that represents the state of all objects. However, this technique has serious drawbacks. For example, if there are n read-write registers and one multi-method *sum* that takes two registers as arguments, the technique will force all registers to be treated as one object. This results in loss of locality and concurrency.

This paper is organized as follows. Section 2 gives our model of a concurrent system with multi-object operations and presents the consistency conditions appropriate in this model. In Section 3 we show the NP-completeness of verification of m-linearizability. Section 4 imposes additional constraints on execution for efficient implementation of distributed objects. In Section 5 we present algorithms for implementation of m-sequential consistency and m-linearizability in an asynchronous distributed systems.

2 Definitions

2.1 System Model

A *concurrent system* consists of a finite set of sequential threads of control called *processes*, denoted by $P_1, P_2, \ldots, P_n$, that communicate through a set of shared data structures called *objects* (or *concurrent objects*) X . Each object can be accessed by *read* and *write* operations. A write into an object defines a new value for the object; a read allows to obtain the value of the object. A write operation on an object x is denoted by $w(x)v$, where v is the value written to x by this operation. A read operation on x is denoted by $r(x)v$, where v is the value of object x returned by this operation.

Processes are sequential and manipulate objects through m-operations. An *m-operation* is a sequence of operations possibly spanning several objects. Intuitively, an m-operation is a "deterministic procedure" of read and write operations on shared objects. Each process applies a sequence of m-operations to objects, alternately issuing an invocation and then receiving the associated response. Let $\alpha(arg, res)$ be an m-operation issued at P_i; *arg* and *res* denote α's input and output parameters respectively. Execution of an m-operation takes certain time; this is modeled by two events, namely an *invocation* event and a *response* event. For an m-operation α, invocation and response events, $inv(\alpha(arg))$ at P_i and $resp(\alpha(res))$ at P_i, will be abbreviated as $inv(\alpha)$ and $resp(\alpha)$ when parameters and process identity are not necessary. An event e *occurs-before* event f, denoted by $e < f$, iff event e precedes event f in real time. We will use greek symbols α, β, γ, δ, etc. to denote m-operations.

If two m-operations α and β are issued by the same process, say P_i, and α is issued before β, then we say α precedes β in P_i's *process order* and is written as $\alpha \rightsquigarrow_{P_i} \beta$. If process identity is not important then process order is denoted by $\rightsquigarrow_P$. In Figure 1, $\alpha \rightsquigarrow_{P_1} \beta$.

If a read operation $r(x)v$ reads the value written by the write operation $w(x)v$, then $r(x)v$ is said to *read-from* $w(x)v$. An m-operation α *reads-from* a distinct m-operation β the value of object x, written as $\beta \rightsquigarrow_{rf} \alpha$, if there exists at least one read operation of α that reads from some write operation of β the value of object x. In Figure 1, $\alpha \rightsquigarrow_{rf} \delta$ and $\eta \rightsquigarrow_{rf} \delta$.

We assume that an imaginary m-operation that writes to all objects is performed to initialize the objects before the first operation by any process is executed. In all the examples considered in this paper, unless specified otherwise, we assume that initial value of all objects is 0.

2.2 Histories

Informally, an execution of a concurrent system is modeled by a *history*, which is a finite sequence of m-operation invocation and response events. Formally, a history $\mathcal{H}$ is denoted by a tuple $\langle op(\mathcal{H}), \rightsquigarrow_{\mathcal{H}} \rangle$, where $op(\mathcal{H})$ is the set of m-operations and $\rightsquigarrow_{\mathcal{H}}$ is some irreflexive transitive relation defined on the set of m-operations which includes the partial order imposed by process orders and reads-from relation.

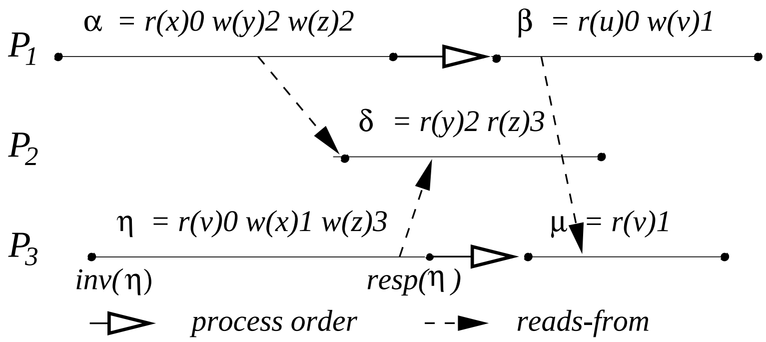

Figure 1: An execution history $\mathcal{H}_0$

A history $\mathcal{S}$ is *sequential* iff (1) its first event is an invocation event, (2) each invocation event is immediately followed by a matching response event, and (3) $\rightsquigarrow_S$ is a total order consistent with the order of m-operation invocation events.

A *process subhistory* or *local history* of P_i of a history $\mathcal{H}$, denoted by $\mathcal{H}|P_i$, is the subsequence of all events in $\mathcal{H}$ associated with the process P_i. A history is *well-formed* iff each process subhistory is sequential. All histories considered in this paper are assumed to be well-formed.

Two histories $\mathcal{H}$ and $\mathcal{G}$ are *equivalent* iff for every process P_i, $\mathcal{H}|P_i = \mathcal{G}|P_i$ and they have the same reads-from relation.

Intuitively, a read operation is legal if it does not read from an overwritten write operation. Note that if there exists a write operation $w(x)v$ before a read operation $r(x)u$ in an m-operation (such that $w(x)v$ is the last write on x before $r(x)u$) then u must be equal to v. Similarly, if there exists a write operation $w(x)v$ after a write operation $w(x)u$ in an m-operation then no read operation of another m-operation can read from $w(x)u$. In the rest of the paper, we ignore such read and write operations. Let $op(op(x)v)$ denote the m-operation associated with the operation $op(x)v$. A read operation $r(x)v$ is *legal* iff there exists a write operation $w(x)v$ such that $r(x)v$ reads from $w(x)v$ and there does not exist another write operation $w'(x)u$ such that $op(w(x)v) \rightsquigarrow_{\mathcal{H}} op(w'(x)u) \rightsquigarrow_{\mathcal{H}} op(r(x)v)$. An m-operation is *legal* iff all its read operations are legal. A history $\mathcal{H}$ is *legal* iff all its m-operations are legal.

A history $\mathcal{H}$ is *admissible* with respect to $\rightsquigarrow_{\mathcal{H}}$ iff it is equivalent to some legal sequential history that respects $\rightsquigarrow_{\mathcal{H}}$. We will omit the phrase "with respect to $\rightsquigarrow_{\mathcal{H}}$" if $\rightsquigarrow_{\mathcal{H}}$ is clear from the context.

2.3 Consistency Conditions

A consistency policy makes the behavior of a concurrent system equivalent to that of a non-concurrent system. A consistency condition provides guarantees about the values returned by data accesses in the presence of interleaved and/or overlapping accesses. Sequential consistency and linearizability are two well known consistency conditions. We extend their definitions to include m-operations to give m-sequential consistency and m-linearizability. Our definitions are based on the definition of admissibility with the partial order $\rightsquigarrow_H$ appropriately defined.

Let $proc(\alpha)$ and $objects(\alpha)$ denote the process and the set of objects respectively associated with an m-operation α. In Figure 1, $proc(\alpha) = P_1$ and $objects(\alpha) = \{x, y, z\}$. The m-operations α and β are related by *real-time order*, denoted by $\alpha \rightsquigarrow_t \beta$, iff the response of α is received before the invocation of β, that is, $resp(\alpha) < inv(\beta)$. The m-operations α and β are related by *object order*, denoted by $\alpha \rightsquigarrow_X \beta$, iff both the m-operations share an object and the response of α is received before the invocation of β, that is, $(objects(\alpha) \cap objects(\beta) \neq \phi) \wedge (resp(\alpha) < inv(\beta))$. In Figure 1, $\alpha \rightsquigarrow_t \mu$, $\eta \rightsquigarrow_t \beta$ and $\eta \rightsquigarrow_X \beta$.

m-Sequential consistency requires that all m-operations appear to have executed atomically, in some sequential order that is consistent with the order seen by individual processes. A history is *m-sequentially consistent* iff it is admissible with respect to process orders and reads-from relation. In other words, let $\mathcal{H} = \langle op(\mathcal{H}), \rightsquigarrow_{\mathcal{H}} \rangle$ be an execution history such that $\rightsquigarrow_{\mathcal{H}}$ consists of process orders and reads-from relation. Then $\mathcal{H}$ is m-sequentially consistent iff it is admissible with respect to $\rightsquigarrow_{\mathcal{H}}$. If m-operations are restricted to a single read or write operation, then our definition reduces to traditional definition of sequential consistency.

m-Linearizability requires that: (1) each m-operation should appear to take effect instantaneously somewhere between its invocation and response, and (2) the order of non-overlapping m-operations should be preserved. A history is *m-linearizable* iff it is admissible with respect to process orders, reads-from relation and real-time order. Formally, let $\mathcal{H} = \langle op(\mathcal{H}), \rightsquigarrow_{\mathcal{H}} \rangle$ be an execution history such that $\rightsquigarrow_{\mathcal{H}}$ consists of process orders, reads-from relation and real-time order. Then $\mathcal{H}$ is m-linearizable iff it is admissible with respect to $\rightsquigarrow_{\mathcal{H}}$.

Garg and Raynal [8] proposed another definition of consistency, namely normality, which is based on object order rather than real-time order. We also extend their definition of normality to give m-normality. A history is *m-normal* iff it is admissible with respect to process orders, reads-from relation and object order. In other words, let $\mathcal{H} = \langle op(\mathcal{H}), \rightsquigarrow_{\mathcal{H}} \rangle$ be an execution history such that $\rightsquigarrow_{\mathcal{H}}$ consists of process orders, reads-from relation and object order. Then $\mathcal{H}$ is m-normal iff it is admissible with respect to $\rightsquigarrow_{\mathcal{H}}$. m-Normality is less restrictive than m-linearizability since it does not order two non-overlapping m-operations unless they act on a common object. The results of Section 3 and Section 4 also hold for m-normality. Since the protocol for m-linearizability also implements m-normality, we will focus on m-linearizability in the rest of the paper.

3 NP-completeness of Consistency Conditions

It has been shown that ascertaining whether a given execution is sequentially consistent when the operations are restricted to a single object is an NP-complete problem [23]. Since our model is a generalization of the traditional DSM model, determining whether a given execution is m-sequentially consistent is NP-complete too. Misra proved that checking whether an execution satisfies atomic consistency is solvable in polynomial time when reads-from relation is known [19]. It turns out that this is not the case when the operations can encompass multiple objects. In this section we show that determining whether a given execution is m-linearizable is an NP-complete problem even when reads-from relation is known. We will use the results in databases to prove the NP-completeness of m-linearizability.

Much work on databases uses *serializability* [21, 5] as the basic correctness condition for concurrent computations. Several notions of equivalence such as *view equivalence*, *strict view equivalence*, and *conflict equivalence* are defined [21]. If we restrict each process to contain a single operation (one for each transaction) then the notion of correctness in the database world can be viewed as special case of the consistency conditions in our model. For instance, view equivalence can be considered as a special case of m-sequential consistency; strict view equivalence can be viewed as a special case of m-linearizability, and conflict equivalence can be considered as a special case of m-normality under OO-constraint (defined later). Since determining whether a schedule is strict view serializable is an NP-complete problem, hence checking whether a history is m-linearizable is also an NP-complete problem. It should be noted that checking for m-linearizability of history $\mathcal{H}$ is not same as checking for acyclicity of $\rightsquigarrow_{\mathcal{H}}$. In particular, $\rightsquigarrow_{\mathcal{H}}$ may be acyclic but $\mathcal{H}$ may not be m-linearizable.

Theorem 1 *Let $\mathcal{H}$ be an execution history. Then it is NP-complete to determine whether $\mathcal{H}$ is m-sequentially consistent.*

Theorem 2 *Let $\mathcal{H}$ be an execution history. Then it is NP-complete to determine whether $\mathcal{H}$ is m-linearizable.*

Proof: To prove that determining whether a history $\mathcal{H}$ is m-linearizable is NP-hard we reduce strict view serializability[2] to m-linearizability. Let $\mathcal{S} = (trans(\mathcal{S}), \rightsquigarrow_S)$ be a schedule of transactions in a database consisting of finite set of entities $E = \{x_1, x_2, \ldots\}$, where $trans(\mathcal{S})$ denote the set of transactions $T_1, T_2, \ldots, T_n$, and $\rightsquigarrow_S$ represents the order of actions in the schedule. We construct a distributed system consisting of sequential processes $P_0, P_1, P_2, \ldots P_n, P_\infty$, one for each transaction in the augmented schedule[3], and shared objects E. For each action in the schedule there is a corresponding operation. An operation a_i reads from operation a_j if the corresponding action a_i reads-from the corresponding action a_j in the schedule $\mathcal{S}$. Each process P_i executes a single m-operation α_i whose operations correspond to the actions of the transaction T_i executed in the same order. The first and last actions of a transaction define the invocation and response events respectively of the corresponding m-operation. It is easy to see that two transactions are non-overlapping in the schedule $\mathcal{S}$ if and only if the corresponding m-operations are non-overlapping in $\mathcal{H}$. The history $\mathcal{H}$ of the system is the history $\langle op(\mathcal{H}), \rightsquigarrow_{\mathcal{H}} \rangle$ where $op(\mathcal{H})$ is the set of transactions and $\rightsquigarrow_{\mathcal{H}}$ consists of reads-from relation and real-time order.

It can be easily proved that schedule $\mathcal{S}$ is strict view serializable if and only if the history $\mathcal{H}$ is m-linearizable. Moreover, it can be easily verified that the problem is indeed in NP since, given a sequential history, we can easily check that it is legal and equivalent to $\mathcal{H}$. ∎

4 Consistency Conditions with Constraints

Due to Theorem 1 and Theorem 2 it is unlikely that there exists an efficient algorithm that realizes m-sequential consistency (m-linearizability), that is, allows all m-sequentially consistent (m-linearizable) histories and only these. Thus, as in concurrency control protocols [13], actual implementations need to enforce constraints on executions. Mizuno *et al* [20] identified two such constraints, namely WW-and OO-constraints, for sequential consistency. We extend their work in two ways: we show that (1) their results extend to the case when the operations can span multiple objects, and (2) similar results also hold for m-linearizability. In the rest of the paper, we label the definitions by prefix "D" and the properties by prefix "P". Before proceeding further, we give some definitions we use in this section.

Let $robjects(\alpha)$ and $wobjects(\alpha)$ denote the objects read and written by α respectively. Note that after execution, the system knows the set of objects read and written by each m-operation. An m-operation is said to be an *update* m-operation iff it writes to some object. An m-operation is a *query* m-operation iff it is not an update m-operation. Two distinct operations are said to be *conflicting* iff both act on the same object and at least one of them is a write operation. Two distinct m-operations are said to be *conflicting* iff one of them contains an operation that conflicts with some operation of the other. Let $rfobjects(\mathcal{H}, \alpha, \beta)$ denote the set of objects that α reads from β in history $\mathcal{H}$. The distinct m-operations α, β and γ are said to *interfere* in history $\mathcal{H}$ iff γ writes to some object that α reads from β. Note that if α, *beta* and γ interfere in $\mathcal{H}$ then they pairwise conflict. In Figure 1, α conflicts with η, and m-operations δ, η and α interfere. Formally,

[2]A schedule $\mathcal{S}$ is *strict view serializable* if it is view equivalent to a serial schedule in which transactions that do not overlap in $\mathcal{S}$ are in the same order as in $\mathcal{S}$.

[3]a schedule augmented with an initial transaction, T_0, writing values to each entity and a final transaction, T_∞, reading values from each entity.

(D 4.1) $$conflict(\alpha,\beta) \stackrel{def}{=} (\alpha \neq \beta) \wedge ((objects(\alpha) \cap wobjects(\beta)) \cup (objects(\beta) \cap wobjects(\alpha)) \neq \phi)$$

(D 4.2) $$interfere(\mathcal{H},\alpha,\beta,\gamma) \stackrel{def}{=} (\alpha, \beta \text{ and } \gamma \text{ are distinct operations}) \wedge (rfobjects(\mathcal{H},\alpha,\beta) \cap wobjects(\gamma) \neq \phi)$$

(P 4.1) $$interfere(\mathcal{H},\alpha,\beta,\gamma) \Rightarrow conflict(\alpha,\beta) \wedge conflict(\beta,\gamma) \wedge conflict(\gamma,\alpha) \wedge (robjects(alpha) \wedge wobjects(\beta) \wedge wobjects(\gamma) \neq \phi)$$

The reads-from relation can be formally stated as follows,

(D 4.3) $$\beta \leadsto_{rf} \alpha \stackrel{def}{=} \langle \exists\, x :: x \in rfobjects(\mathcal{H},\alpha,\beta) \rangle$$

The well-formedness of a history $\mathcal{H}$ can be represented as,

(P 4.2) $$\beta \leadsto_{p} \alpha \Rightarrow resp(\beta) < inv(\alpha)$$

A history $\mathcal{G}$ *extends* history $\mathcal{H}$ iff $\mathcal{G}$ is equivalent to $\mathcal{H}$ and $\leadsto_{\mathcal{G}}$ respects $\leadsto_{\mathcal{H}}$. Note that if $\mathcal{G}$ extends $\mathcal{H}$ then $\mathcal{G}$ and $\mathcal{H}$ have identical set of interfering m-operations and "extends" is transitive. Formally,

(D 4.4) $$extends(\mathcal{G},\mathcal{H}) \stackrel{def}{=} \langle \forall\, i : 1 \leq i \leq n : \mathcal{H}|P_i = \mathcal{G}|P_i \rangle \wedge (\leadsto_{rf}^{\mathcal{H}} = \leadsto_{rf}^{\mathcal{G}}) \wedge (\leadsto_{\mathcal{H}} \subseteq \leadsto_{\mathcal{G}})$$

(P 4.3) $$extends(\mathcal{G},\mathcal{H}) \Rightarrow \langle \forall\, \alpha,\beta,\gamma \in op(\mathcal{G})(= op(\mathcal{H})) :: interfere(\mathcal{G},\alpha,\beta,\gamma) = interfere(\mathcal{H},\alpha,\beta,\gamma) \rangle$$

(P 4.4) $$extends(\mathcal{F},\mathcal{G}) \wedge extends(\mathcal{G},\mathcal{H}) \Rightarrow extends(\mathcal{F},\mathcal{H})$$

The sequentiality, legality, and admissibility of a history can be defined using the above definitions as follows,

(D 4.5) $$sequential(\mathcal{H}) \stackrel{def}{=} \leadsto_{\mathcal{H}} \text{ is a total order}$$

(D 4.6) $$legal(\mathcal{H}) \stackrel{def}{=} \langle \forall\, \alpha,\beta,\gamma \in op(\mathcal{H}) : interfere(\mathcal{H},\alpha,\beta,\gamma) : \neg(\beta \leadsto_{\mathcal{H}} \gamma) \vee \neg(\gamma \leadsto_{\mathcal{H}} \alpha) \rangle$$

(D 4.7) $$admissible(\mathcal{H}) \stackrel{def}{=} \langle \exists\, S : extends(\mathcal{S},\mathcal{H}) \wedge sequential(\mathcal{S}) : legal(\mathcal{S}) \rangle$$

Intuitively, the constraints impose additional ordering on the m-operations such that it is efficiently possible to sequentialize a history to a legal one. In this paper, we focus on OO-and WW-constraints. These constraints are enforced by the underlying system by synchronizing certain m-operations across processes. In WW-constraint all update m-operations must be globally synchronized. If OO-constraint is used, m-operations need to be synchronized only at each object level. However, m-operations that only read an object must also be synchronized with other update m-operations on that object. An alternate approach is to impose constraints on the program execution (*data race free* (DRF) and *concurrent write free* (CWF)) [3]. The system can then provide weaker guarantees and have better performance. The onus of enforcing these constraints then lies with the programmer which makes application building more difficult.

A history $\mathcal{H}$ satisfies WW*-constraint* iff any pair of m-operations performing write operations are ordered under $\leadsto_{\mathcal{H}}$. A history $\mathcal{H}$ satisfies OO*-constraint* iff any pair of conflicting m-operations are ordered under $\leadsto_{\mathcal{H}}$. We define another another constraint, namely WO-constraint, which is the intersection of OO- and

WW-constraints. We use it to prove the results that are common to both OO- and WW-constraints. A history $\mathcal{H}$ satisfies WO*-constraint* iff any pair of m-operations performing write operations on a common object are ordered under $\leadsto_{\mathcal{H}}$. Formally,

(D 4.8) $$OO(\mathcal{H}) \stackrel{def}{=} \langle \forall\, \alpha, \beta \in op(\mathcal{H}) : conflict(\alpha, \beta) : (\alpha \leadsto_{\mathcal{H}} \beta) \vee (\beta \leadsto_{\mathcal{H}} \alpha) \rangle$$

(D 4.9) $$WW(\mathcal{H}) \stackrel{def}{=} \langle \forall\, \alpha, \beta \in op(\mathcal{H}) : (\alpha \neq \beta) \wedge (wobjects(\alpha) \neq \phi) \wedge (wobjects(\beta) \neq \phi) : (\alpha \leadsto_{\mathcal{H}} \beta) \vee (\beta \leadsto_{\mathcal{H}} \alpha) \rangle$$

(D 4.10) $$WO(\mathcal{H}) \stackrel{def}{=} \langle \forall\, \alpha, \beta \in op(\mathcal{H}) : (\alpha \neq \beta) \wedge (wobjects(\alpha) \cap wobjects(\beta) \neq \phi) : (\alpha \leadsto_{\mathcal{H}} \beta) \vee (\beta \leadsto_{\mathcal{H}} \alpha) \rangle$$

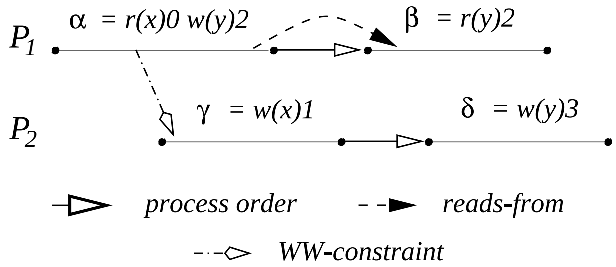

Figure 2: An execution history $\mathcal{H}_1$ under WW-constraint

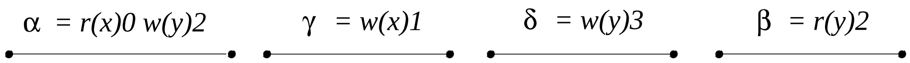

Figure 3: An extension of history $\mathcal{H}_1$ to a nonlegal history $\mathcal{S}_1$

A history under WW-constraint permits the m-operations, one of which only reads from an object and the other writes on the same object, to execute concurrently. Simply extending the partial order $\leadsto_{\mathcal{H}}$ to a total order may give sequential histories that are not legal. In Figure 2, the history $\mathcal{H}_1$ is under WW-constraint. One of the possible extensions of $\leadsto_{\mathcal{H}_1}$ gives us the sequential history $\mathcal{S}_1$, as in Figure 3, which is not legal. Therefore we define a *logical read-write* precedence, denoted by $\leadsto_{rw}$, between two such m-operations which are not ordered under $\leadsto_{\mathcal{H}}$. Let $\mathcal{H}$ be an execution history and let α, β and γ be m-operations that interfere in $\mathcal{H}$. Then $\beta \leadsto_{\mathcal{H}} \gamma \Rightarrow \alpha \leadsto_{rw} \gamma$. Formally,

(D 4.11) $$\alpha \leadsto^{\gamma}_{rw} \stackrel{def}{=} \langle \exists\, \beta : interfere(\mathcal{H}, \alpha, \beta, \gamma) : \beta \leadsto_{\mathcal{H}} \gamma \rangle$$

The intuition is that in any legal sequential history equivalent to $\mathcal{H}$, γ has to occur after α. We define an extended relation, denoted by $\leadsto^{+}_{\mathcal{H}}$, as,

(D 4.12) $$\leadsto^{+}_{\mathcal{H}} = (\leadsto_{\mathcal{H}} \cup \leadsto_{rw})^{+}$$

The natural question now is whether the extended relation, $\leadsto^{+}_{\mathcal{H}}$, is still an irreflexive partial order. Lemma 3 and Lemma 4 prove that legality is a sufficient condition for $\leadsto^{+}_{\mathcal{H}}$ to be irreflexive if the history is under OO- or WW-constraint.

Lemma 3 *Let $\mathcal{H}$ be a legal execution history under OO-constraint. Then $\rightsquigarrow_{\mathcal{H}}^{+}$ is an irreflexive transitive relation.*

Proof: We first show that $\rightsquigarrow_{rw} \subseteq \rightsquigarrow_{\mathcal{H}}$. Consider m-operations α, $\gamma \in op(\mathcal{H})$ such that $\alpha \rightsquigarrow_{rw} \gamma$. Then, by the definition of $\rightsquigarrow_{rw}$, there exists an m-operation β such that α, β and γ interfere in $\mathcal{H}$, and $\beta \rightsquigarrow_{\mathcal{H}} \gamma$. Then,

$$\begin{array}{lll} & \alpha \rightsquigarrow_{rw} \gamma & \\ \Rightarrow & (\alpha \rightsquigarrow_{rw} \gamma) \wedge (\beta \rightsquigarrow_{\mathcal{H}} \gamma) & \text{; D 4.11 (definition of } \rightsquigarrow_{rw}) \\ \Rightarrow & \mathit{conflict}(\alpha, \gamma) \wedge (\beta \rightsquigarrow_{\mathcal{H}} \gamma) & \text{; P 4.1} \\ \Rightarrow & ((\alpha \rightsquigarrow_{\mathcal{H}} \gamma) \vee (\gamma \rightsquigarrow_{\mathcal{H}} \alpha)) \wedge (\beta \rightsquigarrow_{\mathcal{H}} \gamma) & \text{; given } \mathcal{H} \text{ is under } OO\text{-constraint, D 4.8} \\ \equiv & ((\alpha \rightsquigarrow_{\mathcal{H}} \gamma) \wedge (\beta \rightsquigarrow_{\mathcal{H}} \gamma)) \vee ((\gamma \rightsquigarrow_{\mathcal{H}} \alpha) \wedge (\beta \rightsquigarrow_{\mathcal{H}} \gamma)) & \text{; distributivity of } \wedge \text{ over } \vee \\ \Rightarrow & (\alpha \rightsquigarrow_{\mathcal{H}} \gamma) \vee ((\beta \rightsquigarrow_{\mathcal{H}} \gamma) \wedge (\gamma \rightsquigarrow_{\mathcal{H}} \alpha)) & \text{; commutativity of } \wedge \text{, predicate calculus} \\ \Rightarrow & (\alpha \rightsquigarrow_{\mathcal{H}} \gamma) \vee \neg \mathit{legal}(\mathcal{H}) & \text{; D 4.6 (definition of legality)} \\ \Rightarrow & \alpha \rightsquigarrow_{\mathcal{H}} \gamma & \text{; given } \mathcal{H} \text{ is legal} \end{array}$$

Then, from D 4.12 we can conclude that $\rightsquigarrow_{\mathcal{H}}^{+} = \rightsquigarrow_{\mathcal{H}}$, and therefore $\rightsquigarrow_{\mathcal{H}}^{+}$ is an irreflexive and transitive relation. ■

Lemma 4 *Let $\mathcal{H}$ be a legal execution history under WW-constraint. Then $\rightsquigarrow_{\mathcal{H}}^{+}$ is an irreflexive transitive relation.*

Proof: We first prove that $\rightsquigarrow_{\mathcal{H}} \cup \rightsquigarrow_{rw}$ is acyclic. The proof is by induction on the number of pair of m-operations, n, ordered by $\rightsquigarrow_{rw}$ in a cycle. Note that since $\rightsquigarrow_{\mathcal{H}}$ is an irreflexive transitive relation, any cycle consists of at least one pair of m-operations ordered by $\rightsquigarrow_{rw}$.

Base case ($n = 1$): Any cycle is of the form $\alpha \rightsquigarrow_{rw} \gamma \rightsquigarrow_{\mathcal{H}} \alpha$. By definition of $\rightsquigarrow_{rw}$, there exists an m-operation β such that α, β and γ interfere in $\mathcal{H}$. Then,

$$\begin{array}{lll} & (\alpha \rightsquigarrow_{rw} \gamma) \wedge (\gamma \rightsquigarrow_{\mathcal{H}} \alpha) & \\ \Rightarrow & (\beta \rightsquigarrow_{\mathcal{H}} \gamma) \wedge (\gamma \rightsquigarrow_{\mathcal{H}} \alpha) & \text{; D 4.11 (definition of } \rightsquigarrow_{rw}) \\ \equiv & \neg(\neg(\beta \rightsquigarrow_{\mathcal{H}} \gamma) \vee \neg(\gamma \rightsquigarrow_{\mathcal{H}} \alpha)) & \text{; double negation, de morgan's law} \\ \Rightarrow & \neg \mathit{legal}(\mathcal{H}) & \text{; D 4.6 (definition of legality)} \end{array}$$

Hence $\mathcal{H}$ is not legal - a contradiction.

Induction Step ($n > 1$): Let $\mathit{path}(\alpha, \beta)$ denote the fact that there is a path from α to β consisting of a pair of m-operations ordered by $\rightsquigarrow_{\mathcal{H}}$ or $\rightsquigarrow_{rw}$. Let the cycle be denoted by $\alpha \rightsquigarrow_{rw} \beta \rightsquigarrow \cdots \rightsquigarrow \gamma \rightsquigarrow_{rw} \delta \rightsquigarrow \cdots \rightsquigarrow \alpha$, where $\rightsquigarrow$ represents either $\rightsquigarrow_{\mathcal{H}}$ or $\rightsquigarrow_{rw}$. Then,

$$
\begin{array}{lll}
& (\alpha \leadsto_{rw} \beta) \wedge path(\beta,\gamma) \wedge (\gamma \leadsto_{rw} \delta) \wedge path(\delta,\alpha) & \\
\Rightarrow & (\alpha \leadsto_{rw} \beta) \wedge path(\beta,\gamma) \wedge (\gamma \leadsto_{rw} \delta) \wedge path(\delta,\alpha) \wedge (wobjects(\beta) \neq \phi) \wedge (wobjects(\delta) \neq \phi) & \\
& & ;\text{ D 4.11 (definition of } \leadsto_{rw}\text{), P 4.1} \\
\Rightarrow & (\alpha \leadsto_{rw} \beta) \wedge path(\beta,\gamma) \wedge (\gamma \leadsto_{rw} \delta) \wedge path(\delta,\alpha) \wedge (\beta \leadsto_{\mathcal{H}} \delta) \vee (\delta \leadsto_{\mathcal{H}} \beta) & \\
& & ;\text{ given } \mathcal{H} \text{ is under } WW\text{-constraint, D 4.9} \\
\Rightarrow & ((\alpha \leadsto_{rw} \beta) \wedge (\beta \leadsto_{\mathcal{H}} \delta) \wedge path(\delta,\alpha)) \vee (path(\beta,\gamma) \wedge (\gamma \leadsto_{rw} \delta) \wedge (\delta \leadsto_{\mathcal{H}} \beta)) & \\
& & ;\text{ predicate calculus} \\
\Rightarrow & \neg legal(\mathcal{H}) & ;\text{ induction hypothesis}
\end{array}
$$

Hence $\mathcal{H}$ is not legal - a contradiction.
Thus, by induction we can conclude that $\leadsto_{\mathcal{H}} \cup \leadsto_{rw}$ is acyclic. Therefore $\leadsto_{\mathcal{H}}^{+} = (\leadsto_{\mathcal{H}} \cup \leadsto_{rw})^{+}$ is an irreflexive transitive relation. ■

We now show that $\leadsto_{\mathcal{H}}^{+}$ can be extended to any total order to obtain a legal sequential history equivalent to the history $\mathcal{H}$.

Lemma 5 *Let $\mathcal{H}$ be a legal execution history under WO-constraint. If $\leadsto_{\mathcal{H}}^{+}$ is an irreflexive transitive relation then $\mathcal{H}$ is admissible.*

Proof: Let $\mathcal{H}^{+}$ denote the history $\langle op(\mathcal{H}), \leadsto_{\mathcal{H}}^{+} \rangle$. We first prove a stronger result that any extension of $\mathcal{H}^{+}$ is legal if $\leadsto_{\mathcal{H}}^{+}$ is an irreflexive transitive relation and $\mathcal{H}$ is under WO-constraint. Formally,

(P 4.5) $(\leadsto_{\mathcal{H}}^{+}$ is an irreflexive transitive relation$) \wedge extends(\mathcal{G}, \mathcal{H}^{+}) \wedge WO(\mathcal{H}) \Rightarrow legal(\mathcal{G})$

The proof is as follows. Consider m-operations α, β, $\gamma \in op(\mathcal{H})$ that interfere in $\mathcal{H}$. Then,

$$
\begin{array}{lll}
& wobjects(\beta) \cap wobjects(\gamma) \neq \phi & ;\text{ P 4.1} \\
\Rightarrow & (\beta \leadsto_{\mathcal{H}} \gamma) \vee (\gamma \leadsto_{\mathcal{H}} \beta) & ;\text{ given } \mathcal{H} \text{ is under } WO\text{-constraint, D 4.10} \\
\Rightarrow & (\alpha \leadsto_{rw} \gamma) \vee (\gamma \leadsto_{\mathcal{H}} \beta) & ;\text{ D 4.11 (definition of } \leadsto_{rw}\text{)} \\
\Rightarrow & (\alpha \leadsto_{\mathcal{G}} \gamma) \vee (\gamma \leadsto_{\mathcal{G}} \beta) & ;\ \leadsto_{\mathcal{G}} \supseteq (\leadsto_{\mathcal{H}} \cup \leadsto_{rw}) \\
\Rightarrow & \neg(\gamma \leadsto_{\mathcal{G}} \alpha) \vee \neg(\beta \leadsto_{\mathcal{G}} \gamma) & ;\ \leadsto_{\mathcal{G}} \text{ is an irreflexive transitive relation}
\end{array}
$$

Thus, from D 4.6 we can conclude that $\mathcal{G}$ is legal. We now prove the lemma. Let us extend $\leadsto_{\mathcal{H}}^{+}$ to any total order, say $\leadsto_{\mathcal{S}}$, and denote the resulting history by $\mathcal{S}$. Then,

$$
\begin{array}{lll}
& extends(\mathcal{S}, \mathcal{H}^{+}) \wedge sequential(\mathcal{S}) & \\
\Rightarrow & extends(\mathcal{S}, \mathcal{H}^{+}) \wedge sequential(\mathcal{S}) \wedge legal(\mathcal{S}) & ;\text{ given } \mathcal{H} \text{ is under } WO\text{-constraint, P 4.5} \\
\Rightarrow & extends(\mathcal{S}, \mathcal{H}) \wedge sequential(\mathcal{S}) \wedge legal(\mathcal{S}) & ;\ extends(\mathcal{H}^{+}, \mathcal{H}),\text{ P 4.4} \\
\Rightarrow & \langle \exists\, S : extends(\mathcal{S}, \mathcal{H}) \wedge sequential(\mathcal{S}) : legal(\mathcal{S}) \rangle & ;\text{ predicate calculus} \\
\equiv & admissible(\mathcal{H}) & ;\text{ D 4.7 (definition of admissibility)}
\end{array}
$$

Hence $\mathcal{H}$ is admissible. ■

Lemma 3, Lemma 4 and Lemma 5 establish that legality is a sufficient condition for a history under OO- or WW-constraint to be admissible. Lemma 6 show that legality is also necessary for admissibility.

Lemma 6 *Let $\mathcal{H}$ be an execution history. If $\mathcal{H}$ is admissible then it is legal.*

Proof: We first prove a stronger result that if any extension of $\mathcal{H}$ is legal then $\mathcal{H}$ is legal. Formally,

(P 4.6) $$extends(\mathcal{G},\mathcal{H}) \wedge legal(\mathcal{G}) \Rightarrow legal(\mathcal{H})$$

The proof is as follows. Consider m-operations α, β, $\gamma \in op(\mathcal{G})$ that interfere in $\mathcal{G}$. Then,

$$\begin{array}{lll} & \neg(\beta \leadsto_{\mathcal{G}} \gamma) \vee \neg(\gamma \leadsto_{\mathcal{G}} \alpha) & \text{; given } \mathcal{G} \text{ is legal, D 4.6} \\ \Rightarrow & \neg(\beta \leadsto_{\mathcal{H}} \gamma) \vee \neg(\gamma \leadsto_{\mathcal{H}} \alpha) & ;\ \leadsto_{\mathcal{G}} \supseteq \leadsto_{\mathcal{H}} \end{array}$$

Hence, using P 4.3 we can infer that $\mathcal{H}$ is legal. We now prove the lemma.

$$\begin{array}{lll} & admissible(\mathcal{H}) & \text{; given } \mathcal{H} \text{ is admissible} \\ \equiv & \langle \exists\, \mathcal{S} : extends(\mathcal{S},\mathcal{H}) \wedge sequential(\mathcal{S}) : legal(\mathcal{S}) \rangle & \text{; D 4.7 (definition of admissibility)} \\ \Rightarrow & legal(\mathcal{H}) & \text{; P 4.6} \end{array}$$

Hence $\mathcal{H}$ is legal. ■

The next theorem combines the results of Lemma 3-6.

Theorem 7 *Let $\mathcal{H}$ be an execution history under OO- or WW-constraint. Then $\mathcal{H}$ is admissible if and only if it is legal.*

Proof: From Lemma 3 and Lemma 5, we can infer that legality is sufficient for a history under OO-constraint to be admissible. From Lemma 4 and Lemma 5, we can conclude that legality is also sufficient for a history under WW-constraint to be admissible. Lemma 6 implies that legality is necessary for a admissibility. Thus, legality is both necessary and sufficient for a history under OO - or WW-constraint to be admissible.

The next section illustrates how WW-constraint can be used effectively to implement m-sequential consistency and m-linearizability in an asynchronous distributed system.

5 Implementation of Consistency Conditions

Our protocols for implementing consistency conditions introduced in Section 2.3 are based on WW-constraint. The protocols assume that processes and channels are reliable and a message sent is eventually received. However, the messages can get reordered. As discussed in Section 4, to ensure that the execution follows WW-constraint the system need to synchronize all update m-operations. We use atomic broadcast to achieve our objective. In general, the system may not know beforehand the set of objects an m-operation will access during execution. In fact, the set of objects read and written by an

m-operation may actually depend on the values read during its execution. We take a conservative approach and treat an m-operation as an update m-operation if it can potentially write to some object.

In our protocols, each process keeps a local copy of every shared object. On receiving an atomic broadcast, the process applies the m-operation to its local copy. The legality of the read operations of an update m-operation is maintained since atomic broadcast ensures that all processes apply all update m-operations in the same order. The algorithm for maintaining the legality of read operations of a query m-operation depends on the consistency condition in consideration.

Before describing the protocols, we present the properties that the protocols should satisfy to be correct. Let $\mathcal{H} = \langle op(\mathcal{H}), \leadsto_{\mathcal{H}} \rangle$ be an execution history and $\leadsto_{\mathcal{H}}^{-}$ be an irreflexive relation defined on $op(\mathcal{H})$, where $\leadsto_{\mathcal{H}}$ is the irreflexive transitive closure of $\leadsto_{\mathcal{H}}^{-}$, such that $\leadsto_{\mathcal{H}}^{-}$ satisfies the properties,

(P 5.1) $$(\beta \leadsto_{\mathcal{H}}^{-} \alpha) \wedge (wobjects(\beta) = \phi) \wedge (wobjects(\alpha) = \phi) \Rightarrow \beta \leadsto_{t} \alpha$$

(P 5.2) $$(wobjects(\beta) \neq \phi) \wedge (wobjects(\alpha) \neq \phi) \Rightarrow (\alpha \leadsto_{\mathcal{H}}^{-} \beta) \vee (\beta \leadsto_{\mathcal{H}}^{-} \alpha)$$

We associate a *timestamp* with every m-operation. The timestamp is a vector of integers with one entry for every object. Intuitively, it represents the version of an object. Two timestamps are equal iff their corresponding entries are identical. We order timestamps lexicographically. A timestamp ts is less than or equal to timestamp ts', denoted by $ts \preceq ts'$, iff every entry of ts is less than or equal to the corresponding entry of ts'. A timestamp ts is less than timestamp ts', denoted by $ts \prec ts'$, iff ts is less than or equal to ts' and they are not equal.

Let $ts(\alpha)$ denote the timestamp associated with an m-operation α. Lemma 8 gives the properties of $\leadsto_{\mathcal{H}}^{-}$ and ts that ensure that the execution is under WW-constraint. The property P 5.3 states that ts is monotonic with respect to $\leadsto_{\mathcal{H}}^{-}$. The properties P 5.4 and P 5.2 imply that every write to an object establishes a new version for that object. From property P 5.1 we can infer that $\leadsto_{\mathcal{H}}^{-}$ orders query m-operations only when necessary. Lemma 9 states the additional properties needed to ensure the legality of all m-operations. The properties P 5.7 and P 5.8 ensure that only a write can create new versions.

Lemma 8 *If $\leadsto_{\mathcal{H}}^{-}$ and ts satisfy the properties,*

(P 5.3) $$\beta \leadsto_{\mathcal{H}}^{-} \alpha \Rightarrow ts(\beta) \preceq ts(\alpha)$$

(P 5.4) $$(\beta \leadsto_{\mathcal{H}}^{-} \alpha) \wedge (x \in wobjects(\alpha)) \Rightarrow ts(\beta)[x] < ts(\alpha)[x]$$

then $\mathcal{H}$ is under WW-constraint.

Proof: It is easy to prove that properties P 5.3 and P 5.4 also hold for $\leadsto_{\mathcal{H}}$. Formally,

(P 5.5) $$\beta \leadsto_{\mathcal{H}} \alpha \Rightarrow ts(\beta) \preceq ts(\alpha)$$

(P 5.6) $$(\beta \leadsto_{\mathcal{H}} \alpha) \wedge (x \in wobjects(\alpha)) \Rightarrow ts(\beta)[x] < ts(\alpha)[x]$$

We first show that $\leadsto_{\mathcal{H}}^{-}$ is acyclic. Assume, on the contrary, that $\leadsto_{\mathcal{H}}^{-}$ contains a cycle, say $\mathcal{C}$. Let $op(\mathcal{C})$ denote the set of m-operations involved in $\mathcal{C}$. Note that any cycle contains at least two m-operations since $\leadsto_{\mathcal{H}}^{-}$ is irreflexive. Assume that α, $\beta \in op(\mathcal{C})$. There are two cases to consider: $op(\mathcal{C})$ contains an update m-operation ($\langle \exists\, \gamma \in op(\mathcal{C}) :: wobjects(\gamma) \neq \phi \rangle$) or there are no update m-operations in $op(\mathcal{C})$ ($\langle \forall\, \gamma \in op(\mathcal{C}) :: wobjects(\gamma) = \phi \rangle$).

Case 1 $[\langle \exists\, \gamma \in op(\mathcal{C}) :: wobjects(\gamma) \neq \phi \rangle]$**:** Without loss of generality, let β be the m-operation with $wobjects(\beta) \neq \phi$. Then,

$$
\begin{array}{lll}
 & (\alpha \rightsquigarrow_{\mathcal{H}} \beta \rightsquigarrow_{\mathcal{H}} \alpha) \wedge (wobjects(\beta) \neq \phi) & ;\ \text{assumption} \\
\equiv & (\alpha \rightsquigarrow_{\mathcal{H}} \beta \rightsquigarrow_{\mathcal{H}} \alpha) \wedge \langle \exists\, x :: x \in wobjects(\beta) \rangle & ; \\
\Rightarrow & (ts(\alpha) \preceq ts(\beta) \preceq ts(\alpha)) \wedge \langle \exists\, x :: ts(\alpha)[x] < ts(\beta)[x] \rangle & ;\ \text{P 5.5, P 5.6} \\
\Rightarrow & ts(\alpha) \prec ts(\beta) \preceq ts(\alpha) & ; \\
\equiv & false & ;\ \text{contradiction}
\end{array}
$$

Case 2 $[\langle \forall\, \gamma \in op(\mathcal{C}) :: wobjects(\gamma) = \phi \rangle]$**:** Since the cycle $\mathcal{C}$ does not contain any update m-operation, from P 5.1 we can conclude that the cycle will have all the pair of m-operations ordered by $\rightsquigarrow_t$. Furthermore, since $\rightsquigarrow_t$ is a transitive relation therefore $\alpha \rightsquigarrow_t \alpha$. Thus $resp(\alpha) < inv(\alpha)$ - a contradiction.
Therefore $\rightsquigarrow_{\mathcal{H}}^{-}$ is acyclic and hence $\rightsquigarrow_{\mathcal{H}}$ is an irreflexive transitive relation. Thus, $\mathcal{H}$ is indeed a valid execution history. Furthermore, using P 5.2 we can infer that $\mathcal{H}$ is under WW-constraint. ■

Lemma 9 *If ts satisfies P 5.5, P 5.6 and the properties,*

(P 5.7) $$(x \in rfobjects(\mathcal{H}, \alpha, \beta)) \wedge (x \notin wobjects(\alpha)) \Rightarrow ts(\beta)[x] = ts(\alpha)[x]$$

(P 5.8) $$(x \in rfobjects(\mathcal{H}, \alpha, \beta)) \wedge (x \in wobjects(\alpha)) \Rightarrow ts(\beta)[x] = ts(\alpha)[x] - 1$$

then $\mathcal{H}$ is legal.

Proof: Consider m-operations α, β, $\gamma \in op(\mathcal{H})$ and let x denote a shared object. We first prove that if α, β, γ interfere in $\mathcal{H}$ on an object x then γ cannot be ordered between β and α. Formally,

(P 5.9) $$(x \in rfobjects(\mathcal{H}, \alpha, \beta)) \wedge (x \in wobjects(\gamma)) \wedge (\beta \rightsquigarrow_{\mathcal{H}} \gamma \rightsquigarrow_{\mathcal{H}} \alpha) \Rightarrow false$$

There are two cases to consider: α does not write to x ($x \notin wobjects(\alpha)$) or α writes to x ($x \in wobjects(\alpha)$).

Case 1 $[x \notin wobjects(\alpha)]$**:**

$$
\begin{array}{lll}
 & (x \in rfobjects(\mathcal{H}, \alpha, \beta)) \wedge (x \in wobjects(\gamma)) \wedge (x \notin wobjects(\alpha)) \wedge (\beta \rightsquigarrow_{\mathcal{H}} \gamma \rightsquigarrow_{\mathcal{H}} \alpha) & \\
\Rightarrow & (ts(\beta)[x] = ts(\alpha)[x]) \wedge (ts(\beta)[x] < ts(\gamma)[x] \leq ts(\alpha)[x]); & \text{P 5.7, P 5.5, P 5.6} \\
\Rightarrow & ts(\beta)[x] < ts(\beta)[x] & ; \\
\equiv & false & ;\ \text{contradiction}
\end{array}
$$

Case 2 $[x \in wobjects(\alpha))]$**:**

$$
\begin{array}{lll}
 & (x \in rfobjects(\mathcal{H},\alpha,\beta)) \wedge (x \in wobjects(\gamma)) \wedge (x \in wobjects(\alpha)) \wedge (\beta \leadsto_{\mathcal{H}} \gamma \leadsto_{\mathcal{H}} \alpha) & \\
\Rightarrow & (ts(\beta)[x] = ts(\alpha)[x] - 1) \wedge (ts(\beta)[x] < ts(\gamma)[x] < ts(\alpha)[x]) & \\
 & & ;\text{ P 5.8, P 5.5, P 5.6} \\
\Rightarrow & ts(\beta)[x] < ts(\gamma)[x] < ts(\beta)[x] + 1 & ; \\
\Rightarrow & ts(\beta)[x] < ts(\beta)[x] & ; \\
\equiv & false & ;\text{ contradiction}
\end{array}
$$

Therefore,

$$
\begin{array}{lll}
 & \langle \exists\, x :: (x \in rfobjects(\mathcal{H},\alpha,\beta)) \wedge (x \in wobjects(\gamma)) \rangle \wedge (\beta \leadsto_{\mathcal{H}} \gamma \leadsto_{\mathcal{H}} \alpha) \Rightarrow false & \\
 & & ;\text{ conjunction of P 5.9 over all } x \\
\equiv & interfere(\mathcal{H},\alpha,\beta,\gamma) \wedge (\beta \leadsto_{\mathcal{H}} \gamma \leadsto_{\mathcal{H}} \alpha) \Rightarrow false & ;\text{ D 4.2} \\
\equiv & interfere(\mathcal{H},\alpha,\beta,\gamma) \Rightarrow \neg(\beta \leadsto_{\mathcal{H}} \gamma) \vee \neg(\gamma \leadsto_{\mathcal{H}} \alpha) & ;\text{ predicate calculus}
\end{array}
$$

Hence we can conclude from D 4.6 that $\mathcal{H}$ is legal. ■

We combine the results of Lemma 8 and Lemma 9 in Theorem 10

Theorem 10 *If $\leadsto_{\mathcal{H}}^{-}$ and ts satisfy the properties P 5.1-5.4 and P 5.7-5.8 then $\mathcal{H}$ is admissible.*

Proof: Since $\leadsto_{\mathcal{H}}^{-}$ and ts satisfy the properties P 5.1-5.4 therefore from Lemma 8 we can infer that $\mathcal{H}$ is under WW-constraint. Furthermore, since $\leadsto_{\mathcal{H}}^{-}$ and ts also satisfy the properties P 5.7 and P 5.8 therefore from Lemma 9 we can conclude that $\mathcal{H}$ is legal. Finally, using Theorem 7 we can conclude that $\mathcal{H}$ is admissible. ■

5.1 Implementation of m-Sequential Consistency

Our protocol for m-sequential consistency is an extension of Welch and Attiya's protocol [4]. It consists of three actions, each of which is performed locally and atomically. When a process issues an update m-operation, it atomically broadcasts it to all processes (A1). On receiving atomic broadcast of an m-operation, the process applies it to its local copy of the shared objects (A2). On the other hand, a query m-operation simply reads from the local copy of its issuing process (A3). The protocol is formally described in Figure 4. The statements in curly braces are not part of the protocol but are merely used to establish its correctness. Before proving the correctness of the protocol we give some definitions.

Let $\mathcal{H}$ be an execution generated by the protocol in Figure 4. Let e be an event of process P_k. We define the timestamp, ts, associated with an event as,

$$ts(e) \stackrel{def}{=} ts \text{ of } P_k \text{ on occurence of } e$$

Let e and f be the events on the same process. Since ts of any process never decreases,

(P 5.10) $\quad e < f \Rightarrow ts(e) \preceq ts(f) \quad$ (**monotonicity**)

P_i ::
var
  X : array of shared objects, initially $\perp$
  $\{ts$: array$[1..|\mathcal{X}|]$ of integer, initially $0\}$

(A1) On invocation of an m-operation α such that potentially $wobjects(\alpha) \neq \phi$;
    atomically broadcast α to all processes;

(A2) On receiving atomic broadcast of α from P_k;
    apply α to X;
    $\{\forall\, x : x \in wobjects(\alpha) : ts[x]++\}$
    if $proc(\alpha) = P_i$ then
      generate response for α;
    endif;

(A3) On invocation of an m-operation α such that $wobjects(\alpha) = \phi$;
    apply α to X;
    generate response for α;

Figure 4: Implementation of m-Sequential Consistency

In the discussion that follows, consider distinct m-operations α, $\beta \in op(\mathcal{H})$. Let P_i denote the process that issued α and P_k be any process. Let x denote a shared object and $abcast(\alpha)$ denote the fact that α was atomically broadcast in the execution. Furthermore, any property $P(k)$ involving P_k actually implies $\langle \forall\, k : 1 \leq k \leq n : P(k) \rangle$. In addition to $inv(\alpha)$ and $resp(\alpha)$ events, we define $start(k, \alpha)$ and $finish(k, \alpha)$ events for an m-operation α and process P_k. If α is atomically broadcast then $start(k, \alpha)$ and $finish(k, \alpha)$ are defined for every process, otherwise they are only defined for the process that issued α.

$$inv(\alpha) \stackrel{def}{=} \begin{cases} \text{if } abcast(\alpha) \text{ then start event of action (A1) at } P_i \text{ for } \alpha \\ \text{else start event of action (A3) at } P_i \text{ for } \alpha \end{cases}$$

$$resp(\alpha) \stackrel{def}{=} \begin{cases} \text{if } abcast(\alpha) \text{ then finish event of action (A2) at } P_i \text{ for } \alpha \\ \text{else finish event of action (A3) at } P_i \text{ for } \alpha \end{cases}$$

$$start(k, \alpha) \stackrel{def}{=} \begin{cases} \text{if } abcast(\alpha) \text{ then start event of action (A2) at } P_k \text{ for } \alpha \\ \text{else if } P_k = P_i \text{ then start event of action (A3) at } P_k \text{ for } \alpha \end{cases}$$

$$finish(k, \alpha) \stackrel{def}{=} \begin{cases} \text{if } abcast(\alpha) \text{ then finish event of action (A2) at } P_k \text{ for } \alpha \\ \text{else if } P_k = P_i \text{ then finish event of action (A3) at } P_k \text{ for } \alpha \end{cases}$$

Figure 5 illustrates the working of the protocol and labels the various events defined before. It can be easily verified from the protocol that the following property is true.

(P 5.11) $$inv(\alpha) \leq start(i, \alpha) < finish(i, \alpha) \leq resp(\alpha)$$

Let $\rightsquigarrow_{ww}$ denote the order in which update m-operations are atomically broadcast. Then,

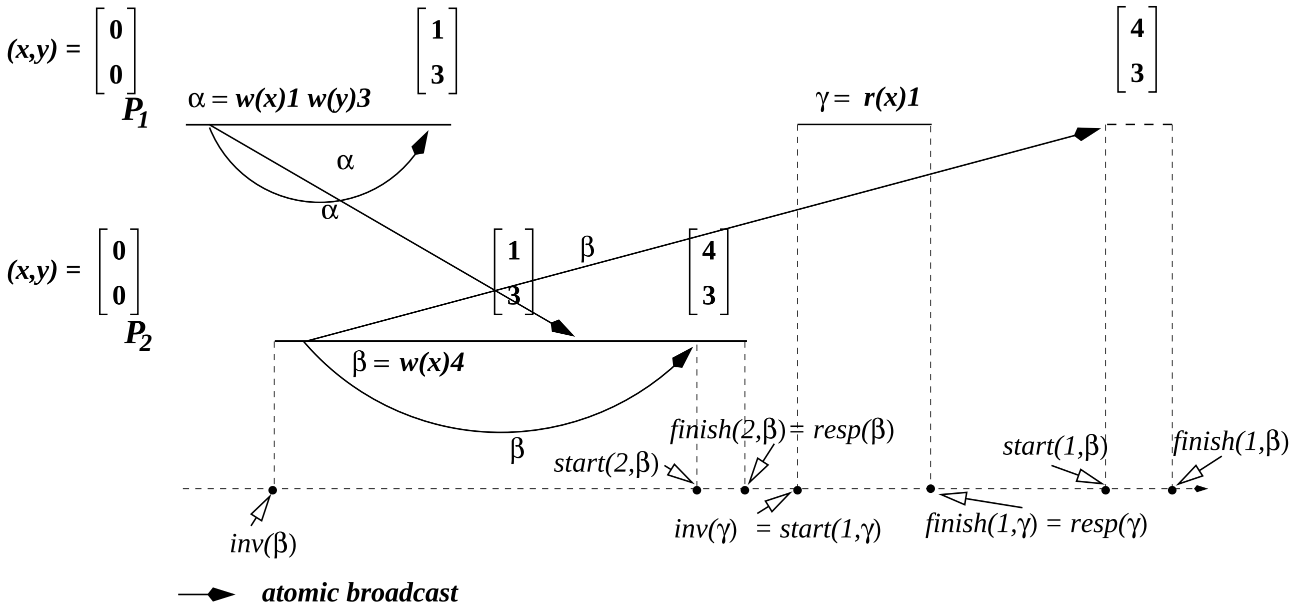

Figure 5: An example execution of the protocol in Figure 4

(P 5.12) $$\mathit{wobjects}(\alpha) \neq \phi \Rightarrow \mathit{abcast}(\alpha)$$

(P 5.13) $$(\mathit{wobjects}(\alpha) \neq \phi) \wedge (\mathit{wobjects}(\beta) \neq \phi) \equiv (\beta \rightsquigarrow_{ww} \alpha) \vee (\alpha \rightsquigarrow_{ww} \beta)$$

(P 5.14) $$\beta \rightsquigarrow_{ww} \alpha \Rightarrow \mathit{start}(k, \beta) < \mathit{start}(k, \alpha)$$

Note that ts of any process is updated only in action (A2), and any two m-operations that are atomically broadcast are executed in the same order by all processes. Therefore the timestamps of "start" and "finish" events, if α is atomically broadcast, are respectively identical for all processes. Formally,

(P 5.15) $$\mathit{abcast}(\alpha) \Rightarrow (ts(\mathit{start}(i, \alpha)) = ts(\mathit{start}(k, \alpha))) \wedge (ts(\mathit{finish}(i, \alpha)) = ts(\mathit{finish}(k, \alpha)))$$

We can use this property to abbreviate $ts(\mathit{start}(k, \alpha))$ and $ts(\mathit{finish}(k, \alpha))$ as $ts(\mathit{start}(\alpha))$ and $ts(\mathit{finish}(\alpha))$ respectively when α is atomically broadcast. Furthermore, when α is not atomically broadcast we use $ts(\mathit{start}(\alpha))$ and $ts(\mathit{finish}(\alpha))$ as a shorthand for $ts(\mathit{start}(i, \alpha))$ and $ts(\mathit{finish}(i, \alpha))$ respectively.

Intuitively, α reads from β the value of object x if no other operation writes to x after β has written to x and before α reads from x. In other words, α reads the version of x written by β. We can use the timestamp ts to capture this notion. The value of ts at the "start" event of α captures the version of the objects read by α. Moreover, if α writes to an object the version of that object in ts is incremented by 1. Thus, the value of ts at the "finish" event gives the new version of the written objects. Formally, the reads-from relation $\rightsquigarrow_{rf}$ can be defined using D 4.3 as follows,

(D 5.1) $$\begin{aligned} x \in \mathit{rfobjects}(\mathcal{H}, \alpha, \beta) \stackrel{def}{=} & (x \in \mathit{robjects}(\alpha)) \wedge (x \in \mathit{wobjects}(\beta)) \wedge \\ & (ts(\mathit{finish}(\beta))[x] = ts(\mathit{start}(\alpha))[x]) \end{aligned}$$

Since after application of α in action (A2) the components of ts for which α performs a write are incremented by one, therefore,

(P 5.16) $$x \notin wobjects(\alpha) \Rightarrow ts(start(\alpha))[x] = ts(finish(\alpha))[x]$$

(P 5.17) $$x \in wobjects(\alpha) \Rightarrow ts(start(\alpha))[x] = ts(finish(\alpha))[x] - 1$$

We define the timestamp, ts, associated with an m-operation α as follows,

(D 5.2) $$ts(\alpha) \stackrel{def}{=} ts(finish(\alpha))$$

We define $\rightsquigarrow_{\mathcal{H}}^{-}$ as,

(D 5.3) $$\rightsquigarrow_{\mathcal{H}}^{-} \stackrel{def}{=} \rightsquigarrow_P \cup \rightsquigarrow_{rf} \cup \rightsquigarrow_{ww}$$

Now we prove that the protocol in Figure 4 implements m-sequential consistency. Let $\mathcal{H}$ be an execution history generated by the protocol. For the proofs that follow consider distinct m-operations α, $\beta \in op(\mathcal{H})$. Let $proc(\alpha) = P_i$ and x be a shared object.

Lemma 11 *If $\beta \rightsquigarrow_{\mathcal{H}}^{-} \alpha$ then $finish(i, \beta)$ is defined. Furthermore, $finish(i, \beta) < start(i, \alpha)$.*

Proof: Intuitively, the lemma says that if β is ordered before α then β is applied to P_i's local copy before α. There are three cases to consider: $\beta \rightsquigarrow_P \alpha$ or $\beta \rightsquigarrow_{rf} \alpha$ or $\beta \rightsquigarrow_{ww} \alpha$.

Case 1 $[\beta \rightsquigarrow_P \alpha]$**:** $finish(i, \beta)$ is defined since $proc(\alpha) = proc(\beta) = P_i$.

$$\begin{array}{lll} & \beta \rightsquigarrow_P \alpha & \\ \Rightarrow & resp(\beta) < inv(\alpha) & \text{; P 4.2 (well-formedness of history)} \\ \Rightarrow & finish(i, \beta) < start(i, \alpha) & \text{; P 5.11, } proc(\beta) = P_i \end{array}$$

Case 2 $[\beta \rightsquigarrow_{rf} \alpha]$**:** Using D 4.3 and D 5.1, we can infer that $wobjects(\beta) \neq \phi$. From P 5.12, we can conclude that β is atomically broadcast and hence $finish(i, \beta)$ is defined. Without loss of generality, assume $x \in rfobjects(\mathcal{H}, \alpha, \beta)$. Therefore $x \in wobjects(\beta)$.

$$\begin{array}{lll} & start(i, \alpha) < finish(i, \beta) & \text{; assumption} \\ \Rightarrow & start(i, \alpha) < start(i, \beta) < finish(i, \beta) & \text{; atomicity of actions} \\ \Rightarrow & ts(start(\alpha)) \preceq ts(start(\beta)) \preceq ts(finish(\beta)) & \text{; P 5.10 (montonicity of } ts) \\ \Rightarrow & (ts(start(\alpha)) \preceq ts(start(\beta)) \preceq ts(finish(\beta))) \wedge (ts(start(\beta))[x] < ts(finish(\beta))[x]) & \\ & & \text{; P 5.17 (}\beta \text{ writes to } x) \\ \Rightarrow & ts(start(i)\alpha)[x] < ts(finish(i)\beta)[x]) & \text{; simplification} \\ \Rightarrow & (ts(start(\alpha))[x] < ts(finish(\beta))) \wedge (ts(finish(\beta))[x] = ts(start(\alpha))[x]) & \end{array}$$

$$; \text{D 5.1 } (\alpha \text{ reads the value of } x \text{ from } \beta)$$

$$\equiv \; false \qquad ; \text{contradiction}$$

Thus, we can conclude that $finish(i, \beta) < start(i, \alpha)$.

Case 3 [$\beta \rightsquigarrow_{ww} \alpha$]**:** From P 5.13, we can infer that $wobjects(\beta) \neq \phi$. Therefore as discussed in previous case $finish(i, \beta)$ is defined.

$$\begin{array}{lll}
 & \beta \rightsquigarrow_{ww} \alpha & \\
\Rightarrow & start(i, \beta) < start(i, \alpha) & ; \text{P 5.14 } (\beta \text{ is received before } \alpha \text{ at } P_i) \\
\Rightarrow & finish(i, \beta) < start(i, \alpha) & ; \text{atomicity of actions}
\end{array}$$

Hence if $\beta \rightsquigarrow_{\mathcal{H}}^{-} \alpha$ then $finish(i, \beta)$ is defined and $finish(i, \beta) < start(i, \alpha)$. ■

Lemma 12 *The protocol in Figure 4 satisfies P 5.1 and P 5.2.*

Proof: The proof of property P 5.1 is as follows,

$$\begin{array}{lll}
 & (\beta \rightsquigarrow_{\mathcal{H}}^{-} \alpha) \wedge (wobjects(\alpha) = \phi) \wedge (wobjects(\beta) = \phi) & \\
\Rightarrow & ((\beta \rightsquigarrow_{P} \alpha) \vee (\beta \rightsquigarrow_{rf} \alpha) \vee (\beta \rightsquigarrow_{ww} \alpha)) \wedge (wobjects(\beta) = \phi) & \\
 & & ; \text{D 5.3 (definition of } \rightsquigarrow_{\mathcal{H}}^{-}) \\
\Rightarrow & ((\beta \rightsquigarrow_{P} \alpha) \vee (wobjects(\beta) \neq \phi) \vee (wobjects(\beta) \neq \phi)) \wedge (wobjects(\beta) = \phi) & \\
 & & ; \text{D 5.1, definition of } \rightsquigarrow_{ww} \\
\Rightarrow & \beta \rightsquigarrow_{P} \alpha & ; \text{distribution of } \wedge \text{ over } \vee \text{, contradiction} \\
\Rightarrow & \beta \rightsquigarrow_{t} \alpha & ; \text{P 4.2 (well-formedness of history)}
\end{array}$$

The property P 5.2 can be proved as follows,

$$\begin{array}{lll}
 & (wobjects(\alpha) \neq \phi) \wedge (wobjects(\beta) \neq \phi) & \\
\equiv & (\alpha \rightsquigarrow_{ww} \beta) \vee (\beta \rightsquigarrow_{ww} \alpha) & ; \text{P 5.13} \\
\Rightarrow & (\alpha \rightsquigarrow_{\mathcal{H}}^{-} \beta) \vee (\beta \rightsquigarrow_{\mathcal{H}}^{-} \alpha) & ; \text{D 5.3 } (\rightsquigarrow_{\mathcal{H}}^{-} \text{ contains } \rightsquigarrow_{ww})
\end{array}$$

Hence P 5.1 and P 5.2 are satisfied by the protocol in Figure 4. ■

Lemma 13 *The protocol in Figure 4 satisfies P 5.3 and P 5.4.*

Proof: The property P 5.3 can be proved as follows,

$$
\begin{array}{lll}
 & \beta \rightsquigarrow_{\mathcal{H}}^{-} \alpha & \\
\Rightarrow & \mathit{finish}(i,\beta) < \mathit{start}(i,\alpha) & \text{; Lemma 11 (}\beta\text{ is applied before }\alpha\text{ at }P_i\text{)} \\
\Rightarrow & \mathit{finish}(i,\beta) < \mathit{start}(i,\alpha) < \mathit{finish}(i,\alpha) & ; \\
\Rightarrow & ts(\mathit{finish}(\beta)) \preceq ts(\mathit{finish}(\alpha)) & \text{; P 5.10 (montonicity of } ts\text{)} \\
\equiv & ts(\beta) \preceq ts(\alpha) & \text{; D 5.2}
\end{array}
$$

The proof of property P 5.4 is as follows,

$$
\begin{array}{lll}
 & (\beta \rightsquigarrow_{\mathcal{H}}^{-} \alpha) \wedge (x \in wobjects(\alpha)) & \\
\Rightarrow & (\mathit{finish}(i,\beta) < \mathit{start}(i,\alpha)) \wedge (x \in wobjects(\alpha)) & \text{; Lemma 11 (}\beta\text{ is applied before }\alpha\text{ at }P_i\text{)} \\
\Rightarrow & (ts(\mathit{finish}(\beta)) \preceq ts(\mathit{start}(\alpha))) \wedge (ts(\mathit{start}(\alpha))[x] < ts(\mathit{finish}(\alpha))[x]) & \\
 & & \text{; P 5.10 (montonicity of } ts\text{), P 5.17} \\
\Rightarrow & ts(\mathit{finish}(\beta))[x] \leq ts(\mathit{start}(\alpha))[x] < ts(\mathit{finish}(\alpha))[x]) & \text{; simplification} \\
\equiv & ts(\beta)[x] < ts(\alpha)[x] & \text{; D 5.2}
\end{array}
$$

Hence P 5.3 and P 5.4 are satisfied by the protocol in Figure 4. ■

Lemma 14 *The protocol in Figure 4 satisfies P 5.7 and P 5.8.*

Proof: The proof of property P 5.7 is as follows,

$$
\begin{array}{lll}
 & (x \in rfobjects(\mathcal{H},\alpha,\beta)) \wedge (x \notin wobjects(\alpha)) & \\
\Rightarrow & (ts(\mathit{finish}(\beta))[x] = ts(\mathit{start}(\alpha))[x])) \wedge (x \notin wobjects(\alpha)) & \\
 & & \text{; D 5.1 (}\alpha\text{ reads the value of }x\text{ from }\beta\text{)} \\
\Rightarrow & (ts(\mathit{finish}(\beta))[x] = ts(\mathit{start}(\alpha))[x]) \wedge (ts(\mathit{start}(\alpha))[x] = ts(\mathit{finish}(\alpha))[x]) & \\
 & & \text{; P 5.16 (}\alpha\text{ does not write to }x\text{)} \\
\Rightarrow & ts(\mathit{finish}(\beta))[x] = ts(\mathit{finish}(\alpha))[x] & \text{; simplification} \\
\equiv & ts(\beta)[x] = ts(\alpha)[x] & \text{; D 5.2}
\end{array}
$$

The property P 5.8 can be proved as follows,

$$
\begin{array}{lll}
 & (x \in rfobjects(\mathcal{H},\alpha,\beta)) \wedge (x \in wobjects(\alpha)) & \\
\Rightarrow & (ts(\mathit{finish}(\beta))[x] = ts(\mathit{start}(\alpha))[x])) \wedge (x \in wobjects(\alpha)) & \\
 & & \text{; D 5.1 (}\alpha\text{ reads the value of }x\text{ from }\beta\text{)} \\
\Rightarrow & (ts(\mathit{finish}(\beta))[x] = ts(\mathit{start}(\alpha))[x]) \wedge (ts(\mathit{start}(\alpha))[x] = ts(\mathit{finish}(\alpha))[x] - 1) &
\end{array}
$$

$$; \text{P 5.17 } (\alpha \text{ writes to } x)$$

$$\Rightarrow \quad ts(\mathit{finish}(\beta))[x] = ts(\mathit{finish}(\alpha))[x] - 1 \qquad ; \text{simplification}$$

$$\equiv \quad ts(\beta)[x] = ts(\alpha)[x] - 1 \qquad ; \text{D 5.2}$$

Hence P 5.7 and P 5.8 are satisfied by the protocol in Figure 4. ∎

Theorem 15 *All the executions generated by the protocol in Figure 4 are m-sequentially consistent.*

Proof: From Lemma 12-14, we can conclude that the protocol in Figure 4 satisfy properties P 5.1-5.4 and P 5.7-5.8. Therefore, from Theorem 10, we can infer that all the executions generated by the protocol in Figure 4 are m-sequentially consistent. ∎

5.2 Implementation of m-Linearizability

Our protocol for m-linearizability consists of six actions, each of which is performed locally and atomically. The actions on invocation of an update m-operations are identical to those in the protocol for m-sequential consistency (A1 and A2). But to ensure that a query m-operation does not read a stale value, the process sends a "query" message to all processes asking them for their copy of the shared objects and the associated timestamps (A3). On receiving a "query" message the process sends its local copy along with the timestamps to the requesting process (A4). The requesting process then constructs a copy of the shared objects from the information received selecting the most recent version for all objects (A5). Finally, when responses from all processes have been received the query m-operation reads from the copy constructed (A6). The protocol is formally described in Figure 6. Again as in Section 5.1, before proving the correctness of the protocol we give some definitions.

Let $\mathcal{H}$ be an execution generated by the protocol in Figure 6. Let e be an event of process P_k. We define the timestamps, *myts* and *othts*, associated with an event as,

$$myts(e) \stackrel{def}{=} myts \text{ of } P_k \text{ on occurence of } e$$

$$othts(e) \stackrel{def}{=} othts \text{ of } P_k \text{ on occurence of } e$$

Let e and f be the events on the same process. Since *myts* of any process never decreases,

(P 5.18) $$e < f \Rightarrow myts(e) \preceq myts(f) \qquad (\textbf{monotonicity})$$

In the discussion that follows, consider distinct m-operations α, $\beta \in op(\mathcal{H})$. Let P_i denote the process that issued α and P_k be any process. Let x denote a shared object and $abcast(\alpha)$ denote the fact that α was atomically broadcast in the execution. Moreover, any property $P(k)$ involving P_k actually implies $\langle \forall\, k : 1 \leq k \leq n : P(k) \rangle$. In addition to $inv(\alpha)$ and $resp(\alpha)$ events, we define $start(k, \alpha)$ and $\mathit{finish}(k, \alpha)$ events for an m-operation α and process P_k as in Section 5.1. If α is atomically broadcast then $start(k, \alpha)$ and $\mathit{finish}(k, \alpha)$ are defined for every process, otherwise they are only defined for the process that issued α.

P_i ::
var
  myX, $othX$: array of shared objects, initially $\perp$;
  $myts$, $othts$: array$[1..|\mathcal{X}|]$ of integer, initially 0;

(A1) On invocation of an m-operation α such that potentially $wobjects(\alpha) \neq \phi$;
    atomically broadcast α to all processes;

(A2) On receiving atomic broadcast of α from P_k;
    apply α to myX;
    $\forall\, x : x \in wobjects(\alpha) : myts[x]++$;
    if $proc(\alpha) = P_i$ then
      generate response for α;
    endif;

(A3) On invocation of an m-operation α such that $wobjects(\alpha) = \phi$;
    $othts := 0$;
    send "query" to all processes;

(A4) On receiving a "query" for α from P_k;
    send $\langle myX, myts\rangle$ to P_k;

(A5) On receiving "query response", $\langle X, ts\rangle$, from P_k;
    if $(othts \prec ts)$ then $\langle othX, othts\rangle := \langle X, ts\rangle$;

(A6) If all the responses for the "query" of α have been received then
    apply α to $othX$;
    generate response for α;

Figure 6: Implementation of m-Linearizability

$$inv(\alpha) \stackrel{def}{=} \begin{cases} \text{if } abcast(\alpha) \text{ then start event of action (A1) at } P_i \text{ for } \alpha \\ \text{else start event of action (A3) at } P_i \text{ for } \alpha \end{cases}$$

$$resp(\alpha) \stackrel{def}{=} \begin{cases} \text{if } abcast(\alpha) \text{ then finish event of action (A2) at } P_i \text{ for } \alpha \\ \text{else finish event of action (A6) at } P_i \text{ for } \alpha \end{cases}$$

$$start(k,\alpha) \stackrel{def}{=} \begin{cases} \text{if } abcast(\alpha) \text{ then start event of action (A2) at } P_k \text{ for } \alpha \\ \text{else if } P_k = P_i \text{ then start event of action (A6) at } P_k \text{ for } \alpha \end{cases}$$

$$finish(k,\alpha) \stackrel{def}{=} \begin{cases} \text{if } abcast(\alpha) \text{ then finish event of action (A2) at } P_k \text{ for } \alpha \\ \text{else if } P_k = P_i \text{ then finish event of action (A6) at } P_k \text{ for } \alpha \end{cases}$$

In addition, we also define a $query(k,\alpha)$ for every process when α is not atomically broadcast as follows,

$$query(k,\alpha) \stackrel{def}{=} \text{if } \neg abcast(\alpha) \text{ then start event of action (A4) at } P_k \text{ for } \alpha$$

It can be easily verified from the protocol that the following property is true.

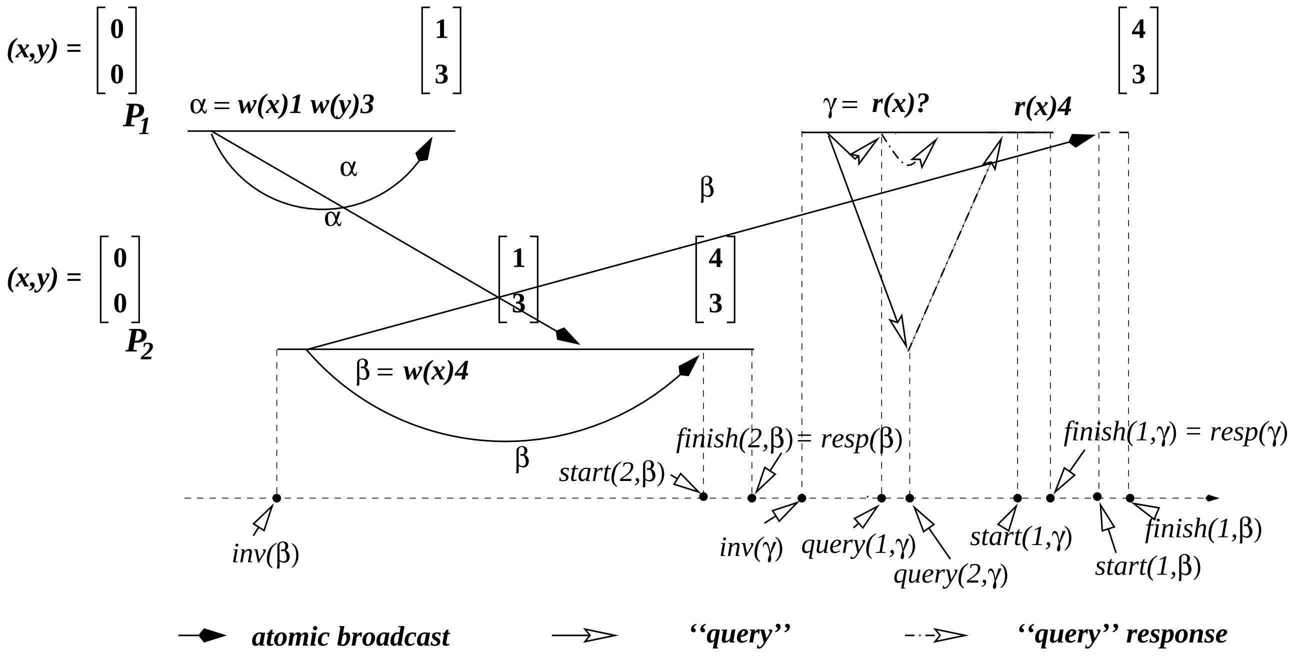

Figure 7: An example execution of the protocol in Figure 6

(P 5.19) $$inv(\alpha) \leq start(i, \alpha) < finish(i, \alpha) \leq resp(\alpha)$$

A "query" for an m-operation is sent only after its invocation and the issuing process generates the response event only after it has received response for its "query" from all the processes. Therefore,

(P 5.20) $$\neg abcast(\alpha) \Rightarrow inv(\alpha) < query(k, \alpha) < resp(\alpha)$$

Similarly, since an m-operation is atomically broadcast only after its invocation therefore,

(P 5.21) $$abcast(\alpha) \Rightarrow inv(\alpha) < start(k, \alpha)$$

Let $\leadsto_{ww}$ denote the order in which update m-operations are atomically broadcast. Then,

(P 5.22) $$wobjects(\alpha) \neq \phi \Rightarrow abcast(\alpha)$$

(P 5.23) $$(wobjects(\alpha) \neq \phi) \wedge (wobjects(\beta) \neq \phi) \equiv (\beta \leadsto_{ww} \alpha) \vee (\alpha \leadsto_{ww} \beta)$$

(P 5.24) $$\beta \leadsto_{ww} \alpha \Rightarrow start(k, \beta) < start(k, \alpha)$$

Note that $myts$ of any process is updated only in action (A2), and any two m-operations that are atomically broadcast are executed in the same order by all processes. Therefore the timestamps, $myts$, of "start" and "finish" events, if α is atomically broadcast, are respectively identical for all processes. Formally,

(P 5.25) $$abcast(\alpha) \Rightarrow (myts(start(i,\alpha)) = myts(start(k,\alpha))) \wedge (myts(finish(i,\alpha)) = myts(finish(k,\alpha)))$$

We can use this property to abbreviate $myts(start(k,\alpha))$ and $myts(finish(k,\alpha))$ as $myts(start(\alpha))$ and $myts(finish(\alpha))$ respectively when α is atomically broadcast. Furthermore, when α is not atomically broadcast we use $othts(start(\alpha))$ and $othts(finish(\alpha))$ as a shorthand for $othts(start(i,\alpha))$ and $othts(finish(i,\alpha))$ respectively. Note that action (A6) for α is executed only after responses for its "query" from all the processes have been received and as soon as a response is received $othts$ is assigned the maximum of $othts$ and the timestamp in the "query" response (action (A5)). Therefore,

(P 5.26) $$\neg abcast(\alpha) \Rightarrow (othts(start(\alpha)) = othts(finish(\alpha)) = \max_{1 \leq k \leq n}\{myts(query(k,\alpha))\})$$

Intuitively, if α is not atomically broadcast then the value of $othts$ at the "start" event of α gives the version of the objects read by α. Similarly, if α is atomically broadcast then the value of $myts$ at the "start" event of α captures the version of objects read by α and its value at the "finish" event gives the new version of the written objects. Thus, we define the timestamp ts associated with the "start" and "finish" events of an m-operation α as follows,

(D 5.4) $$ts(start(\alpha)) \stackrel{def}{=} \begin{cases} \text{if } abcast(\alpha) \text{ then } myts(start(\alpha)) \\ \text{else } othts(start(\alpha)) \end{cases}$$

(D 5.5) $$ts(finish(\alpha)) \stackrel{def}{=} \begin{cases} \text{if } abcast(\alpha) \text{ then } myts(finish(\alpha)) \\ \text{else } othts(finish(\alpha)) \end{cases}$$

The reads-from relation $\leadsto_{rf}$ can be defined identically to D 5.1 as follows,

(D 5.6) $$x \in rfobjects(\mathcal{H},\alpha,\beta) \stackrel{def}{=} (x \in robjects(\alpha)) \wedge (x \in wobjects(\beta)) \wedge (ts(finish(\beta))[x] = ts(start(\alpha))[x])$$

Since after application of α in action (A2) the components of $myts$ for which α performs a write are incremented by one and $othts$ is not modeified in (A6), therefore,

(P 5.27) $$x \notin wobjects(\alpha) \Rightarrow ts(start(\alpha))[x] = ts(finish(\alpha))[x]$$

(P 5.28) $$x \in wobjects(\alpha) \Rightarrow ts(start(\alpha))[x] = ts(finish(\alpha))[x] - 1$$

We define the timestamp associated with an m-operation α as follows,

(D 5.7) $$ts(\alpha) \stackrel{def}{=} ts(finish(\alpha))$$

Since $\leadsto_P \subseteq \leadsto_t$, we define $\leadsto_{\mathcal{H}}^{-}$ as,

(D 5.8) $$\leadsto_{\mathcal{H}}^{-} \stackrel{def}{=} \leadsto_{rf} \cup \leadsto_t \cup \leadsto_{ww}$$

Now we prove that the protocol in Figure 6 implements m-linearizability. Let $\mathcal{H}$ be an execution history generated by the protocol. For the proofs that follow consider distinct m-operations α, $\beta \in op(\mathcal{H})$. Let $proc(\alpha) = P_i$ and $proc(\beta) = P_j$, and let x denote a shared object.

Lemma 16 *If* $\beta \leadsto_{\mathcal{H}}^{-} \alpha$ *then* $ts(finish(\beta)) \preceq ts(start(\alpha))$.

Proof: This lemma is weaker than Lemma 16 in Section 5.1 for m-sequential consistency. Again, there are three cases to consider: $\beta \rightsquigarrow_{rf} \alpha$ or $\beta \rightsquigarrow_{t} \alpha$ or $\beta \rightsquigarrow_{ww} \alpha$.

Case 1 $[\beta \rightsquigarrow_{rf} \alpha]$**:** Using D 4.3 and D 5.6, we can infer that $wobjects(\beta) \neq \phi$. From P 5.22, we can conclude that β is atomically broadcast. Without loss of generality, assume $x \in rfobjects(\mathcal{H}, \alpha, \beta)$. Therefore $x \in wobjects(\beta)$. There are two subcases to consider depending on whether α is atomically broadcast.

Case 1.1 $[\neg abcast(\alpha)]$**:**

$$
\begin{array}{lll}
 & \langle \forall\, k :: query(k,\alpha) < start(k,\beta) < finish(k,\beta) \rangle & ; \text{assumption} \\
\Rightarrow & \langle \forall\, k :: myts(query(k,\alpha)) \preceq myts(start(\beta)) \preceq myts(finish(\beta)) \rangle & \\
 & & ; \text{P 5.18 (monotonicity of } myts) \\
\Rightarrow & \max_{1 \le k \le n}\{myts(query(k,\alpha))\} \preceq myts(start(\beta)) \preceq myts(finish(\beta)) & \\
\Rightarrow & othts(start(\alpha)) \preceq myts(start(\beta)) \preceq myts(finish(\beta)) & ; \text{P 5.26, P 5.18 (montonicity of } myts) \\
\Rightarrow & (ts(start(\alpha)) \preceq ts(start(\beta)) \preceq ts(finish(\beta))) \wedge (ts(start(\beta))[x] < ts(finish(\beta))[x]) & \\
 & & ; \text{D 5.4, D 5.5, P 5.28 (} \beta \text{ writes to } x) \\
\Rightarrow & (ts(start(\alpha))[x] < ts(finish(\beta))[x]) \wedge (ts(finish(\beta))[x] = ts(start(\alpha))[x]) & \\
 & & ; \text{D 5.6 (} \alpha \text{ reads from } \beta \text{ the value of } x) \\
\equiv & false & ; \text{contradiction}
\end{array}
$$

Therefore,

$$
\begin{array}{lll}
 & \langle \exists\, k :: start(k,\beta) < finish(k,\beta) < query(k,\alpha) \rangle & ; \text{atomicity of actions} \\
\Rightarrow & \langle \exists\, k :: myts(finish(\beta)) \preceq myts(query(k,\alpha)) \rangle & ; \text{P 5.18 (monotonicity of } myts) \\
\Rightarrow & myts(finish(\beta)) \preceq \max_{1 \le k \le n}\{myts(query(k,\alpha))\} = othts(start(\alpha)) & \\
 & & ; \text{P 5.26} \\
\equiv & ts(finish(\beta)) \preceq ts(start(\alpha)) & ; \text{D 5.4, D 5.5}
\end{array}
$$

Case 1.2 $[abcast(\alpha)]$**:**

$$
\begin{array}{lll}
 & start(i,\alpha) < finish(i,\beta) & ; \text{assumption} \\
\Rightarrow & start(i,\alpha) < start(i,\beta) < finish(i,\beta) & ; \text{atomicity of actions} \\
\Rightarrow & myts(start(\alpha)) \preceq myts(start(\beta)) \preceq myts(finish(\alpha)) & ; \text{P 5.18 (monotonicity of } myts) \\
\Rightarrow & (ts(start(\alpha)) \preceq ts(start(\beta)) \preceq ts(finish(\beta))) \wedge (ts(start(\beta))[x] < ts(finish\beta)[x]) & \\
 & & ; \text{D 5.4, D 5.5, P 5.28 (} \beta \text{ writes to } x)
\end{array}
$$

$\Rightarrow \quad (ts(start(\alpha))[x] < ts(finish(beta))[x]) \wedge (ts(finish(\beta))[x] = ts(start(\alpha))[x])$

; D 5.6 (α reads the value of x from β)

$\equiv \quad false$; contradiction

Therefore,

$$
\begin{array}{lll}
 & finish(i,\beta) < start(i,\alpha) & \\
\Rightarrow & myts(finish(\beta)) \preceq myts(start(\alpha)) & \text{; P 5.18 (monotonicity of } myts) \\
\equiv & ts(finish(\beta)) \preceq ts(start(\alpha)) & \text{; D 5.4, D 5.5}
\end{array}
$$

Case 2 [$\beta \rightsquigarrow_t \alpha$]**:** From the definition of $\rightsquigarrow_t$, we can infer that $resp(\beta) < inv(\alpha)$. There are four subcases to consider depending on whether α and β are atomically broadcast.

Case 2.1 [$\neg abcast(\beta) \wedge \neg abcast(\alpha)$]**:**

$$
\begin{array}{lll}
 & resp(\beta) < inv(\alpha) & \\
\Rightarrow & \langle \forall\, k :: query(k,\beta) < query(k,\alpha) \rangle & \text{; P 5.20} \\
\Rightarrow & \langle \forall\, k :: myts(query(k,\beta)) \preceq myts(query(k,\alpha)) \rangle & \text{; P 5.18 (monotonicity of } myts) \\
\Rightarrow & \max_{1\le k\le n}\{myts(query(k,\beta))\} \preceq \max_{1\le k\le n}\{myts(query(k,\alpha))\} & \\
\Rightarrow & othts(finish(\beta)) \preceq othts(start(\alpha)) & \text{; P 5.26} \\
\equiv & ts(finish(\beta)) \preceq ts(start(\alpha)) & \text{; D 5.4, D 5.5}
\end{array}
$$

Case 2.2 [$abcast(\beta) \wedge \neg abcast(\alpha)$]**:**

$$
\begin{array}{lll}
 & resp(\beta) < inv(\alpha) & \\
\Rightarrow & finish(j,\beta) < query(j,\alpha) & \text{; P 5.19, P 5.20} \\
\Rightarrow & myts(finish(\beta)) \preceq myts(query(j,\alpha)) & \text{; P 5.18 (monotonicity of } myts) \\
\Rightarrow & myts(finish(\beta)) \preceq \max_{1\le k\le n}\{myts(query(k,\alpha))\} = othts(start(\alpha)) & \\
 & & \text{; P 5.26} \\
\equiv & ts(finish(\beta)) \preceq ts(start(\alpha)) & \text{; D 5.4, D 5.5}
\end{array}
$$

Case 2.3 [$\neg abcast(\beta) \wedge abcast(\alpha)$]**:**

$$
\begin{array}{lll}
 & resp(\beta) < inv(\alpha) & \\
\Rightarrow & \langle \forall\, k :: query(k,\beta) < start(k,\alpha) \rangle & \text{; P 5.20, P 5.21}
\end{array}
$$

$$\Rightarrow \langle \forall\, k :: myts(query(k,\beta)) < myts(start(\alpha))\rangle \quad ; \text{P 5.18 (monotonicity of } myts)$$

$$\Rightarrow \max_{1\le k\le n}\{myts(query(k,\beta))\} \preceq myts(start(\alpha)) \quad ;$$

$$\Rightarrow othts(finish(\beta)) \preceq myts(start(\alpha)) \quad ; \text{P 5.26}$$

$$\equiv ts(finish(\beta)) \preceq ts(start(\alpha)) \quad ; \text{D 5.4 , D 5.5}$$

Case 2.4 $[abcast(\beta) \wedge abcast(\alpha)]$**:**

$$resp(\beta) < inv(\alpha)$$

$$\Rightarrow finish(j,\beta) < start(j,\alpha) \quad ; \text{P 5.19, P 5.21}$$

$$\Rightarrow myts(finish(\beta)) \preceq myts(start(\alpha)) \quad ; \text{P 5.18 (monotonicity of } myts)$$

$$\equiv ts(finish(\beta)) \preceq ts(start(\alpha)) \quad ; \text{D 5.4, D 5.5}$$

Case 3 $[\beta \leadsto_{ww} \alpha]$**:** Using P 5.23 and P 5.22, we can infer that both α and β are atomically broadcast.

$$\beta \leadsto_{ww} \alpha$$

$$\Rightarrow start(i,\beta) < start(i,\alpha) \quad ; \text{P 5.24 (} \beta \text{ is received before } \alpha \text{ at } P_i)$$

$$\Rightarrow finish(i,\beta) < start(i,\alpha) \quad ; \text{atomicity of actions}$$

$$\Rightarrow myts(finish(\beta)) \preceq myts(start(\alpha)) \quad ; \text{P 5.18 (monotonicity of } myts)$$

$$\equiv ts(finish(\beta)) \preceq ts(start(\alpha)) \quad ; \text{D 5.4, D 5.5}$$

Hence if $\beta \leadsto_{\mathcal{H}}^{-} \alpha$ then $ts(finish(\beta)) < ts(start(\alpha))$. ■

Lemma 17 *The protocol in Figure 6 satisfies P 5.1 and P 5.2.*

Proof: The proof of property P 5.1 is as follows,

$$(\beta \leadsto_{\mathcal{H}}^{-} \alpha) \wedge (wobjects(\alpha) = \phi) \wedge (wobjects(\beta) = \phi)$$

$$\Rightarrow ((\beta \leadsto_{rf} \alpha) \vee (\beta \leadsto_{t} \alpha) \vee (\beta \leadsto_{ww} \alpha)) \wedge (wobjects(\beta) = \phi)$$

$$; \text{D 5.8 (definition of } \leadsto_{\mathcal{H}}^{-})$$

$$\Rightarrow ((wobjects(\beta) \neq \phi) \vee (\beta \leadsto_{t} \alpha) \vee (wobjects(\beta) \neq \phi)) \wedge (wobjects(\beta) = \phi)$$

$$; \text{D 5.6, definition of } \leadsto_{ww}$$

$$\Rightarrow \beta \leadsto_{t} \alpha \quad ; \text{distribution of } \wedge \text{ over } \vee, \text{ contradiction}$$

The property P 5.2 can be proved as follows,

$$
\begin{array}{lll}
& (wobjects(\alpha) \neq \phi) \wedge (wobjects(\beta) \neq \phi) & \\
\equiv & (\alpha \rightsquigarrow_{ww} \beta) \vee (\beta \rightsquigarrow_{ww} \alpha) & ;\text{P 5.23} \\
\Rightarrow & (\alpha \rightsquigarrow^{-}_{\mathcal{H}} \beta) \vee (\beta \rightsquigarrow^{-}_{\mathcal{H}} \alpha) & ;\text{D 5.8 } (\rightsquigarrow^{-}_{\mathcal{H}} \text{ contains } \rightsquigarrow_{ww})
\end{array}
$$

Hence P 5.1 and P 5.2 are satisfied by the protocol in Figure 6. ■

Lemma 18 *The protocol in Figure 6 satisfies P 5.3 and P 5.4.*

Proof: The property P 5.3 can be proved as follows,

$$
\begin{array}{lll}
& \beta \rightsquigarrow^{-}_{\mathcal{H}} \alpha & \\
\Rightarrow & ts(finish(\beta)) \preceq ts(start(\alpha)) & ;\text{Lemma 16} \\
\Rightarrow & ts(finish(\beta)) \preceq ts(start(\alpha)) \preceq ts(finish(\alpha)) & ;\text{conjunction of P 5.27 and P 5.28} \\
\Rightarrow & ts(finish(\beta)) \preceq ts(finish(\alpha)) & ;\text{simplification} \\
\equiv & ts(\beta) \preceq ts(\alpha) & ;\text{D 5.7}
\end{array}
$$

The proof of property P 5.4 is as follows,

$$
\begin{array}{lll}
& (\beta \rightsquigarrow^{-}_{\mathcal{H}} \alpha) \wedge (x \in wobjects(\alpha)) & \\
\Rightarrow & (ts(finish(\beta)) \preceq ts(start(\alpha))) \wedge (x \in wobjects(\alpha)) & ;\text{Lemma 16} \\
\Rightarrow & (ts(finish(\beta)) \preceq ts(start(\alpha))) \wedge (ts(start(\alpha))[x] < ts(finish(\alpha))[x]) & \\
& & ;\text{P 5.28 } (\alpha \text{ writes to } x) \\
\Rightarrow & ts(finish(\beta))[x] \leq ts(start(\alpha))[x] < ts(finish(\alpha))[x]) & ;\text{simplification} \\
\equiv & ts(\beta)[x] < ts(\alpha)[x] & ;\text{D 5.7}
\end{array}
$$

Hence P 5.3 and P 5.4 are satisfied by the protocol in Figure 6. ■

Lemma 19 *The protocol in Figure 6 satisfies P 5.7 and P 5.8.*

Proof: The proof of property P 5.7 is as follows,

$$
\begin{array}{lll}
& (x \in rfobjects(\mathcal{H}, \alpha, \beta)) \wedge (x \notin wobjects(\alpha)) & \\
\Rightarrow & (ts(finish(\beta))[x] = ts(start(\alpha))[x])) \wedge (x \notin wobjects(\alpha)) & \\
& & ;\text{D 5.6 } (\alpha \text{ reads the value of } x \text{ from } \beta) \\
\Rightarrow & (ts(finish(\beta))[x] = ts(start(\alpha))[x]) \wedge (ts(start(\alpha))[x] = ts(finish(\alpha))[x]) & \\
& & ;\text{P 5.27 } (\alpha \text{ does not write to } x) \\
\Rightarrow & ts(finish(\beta))[x] = ts(finish(\alpha))[x] & ;\text{simplification} \\
\equiv & ts(\beta)[x] = ts(\alpha)[x] & ;\text{D 5.7}
\end{array}
$$

The property P 5.8 can be proved as follows,

$$
\begin{array}{lll}
 & (x \in rfobjects(\mathcal{H}, \alpha, \beta)) \wedge (x \in wobjects(\alpha)) & \\
\Rightarrow & (ts(\mathit{finish}(\beta))[x] = ts(start(\alpha))[x])) \wedge (x \in wobjects(\alpha)) & \\
 & & ;\ \text{D 5.6 } (\alpha \text{ reads the value of } x \text{ from } \beta) \\
\Rightarrow & (ts(\mathit{finish}(\beta))[x] = ts(start(\alpha))[x]) \wedge (ts(start(\alpha))[x] = ts(\mathit{finish}(\alpha))[x] - 1) & \\
 & & ;\ \text{P 5.28 } (\alpha \text{ writes to } x) \\
\Rightarrow & ts(\mathit{finish}(\beta))[x] = ts(\mathit{finish}(\alpha))[x] - 1 & ;\ \text{simplification} \\
\equiv & ts(\beta)[x] = ts(\alpha)[x] - 1 & ;\ \text{D 5.7}
\end{array}
$$

Hence P 5.7 and P 5.8 are satisfied by the protocol in Figure 6. ∎

Theorem 20 *All the executions generated by the protocol in Figure 6 are m-linearizable.*

Proof: From Lemma 17-19, we can conclude that the protocol in Figure 6 satisfy properties P 5.1-5.4 and P 5.7-5.8. Therefore, from Theorem 10, we can infer that all the executions generated by the protocol in Figure 6 are m-linearizable. ∎

Although in this protocol a process sends the whole copy of shared objects along with their timestamps in response to the "query", but it is easy to verify that the protocol is still correct if only the relevant copies of the shared objects ($objects(.)$) and their timestamp is sent.

6 Conclusion

We extend the traditional model of concurrent objects to allow operations that span multiple objects. We give the consistency conditions in this model, analyze their verification complexity and give efficient algorithms for ensuring them in distributed systems.

References

[1] Sarita V. Adve and K. Gharachorloo. "Shared Memory Consistency Models: A Tutorial". *IEEE Computer*, pages 66–76, December 1996.

[2] Y. Afek, G. Brown, and M. Merritt. "Lazy Caching". *ACM Transactions on Programming Language and Systems*, 15(1):182–205, January 1993.

[3] Mustaque Ahamad, Phillip W. Hutto, and Ranjit John. "Causal memory: Definitions, Implementation and Programming". Technical Report 93/55, College of Computing, Georgia Institute of Technology, September 1993.

[4] Hagit Attiya and Jennifer L. Welch. "Sequential Consistency versus Linearizability". *ACM Transactions on Computer Systems*, 12(2):91–122, May 1994.

[5] P. Bernstein, V. Hadzilacos, and N. Goodman. *"Concurrency Control and Recovery in Database Systems"*. Addison-Wesley, Reading, MA, 1987.

[6] Robert D. Blumofe, Matteo Frigo, Christopher F. Joerg, Charles E. Leiserson, and Keith H. Randall. "Dag-Consistent Distributed Shared Memory". In *Proceedings of the 10th International Parallel Processing Symposium (IPPS)*, pages 132–141, April 15-19, 1996.

[7] C. J. Fidge. "Logical Time in Distributed Computing Systems". *IEEE Computer*, 24(8):28–33, 1991.

[8] Vijay K. Garg and Michel Raynal. "Normality: A Consistency Conditions for Concurrent Objects". Technical Report TR-PDS-1996-010, The University of Texas at Austin, May 1996. To appear in Parallel Processing Letters.

[9] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors". In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 15–26, May 1990.

[10] Michael Greenwald and David Cheriton. "The Synergy Between Non-blocking Synchronization and Operating System Structure". In *Proceedings of the Second Symposium on Operating System Design and Implementation*, pages 123–136, USENIX, Seattle, October 1996.

[11] Maurice Herlihy. "Wait-Free Synchronization". *ACM Transactions on Programming Language and Systems*, 11(1):124–149, January 1991.

[12] Maurice P. Herlihy and Jeannette M. Wing. "Linearizability: A correctness condition for concurrent objects". *ACM Transactions on Programming Language and Systems*, 12(3):463–492, July 1990.

[13] T. Ibaraki, T. Kameda, and T. Minoura. "Serializability with Constraints". *ACM Transactions on Database Systems*, 12(3):429–452, 1987.

[14] P. Keleher, A. L. Cox, S. Dwarkadas, and W. Zwaenepoel. "Treadmarks: Distributed Shared Memory on Standard Workstations and operating systems". In *Proceedings of the 1994 Winter Usenix Conference*, pages 115–132, January 1994.

[15] Leslie Lamport. "How to make a multiprocessor computer that correctly executes multiprocess programs". *IEEE Transactions on Computers*, C28(9):690–691, September 1979.

[16] Richard J. Lipton and Jonathan S. Sandberg. "PRAM: A scalable shared memory". Technical Report 180-88, Department of Computer Science, Princeton University, September 1988.

[17] Friedemann Mattern. "Virtual time and global states of distributed systems". *International Workshop on Parallel and Distributed Algorithms*, pages 215–226, October 1988.

[18] Marios Mavronicolas and Dan Roth. "Sequential Consistency and Linearizability: Read/Write objects". In *Proceedings of Twenty-Ninth Annual Allerton Conference on Communication, Control and Computing*, pages 683–692, October 1991.

[19] Jayadev Misra. "Axioms for memory access in asynchronous hardware systems". *ACM Transactions on Programming Language and Systems*, 8(1):142–153, January 1986.

[20] M. Mizuno, M. Raynal, and J.Z. Zhou. "Sequential Consistency in Distributed Systems". In K. Birman, F. Mattern, and A. Schiper, editors, *Proceedings of International Workshop on "Theory and Practice in Distributed Systems"*, Springer-Verlag LNCS 938, pages 227–241, Dagstuhl, Germany, 1994.

[21] C. H. Papadimitriou. *"The Theory of Concurrency Control"*. Computer Science Press, May 1986.

[22] M. Raynal, G. Thia-Kime, and M. Ahamad. "From Serializable to Causal Transactions for Collaborative Applications". Technical Report 983, Irisa - Rennes, February 1996. 22 pages.

[23] Richard N. Taylor. "Complexity of Analyzing the Synchronization Structure of Concurrent Programs". *Acta Informatica*, 19:57–84, 1983.