Significance and Uses of Fine-Grained Synchronization Relations

Ajay D. Kshemkalyani

Dept. of Electrical & Computer Engg. and Computer Science
University of Cincinnati, P. O. Box 210030, Cincinnati, OH 45221-0030, USA
ajayk@ececs.uc.edu

Abstract. In a distributed system, high-level actions can be modeled by nonatomic events. Synchronization relations between distributed nonatomic events have been proposed to allow applications a fine choice in specifying synchronization conditions. This paper shows how these fine-grained relations can be used for various types of synchronization and coordination in distributed computations.

1 Introduction

High-level actions in several distributed application executions are realistically modeled by nonatomic poset events (where at least some of the component atomic events of a nonatomic event occur concurrently), for example, in distributed multimedia support, coordination in mobile computing, distributed debugging, navigation, planning, industrial process control, and virtual reality. It is important to provide these and emerging sophisticated applications a fine level of granularity in the specification of various synchronization/causality relations between nonatomic poset events. In addition, [20] stresses the theoretical and practical importance of the need for such relations. Most of the existing literature, e.g., [1, 2, 4, 5, 7, 10, 13, 15, 18–20] does not address this issue. A set of causality relations between nonatomic poset events was proposed in [11, 12] to specify and reason with a fine-grained specification of causality. This set of relations extended the hierarchy of the relations in [9, 14]. An axiom system on the proposed relations was given in [8]. The objective of this paper is to demonstrate the use of the relations in [11, 12].

The following poset event structure model is used as in [4, 9, 10, 12, 14–16, 20]. $(E, \prec)$ is a poset such that E represents points in space-time which are the primitive atomic events related by the causality relation $\prec$ which is an irreflexive partial ordering. Elements of E are partitioned into local executions at a coordinate in the space dimensions. In a distributed system, E represents a set of events and is discrete. Each local execution E_i is a linearly ordered set of events in partition i. An event e in partition i is denoted e_i. For a distributed application, points in the space dimensions correspond to the set of processes (also termed *nodes*), and E_i is the set of events executed by process i. Causality between events at different nodes is imposed by message communication.

High-level actions in the computation are nonempty subsets of E. Formally, if $\mathcal{E}$ denote the power set of E and $\mathcal{A}$ $(\neq \emptyset) \subseteq (\mathcal{E} - \emptyset)$, then each element A of $\mathcal{A}$ is termed an *interval* or a *nonatomic event.* It follows that if $A \bigcap E_i \neq \emptyset$, then $(A \bigcap E_i)$ has a least and a greatest event. $\mathcal{A}$ is the set of all the sets that represent a higher level grouping of the events of E that is of interest to an application.

Table 1. Some causality relations [9].

Relation r	Expression for $r(X, Y)$
$R1$	$\forall x \in X \forall y \in Y, x \prec y$
$R1'$	$\forall y \in Y \forall x \in X, x \prec y$
$R2$	$\forall x \in X \exists y \in Y, x \prec y$
$R2'$	$\exists y \in Y \forall x \in X, x \prec y$
$R3$	$\exists x \in X \forall y \in Y, x \prec y$
$R3'$	$\forall y \in Y \exists x \in X, x \prec y$
$R4$	$\exists x \in X \exists y \in Y, x \prec y$
$R4'$	$\exists y \in Y \exists x \in X, x \prec y$

The relations in [9] formed an exhaustive set of causality relations to express the possible interactions between a pair of linear intervals. These relations $R1$ - $R4$ and $R1'$ - $R4'$ from [9] are expressed in terms of the quantifiers over X and Y in Table 1. Observe that $R2'$ and $R3'$ are different from $R2$ and $R3$, respectively, when applied to posets. Table 1 gives the hierarchy and inclusion relationship of the causality relations $R1$ - $R4$. Each cell in the grid indicates the relationship of the row header to the column header. The notation for the inclusion relationship between causality relations is as follows. The inclusion relation "is a subrelation of" is denoted '$\sqsubseteq$' and its inverse is '$\sqsupseteq$'. For relations r_1 and r_2, we define $r_1 \mid\mid r_2$ to be $(r_1 \not\sqsubseteq r_2 \bigwedge r_2 \not\sqsubseteq r_1)$. The relations { $R1$, $R2$, $R3$, $R4$ } form a lattice hierarchy ordered by $\sqsubseteq$.

Table 2. Inclusion relationships between relations of Table 1 [9].

relation of row header to column header	$R1$	$R2$	$R3$	$R4$
$R1$	$=$	$\sqsubseteq$	$\sqsubseteq$	$\sqsubseteq$
$R2$	$\sqsupseteq$	$=$	$\mid\mid$	$\sqsubseteq$
$R3$	$\sqsupseteq$	$\mid\mid$	$=$	$\sqsubseteq$
$R4$	$\sqsupseteq$	$\sqsupseteq$	$\sqsupseteq$	$=$

The relations in [9] formed a comprehensive set of causality relations to express all possible interactions between a pair of linear intervals using only the $\prec$ relation between atomic events, and extended the partial hierarchy of rela-

tions of [14]. However, when the relations of [9] are applied to a pair of poset intervals, the hierarchy they form is incomplete. Causality relations between a pair of nonatomic poset intervals were formulated by extending the results [8, 9] to nonatomic poset events [11, 12]. The relations form a "comprehensive" set of causality relations between nonatomic poset events using first-order predicate logic and only the $\prec$ relation between atomic events, and fill in the existing partial hierarchy formed by relations in [9, 14]. A relational algebra for the relations in [11, 12] is given in [8]. Given any relation(s) between X and Y, the relational algebra allows the derivation of conjunctions, disjunctions, and negations of all other relations that are also valid between X and Y, as well as between Y and X.

Section 2 reviews the hierarchy of causality relations [11, 12]. Section 3 gives the uses of each of the relations. Some uses of the relations include modeling various forms of synchronization for group mutual exclusion, initiation of nested computing, termination of nested processing, and monitoring the start of a computation. Section 4 gives concluding remarks. The results of this paper are included in [8].

2 Relations between Nonatomic Poset Events

Previous work on linear intervals and time durations, e.g., [1, 2, 4, 5], identifies an interval by the instants of its beginning and end. Given a nonatomic poset interval, one needs to define counterparts for the beginning and end instants. These counterparts serve as "proxy" events for the poset interval just as the events at the beginning and end of linear intervals such as time durations serve as proxies for the linear interval. The proxies identify the durations on each node, in which the nonatomic event occurs. Two possible definitions of proxies are (i) $L_X = \{e_i \in X \mid \forall e_i' \in X, e_i \preceq e_i'\}$ and $U_X = \{e_i \in X \mid \forall e_i' \in X, e_i \succeq e_i'\}$, and (ii) $L_X = \{e \in X \mid \forall e' \in X, e \not\succ e'\}$ and $U_X = \{e \in X \mid \forall e' \in X, e \not\prec e'\}$. Assume that one definition of proxies is consistently used, depending on context and application. Fig. 1 depicts the proxies of X and Y.

There are two aspects of a relation that can be specified between poset intervals. One aspect deals with the determination of an appropriate *proxy* for each interval. A proxy for X and Y can be chosen in four ways corresponding to the relations in $\{R1, R2, R3, R4\}$. From Table 1, it follows that these four relations form a lattice ordered by $\sqsubseteq$. The second aspect deals with how the atomic elements of the chosen proxies are related by causality. The chosen proxies can be related by the eight relations $R1$, $R1'$, $R2$, $R2'$, $R3$, $R3'$, $R4$, $R4'$ of Table 1, which are renamed a, a', b, b', c, c', d, d', respectively, to avoid confusion with their original names used for the first aspect of specifying the relations between poset intervals. The inclusion hierarchy among the six distinct relations forms a lattice ordered by $\sqsubseteq$; see Table 3.

The two aspects of deriving causality relations, described above, are combined to define the relations. The lattice of relations { $R1^*$, $R2^*$, $R3^*$, $R4^*$ } between proxies of X and Y, and the lattice of relations { a, a', b, b', c, c', d, d' } between

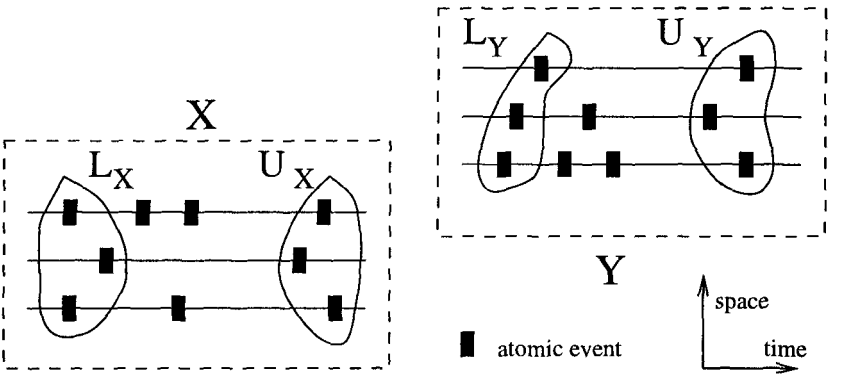

Fig. 1. Poset events X and Y and their proxies.

Table 3. Full hierarchy of relations of Table 1 [9]. Relations $R1$, $R1'$, $R2$, $R2'$, $R3$, $R3'$, $R4$, $R4'$ of Table 1 are renamed a, a', b, b', c, c', d, d', respectively. Relations in the row and column headers are defined between X and Y.

Relation names: its quantifiers for $x \prec y$	$R1,a$ (=$R1',a'$): $\forall x \forall y$ (=$\forall y \forall x$)	$R2,b$: $\forall x \exists y$	$R2',b'$: $\exists y \forall x$	$R3,c$: $\exists x \forall y$	$R3',c'$: $\forall y \exists x$	$R4,d$ (=$R4',d'$): $\exists x \exists y$ (=$\exists y \exists x$)
$R1,a$ (=$R1',a'$) : $\forall x \forall y (= \forall y \forall x)$	$=$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$
$R2,b$: $\forall x \exists y$	$\supseteq$	$=$	$\supseteq$	$\parallel$	$\parallel$	$\subseteq$
$R2',b'$: $\exists y \forall x$	$\supseteq$	$\subseteq$	$=$	$\parallel$	$\parallel$	$\subseteq$
$R3,c$: $\exists x \forall y$	$\supseteq$	$\parallel$	$\parallel$	$=$	$\subseteq$	$\subseteq$
$R3',c'$: $\forall y \exists x$	$\supseteq$	$\parallel$	$\parallel$	$\supseteq$	$=$	$\subseteq$
$R4,d$ (=$R4',d'$): $\exists x \exists y (= \exists y \exists x)$	$\supseteq$	$\supseteq$	$\supseteq$	$\supseteq$	$\supseteq$	$=$

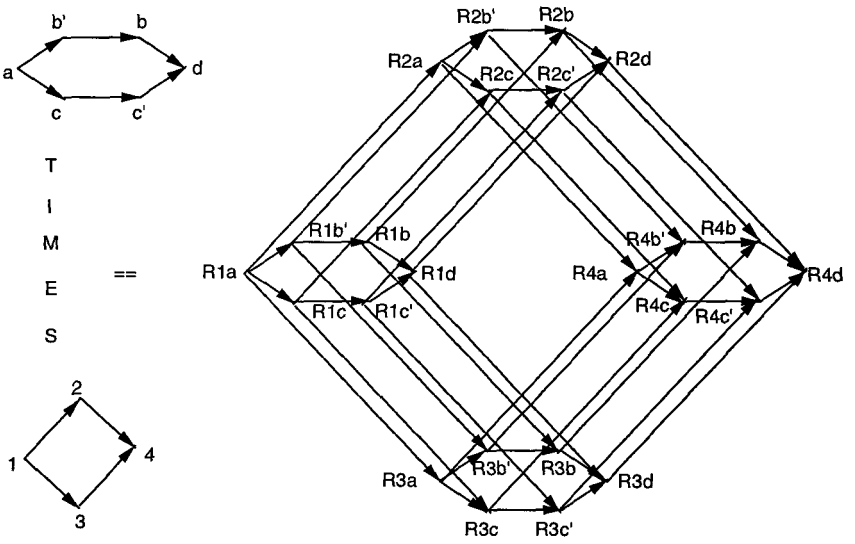

Fig. 2. Hierarchy of relations in [11, 12].

the elements of the proxies, give a product lattice of 32 relations over $\mathcal{A} \times \mathcal{A}$ to express $r(X, Y)$. The resulting set of poset relations, denoted $\mathcal{R}$, is given in Table 4. The relations in $\mathcal{R}$ form a lattice of 24 unique elements as shown in Fig. 2. $\mathcal{R}$ is comprehensive using first-order predicate logic and only the $\prec$ relation between atomic events. Relation $R?\#(X, Y)$ means that proxies of X and Y are chosen as per ?, and events in the proxies are related by $\#$. The two relations in [14], viz., $\longrightarrow$ and $--\rightarrow$, correspond to $R1a$ and $R4d$, respectively, whereas the relations in [9] and listed in Table 1 correspond to the these relations as follows: $R1$=$R1'$, $R2$, $R2'$, $R3$, $R3'$, $R4$=$R4'$ correspond to $R1a$, $R2b$, $R2b'$, $R3c$, $R3c'$, $R4d$, respectively.

3 Significance of the Relations

The hierarchy of causality relations is useful for applications which use nonatomicity in reasoning and modeling and need a fine level of granularity of causality relations to specify synchronization relations and their composite global predicates between nonatomic events. Such applications include industrial process control applications, distributed debugging, navigation, planning, robotics, diagnostics, virtual reality, and coordination in mobile systems. The hierarchy provides a range of relations, and an application can use those that are relevant and useful to it. The relations and their composite (global) predicates provide a precise handle to *express* a naturally occurring or *enforce* a desired fine-grained level of causality or synchronization in the computation. A specific meaning of each relation and a brief discussion of how it can be enforced is now given. In

Table 4. Relations $r(X, Y)$ in $\mathcal{R}$ [11, 12].

Relation $r(X,Y)$	Relation definition specified by quantifiers for $x \prec y$, where $x \in X$, $y \in Y$	Relation $r(X,Y)$	Relation definition specified by quantifiers for $x \prec y$, where $x \in X$, $y \in Y$
$R1a$	$\forall x \in U_X \forall y \in L_Y$	$R3a$	$\forall x \in L_X \forall y \in L_Y$
$R1a'$ $(=R1a)$	$\forall y \in L_Y \forall x \in U_X$	$R3a'$ $(=R3a)$	$\forall y \in L_Y \forall x \in L_X$
$R1b$	$\forall x \in U_X \exists y \in L_Y$	$R3b$	$\forall x \in L_X \exists y \in L_Y$
$R1b'$	$\exists y \in L_Y \forall x \in U_X$	$R3b'$	$\exists y \in L_Y \forall x \in L_X$
$R1c$	$\exists x \in U_X \forall y \in L_Y$	$R3c$	$\exists x \in L_X \forall y \in L_Y$
$R1c'$	$\forall y \in L_Y \exists x \in U_X$	$R3c'$	$\forall y \in L_Y \exists x \in L_X$
$R1d$	$\exists x \in U_X \exists y \in L_Y$	$R3d$	$\exists x \in L_X \exists y \in L_Y$
$R1d'$ $(=R1d)$	$\exists y \in L_Y \exists x \in U_X$	$R3d'$ $(=R3d)$	$\exists y \in L_Y \exists x \in L_X$
$R2a$	$\forall x \in U_X \forall y \in U_Y$	$R4a$	$\forall x \in L_X \forall y \in U_Y$
$R2a'$ $(=R2a)$	$\forall y \in U_Y \forall x \in U_X$	$R4a'$ $(=R4a)$	$\forall y \in U_Y \forall x \in L_X$
$R2b$	$\forall x \in U_X \exists y \in U_Y$	$R4b$	$\forall x \in L_X \exists y \in U_Y$
$R2b'$	$\exists y \in U_Y \forall x \in U_X$	$R4b'$	$\exists y \in U_Y \forall x \in L_X$
$R2c$	$\exists x \in U_X \forall y \in U_Y$	$R4c$	$\exists x \in L_X \forall y \in U_Y$
$R2c'$	$\forall y \in U_Y \exists x \in U_X$	$R4c'$	$\forall y \in U_Y \exists x \in L_X$
$R2d$	$\exists x \in U_X \exists y \in U_Y$	$R4d$	$\exists x \in L_X \exists y \in U_Y$
$R2d'$ $(=R2d)$	$\exists y \in U_Y \exists x \in U_X$	$R4d'$ $(=R4d)$	$\exists y \in U_Y \exists x \in L_X$

the following discussion, the "X computation" and "Y computation" refer to the computation performed by the nonatomic events X and Y, respectively. A proxy of X is denoted $\hat{X}$.

We first consider the significance of the groups of relations $R^*a(X,Y)$, $R^*b(X,Y)$, $R^*b'(X,Y)$, $R^*c(X,Y)$, $R^*c'(X,Y)$, and $R^*d(X,Y)$. Each group deals with a particular proxy $\hat{X}$ and $\hat{Y}$.

- $R^*a(X,Y)$: All events in $\hat{Y}$ know the results of the X computation (if any,) upto all the events in $\hat{X}$. This is a strong form of synchronization between $\hat{X}$ and $\hat{Y}$.
- $R^*b(X,Y)$: For each event in $\hat{X}$, some event in $\hat{Y}$ knows the results of the X computation (if any,) upto that event in $\hat{X}$. The Y computation may then exchange information about the X computation upto $\hat{X}$, among the nodes participating in the Y computation.
- $R^*b'(X,Y)$: Some event in $\hat{Y}$ knows the results of the X computation (if any,) upto all events in $\hat{X}$. These relations are useful when it is sufficient for one node in $N_{\hat{Y}}$ to detect a global predicate across all nodes in $N_{\hat{X}}$. If the event in $\hat{Y}$ is at a node that behaves as the group leader of $N_{\hat{Y}}$, then it can either inform the other nodes in $N_{\hat{Y}}$ or make decisions on their behalf.
- $R^*c(X,Y)$: All events in $\hat{Y}$ know the results of the X computation (if any,) upto some common event in $\hat{X}$. This group of relations is useful when it is sufficient for one node in $N_{\hat{X}}$ to inform all the nodes in $N_{\hat{Y}}$ of its state,

such as when all the nodes in $N_{\hat{X}}$ have a similar state. If the node at which the event in $\hat{X}$ occurs has already collected information about the results/ states of the X computation upto $\hat{X}$ from other nodes in $N_{\hat{X}}$ (thus, that node behaves as the group leader of $\hat{X}$), then the events in $\hat{Y}$ will know the states of the X computation upto $\hat{X}$.

- $R^*c'(X,Y)$: Each event in $\hat{Y}$ knows the results of the X computation (if any,) upto some event in $\hat{X}$. If it is important to the application, then the state at each event in $\hat{X}$ should be communicated to some event in $\hat{Y}$.
- $R^*d(X,Y)$: Some event in $\hat{Y}$ knows the results of the X computation (if any,) upto some event in $\hat{X}$. The nodes under consideration at which the events in $\hat{Y}$ and $\hat{X}$, respectively, occur may be the group leaders of $N_{\hat{Y}}$ and $N_{\hat{X}}$, respectively. This group leader of $N_{\hat{X}}$ may have collected relevant state information from other nodes in $N_{\hat{X}}$, and conveys this information to the group leader of $N_{\hat{Y}}$, which in turn distributes the information to all nodes in $N_{\hat{Y}}$.

The above significance of each group of relations applies to each individual relation of that group. The specific use and meaning of each of the 24 relations in $\mathcal{R}$ is given next. We do not restrict the explanation that follows to any specific application.

$1^*(X,Y)$: This group of relations deals with U_X and L_Y. Each relation signifies different degree of transfer of control for synchronization, as in group mutual exclusion (*gmutex*), from the X computation to the Y computation.

- $R1a(X,Y)$: The Y computation at any node in N_Y begins only after that node knows that the X computation at each node in N_X has ended, e.g., a conventional distributed gmutex in which each node in N_Y waits for an indication from each node in N_X that it has relinquished control.
- $R1b(X,Y)$: For every node in N_X, the final value of its X computation is known by (or its mutex token is transferred to) some node in N_Y before that node in N_Y begins its Y computation. Thus, nodes in N_Y collectively (but not individually) know the final value of the X computation before the last among them begins its Y computation. This is a weak version of synchronization/gmutex.
- $R1b'(X,Y)$: Before beginning its Y computation, some node in N_Y knows the final value of the X computation at each node in N_X. This is a weak version of synchronization/gmutex (but stronger than $R1b$) with the property that at least one node in N_Y cannot begin its Y computation until the final value of the X computation at each node in N_X is known to it.
- $R1c(X,Y)$: The final value of the X computation at some node in N_X is known to all the nodes in N_Y before they begin their Y computation. This is a weak form of synchronization/gmutex which is useful when it suffices for a particular node in N_X to grant all the nodes in N_Y gmutex permission to proceed with the Y computation; this node in N_X may be the group leader of N_X, or simply all the nodes in N_X have the same final local state of the X computation within this application.

- $R1c'(X,Y)$: Each node in N_Y begins its Y computation only after it knows the final value of the X computation of some node in N_X. This is a weak form of synchronization/gmutex (weaker than $R1c$) which requires each node in N_Y to receive a final value (or gmutex token) from at least one node in N_X before starting its Y computation. This relation is sufficient for some applications such as those requiring that at most one (additional) process be admitted to join those in the critical section when one process leaves it.
- $R1d(X,Y)$: Some node in N_Y begins its Y computation only after it knows the final value of (or receives a gmutex token from) the X computation at some node in N_X. This is the weakest form of synchronization/gmutex.

$R2^*(X,Y)$: This group of relations deals with U_X and U_Y. The relations can signify various degrees of synchronization between the termination of computations X and Y, where X is nested within Y or X is a subcomputation of Y. Alternately, Y could denote activity at processes that have already spawned X activity in threads, and Y can complete only after X completes.

- $R2a(X,Y)$: The Y computation at any node in N_Y can terminate only after that node knows the final value of (or termination of) the X computation at each node in N_X. This is a strong synchronization before termination, between X and Y.
- $R2b(X,Y)$: For every node in N_X, the final value of its X computation is known by at least one node in N_Y before that node in N_Y terminates its Y computation. Thus, all the nodes in N_Y collectively (but not individually) know the final values of the X computation before they terminate their Y computation. This is a weak synchronization before termination.
- $R2b'(X,Y)$: Before terminating its Y computation, some node in N_Y knows the final value of the X computation at all nodes in N_X. This is a stronger synchronization before termination than $R2b$, wherein at least one node in N_Y cannot terminate its Y computation without knowing the final state of the X computation at all nodes in N_X. This suffices for all applications in which it is adequate for one node in N_Y to detect the termination of the X computation at each node in N_X before that node terminates its Y computation.
- $R2c(X,Y)$: The final value of the X computation at some node in N_X is known to all the N_Y nodes before they terminate the Y computation. This is a weak form of synchronization. The pertinent node in N_X could represent a critical thread in the X computation, or could be the group leader of N_X that represents the X computation at all nodes in N_X.
- $R2c'(X,Y)$: Each node in N_Y terminates its Y computation only after it knows the final value of the X computation at some node in N_X. This is a weak form of synchronization before termination (weaker than $R2c$), but is adequate when all the nodes in N_X are performing a similar X computation.
- $R2d(X,Y)$: Some node in N_Y terminates its Y computation after it knows the final value of the X computation at some node in N_X. This is a weak form of synchronization; however, if the concerned nodes in N_X and N_Y are the respective group leaders of the X and Y computations and, respectively,

collect/distributed information from/to their groups, then a strong form of synchronization can be implicitly enforced because when Y terminates, it is known to each node in N_Y that the X computation has terminated.

$R3^*(X,Y)$: This group of relations deals with L_X and L_Y. The relations can signify various degrees of synchronization between the initiation of computations X and Y, where Y is nested within X or Y is a subcomputation of X. Alternately, X could denote activity at processes that have already spawned Y activity in threads.

- $R3a(X,Y)$: The Y computation at any node in N_Y begins after that node knows the initial values of the X computation at each node in N_X. This is a strong form of synchronization between the beginnings of the X and Y computations.
- $R3b(X,Y)$: For each node in N_X, the initial state of its X computation is known to some node in N_Y before that node in N_Y begins its Y computation. Thus, all the nodes in N_Y collectively (but not individually) know the initial state of the X computation. This synchronization is sufficient when the forked Y computations at each node in N_Y are only loosely coupled and should not know each others' initial states communicated by the X computation; while at the same time ensuring that the initial state of the X computation at each node in N_X is available to at least one node in N_Y before it commences its Y computation.
- $R3b'(X,Y)$: Before beginning its Y computation, some node in N_Y knows the initial state of the X computation at all the nodes in N_X. Thus the Y computation at this node can run a parallel X computation for fault-tolerance, or can be made an entirely deterministic function of the inputs to the X computation. The subject node in N_Y can coordinate the Y computation of the other nodes in N_Y. This synchronization is weaker than $R3a$ but stronger than $R3b$.
- $R3c(X,Y)$: The initial state of the X computation at some node in N_X is known to all the nodes in N_Y before they begin their Y computation. This is a weak synchronization; however, it is adequate when the subject node in N_X has forked all the threads that will perform Y, and behaves as the group leader of X that initiates the nested computation Y.
- $R3c'(X,Y)$: Each node in N_Y begins its Y computation only after it knows the initial state of the X computation at some node in N_X. Thus each node executing the computation Y has its Y computation forked or spawned by some node in N_X and its Y computation corresponds to a nested branch of X. The nodes in N_Y may not know each others' initial values for the Y computation; the X computations at (some of) the N_X nodes have semi-independently forked the Y computations at the nodes in N_Y.
- $R3d(X,Y)$: Some node in N_Y begins its Y computation only after it knows the initial state of the X computation at some node in N_X. This is a weak form of synchronization in which only one node in N_X and one node in N_Y coordinate their respective initial states of their local X and Y computations. However, if the node in N_X that initiated the X computation forks off

the main thread for the Y computation, then this form of synchronization between the initiations of X and Y is adequate to have Y as an entirely nested computation within X.

$R4^*(X,Y)$: This group of relations deals with L_X and U_Y. The relations signify different degrees of synchronization between a monitoring computation Y that knows the initial values with which the X computation begins, and then the monitoring computation Y terminates.

- $R4a(X,Y)$: The Y computation at any node in N_Y terminates only after that node knows the initial values of the X computation at each node in N_X. This is a strong form of synchronization between the start of X and the end of Y.
- $R4b(X,Y)$: For every node in N_X, the initial state of its X computation is known by at least one node in N_Y before that node in N_Y terminates its Y computation. Even if there is no exchange of information in the Y computation about the state of the X computation at individual nodes in N_X, this relation guarantees that when Y completes, the (initial) local states at each of the N_X nodes are collectively (but not individually) known by N_Y.
- $R4b'(X,Y)$: Before terminating its Y computation, some node in N_Y knows the initial state of the X computation at all the nodes in N_X. This node in N_Y can detect if an initial global predicate of the X computation across the nodes in N_X is satisfied, before it terminates its Y computation. If this node in N_Y is a group leader, it can then inform the other nodes in N_Y to terminate their Y computations.
- $R4c(X,Y)$: The initial state of the X computation at some node in N_X is known to all the nodes in N_Y before they terminate their Y computation. This weak synchronization is adequate for applications where all the N_X nodes start their X computation with similar values. Alternately, if the node in N_X behaves as a group leader, it can first detect the initial global state of the X computation and then inform all the nodes in N_Y.
- $R4c'(X,Y)$: Each node in N_Y terminates its Y computation only after it knows the initial state of the X computation at some node in N_X. This is a weaker form of synchronization than $R4c$ because the states of all nodes in N_X may not be observed before the nodes in N_Y terminate their Y computation. But this will be adequate for applications in which each node in N_X is reporting the same state/value of the X computation, and each node in N_Y simply needs a confirmation from some node in N_X before it terminates its Y computation. For example, a mobile host (an N_Y node) may simply need a confirmation from some base station (an N_X node) before it exits its Y computation.
- $R4d(X,Y)$: Some node in N_Y terminates its Y computation after it knows the initial state of the X computation at some node in N_X. This weak form of synchronization is sufficient when the group leader of X which is responsible for kicking off the rest of X informs some node (or the group leader) of the monitoring distributed program Y that computation X has successfully begun.

Consider enforcing group mutual exclusion (*gmutex*) between two groups of processes G_1 and G_2. Multiple processes of either group, but not both groups, are permitted to access some critical resources, such as distributed database records, at any time. Relations $R1*$ represent different degrees of gmutex that can be enforced, as explained for $R1^*(X,Y)$ earlier in this section. Also, the strongest form of gmutex, $R1a$, can also be enforced by $R1b'$, $R1c$, and $R1d$, if the communicating nodes in G_1 /G_2 are the respective group leaders. Thus, for $R1b'$, the nodes in G_1 communicate their states (gmutex tokens) to the group leader of G_2 which then collects all these states (gmutex tokens), and distributes them within G_2. For $R1c$, the group leader of G_1 collects all the gmutex tokens from G_1, then informs all the nodes in G_2. For $R1d$, the group leader of G_1 collects all the gmutex tokens from G_1, then informs the group leader of G_2 which then informs all the nodes in G_2. The above four ways of expressing the distributed mutual exclusion provide a choice in trade-offs of (i) knowledge of membership of G_1 and/or G_2, by members and/or group leaders, within each group and across groups (this is further complicated with mobile processes), (ii) different delay or the number of message exchange phases to achieve gmutex, (it is critical to have rapid exchange of access rights to the distributed database), (iii) different number of messages exchanged to achieve gmutex, (bandwidth is a constraint, particularly with the use of crypto techniques), and (iv) fault-tolerance implications (critical to sensitive applications).

4 Concluding Remarks

We showed the uses of fine-grained synchronization relations by applications that use nonatomicity in modeling actions and need a fine degree of granularity to specify synchronization relations and their composite global predicates. We showed the specific meaning and significance of each relation. Some uses of the relations in a distributed system include modeling various forms of synchronization for group mutual exclusion, initiation of nested computing, termination of nested computing, and monitoring the start of a computation. The synchronization between any X and Y computations can be performed at any "synchronization barrier" for the X and Y computations of any application. For example, $R2^*(X,Y)$ synchronization can be performed at a barrier to ensure that subcomputation X which is nested in subcomputation Y completes before subcomputation Y completes, following which $R3^*(Y,X)$ synchronization can be performed to kick off another nested subcomputation X within Y. This barrier synchronization may involve blocking of processes which need to wait for expected messages, analogous to the barrier synchronization for multiprocessor systems [17].

The synchronization relations provide a precise handle to express various types of synchronization conditions in first-order logic. Each synchronization performed satisfies a global predicate in the execution. (A classification of some types of global predicates is given in [3,6].) Complex global predicates can be formed by logical expressions on such synchronization relations. It is an inter-

esting problem to classify the global predicates that can be specified using the synchronization relations.

Observe that performing a synchronization (corresponding to one of the relations) involves message passing, and hence also provides a direct way to *evaluate* global state functions and global predicates involving the nodes participating in the synchronization. Thus, performing the synchronization enables the detection of global predicates. Also, global predicates can be *enforced* by initiating the synchronization only when some local predicates become true. Identifying the types of global predicates that can be detected or enforced using the synchronization relations is an interesting problem. The design of algorithms to enforce global predicates is also a topic for study.

References

1. *Linear time, branching time, and partial orders in logics and models of concurrency*, J. W. de Bakker, W. P. de Roever, G. Rozenberg (Eds.), LNCS 354, Springer-Verlag, 1989.
2. J. V. Benthem, *The Logic of Time*, Kluwer Academic Publishers, (1ed. 1983), 2ed. 1991.
3. R. Cooper, K. Marzullo, Consistent detection of global predicates, *ACM/ONR Workshop on Parallel and Distributed Debugging*, 163-173, May 1991.
4. C. A. Fidge, Timestamps in message-passing systems that preserve partial ordering, *Australian Computer Science Communications*, Vol. 10, No. 1, 56-66, Feb. 1988.
5. P. C. Fishburn, *Interval Orders and Interval Graphs: A Study of Partially Ordered Sets*, J. Wiley & Sons, 1985.
6. V. Garg, B. Waldecker, Detection of weak unstable predicates in distributed programs, *IEEE Transactions on Parallel and Distributed Systems*, 5(3), 299-307, March 1994.
7. W. Janssen, M. Poel, J. Zwiers, Action systems and action refinement in the development of parallel systems, In J.C. Baeten, J.F. Groote, (Eds.) *Concur91*, LNCS 527, Springer-Verlag, 298-316, 1991.
8. A. Kshemkalyani, Temporal interactions of intervals in distributed systems, *TR-29.1933, IBM*, Sept. 1994.
9. A. Kshemkalyani, Temporal interactions of intervals in distributed systems, *Journal of Computer and System Sciences*, 52(2), 287-298, April 1996. (Contains some parts of [8]).
10. A. Kshemkalyani, Framework for viewing atomic events in distributed computations, *Theoretical Computer Science*, 196(1-2): 45-70, April 1998.
11. A. Kshemkalyani, Relative timing constraints between complex events, *8th IASTED Conf. on Parallel and Distributed Computing and Systems*, 324-326, Oct. 1996.
12. A. Kshemkalyani, Synchronization for distributed real-time applications, *5th Workshop on Parallel and Distributed Real-time Systems*, IEEE CS Press, 81-90, April 1997.
13. L. Lamport, Time, clocks, and the ordering of events in a distributed system, *CACM*, 558-565, 21(7), July 1978.
14. L. Lamport, On interprocess communication, Part I: Basic formalism, Part II: Algorithms, *Distributed Computing*, 1:77-101, 1986.

15. F. Mattern, Virtual time and global states of distributed systems, *Parallel and Distributed Algorithms*, North-Holland, 215-226, 1989.
16. F. Mattern, On the relativistic structure of logical time in distributed systems, In: Datation et Controle des Executions Reparties, *Bigre*, 78 (ISSN 0221-525), 3-20, 1992.
17. J. Mellor-Crummey, M. Scott, Algorithms for scalable synchronization on shared-memory multiprocessors, *ACM Transactions on Computer Systems*, 9(1): 21-65, Feb. 1991.
18. E.R. Olderog, *Nets, Terms, and Formulas*, Cambridge Tracts in Theoretical Computer Science, 1991.
19. A. Rensink, *Models and Methods for Action Refinement*, Ph.D. thesis, University of Twente, The Netherlands, Aug. 1993.
20. R. Schwarz, F. Mattern, Detecting causal relationships in distributed computations: In search of the holy grail, *Distributed Computing*, 7:149-174, 1994.