Concurrent Common Knowledge: A New Definition of Agreement for Asynchronous Systems

Prakash Panangaden*
Kim Taylor**

TR 89-1011
May 1989
(Revised July 1990)

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Supported in part by NSF Grants DCR-8602072 and CCR-8818979.
**Supported in part by an AT&T Ph.D Scholarship.

Concurrent Common Knowledge: A New Definition of Agreement for Asynchronous Systems*

Prakash Panangaden[†‡]
Kim Taylor[§¶]

Cornell University, Ithaca, New York

Abstract

In this paper we present a new, knowledge-theoretic definition of agreement designed for asynchronous systems. In analogy with common knowledge, it is called *concurrent common knowledge.* Unlike common knowledge, it is a form of agreement that is attainable asynchronously. In defining concurrent common knowledge, we give a logic with new modal operators and a formal semantics, both of which are based on causality and consequently capture only the relevant structure of purely asynchronous systems. We give general conditions by which protocols attain concurrent common knowledge and prove that two simple and efficient protocols do so. We also present several applications of our logic. We show that concurrent common knowledge is a necessary and sufficient condition for the concurrent performance of distributed actions. We also demonstrate the role of knowledge in checkpointing and asynchronous broadcasts. In general, applications that involve all processes reaching agreement about some property of a consistent global state can be understood in terms of concurrent common knowledge.

*An earlier version of this work by the same title appears in *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, August 1988.

†Supported in part by NSF grants DCR-8602072 and CCR-8818979.

‡Current address: School of Computer Science, McGill University, Montreal, Quebec, Canada H3A 2A7.

§Supported in part by an AT&T Ph.D. Scholarship.

¶Current address: Department of Information and Computing Sciences, University of California at Santa Cruz, Santa Cruz, California 95064.

1 Introduction

Knowledge-theoretic logic has become an important tool for reasoning about communication and cooperation in distributed systems [11,8,16,10]. In this approach, one reasons about what processes "know" about the states of other processes. Cooperation on a distributed task is characterized by its requirement of some form of "group knowledge," and communication is viewed as a means of transferring knowledge. In [11], *common knowledge* is proposed as a definition for agreement in distributed systems. Common knowledge of a fact ϕ implies all knowledge states of the form, "everyone knows ϕ and everyone knows that everyone knows ϕ and everyone knows that everyone knows that everyone knows ϕ" and so on. Common knowledge, however, requires simultaneous action for its achievement and is consequently unattainable in asynchronous systems.

In this paper we discuss a new, knowledge-theoretic definition of agreement appropriate for asynchronous systems. This definition has two important features: first, it uses the causality relation between events in its definition [15] rather than physical time and, second, this form of knowledge is actually attainable in an asynchronous system. In analogy with common knowledge, we call it *concurrent common knowledge.* The idea behind concurrent common knowledge is quite natural. Given that ordinary common knowledge must be attained *simultaneously* by all processes, it seems clear that a viable alternative for asynchronous systems could use causality rather than real time.

The idea that causal structure is fundamental to the analysis of asynchronous systems was brought into computer science by Lamport[15]. The appropriate causal analog of a *real-time global state*, i.e. a global state corresponding to the system at an instant of real time, is a *possible global state* or, as it is now widely called, a *consistent cut.* It is the appropriate analog because—in asynchronous systems—no process can distinguish whether or not a consistent cut is, in fact, a real-time global state. Intuitively, we define *everyone concurrently knows* to be true at a consistent cut if all processes know that ϕ is true of some "indistinguishable" consistent cut. Concurrent common knowledge of a fact ϕ then implies all formulas of the form "everyone concurrently knows ϕ and everyone concurrently knows that everyone concurrently knows ϕ," and so on.

In order to define concurrent common knowledge, we present a logic with new modal operators. Truth values are assigned to the formulas of this logic via a new *asynchronous-runs semantics* in contrast to the commonly-used timed-runs semantics of Halpern and Moses[11,10]. We find this new semantics more natural for expressing our formulas as it contains only the causal structure relevant to asynchronous systems and not real time, which is unobservable in such systems. We give a translation of our semantics to the timed-runs semantics; this allows us to compare rigorously concurrent common knowledge to knowledge forms defined in the timed-runs semantics.

We prove a general condition under which protocols achieve concurrent common knowledge and give two simple and efficient protocols that do so. Several applications of our new logic are given. We show that concurrent common knowledge is a necessary and sufficient condition for performing *concurrent actions* in asynchronous distributed systems, analogously to simultaneous actions and common knowledge [16] in synchronous systems. It is shown that the checkpointing algorithm of [3] achieves two forms of concurrent common knowledge. In general, applications that involve all processes reaching agreement about some property of a consistent global state can be understood in terms of concurrent common knowledge; thus we have isolated the form of knowledge underlying many existing protocols. We also give results pertaining to broadcast message ordering and replicated data updates.

The paper is organized as follows. Section 2 contains our system model. In Section 3 we define our logic and its formal semantics. Section 4 contains our Attainment Theorem, followed by two protocols which satisfy conditions of that theorem and hence attain concurrent common knowledge. In Section 5 several applications of our logic are presented. In Section 6 we give a translation of our semantics into the standard timed-runs semantics [11], and formally compare concurrent common knowledge to common knowledge and other variants of common knowledge. Section 7 contains concluding remarks.

2 System Model

The definitions that we give in this section describe asynchronous, distributed systems [15]. By the term *distributed*, we mean that the system is composed of a set of processes that can communicate only by sending messages along a fixed set of channels. The network is not necessarily completely connected. By *asynchronous*, we mean that there is no global clock in the system, the relative speeds of processes are independent, and the delivery time of messages is finite but unbounded.

It is our intention to give a definition of the model that uses the structures that are relevant to such systems. Thus we do not use timed runs to describe these systems [11,10]. The resulting definitions turn out to be more natural than if we had detailed timing information in the model description. Of course, the timed runs model is more generally applicable than ours. In Section 6.1 we give a precise translation of our formalism in the timed runs formalism. Our model turns out to be similar to that of Chandy and Misra [4].

The description of distributed systems is based on the behaviors of the individual components or processes in the system. We take the notion of a *local state* of a process to be primitive. *Actions*, as in [14,10], are state transformers.

Definition 1. An *action* is a function from local states to local states. There are three types of actions: *send actions* denoted *send*(m) where m is a message (described later), *receive actions* denoted *receive*(m), and *internal actions.*

We use local states and actions to compose *local histories* as in [14,10].

Definition 2. A *local history*, h_i, of process i, is a (possibly infinite) sequence of alternating *local states*—beginning with a distinguished *initial state*—and *actions.* We write such a sequence as follows.

$$h_i = s_i^0 \xrightarrow{\alpha_i^1} s_i^1 \xrightarrow{\alpha_i^2} s_i^2 \xrightarrow{\alpha_i^3} s_i^3 \ldots$$

We use s_i^j (α_i^j) to refer to the j^{th} state (action) in process i's local history.

Definition 3. An *event* is a state transition $\langle s_i^{j-1}, \alpha_i^j, s_i^j \rangle$ and is denoted e_i^j.

Multiple events in a history having the same action correspond to different instantiations of that action. We denote *receive events* and *send events* as for the corresponding action, with an additional parameter to uniquely identify instantiations: $send(k, m)$ and $receive(k, m)$, where k is a natural number and m is a message. We require that the k^{th} occurrence of the sending of a particular message in a local history have instantiation number k. (A similar requirement for receives depends on the channel behavior and is discussed later.)

Constraint (1) If e_i^j is the k^{th} event of h_i of the form $send(k', m)$, where m is a message, then $k' = k$.

The instantiation number is only part of the model and is used to describe certain behaviors in the remainder of this section; however, it is not necessarily a part of the messages themselves. Instantiation numbers in messages would require that they be arbitrarily long; we do not impose such a requirement.

The state of a process can be obtained from its initial state and the sequence of actions or events that have occurred up to the current state. Hence the local history may be equivalently described as either of the following:

$$h_i = s_i^0, \alpha_i^1, \alpha_i^2, \alpha_i^3 \ldots$$
$$h_i = s_i^0, e_i^1, e_i^2, e_i^3 \ldots$$

where $e_i^j = \langle s_i^{j-1}, \alpha_i^j, s_i^j \rangle$. If it is additionally assumed that the local state includes a description of all past actions in the local history (corresponding to the *complete history* interpretation of [11,10]), then

$$h_i = s_i^0, s_i^1, s_i^2, s_i^3 \ldots$$

is also equivalent to the above formulas. We will assume such an interpretation; note that this interpretation results in a maximum amount of information being available to a process based on its local state. We often omit the subscript or superscript on states, events, and actions when it is obvious or irrelevant.

An asynchronous system consists of the following sets.

1. A set $Proc = \{1, \ldots, N\}$ of *process identifiers*, where N is the total number of processes in the system.

2. A set $C \subseteq \{(i, j) : i, j \in Proc\}$ of *channels*. The occurrence of (i, j) in C indicates that process i can send messages to process j.

3. A set H_i of possible *local histories* for each process i in *Proc*.

4. A set A of *asynchronous runs.* Each asynchronous run is a vector of local histories, one per process, indexed by process identifiers. Thus we use the notation

$$a = \langle h_1, h_2, h_3, ...h_N \rangle.$$

   Constraints on the set A are described throughout this section.

5. A set M of *messages.* A message is a triple $\langle i, j, B\rangle$ where $i \in Proc$ is the sender of the message, $j \in Proc$ is the message recipient, and B is the body of the message. B can be either a special value (e.g. a tag to denote a special-purpose message), or some proposition about the run (e.g. "i has reset variable X to zero"), or both.

The set of channels C and our assumptions about their behavior induce two constraints on the runs in A. The first constraint corresponds to our intuitive notion of channels: i cannot send a message to j unless (i, j) is a channel. The second constraint says that, if the reception of a message m is in the run, then the sending of m must also be in that run; this implies that the network cannot introduce spurious messages or alter messages.

Constraint (2) $send(k, \langle i, j, \phi\rangle) \in h_i \Rightarrow (i, j) \in C$

Constraint (3) $receive(k, \langle i, j, \phi\rangle) \in h_j \Rightarrow send(k, \langle i, j, \phi\rangle) \in h_i$

In addition, we introduce two optional channel constraints: reliability and FIFO. Reliability says that if a message is sent then it is received, i.e. no message loss occurs. FIFO indicates that channels exhibit first-in-first-out behavior. These properties are not necessary for our definitions, but we will want to address systems that satisfy them when we address the attainability of concurrent common knowledge. Unless otherwise stated, they will not be assumed in the model.

Reliability Constraint: $send(k, \langle i, j, \phi\rangle) \in h_i \Rightarrow receive(k, \langle i, j, \phi\rangle) \in h_j$

FIFO Constraint: if $\alpha_i^w = send(k_1, \langle i, j, \phi_1\rangle)$, $\alpha_i^x = send(k_2, \langle i, j, \phi_2\rangle)$, and $w < x$, then there exists actions $\alpha_j^y = receive(k_1, \langle i, j, \phi_1\rangle)$ and $\alpha_j^z = receive(k_2, \langle i, j, \phi_2\rangle)$ such that $y < z$.

Our model of an asynchronous system does not mention time. There is, however, an ordering of events in the system due to the fact that certain events are known to precede other events. We can define this order using *potential causality* as done by Lamport [15]. Intuitively, two events exhibit potential causality if it is possible for one to have an effect on the other. In an asynchronous system, potential causality results only from sequential execution on single processes and from message passing between separate processes. It is described using the *happens-immediately-before* relation $\mapsto$ and the *happens-before* relation $\rightarrow$.

Definition 4. Event e_i^x *happens-immediately-before* event e_j^y, denoted $e_i^x \mapsto e_j^y$, if and only if (1) e_i^x and e_j^y are different events in the history of some process i and e_i^x occurs first in the sequence, i.e. $i = j$ and $x < y$, or (2) e_i^x is an instantiation of the sending of a message and e_j^y is the same instantiation of the reception of that message; i.e. there exists m and k such that $e_i^x = send(k, m)$ and $e_j^y = receive(k, m)$.

Definition 5. The *happens-before* relation, denoted $\rightarrow$, is the transitive closure of happens-immediately-before.

Thus if $e_i^x \rightarrow e_j^y$, then either $e_i^x \mapsto e_j^y$ or there exists an event e_k^z such that $e_i^x \rightarrow e_k^z$ and $e_k^z \mapsto e_j^y$.

Our final requirement is that the reflexive closure of $\rightarrow$ be anti-symmetric, which is necessary if the system is to model actual executions.

Constraint (4) For no two events e_1 and e_2 does $e_1 \rightarrow e_2$ and $e_2 \rightarrow e_1$.

Our requirements on asynchronous runs are equivalent to those in [4], with the exception that we limit message sending to occur along the set of designated channels. Chandy and Misra express the possible behaviors of systems in terms of totally ordered sets of events called *system computations.* Their conditions on system computations are that (i) projections on each process are possible local histories, and (ii) the reception of a message is preceded by its sending. These are equivalent to stating that the system computations are linearizations of the $\rightarrow$ relation.

We can now use Lamport's theory to talk about global states of an asynchronous system. A global state is some prefix of a run, as defined below.

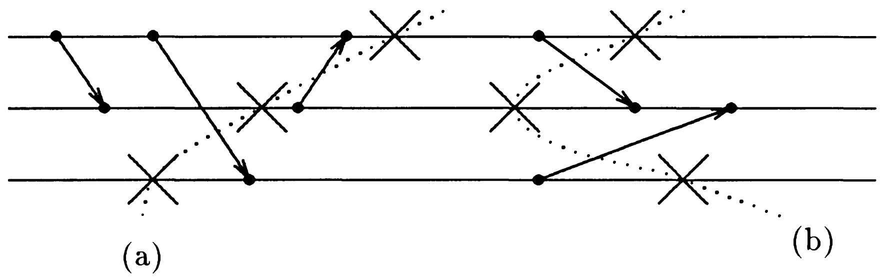

Figure 1: Inconsistent (a) vs. consistent (b) cuts.

Definition 6. A *global state* of run a is an N-vector of prefixes of local histories of a, one prefix per process.

The happens-before relation can be used to define a *consistent* global state [19,3], often termed a *consistent cut*, as follows.

Definition 7. A *consistent cut* of a system is any global state such that if $e_i^x \to e_j^y$ and e_j^y is in the global state, then e_i^x is also in the global state.

(See Figure 1. The states of (b) form a consistent cut whereas those of (a) do not.) We will denote a consistent cut by a vector of local states $c = \langle s_1, \ldots, s_N \rangle$; we also use the notation $c[i]$ to indicate the local state of i in cut c.

For simplicity in the remainder of the paper, we will not explicitly mention the instantiation number of events.

We often refer to causally-related *message chains* as defined below.

Definition 8. In an asynchronous run, a *message chain* is a (possibly infinite) sequence of messages $m_1, m_2, m_3, \ldots$ such that, for all i, $receive(m_i) \to send(m_{i+1})$. Consequently,

$$send(m_1) \to receive(m_1) \to send(m_2) \to receive(m_2) \to send(m_3) \ldots.$$

Finally, the following lemma establishes a desirable property of asynchronous runs; its proof is contained in Appendix A.1.

Lemma 1. In any asynchronous run of any system, each local state of each process is included in some consistent cut of the system.

In any state of the history of process i, i cannot determine which of the possible consistent cuts including its current state is an actual real-time global state, i.e. a set of local states that actually occur at the same instant of physical time during the execution. In this sense, a consistent cut is indistinguishable from a real-time global state. In defining epistemic concepts, the notion of indistinguishability plays a key role. For this reason we have chosen to use consistent cuts rather than real time in our logic for reasoning about asynchronous distributed systems.

3 Semantics of Concurrent Knowledge

The definition of concurrent common knowledge follows the standard pattern of defining a form of group knowledge and then using a greatest fixed point operator to define the appropriate variant of common knowledge [11].

In order to give a Kripkean interpretation of the knowledge modality, we need to identify an appropriate set of possible worlds and a family of possibility relations between those worlds. The discussion of concurrent knowledge really involves two modal operators and, hence, two different collections of accessibility relations in the semantics. This kind of situation is also seen in other variants of common knowledge. Discussions of eventual, epsilon, and timestamped common knowledge [11,17] involve a temporal modality in addition to an epistemic modality.

3.1 The Logic

We will first introduce the symbols contained in our logic. Later we will define a formal semantics by stating when a formula is satisfied by a pair (a, c), where c is a consistent cut in asynchronous run a.

We assume that there is a set of primitive propositions *Prop*; these typically will be statements like "variable x in process i is 0" or "process i has sent a message m to process j". We represent these by lower-case letters $p, q, \ldots$.

We introduce two families of modal operators, each family indexed by process identifiers. They are written K_i and P_i respectively. Intuitively, $K_i(\phi)$ represents the statement "i knows ϕ," which in terms of asynchronous systems means "ϕ is true in all possible consistent global states that include

i's local state." The formula $P_i(\phi)$ represents the statement "there is some consistent global state in this run, including i's local state, in which ϕ is true." P has, roughly speaking, a role similar to a temporal modality. It is quite different, however, from the familiar temporal operators like $\Diamond$.

The next modal operator is written E^C and stands for "everyone concurrently knows." The definition of $E^C(\phi)$ is

$$E^C(\phi) \Leftrightarrow \bigwedge_{i \in Proc} K_i P_i(\phi)$$

The last modal operator that we introduce is C^C, concurrent common knowledge. Analogously to common knowledge, we wish to define a state of process knowledge that implies that all processes are in that same state of knowledge, with respect to ϕ, along some cut of the run. In other words, we want a state of knowledge X satisfying

$$X = E^C(\phi \wedge X).$$

Thus we want concurrent common knowledge to be a fixed point of $E^C(\phi \wedge X)$. C^C will be defined semantically as the weakest such fixed point, namely as the greatest fixed-point of $E^C(\phi \wedge X)$. It therefore satisfies

$$C^C(\phi) \Leftrightarrow E^C(\phi \wedge C^C(\phi))$$

and

$$C^C(\phi) \Rightarrow E^C(\phi) \wedge (E^C)^2(\phi) \wedge (E^C)^3(\phi) \ldots,$$

i.e. that $(E^C)^k \phi$ holds for any k. The greatest fixed point definition is, however, stronger than the infinite conjunction.

3.2 Formal Semantics

In an asynchronous system, the possible worlds are the consistent cuts of the set of possible asynchronous runs A. We use pair (a, c) to stand for the consistent cut c in the asynchronous run a. Recall that a cut is an N-vector of local states, one for each process in $Proc$. Two cuts are viewed as indistinguishable by process i if they contain the same local state of process i. This is clearly an equivalence relation.

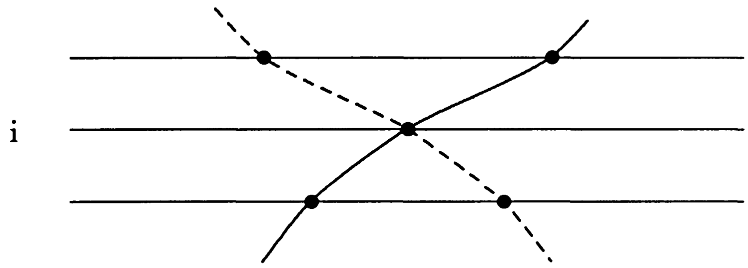

Figure 2: Satisfaction of $(a, c) \models P_i(\phi)$.

Definition 9. We write $(a, c) \sim_i (a', c')$ to represent the indistinguishability of (a, c) and (a', c') to i:

$$(a, c) \sim_i (a', c') \Leftrightarrow (a, c)[i] = (a', c')[i]$$

The formal semantics is given via the definition of the satisfaction relation $\models$. We assume that we are given a function π that assigns a truth value to each primitive proposition p and local state s of process i. The truth of a primitive proposition p in (a, c) is determined by π and c. This defines $(a, c) \models p$. The satisfaction relation is defined in the obvious way for formulas built up using the logical connectives. The following defines the meaning of K_i in our setting:

$$(a, c) \models K_i(\phi) \Leftrightarrow \forall (a', c') \ : \ (a', c') \sim_i (a, c) \ : \ (a', c') \models \phi$$

This is practically the same as the definition in Halpern and Moses [11], except that we use asynchronous runs rather than timed runs. This definition also validates the standard axioms for knowledge.

The meaning of P_i is given by the following definition

$$(a, c) \models P_i(\phi) \Leftrightarrow \exists (a, c') \ : \ (a, c') \sim_i (a, c) \ : \ (a, c') \models \phi$$

In other words, $P_i(\phi)$ states that there is some cut, *in the same asynchronous run*, including i's current state, such that ϕ is true in that cut. (See Figure 2.) Another way of viewing the meaning of P_i is to define the equivalence relation $\approx_i$ to stand for indistinguishable cuts in the same run; this is a refinement of the $\sim_i$ relation.

$$(a, c) \approx_i (a', c') \Leftrightarrow (a = a') \wedge (a, c) \sim_i (a', c')$$

Given the definition of $\approx_i$ we can equivalently define P_i as

$$(a, c) \models P_i(\phi) \Leftrightarrow \exists (a', c') \ : \ (a', c') \approx_i (a, c) \ : \ (a', c') \models \phi.$$

Note that $\phi \Rightarrow P_i(\phi)$. This makes $E^C(\phi) \equiv \wedge_{i \in Proc} K_i P_i(\phi)$, concurrent knowledge, weaker than $E(\phi) \equiv \wedge_{i \in Proc} K_i(\phi)$, "everyone knows". Consequently, unlike E, its greatest fixed point will be attainable asynchronously.

It is not the case, in general, that $P_i(\phi) \Rightarrow \phi$ or that $E^C(\phi) \Rightarrow \phi$. Note that the truth of $E^C(\phi)$ is determined with respect to some cut (a, c). A process cannot distinguish which cut, of the perhaps multiple cuts that are in the run and consistent with its local state, satisfies ϕ, i.e. whether or not c does; it can only know the existence of such a cut. $E^C(\phi)$ does imply ϕ for certain types of facts, as we will discuss at the end of this section.

The remainder of our formal semantics outlines the definition of C^C using greatest fixed points. In order to define the meaning of C^C using fixed points we need to define the meaning of formulas with free variables, written $X, Y \ldots$, in them. We think of the meaning of such a formula as a function from sets of consistent cuts to sets of consistent cuts. Let us call the set of consistent cuts W. Then we can define the following meaning function for all formulas. We let Z stand for a generic subset of W. The meaning of the formulas is given by the inductively defined function $\mathcal{M}$. The meaning of the primitive propositions $p, q, \ldots$ is given by a function π as discussed above. The meaning function defined below follows, very closely, the definition given by Halpern and Moses [11,13].

1. $\mathcal{M}[\![p]\!](Z) = \{u \in W | \pi(u, p) = \mathbf{true}\}$ where p is a primitive proposition.

2. $\mathcal{M}[\![\neg\phi]\!](Z) = W - \mathcal{M}[\![\phi]\!](Z)$.

3. $\mathcal{M}[\![\phi \wedge \psi]\!](Z) = \mathcal{M}[\![\phi]\!](Z) \cap \mathcal{M}[\![\psi]\!](Z)$.

4. $\mathcal{M}[\![X]\!](Z) = Z$.

5. $\mathcal{M}[\![K_i(\phi)]\!](Z) = \{(a,c) \in W | \forall (a',c') \in W : (a,c) \sim_i (a',c') \Rightarrow (a',c') \in \mathcal{M}[\![\phi]\!](Z)\}$.

6. $\mathcal{M}[\![P_i(\phi)]\!](Z) = \{(a,c) \in W | \exists (a',c') \in W : (a,c) \approx_i (a',c') \Rightarrow (a',c') \in \mathcal{M}[\![\phi]\!](Z)\}$.

If a formula does not contain a free variable then its meaning is a constant function. The truth value definition of the semantics can be recovered by defining

$$(a,c) \models \phi \text{ iff } (a,c) \in \mathcal{M}[\![\phi]\!](\emptyset).$$

In fact the semantic clauses just given are exactly what one would expect for the Tarski-style truth definition except that they have been given in terms of sets and set operations instead of truth values and logical connectives.

We will define C^C via a greatest fixed point operator. We extend the syntax by $\nu X.\phi$. The interpretation of this proceeds as follows. The existence of the greatest fixed point depends upon the *monotonicity* of $\mathcal{M}[\![\phi]\!](Z)$; a function f is monotonic if $A \subseteq B$ implies $f(A) \subseteq f(B)$. To guarantee monotonicity, we require that free occurrences of X in ϕ be *positive*, i.e. all occurrences of X are in the scope of an even number of negation signs. This is clearly a syntactic property. It is easy to see (by induction on the structure of formulas) that $\mathcal{M}[\![\phi]\!](Z)$ will be a monotonic function if X appears positively. Any monotonic function on a complete lattice has a greatest fixed point [20]. The powerset 2^W ordered by inclusion is certainly a complete lattice. We can thus give meaning to $\nu X.\phi$ as

$$\mathcal{M}[\![\nu X.\phi]\!](Z) = \cup\{B | \mathcal{M}[\![\phi]\!](B) = B\}$$

and $C^C(\phi)$ can be viewed as a special case of this as follows:

$$C^C(\phi) = \nu X.E^C(\phi \wedge X)$$

It is not true that $E^C(\phi \wedge X)$ defines a continuous function so the fixed point is not attained by simply iterating up to ω; in this sense C^C is rather like $C^\diamond$ [11]. Note that in the logic the only occurrence of greatest fixed points is through occurrences of C^C. In such occurrences one never has to interpret formulas like $\nu X.\nu Y.X \wedge Y$ where there are fixed points of expressions containing more than one free variable.

The fact that concurrent common knowledge is a greatest fixed point is expressed by an induction rule. The following theorem gives this rule and establishes its soundness.

Theorem 1. The following induction rule is sound with respect to the semantics defined above. If $\phi \Rightarrow E^C(\phi)$ is valid then so is $\phi \Rightarrow C^C(\phi)$.

Proof: Assume that $\phi \Rightarrow E^C(\phi)$. Semantically, this means that $\mathcal{M}[\![\phi]\!](\emptyset) \subseteq \mathcal{M}[\![E^C(\phi)]\!](\emptyset)$. Recall, the meaning of $C^C(\phi)$ is the greatest fixed point of the functional $F = \lambda u.\mathcal{M}[\![E^C(\phi \wedge X)]\!](u)$, where u is an element of W. Now

$$\begin{aligned}&F(\mathcal{M}[\![\phi]\!](\emptyset)) = \mathcal{M}[\![E^C(\phi \wedge X)]\!](\mathcal{M}[\![\phi]\!](\emptyset))\\ &= \mathcal{M}[\![E^C(\phi)]\!](\mathcal{M}[\![\phi]\!](\emptyset)) \cap \mathcal{M}[\![E^C(X)]\!](\mathcal{M}[\![\phi]\!](\emptyset))\\ &= \mathcal{M}[\![E^C(\phi)]\!](\emptyset).\end{aligned}$$

The first equality is by definition of F; the second and third are by explicit calculation, using the semantic definitions. Thus, by our assumption, $\mathcal{M}[\![\phi]\!](\emptyset) \subseteq F(\mathcal{M}[\![\phi]\!](\emptyset))$. Since F is monotonic, we have the following chain of inclusions

$$\mathcal{M}[\![\phi]\!](\emptyset) \subseteq F(\mathcal{M}[\![\phi]\!](\emptyset)) \subseteq F(F(\mathcal{M}[\![\phi]\!](\emptyset)))\ldots$$

Thus $\mathcal{M}[\![\phi]\!](\emptyset) \subseteq \cup_\alpha F^\alpha(\mathcal{M}[\![\phi]\!](\emptyset))$, where α ranges through the ordinals. Now $\cup_\alpha F^\alpha(\mathcal{M}[\![\phi]\!](\emptyset))$ is a fixed point of F, hence, $\cup_\alpha F^\alpha(\mathcal{M}[\![\phi]\!](\emptyset)) \subseteq \mathcal{M}[\![C^C(\phi)]\!]$. Thus we get $\mathcal{M}[\![\phi]\!](\emptyset) \subseteq \mathcal{M}[\![C^C(\phi)]\!]$; which establishes the soundness of the induction rule.∎

As noted earlier, it follows from these definitions that

$$C^C(\phi) \Leftrightarrow E^C(\phi \wedge C^C(\phi))$$

and that $C^C(\phi) \Rightarrow (E^C)^k(\phi)$ for any natural number k. It does not follow from the definitions, in general, that

$$C^C(\phi \Rightarrow \psi) \wedge C^C(\phi) \Rightarrow C^C(\psi);$$

or that $C^C(\phi) \Rightarrow \phi$. As noted earlier for $E^C(\phi)$, this is because processes cannot distinguish whether or not ϕ holds on the same cut on which $C^C\phi$ holds; rather, they know that it holds on some indistinguishable cut in the current run.

The operators P_i, E^C, and C^C have stronger properties for *local* facts. A local fact is one that is determined solely by the local state of some process; for example, a fact regarding a value contained only in the local memory of that process. The following definition is equivalent to that of Chandy and Misra [4].

Definition 10. A fact ϕ is *local to process i* if $\phi \Rightarrow K_i\phi$.

For a fact ϕ that is local to process i, it is the case that $P_i \Rightarrow \phi$. Furthermore, if a fact ϕ is local to *any* process, then $E^C(\phi) \Rightarrow \phi$ and $C^C(\phi) \Rightarrow \phi$.

Theorem 2. If ϕ is local to process i, then $P_i(\phi) \Rightarrow \phi$.

Proof: Suppose that ϕ is local to process i. Suppose also that for some cut (a,c), $(a,c) \models P_i(\phi)$ but $(a,c) \not\models \phi$. By the definition of P_i, there is some cut c' in run a for which $(a,c) \sim_i (a,c')$ and $(a,c') \models \phi$. By the definition of local, it must also be the case that $(a,c) \models \phi$. This contradicts the assumption that $(a,c) \models P_i(\phi)$ but $(a,c) \not\models \phi$. ∎

A corollary to the previous theorem follows because, for any fact ϕ, $E^C(\phi) \Rightarrow P_i(\phi)$ for every process i and $C^C(\phi) \Rightarrow E^C(\phi)$.

Corollary 1. If fact ϕ is local to any process, then $E^C(\phi) \Rightarrow \phi$ and furthermore $C^C(\phi) \Rightarrow \phi$.

4 Attainment of CCK

For ordinary common knowledge, C, it is a theorem that if C is attained then all processes learn it simultaneously [11,18]. An analogous theorem holds for concurrent common knowledge. Before stating the theorem, we will first formalize the notions of "attainment" and "learning."

Definition 11. A fact ϕ is *attained* in run a if $\exists c\ :\ (a,c) \models \phi$.

Likewise, we say that a system attains ϕ if every run of the system attains ϕ.

In this section and the following section we will often refer to "knowing" a fact in a state rather than in a consistent cut. Recall that knowledge is dependent only on the local state of a process, since $(a, c) \sim_i (a', c')$ iff $(a, c)[i] = (a', c')[i]$; therefore such terminology is reasonable. Formally, i *knows* ϕ *in state* s is shorthand for

$$\forall (a, c) \ : \ (a, c)[i] = s \ : \ (a, c) \models \phi.$$

Definition 12. Process i *learns* ϕ *in state* s_i^j of run a if i knows ϕ in s_i^j and, for all states s_i^k in run a, $k < j$, i does not know ϕ.

The following theorem says that if $C^C(\phi)$ is attained in a run then all processes i learn $P_i C^C(\phi)$ along a single consistent cut.

Theorem 3. If $C^C(\phi)$ is attained in a run a, then the set of states in which all processes learn $P_i C^C(\phi)$ forms a consistent cut in a.

Proof: $C^C(\phi)$ is attained in a implies that there exists some consistent cut where $C^C(\phi)$ holds. Since $C^C(\phi) \Rightarrow \wedge_i K_i P_i C^C(\phi)$, there must exist states $s_1, \ldots, s_N$ such that s_i is the state in which i learns $P_i C^C(\phi)$. We will show that cut $c = \langle s_1, \ldots, s_N \rangle$ must be consistent.

Suppose that c is inconsistent. Then there must be a message m, say from process j to process k, such that *receive*(m) is included in state s_k but *send*(m) is not included in state s_j. Any *consistent* cut c' where $(a, c')[j] = s_j$ cannot include the reception of m, since s_j does not include *send*(m). Furthermore, by the definition of s_k, the reception of m occurs before k learns $P_k C^C(\phi)$. Therefore in any consistent cut c' where $(a, c')[j] = s_j$, k does not know $P_k C^C(\phi)$. We next show that this is impossible.

By the definition of s_j, j knows $P_j C^C(\phi)$ in s_j, i.e. in any consistent cut c_1 where $(a, c_1)[j] = s_j$, $(a, c_1) \models P_j C^C(\phi)$. By the definition of P, this means that there is some consistent cut c_2, $(a, c_2)[j] = (a, c_1)[j] = s_j$, for which $(a, c_2) \models C^C(\phi)$. The definition of C^C implies furthermore that $(a, c_2) \models \wedge_{i \in Proc} K_i P_i C^C(\phi)$ which in turn implies $(a, c_2) \models K_k P_k C^C(\phi)$. This contradicts the statement above that, in any consistent cut c' where $(a, c')[j] = s_j$ (including (a, c_2)), k does not know $P_k C^C(\phi)$. Hence the supposition that c is inconsistent must be false, making the theorem true. ∎

This theorem can be trivially extended to address cases in which $C^C(\phi)$ is attained periodically. Whenever $C^C(\phi)$ does not hold on a particular cut but does hold on some extending cut, between those cuts all processes i first reach a state in which $P_i C^C(\phi)$ is known along a single consistent cut.

The previous theorem illustrates an important difference between $C(\phi)$ and $C^C(\phi)$. In asynchronous systems, simultaneous action of any kind is impossible. Action coordinated to occur along a consistent cut is, however, easily achievable. We proceed by first presenting our Attainment Theorem, which gives a general criterion by which concurrent common knowledge may be attained in distributed protocols. Following the Attainment Theorem we give two protocols and prove that they meet the criterion of the theorem.

In order to achieve $C^C(\phi)$, it will be sufficient that a system have a set $\mathcal{S}$ of cuts, at least one per run, with the following property: when the local state of any process is in a cut of $\mathcal{S}$ in some run, then the same local state of that process is at some cut of $\mathcal{S}$ in every run in which it occurs. In other words, the process *knows* that it is at one of the cuts. We describe this more formally by defining *locally-distinguishable cut sets* below.

Definition 13. A *locally-distinguishable cut set* $\mathcal{S}$ of a system A is a set of cuts $\mathcal{S}$ such that

$$\forall a \in A \ : \ \exists (a, c) \in \mathcal{S}$$

and

$$\forall i \in \mathit{Proc} \ : \ \forall (a, c) \in \mathcal{S} \ :$$
$$\forall (a', c') \sim_i (a, c) \ : \ \exists (a', d) \in \mathcal{S} \ : \ (a', d) \sim_i (a, c).$$

In the definition above, suppose that we let $\mathit{in}\mathcal{S}$ stand for a formula such that $(a, c) \models \mathit{in}\mathcal{S}$ iff $(a, c) \in \mathcal{S}$. Given the formula $\mathit{in}\mathcal{S}$, the second condition for a locally-distinguishable cut set can be rewritten as simply

$$\mathit{in}\mathcal{S} \Rightarrow E^C(\mathit{in}\mathcal{S}).$$

We now show the primary result of this section: any system that guarantees that there is a locally-distinguishable cut set where a fact ϕ holds attains concurrent common knowledge of ϕ. We will later give two protocols to guarantee that a system does so, given particular assumptions on the fact ϕ.

Theorem 4. (Attainment Theorem) If a system A has a locally-distinguishable cut set $\mathcal{S}$ such that

$$\forall (a,c) \in \mathcal{S} : (a,c) \models \phi$$

then

$$\forall (a,c) \in \mathcal{S} : (a,c) \models C^C(\phi),$$

i.e. the system attains concurrent common knowledge of ϕ.

Proof: Let $in\mathcal{S}$ stand for a formula such that $(a,c) \models in\mathcal{S}$ iff $(a,c) \in \mathcal{S}$, as above. By the definition of a locally-distinguishable cut set, $in\mathcal{S} \Rightarrow E^C(in\mathcal{S})$. By the conditions of the theorem, $in\mathcal{S} \Rightarrow \phi$. Note that, since $in\mathcal{S} \Rightarrow \phi$, $\phi \wedge in\mathcal{S}$ is equivalent to $in\mathcal{S}$. Thus we have

$$\phi \wedge in\mathcal{S} \Rightarrow E^C(\phi \wedge in\mathcal{S}).$$

Then, by the induction axiom,

$$\phi \wedge in\mathcal{S} \Rightarrow C^C(\phi \wedge in\mathcal{S}).$$

It follows from the semantic definitions and the validity of $\phi \wedge in\mathcal{S} \Rightarrow \phi$ that $C^C(\phi \wedge in\mathcal{S}) \Rightarrow C^C(\phi)$. Thus we have that $in\mathcal{S} \Rightarrow C^C(\phi)$, or equivalently $\forall (a,c) \in \mathcal{S} : (a,c) \models C^C(\phi)$. ∎

We now proceed to a discussion of attaining C^C of a fact using specific protocols. A *protocol* is a partial specification on the set of runs of a system. It includes a set of actions with conditions on those actions. These may be conditions on the entire run, such as "process i executes α at some point in the run," or conditions on the state preceding the action, such as "process i sends m immediately after receiving m'." We say that a system *implements a protocol* if all runs of the system satisfy the specification. Note that one system may implement multiple protocols.

Before giving our protocols, we must first discuss for which facts ϕ attaining $C^C(\phi)$ is possible. $C^C(\phi)$ cannot be guaranteed to be attained by a protocol implementation if ϕ is false or may be falsified during execution of the protocol. We say that ϕ is *locally controllable by* i if, whenever i knows ϕ in any state, i can prevent falsifying ϕ for any finite number of events. Note that a stable fact—one that, once true, remains true forever—is always locally controllable by any process. Unstable facts that are local to a process i are also typically locally controllable by i. An example of this is $x_i = 1$, where x_i is a local variable of i. Any fact that becomes known to some process and is locally controllable by that process can become concurrent common knowledge of all processes.

By our definitions, in order for a system to implement the following protocols the fact ϕ must become known at some point in all runs. Alternatively, we could weaken our requirements so that $C^C(\phi)$ is only attained if ϕ is ever known, and only runs in which ϕ becomes known must contain an element of the locally-distinguishable cut set. For simplicity we use the current scheme.

The two protocols that follow differ in three primary ways: their message complexities, the degree to which they suspend activity of the underlying system, and the requirement of FIFO channels. Protocol 1 causes less suspension of system events but requires FIFO channels, whereas Protocol 2 requires fewer protocol messages and does not require FIFO behavior. We will discuss these issues further after presenting the protocols and proving their correctness.

In the presentation of each protocol, a *cut state* refers to the local state of a particular process that is included in the protocol cut. Local distinguishability is guaranteed because each cut state occurs immediately upon the completion of specific actions by the process.

Our first protocol is similar to the checkpointing protocol of Chandy and Lamport [3] and to echo algorithms of Chang [5]. It causes messages to be sent along every channel in the system. Intuitively, it creates a consistent cut because—since channels are FIFO—any message sent after execution of the protocol must be received after any messages the protocol sent along the same channel. Below, *CCK* identifies messages of the protocol.

Protocol 1. Attainment of $C^C(\phi)$.

- The initiator I, at some point in its local history where I knows ϕ, sends $\langle I, j, (\phi, CCK)\rangle$ to all neighbors j and then immediately reaches its cut state. Between sending the first message and reaching the cut state, I receives no messages and prevents falsifying ϕ.

- All other processes, i, upon first receiving a message of the form $\langle j, i, (\phi, CCK)\rangle$, sends $\langle i, k, (\phi, CCK)\rangle$ to all neighbors $k \neq j$ and then

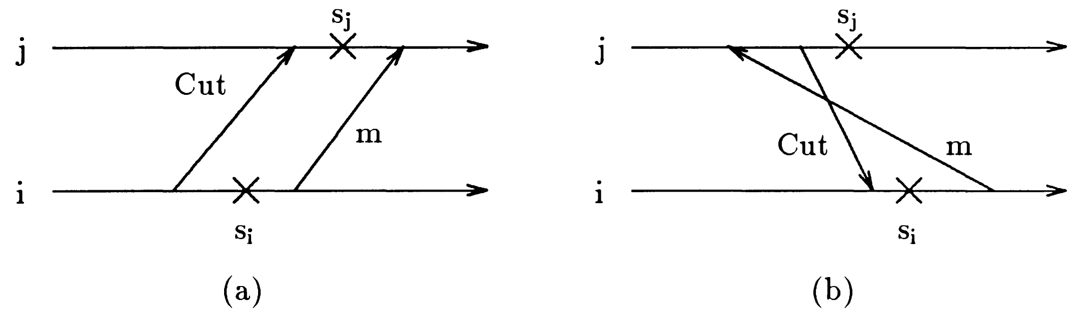

Figure 3: Proof of Theorem 5: (a) case 1, (b) case 2.

immediately reaches its cut state. Between sending the first message and reaching the cut state, i receives no messages.

Theorem 5. Let A be a system with reliable, FIFO channels in which ϕ is locally controllable by I. If A implements Protocol 1, then A attains $C^C(\phi)$.

Proof: For each process i, let s_i be the cut state indicated in the specification of the protocol. We show that states $s_1, s_2, \ldots, s_N$ form a consistent cut by contradiction. Suppose to the contrary that there is a message m sent after s_i but received before s_j. Note that m must be a system message, since cut states are not reached until all protocol messages have been sent. There are two cases to consider: (1) i sends a protocol message to j, or (2) i does not send a protocol message to j, which implies that i received its first protocol message from j.

In case (1), i must have sent m to j after s_i, by assumption, and consequently after sending $\langle i, j, (\phi, CCK)\rangle$ to j. Since channels are FIFO, the protocol message must reach j before message m. Unless j has already reached its cut state when receiving $\langle i, j, (\phi, CCK)\rangle$, it sends out its protocol messages and reaches its cut state before receiving any further messages. In either case, j reaches s_j before receiving m and the assumption that m was received before s_j is false. (See Figure 3(a).)

In case (2), the protocol message from j arrives before s_i and consequently before the sending of m. By assumption m is received before s_j.

Since j must send all protocol messages before reaching its cut state with no intervening receives, m must be received by j before sending the protocol message to i. (See Figure 3(b).) However, this implies causal circularity, since

$$\begin{aligned} receive(m) \rightarrow send(j, i, (\phi, CCK)) \rightarrow receive(j, i, (\phi, CCK)) \\ \rightarrow send(m) \rightarrow receive(m). \end{aligned}$$

Again the assumption that m was received before s_j must be false.

Let $\mathcal{S}$ be the set of possible consistent cuts characterized as above. Every run of a system implementing the protocol contains one of these cuts. Since the state of each process contained in the cut always immediately follows the sending of the protocol messages (and is therefore distinguishable), $\mathcal{S}$ is a locally-distinguishable cut set. Since ϕ holds initially and on any cut up through I's completion of the protocol, for any (a, c) in $\mathcal{S}$ $(a, c) \models \phi$. The theorem then follows from Theorem 4. ∎

In Protocol 2, three sets of messages *Prepare, Cut*, and *Resume* are sent respectively from the initiator to all processes, back to the initiator, and back to processes. We assume that messages are forwarded as necessary on paths of length d or less, where d is the diameter of the network (recall that the network is not completely connected, so there may not be channels between the initiator and some processes). The sending of non-protocol messages is suppressed between non-initiators sending *Cut* and receiving *Resume*.

Protocol 2. Attainment of $C^C(\phi)$.

1. The initiator I, at some point in its local history when I knows ϕ, sends $\langle I, j, (\phi, Prepare)\rangle$ to each process $j \neq I$.

2. Each process $j \neq I$, upon receiving $\langle I, j, (\phi, Prepare)\rangle$, begins suppression of system send events, sends $\langle j, I, (\phi, Cut)\rangle$ to the initiator then reaches its cut state.

3. The initiator I, after receiving $\langle j, I, (\phi, Cut)\rangle$ from all processes $j \neq I$, immediately reaches its cut state and then sends $\langle I, j, (\phi, Resume)\rangle$ to all processes $j \neq I$.

4. Each process $j \neq I$, upon receiving $\langle I, j, (\phi, Resume)\rangle$, resumes sending of system messages.

- The initiator I prevents falsifying ϕ from the beginning of step (1) until the end of step (3).

Theorem 6. Let A be a system with reliable channels in which ϕ is locally controllable by I. If A implements Protocol 2, then A attains $C^C(\phi)$.

Proof: Again let state s_i be the cut state of process i, as indicated in the protocol. We show by contradiction that this set of states forms a consistent cut. Suppose message m is sent after one of these states but received before another. There are three cases to consider: (1) m is from I, to some $j \neq I$, (2) m is from some $j \neq I$, to I, and (3) m is from some $i \neq I$ to some $j \neq I$.

In case (1), illustrated in Figure 4(a), let $m_1, ...m_k$ be the sequence of Cut messages from j to the initiator. In the figure, "$\times$" denotes the cut states of s_j and s_I which immediately follow the sending of m_1 and the initiator's last reception of a Cut message from its children (not necessarily immediately after the reception of m_k), respectively. Since the inconsistent message m is received before s_j, it is received before $send(m_1)$, which immediately precedes s_j. Clearly $send(m_1)$ happens-before $receive(m_k)$. But $receive(m_k)$ is received by the initiator before s_I and hence before sending m. Thus the inconsistency of m causes circular causality, producing an invalid run.

In case (2), illustrated in Figure 4(b), the sending of an inconsistent message m by j cannot occur until system sends are enabled, which happens after $Resume$ is received. Let $m_1, ...m_k$ be the sequence of $Resume$ messages from the initiator to j, so $receive(m_k)$ happens-before $send(m)$. If m is received before s_I then it is received before I sends m_1, and again an invalid circularity results.

Case (3), in which the inconsistent message m is between two non-initiators, is essentially a concatenation of case (1) and case (2). After the reception of m, j must begin a sequence of Cut messages to the initiator who then begins a sequence of $Resume$ messages to i. Message m cannot be sent by i until after the resume, with the same result. Therefore the states indicated form a consistent cut. The remainder of the proof confirms a locally-distinguishable cut set where ϕ holds exactly as in the proof of Theorem 5. ∎

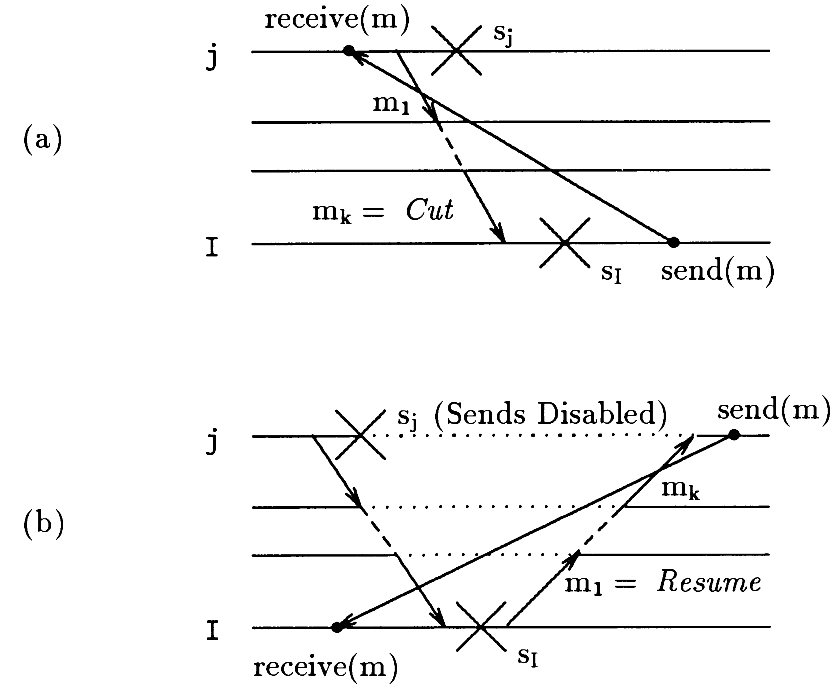

Figure 4: Proof of Theorem 6: (a) case 1, (b) case 2.

Protocol 2 does not require FIFO channels and uses only $3nd$ messages, where d is the diameter of the network. If all channels are bi-directional, this can be further optimized using a spanning tree of the network, to require only $3(n-1)$ messages [21]. In contrast, Protocol 1 does require FIFO channels and uses up to two messages per pair of neighboring processes, or $O(n^2)$ messages. However, Protocol 2 suspends system send events between its phases; communication from the initiator is required to resume send activity, which in turn occurs only after the initiator receives communication from every process indirectly through its children. Protocol 1 only suspends activity while a process is sending protocol messages to its neighbors. Hence Protocol 1 interferes less with the underlying system. The trade-off between these two protocols depends on the degree that the system may be degraded by the suspension of activity. This suspension is termed *inhibition* and is studied extensively in [21,6,7].

We have shown in this section that concurrent common knowledge is attainable in asynchronous systems by giving two simple and efficient protocols that do so. This makes it a potentially useful form of knowledge, as it describes states that can and do arise in such systems. Given a problem that can be formulated in terms of C^C, a solution immediately follows from these results.

5 Applications

The logic that we have presented, along with the semantics for concurrent knowledge and concurrent common knowledge, can have the following roles in the development and analysis of distributed algorithms: (1) simplification of solutions and proofs for problems that can be formulated in terms of concurrent knowledge or concurrent common knowledge, (2) characterization of implicit agreement present in certain algorithms, and (3) a tool for reasoning about asynchronous distributed algorithms, particularly with respect to causality.

This section contains several examples in which these goals are realized. We prove necessary and sufficient conditions for concurrent actions to take place in distributed systems. We prove a sufficient condition for broadcasts from a common initiator to arrive in their original order at all locations in the network without using sequence numbers, and apply this to updating replicated data. We show that concurrent common knowledge characterizes the knowledge between two processes attained by a single message transfer along reliable FIFO channels. Finally, we give a novel analysis of the Chandy-Lamport checkpointing algorithm with regard to both process states and channel states. Importantly, the expressiveness of our logic has led to short, straightforward proofs for these applications. We assume reliable channels throughout this section.

5.1 Concurrent Actions

In the theorem that follows, we use our logic to show a precise necessary and sufficient condition on the performance of *concurrent actions* in distributed systems. Concurrent actions are sets of events which are to be performed concurrently—immediately following a single consistent cut of the system—or not at all. The relationship between concurrent common knowledge and concurrent actions is analogous to that between common knowledge and *simultaneous actions* [16] in synchronous systems.

Definition 14. A vector of actions $\overline{\alpha} = \langle \alpha_1, \alpha_2, ..., \alpha_N \rangle$ is a *concurrent action* of system A iff the following holds. If any element α_i occurs in history $a[i]$ of A, following state s_i, then there is a consistent cut c of a where $(a, c)[i] = s[i]$ and for every j, α_j follows state $(a, c)[j]$.

For example, suppose that each of a set of processes has a local clock, and those local clocks are to be reset concurrently. Then

$$\{ \alpha_i \mid \alpha_i \equiv \text{“reset local clock } i \text{ ”}\}$$

is the corresponding vector of actions. We use the operator P_i to give a precise necessary and sufficient condition on the concurrent performance of actions in the next theorem.

In the theorem, if ϕ_i is a *precondition* of an action then i will execute the action immediately following $(a, c)[i]$ iff $(a, c) \models \phi_i$. Reasonable preconditions are local, i.e. $\phi_i \Rightarrow K_i\phi_i$. This follows automatically from the positive introspection axiom if ϕ_i is of the form $K_i\psi$ for any ψ. We assume locality of preconditions.

Theorem 7. Let $\overline{\alpha} = \langle \alpha_1, \alpha_2, \ldots, \alpha_N \rangle$, for each i let ϕ_i be the precondition of α_i, and let $\overline{\phi} \equiv \bigwedge_{i \in Proc} \phi_i$. Then $\overline{\alpha}$ is a concurrent action in A iff

$$\bigwedge_{i \in Proc} (\phi_i \Rightarrow K_i P_i \overline{\phi})$$

is valid in A.

Proof: First we show that if the formula is not valid then $\overline{\alpha}$ is not a concurrent action. If the formula is not valid then, for some i and some (a, c), $(a, c) \models \phi_i$ but $\phi_i \not\models K_i P_i \overline{\phi}$. Then, by the definitions of K_i and P_i, for some cut (a', c') that is indistinguishable from (a, c) there is not a cut (a', c'') such that $(a', c'') \sim_i (a', c')$ and where ϕ_j holds for all other processes j. By locality $(a', c') \models \phi_i$, so i executes α_i in a'. However, all other processes

j do not execute α_j concurrently in a'. Thus $\overline{\alpha}$ is not a concurrent action in A.

Next we show that if the formula is valid then $\overline{\alpha}$ is a concurrent action. Suppose that, for all i and all (a,c), $\phi_i \Rightarrow K_i P_i \overline{\phi}$ and that process i executes α_i following state $(a,c)[i]$. By the definitions of K_i and P_i, there must be a cut (a,c') such that $(a,c')[i] = (a,c)[i]$ and where $\overline{\phi}$ holds. By the definition of precondition all processes j perform action α_j following $(a,c')[j]$. Therefore $\phi_i \Rightarrow K_i P_i \overline{\phi}$ guarantees concurrent performance. ∎

This result is extended by the corollaries below; they give necessary and sufficient conditions on concurrent performance based on concurrent common knowledge.

Corollary 2. Given the conditions of Theorem 7, if $\overline{\alpha}$ is a concurrent action then $\overline{\phi} \Rightarrow C^C \overline{\phi}$ is valid in A.

Proof: From Theorem 7, we have that $\overline{\alpha}$ is a concurrent action implies that $\bigwedge_{i \in Proc}(\phi_i \Rightarrow K_i P_i \overline{\phi})$ is valid. This is turn implies that $\bigwedge_{i \in Proc} \phi_i \Rightarrow \bigwedge_{i \in Proc} K_i P_i \overline{\phi}$ is valid, which is equivalent to $\overline{\phi} \Rightarrow E^C \overline{\phi}$. By the induction axiom, we have that $\overline{\phi} \Rightarrow C^C \overline{\phi}$ is valid. ∎

Corollary 3. Given the conditions of Theorem 7, if $\bigwedge_{i \in Proc}(\phi_i \Rightarrow C^C \overline{\phi})$ is valid in A, then $\overline{\alpha}$ is a concurrent action.

Proof: By the definition of C^C, $C^C(\overline{\phi}) \Rightarrow E^C(\overline{\phi}) \Rightarrow \bigwedge_{i \in Proc} K_i P_i(\overline{\phi})$. Thus if $\bigwedge_{i \in Proc}(\phi_i \Rightarrow C^C \overline{\phi})$ is valid in A then $\bigwedge_{i \in Proc}(\phi \Rightarrow K_i P_i(\overline{\phi})$ is valid in A. By Theorem 7, this implies that $\overline{\alpha}$ is a concurrent action. ∎

5.2 Broadcast Ordering

In this section we consider the problem of one process *broadcasting* a sequence of facts $\phi_1, \phi_2, \ldots$ to all processes such they arrive everywhere in the order that they are sent.

Definition 15. In an asynchronous run, a *broadcast of ϕ by process i* is a set of $N-1$ message chains, possibly having some messages in multiple chains, such that (1) each message chain begins with a message from i, (2) all messages contain ϕ in the message body, and (3) each process $j \neq i$ is the recipient of the last message of one of the message chains. Additionally, we say that process i *initiates* the broadcast when it sends the first message in its history that is the first message of one of the message chains.

Note that both Protocol 1 and Protocol 2 are broadcasts.

In general, broadcasting a series of facts one at a time—sending the first messages of all chains of one broadcast before sending any of the next—does not guarantee that they arrive in the same order everywhere, even in FIFO systems, because messages may take different routes of differing transmission speeds. We give a theorem and corollary relating sufficient conditions so that (1) facts are guaranteed to arrive in the correct order everywhere, and (2) all processes know that they arrive in the correct order everywhere. Let $\rho(\phi)$ denote "fact ϕ has been received by all processes."

Theorem 8. If i knows $P_i(\rho(\phi_k))$ before initiating a broadcast of ϕ_{k+1}, then ϕ_k is guaranteed to arrive before message ϕ_{k+1} at all processes.

Proof: Consider any run a in which i knows $P_i(\rho(\phi_k))$ before initiating a broadcast of ϕ_{k+1}. By the definition of P_i, there must be a consistent cut (a, c') where $\rho(\phi_k)$ holds; furthermore, $(a, c')[i]$ precedes the beginning of all of the broadcast message chains for ϕ_{k+1}. If ϕ_{k+1} arrived at some process j before ϕ_k, hence before $(a, c')[j]$, then the message chain from i to j must begin after $(a, c')[i]$ and end before $(a, c')[j]$, making cut c' inconsistent. ∎

This is a good example of a simple situation where $K_i P_i \psi$ is sufficient to perform an action rather than the stronger traditional knowledge $K_i \psi$. Since it is not necessary to know that all messages have been received, now, in order to know $P_i \rho(\phi_k)$, this alleviates the latency of waiting for acknowledgements.

A result of this theorem is that Protocol 1 and Protocol 2 can be used to insure that multiple messages sent to all processes are ordered properly. Just as $C^C(\alpha)$ is attained with the protocols invoked with locally-controllable parameter α, the fact $\rho(\alpha)$ becomes concurrent common knowledge also. This follows from the Attainment Theorem (Theorem 4) because $\rho(\alpha)$ always holds on the locally-distinguishable cuts of the protocols. Thus the protocols can achieve $C^C \rho(\phi_k)$, which in turn implies $K_i P_i \rho(\phi_k)$.

The theorem above does not imply that the non-initiating processes know that they have received messages in the proper order. This is clearly guaranteed, however, if it is common knowledge that $K_iP_i\rho(\phi_k)$ is a precondition to i initiating the broadcast of ϕ_{k+1}.

Corollary 4. If it is common knowledge that, for all facts ϕ_k, $K_iP_i\rho(\phi_k)$ is a precondition to i initiating the broadcast of ϕ_{k+1}, then all processes know that they receive the facts in the order that they are sent by i.

5.3 Updates to Replicated Data

In this example, we use the results on broadcast message ordering to develop a protocol for maintaining consistency of updates to replicated data items. Consider a replicated data item $\overline{x}$, where x_i indicates i's copy of the data. Suppose that process I must perform a sequence of updates to $\overline{x}$ such that these updates occur in the same order at all copies as they do at I.

> **Operation Ordering Problem:** When a process I performs a series of operations, $Op_1, Op_2, \ldots$ that modify its copy x_I of replicated variable $\overline{x}$, ensure that the operations are carried out on each additional copy x_i so that $Op_1(x_i) \rightarrow Op_2(x_i) \rightarrow \ldots$.

Let ψ_k denote "operation Op_k has been performed on x_I." The sequence numbers are a notational convenience and are not necessary in the messages sent by processes. In the previous section we observed that Protocol 1 and Protocol 2 can insure that a series of broadcasts sent by I arrive at all processes in the order that they are sent. Therefore, a method which uses one of those protocols for broadcasting operations and in which each process performs operation Op_k immediately upon reaching its cut state—i.e. upon learning $C^C\rho(\psi_k)$—solves the Operation Ordering Problem. Using Protocol 1 as a basis, we obtain the following protocol.

Protocol 3. Update of Replicated Data $\overline{x}$.

- The initiator I, after performing operation $Op_k(x_I)$ and before performing operation $Op_{k+1}(x_I)$, sends $\langle I, j, \psi_k\rangle$ to all neighbors j. While sending the protocol messages, I receives no messages.

- All other processes, i, upon first receiving a message of the form $\langle j, i, \psi_k\rangle$, sends $\langle i, k, \psi_k\rangle$ to all neighbors $k \neq j$, and then performs $Op_k(x_i)$. Between sending the first message and performing $Op_k(x_i)$, i receives no messages.

In contrast, a typical method for ordering operations would be to give each a unique sequence number, then buffer operations until all of those with lower numbers have arrived and been executed. This requires unbounded messages—to accommodate sequence numbers—and buffering of information, neither of which is necessary in Protocol 3.

Note, however, that even concurrent common knowledge of the operations does not guarantee that concurrent updates from multiple initiators are ordered the same everywhere; nor does the sequence number method above. If process i makes ϕ_1 concurrent common knowledge and process j makes ϕ_2 concurrent common knowledge, some other processes may perform Op_1 first whereas others perform Op_2 first. Timestamped common knowledge [11,17], C^T, can guarantee that concurrent broadcasts from different initiators are ordered the same everywhere. As we will discuss in Section 6.2.4, if local clocks are *logical clocks* and if the timestamp of interest is known to be reached by all processes, timestamped common knowledge implies concurrent common knowledge when the appropriate local times are reached. However, protocols to achieve C^T [17] require two rounds of messages during which underlying communication is suspended, as in Protocol 2. There do not appear to be lower-latency protocols such as Protocol 1 for C^T. Also, C^C does not require the use of local clocks.

Broadcast protocols which achieve each of these two forms of knowledge, C^C and C^T, may be combined to handle replicated data updates efficiently. A C^T protocol can be used to obtain locks for concurrency control of transactions. Once locks are obtained, a C^C protocol which is faster and causes less latency can be used to issue operations within each transaction. A similar scheme is used in the ISIS project [1], using two broadcast primitives, CBCAST (*causal broadcast*) and ABCAST (*atomic broadcast*). This example illustrates situations in practical systems where two different forms of knowledge are both appropriate characterizations of agreement.

5.4 Single Message Transfer

In this section we show that concurrent common knowledge characterizes the knowledge between two processes attained by a single message transfer along a reliable FIFO channel. A formula subscripted with a set of process identifiers, such as $\{i, j\}$, refers to the subsystem containing only that set of processes.

Theorem 9. A single message m sent along a reliable FIFO channel from i to j achieves

$$C^C_{\{i,j\}}(\text{“}m\text{ has been received”}).$$

Proof: The states immediately following the sending and receiving of m form a locally-distinguishable cut in the $\{i,j\}$ subsystem. Furthermore, "m has been received" holds on this cut. By the Attainment Theorem (Theorem 4), concurrent common knowledge of the fact holds as well. ∎

Theorem 9 enables us to better explain the role of certain messages in the checkpointing example which follows.

5.5 Checkpointing

Our example is an analysis of the agreement implicit in a variant of the well-known checkpointing protocol by Chandy and Lamport [3]. The protocol is used to record global states—including that of channels—for the purpose of detecting global system properties or for rollback recovery. It is designed for asynchronous systems with reliable FIFO channels as in our model.

The protocol works by using rules for sending and receiving special types of messages called *markers*. Initialization occurs by one process following the marker sending rule.

Marker sending rule for a process i: Before sending or receiving any other messages, i sends one marker to *each* neighbor j, then records its state.

Marker receiving rule for a process j: Upon receiving a marker from i, if j has not recorded its state, then j follows the marker sending rule, then records the state of channel (i,j) as the empty sequence. Otherwise, j records the state of channel (i,j) as the sequence of messages received from i after j's state was recorded and before j received the marker from i.

A perhaps subtle but important difference between this protocol and Protocol 1 is that a marker is sent along every channel, including a marker back along the channel where a process received its first marker. These extra messages give information about the state of channels during the checkpoint. The next theorem summarizes the agreement attained in the protocol regarding both the process states and the channel states.

The theorem statement makes use of some additional notation. Given any consistent global state, $current_i$ will denote i's local state. $Saved_i$ denotes the set of states checkpointed by process i. $Learn_i(\psi)$ indicates the local state in which i learns ψ. Finally, we extend $\rightarrow$ to include states in the obvious way.

Theorem 10. In any run of the Chandy-Lamport protocol:

1. $C^C(\phi_{states})$ is attained, where

$$\phi_{states} \equiv \forall i : current_i \in Saved_i$$

2. for each channel (i, j), $C^C_{\{i,j\}}(\phi_{channels})$ is attained, where

$$\phi_{channels} \equiv \forall m :$$
$$send(m) \rightarrow Learn_i P_i C^C(\phi_{states}) \Rightarrow receive(m) \rightarrow current_j$$

   where m is a message from i to j.

Informally, the first statement above says that there is a consistent cut of the system in which all processes have concurrent common knowledge that all processes are taking a checkpoint. We will not give a proof as this is a simple variation of Protocol 1.

The second statement says that—within each (i, j) subsystem—it becomes concurrent common knowledge that all messages sent prior to i recording its state have been received by j. Hence when this is attained, the messages in the (i, j) channel during the checkpoint are exactly those that have been received since j recorded its state. This follows from the fact that all messages from i to j are in one of three states at the checkpoint:

1. Received before the checkpoint. These are part of j's local state at the time of the checkpoint.

2. In the channel during the checkpoint. All messages not in (1) received prior to $Learn_j P_j(\phi_{channels})$ must be in this set.

3. Sent after the checkpoint. These are not received by j until after $Learn_j P_j(\phi_{channels})$.

Proof of Theorem 10, part 2: From Theorem 9, the reception of the marker by j becomes concurrent common knowledge between i and j in the subsystem cut formed by the states immediately following the sending and receiving of the marker. Since no messages are sent between the sending of the marker and $Learn_i P_i(C^C(\phi_{states}))$, and channels are FIFO, all non-marker messages sent prior to $Learn_i P_i(C^C(\phi_{states}))$ must be received before the marker. Therefore, at the subsystem cut it becomes concurrent common knowledge of i and j that all messages sent prior to $Learn_i P_i(C^C(\phi_{states}))$ have been received. ∎

Any problem that requires only the detection of some property of a consistent global state can be solved using the Chandy-Lamport protocol. Some examples of this type of problem are termination detection [9], deadlock detection [2], and rollback recovery [12]. Hence concurrent common knowledge can be used for a solution and formal analysis of such problems.

6 Comparisons

In this section we compare our semantics and the definition of concurrent common knowledge to other standard knowledge-theoretic semantics and agreement definitions. First, we give a translation from our asynchronous-runs semantics to the timed-runs semantics of Halpern and Moses [11]. Then we compare C^C to common knowledge and to other weakenings of common knowledge, namely epsilon common knowledge C^ϵ, eventual common knowledge $C^\diamond$, and timestamped common knowledge C^T.

6.1 Translation to Timed Runs Semantics

Timed runs have been used by Halpern and Moses to provide formal semantics for common knowledge and the other variants of common knowledge that they introduce. For our purposes, the asynchronous runs provide a better choice since time does not enter our system model. Nevertheless, it has become standard to use timed runs to model a variety of different systems; we will show that our logic can be given a timed-runs semantics as well. Also, this will be useful in the following section when we compare C^C to knowledge forms defined in the timed-runs model. We show that our semantics in terms of timed runs is essentially equivalent to the asynchronous runs semantics.

The translation proceeds by first defining an appropriate set of possible timed runs R and a primitive proposition function π_R, given an initial set of possible asynchronous runs A and a primitive proposition function π_A. Next, we define our new modal operators using timed-runs semantics. Finally, we state a theorem which formally relates our asynchronous-runs semantics to the timed-runs semantics, and prove it by structural induction on the formulas in the logic.

6.1.1 The Set of Possible Timed Runs

We first give a definition for timed runs using our notation that is consistent with that of [11].

Definition 16. A *timed run* r is a sequence of N-vectors of local states (or equivalently, event or action sequences) indexed by a possibly-infinite sequence of natural numbers, such that, for each processor i, $(r,t)[i]$ is a prefix of $(r,t+1)[i]$. We let $r[i]$ denote the sequence $(r,1)[i],(r,2)[i],\ldots$.

We associate an asynchronous run a with $timing(a)$, the set of all timed runs having the same events and causal structure. To preserve causal structure, we will require that the real-time values associated with events be consistent with the happens-before relation. Let $time(r,e)$ be the time value t of the latest global state (r,t) preceding the occurrence of event e; *time* is a partial function from runs and events to natural numbers.

Definition 17. Given a run r and event e in $r[i]$, $time(r,e)$ is the natural number t such that $e \notin (r,t)[i] \wedge e \in (r,t+1)[i]$.

Definition 18. Given an asynchronous run a, $timing(a)$ is the set of all timed runs r such that:

1. A state is in r iff it is in a, i.e.

$$\forall t : \forall i : \exists c : (r,t)[i] = (a,c)[i] \text{ and } \forall c : \forall i : \exists t : (a,c)[i] = (r,t)[i].$$

2. Causal structure of events is preserved, i.e. for all events e_1 and e_2, if $e_1 \rightarrow e_2$ then $time(e_1) < time(e_2)$.

Note that for any one asynchronous run there are, in general, infinitely many corresponding timed runs. However, there is exactly one asynchronous run corresponding to each timed run, because a timed run has exactly one causal structure. Now we can define the set of timed runs to be the union of all asynchronous timings, so that $R \equiv \cup_{a \in A} timing(a)$. Throughout this section, unless otherwise noted, the ranges of quantification for asynchronous runs and timed runs are the sets A and R, respectively.

We next prove that every real-time global state has a corresponding consistent cut. Note that we can assert the equality of real-time global states and consistent cuts since they are both N-vectors of local states.

Theorem 11. Given a run r in $timing(a)$, $\forall t : \exists! c : (r,t) = (a,c)$.

Proof: The proof is by contradiction. By condition (1) in the definition of $timing(a)$, every local state of r is a local state of a. Therefore for the theorem to be false, for some t the local states of (r,t) must be inconsistent. Thus run r must contain events e_1 and e_2 such that $e_1 \rightarrow e_2$, e_2 is contained in (r,t), and e_1 is not contained in (r,t). Since e_1 is in r but not in (r,t), $time(r,e_1) \geq t$ and similarly $time(r,e_2) < t$. However, this contradicts condition (2) of the definition of $timing(a)$, which requires that if $e_1 \rightarrow e_2$ then $time(r,e_1) < time(r,e_2)$. Uniqueness follows because a set of local states uniquely specifies a cut in an asynchronous run. ∎

In view of the preceding theorem, we can define $Cut(r,t)$ to be the consistent cut corresponding to real-time global state (r,t).

Definition 19. $Cut(r,t)$ is the pair (a,c) such that r is in $timing(a)$ and $(r,t) = (a,c)$.

It is also true that for any (a,c) there is some (r,t) such that $Cut(r,t) = (a,c)$. This is a straightforward consequence of the definitions of consistent cuts and $timing(a)$; we leave the proof of the theorem to the reader.

Theorem 12. $\forall (a,c) : \exists (r,t) : Cut(r,t) = (a,c)$

We next define the primitive proposition function π_R from the primitive proposition function π_A in the obvious way.

Definition 20. $\pi_R((r,t),\phi) \equiv true$ iff $\pi_A(Cut(r,t),\phi) = true$.

6.1.2 Timed-Runs Semantics

The timed-runs semantics for the ordinary logical connectives, the primitive propositions, the knowledge modality K_i, and the greatest fixed-point operator are exactly as in Halpern and Moses [11]. In this subsection we only discuss the new modal operator, P_i. In order to avoid confusion, we use the symbols $\models_T$ and $\models_A$ to stand for timed-run semantics and asynchronous-runs semantics, respectively.

First we define the relation $\bowtie$ on timed runs, to represent timed runs that are timings of the same asynchronous run.

Definition 21. Given timed runs r and r', $r \bowtie r'$ iff there exists an asynchronous run a such that $r \in timing(a)$ and $r' \in timing(a)$.

Clearly a real-time global state (r', t') of run r' is also a consistent cut of run r if $r \bowtie r'$. This motivates the following definition for the meaning of P_i in a timed-runs semantics.

Definition 22.

$$(r,t) \models_T P_i(\phi) \Leftrightarrow \exists (r',t') : (r' \bowtie r) \wedge ((r',t') \sim_i (r,t)) : (r',t') \models \phi$$

The definition of E^C is derivable from the definitions of $\wedge$, K_i, and P_i and is as follows:

$$(r,t) \models_T E^C \phi \Leftrightarrow \forall (r',t') : (r',t') \sim_i (r,t) :$$

$$\exists (r'',t'') : (r'' \bowtie r') \wedge ((r'',t'') \sim_i (r',t')) : (r'',t'') \models_T \phi$$

Finally, $(r,t) \models_T C^C \phi$ is defined using the same greatest fixed-point interpretation as in [11] and as in our asynchronous runs semantics.

6.1.3 The Equivalence of the Two Semantics

Two preliminary facts are trivial consequences of our definitions.

Fact 1. Let $(a,c) = Cut(r,t)$ and $(a',c') = Cut(r',t')$. Then

(a) $(a,c) \sim_i (a',c')$ iff $(r,t) \sim_i (r',t')$, and
(b) $a = a'$ iff $r \bowtie r'$.

The following theorem relates the truth of formulas in our asynchronous-runs semantics to the truth of formulas in the timed-runs semantics.

Theorem 13. $(r,t) \models_T \phi \Leftrightarrow Cut(r,t) \models_A \phi$

Proof: By structural induction on formulas.

1. Primitive propositions:
   Follow immediately from definition of π_r.

2. $\neg, \wedge$: From the structural induction hypothesis,
   $(r,t) \models_T \phi \Leftrightarrow Cut(r,t) \models_A \phi$, it follows immediately that
   $(r,t) \models_T \neg\phi \Leftrightarrow Cut(r,t) \models_A \neg\phi$ and
   $(r,t) \models_T \phi \wedge \psi \Leftrightarrow Cut(r,t) \models_A \phi \wedge \psi$.

3. K_i: $(\Rightarrow)$ It is given that $(r,t) \models_T K_i\phi$, i.e. that

   $$\forall (r',t') : (r',t') \sim_i (r,t) : (r',t') \models_T \phi.$$

   By Theorem 12,

   $$\forall (a,c) : (a,c) \sim_i Cut(r,t) : \exists (r'',t'') : Cut(r'',t'') = (a,c).$$

   Furthermore, Fact 1(a) implies that $(r'',t'') \sim_i (r,t)$ and since $(r,t) \models_T K_i\phi$, $(r'',t'') \models_T \phi$. By the structural induction hypothesis we then have that $\forall (a,c) : (a,c) \sim_i Cut(r,t) : (a,c) \models_A \phi$, therefore $Cut(r,t) \models_A K_i\phi$ as desired.

   $(\Leftarrow)$ It is given that $Cut(r,t) \models_A K_i\phi$, i.e. that

   $$\forall (a,c) : (a,c) \sim_i Cut(r,t) : (a,c) \models_A \phi.$$

   From Fact 1(a),

   $$\forall (r',t') : (r',t') \sim_i (r,t) : Cut(r',t') \sim_i Cut(r,t).$$

   Since $Cut(r,t) \models_A K_i\phi$ it follows that

   $$\forall (r',t') : (r',t') \sim_i (r,t) : Cut(r',t') \models_A \phi.$$

   Then $\forall (r',t') : (r',t') \sim_i (r,t) : (r',t') \models_T \phi$ follows from the structural induction hypothesis, hence $(r,t) \models_T K_i\phi$.

4. P_i: ($\Rightarrow$) It is given that $(r,t) \models_T P_i\phi$, i.e.

$$\exists(r',t') : (r' \bowtie r) \wedge (r',t') \sim_i (r,t) : (r',t') \models_T \phi.$$

Let $(a,c) = Cut(r,t)$ and $(a',c') = Cut(r',t')$. Then by Fact 1, $a = a'$ and $(a,c) \sim_i (a',c')$. Therefore

$$\exists(a,c') : (a,c') \sim_i (a,c) : (a,c') = Cut(r',t') \wedge (r',t') \models_T \phi.$$

$(a,c') \models_A \phi$ by the induction hypothesis, so that $(a,c) \models_A P_i\phi$.

($\Leftarrow$) It is given that $Cut(r,t) \models_A P_i\phi$, so that

$$\exists(a,c') : (a,c') \sim_i Cut(r,t) : (a,c') \models_A \phi.$$

By Theorem 12 there exists (r',t') such that $Cut(r',t') = (a,c')$. By Fact 1, $(r',t') \sim_i (r,t)$ and $r' \bowtie r$. Then

$$\exists(r',t') : (r' \bowtie r) \wedge (r',t') \sim_i (r,t) : Cut(r',t') \models_A \phi,$$

giving the desired result by the induction hypothesis.

5. C^C: For C^C we return to the view that formulas are functions from sets of global states to sets of global states; again we use subscripts to distinguish the timed-runs semantics from the asynchronous-runs semantics. We also use the notation that, for any set of consistent cuts Z, Z^* is the set $\{(r,t) | Cut(r,t) \in Z\}$.

We need to show that (r,t) is contained in the greatest fixed point of $\mathcal{M}_T[\![E^C(\phi \wedge X)]\!]$ iff $Cut(r,t)$ is contained in the greatest fixed point of $\mathcal{M}_A[\![E^C(\phi \wedge X)]\!]$, i.e. set B is the latter greatest fixed point iff B^* is the former. Repeating the technique for K_i and P_i above, it can be shown that, for any Z,

$$(r,t) \in \mathcal{M}_T[\![E^C(\phi \wedge X)]\!](Z^*) \text{ iff } Cut(r,t) \in \mathcal{M}_A[\![E^C(\phi \wedge X)]\!](Z).$$

It then follows that B^* is a fixed point of $\mathcal{M}_T[\![E^C(\phi \wedge X)]\!]$ iff B is a fixed point of $\mathcal{M}_A[\![E^C(\phi \wedge X)]\!]$, i.e.

$$((r,t) \in B^* \Leftrightarrow (r,t) \in \mathcal{M}_T[\![E^C(\phi \wedge X)]\!](B^*))$$

iff

$$(Cut(r,t) \in B \Leftrightarrow Cut(r,t) \in \mathcal{M}_A[\![E^C(\phi \wedge X)]\!](B)).$$

It just remains to be shown that B is the *greatest* fixed point iff B^* is the greatest fixed point also. Suppose that B is but B^* is not. Then there is some set C^*, not a subset of B^*, such that

$$B^* \cup C^* = \mathcal{M}_T[\![E^C(\phi \wedge X)]\!](B^* \cup C^*).$$

However, since (r,t) is contained in $\mathcal{M}_T[\![E^C(\phi \wedge X)]\!](Z^*)$ iff $Cut(r,t)$ is contained in $\mathcal{M}_A[\![E^C(\phi \wedge X)]\!](Z)$, this would be imply that

$$B \cup C = \mathcal{M}_A[\![E^C(\phi \wedge X)]\!](B \cup C).$$

This contradicts the assumption that B is the greatest fixed point. An analogous argument holds in the reverse direction, thus concluding the proof. ∎

6.2 Other Knowledge Forms

There have been other common knowledge variants that are based on replacing simultaneity with weaker temporal notions [11]; namely, epsilon common knowledge, C^ϵ, eventual common knowledge, $C^\diamond$, and timestamped common knowledge, C^T. In this section we compare concurrent common knowledge to common knowledge and to each of these variants. As the names indicate, epsilon common knowledge corresponds to agreement within ϵ time units, for some ϵ, eventual common knowledge corresponds to agreement at some future global state of the system, and timestamped common knowledge corresponds to agreement at local states having the same local clock value. The strength of timestamped common knowledge, consequently, depends upon characteristics of the local clocks used.

In the discussions to follow, we will demonstrate systems in which one knowledge form is attained but another is not. Recall that we say a fact ϕ is *attained* in run a iff there is a cut c such that $(a,c) \models_A \phi$. Similarly, we say that a fact ϕ is attained in timed run r iff there is a time t such that $(r,t) \models_T \phi$. Recall that if $r \in timing(a)$ then r is one of the possible timed runs corresponding to asynchronous run a. When comparing C^C to a knowledge form, say C^X, defined in the timed-runs model, we will consider whether or not attaining $C^X\phi$ in a run r implies attaining $C^C\phi$ in the asynchronous run a such that $r \in timing(a)$, or vice versa.

In Section 6.2.1 we demonstrate that common knowledge is strictly stronger than C^C, by showing that $C \Rightarrow C^C$ and that there are systems where C^C is attained but C is not. In Section 6.2.2, we define C^ϵ and show that it is incomparable to C^C in a strict sense; namely, there are systems and facts ϕ for which $C^\epsilon\phi$ is attainable and $C^C\phi$ is not attainable, and vice versa.

In Section 6.2.3, we demonstrate that C^C and $C^\diamond$ are also incomparable. It should be intuitively clear that eventually agreeing on a fact does not guarantee causal consistency; the converse, however, may not be so clear. It turns out that eventual common knowledge guarantees that a fact ϕ is *known* at points in the run. Concurrent common knowledge, in contrast, only guarantees knowledge of $P_i\phi$ for each i, which is weaker. Consequently, C^C and $C^\diamond$ are indeed incomparable.

In Section 6.2.4, we demonstrate that, in general, timestamped common knowledge, C^T, is incomparable to concurrent common knowledge. We also demonstrate that, in the special case of local clocks being *logical clocks* [15], $C^T\phi$ implies $C^C\phi$ at a consistent cut where local time T is reached by all processes.

6.2.1 Common Knowledge

$C\phi$ implies all formulas of the form $E^k\phi$ for any natural number k, where E is defined as follows.

$$E\phi \equiv \bigwedge_{i \in \mathcal{I}} K_i\phi$$

More precisely, C is defined as the greatest fixed point of $E(\phi \wedge X)$, where X is the free variable in the greatest fixed-point operator. We show that this is strictly stronger than C^C below.

Theorem 14. Common knowledge is strictly stronger than concurrent common knowledge, i.e. $C\phi \Rightarrow C^C\phi$ but $C^C\phi \not\Rightarrow C\phi$. Furthermore, there are systems and facts ϕ for which $C^C\phi$ is attained but $C\phi$ is never attained.

Proof: To see that $C\phi \Rightarrow C^C\phi$, recall that $\phi \Rightarrow P_i\phi$. Consequently, $K_i\phi \Rightarrow K_iP_i\phi$ and furthermore, $E\phi \Rightarrow E^C\phi$. Since $C^C\phi$ is the greatest fixed point of $E^C(\phi \wedge X)$ and $C\phi$ is the greatest fixed point of $E(\phi \wedge X)$, the desired result follows.

From [11], common knowledge of any fact not initially common knowledge of a system is unattainable in any asynchronous system. We have demonstrated in the current work that concurrent common knowledge of certain facts is attainable in such systems. Thus there are systems and facts ϕ for which $C^C\phi$ is attained and $C\phi$ is not; it follows that $C^C\phi \not\Rightarrow C\phi$. ∎

6.2.2 Epsilon Common Knowledge

Epsilon common knowledge corresponds to agreement within ϵ time units, for some natural number ϵ. Its definition is consequently dependent upon the timed-runs model, since we cannot express "in ϵ time units" in the asynchronous runs model. The formula $\bigcirc^\epsilon\phi$ indicates that ϕ will hold in exactly ϵ time units, as follows.

$$(r,t) \models_T \bigcirc^\epsilon\phi \text{ iff } (r, t+\epsilon) \models_T \phi$$

Epsilon common knowledge is then the greatest fixed point of $\bigcirc^\epsilon E(\phi \wedge X)$, where X is the free variable in the greatest fixed-point operator. $C^\epsilon\phi$ can also be expressed as:

$$C^\epsilon\phi \equiv \forall k : (\bigcirc^\epsilon E)^k\phi$$

We show that C^C and C^ϵ are incomparable in a strict sense: that there are systems and facts ϕ for which $C^C\phi$ is attained and $C^\epsilon\phi$ is not, and vice versa.

Theorem 15. Epsilon common knowledge is incomparable to C^C, i.e. $C^C\phi \not\Rightarrow C^\epsilon\phi$ and $C^\epsilon\phi \not\Rightarrow C^C\phi$. Moreover, there are systems and facts ϕ for which $C^C\phi$ is attained but $C^\epsilon\phi$ is not, and vice versa.

Proof: First, we demonstrate a system $\mathcal{S}_1$ and fact ϕ_1 for which $C^C\phi_1$ is attained, but $C^\epsilon\phi_1$ is not. Let system $\mathcal{S}_1$ contain only two processes, i and j. In this system, the only communication occurs when, at some point, i sends a message m to j along a reliable channel with unbounded transmission time. Let a be any run of this system. Let c be the consistent cut of a consisting of i's local state immediately after sending m and j's local state immediately after receiving m. Clearly c is a cut, since no message is sent after it. It is distinguishable, because its occurrence in each local state is determined by specific events, namely the sending and receiving of m. Let ϕ_1 be the fact "message m has been received." Since ϕ_1 always holds on cut c, from Theorem 4 it follows that $(a,c) \models_A C^C\phi_1$. Therefore $C^C\phi_1$ is attained in any run of the system.

Now consider any r in $timing(a)$. For $C^\epsilon\phi_1$ to be attained, there must be a t such that $(r,t) \models_T C^\epsilon\phi_1$. This implies further that $(r,t+\epsilon) \models_T K_i\phi_1$. However, process i never *knows* ϕ_1, because every local state of i in r is part of a possible real-time state in which message m has not yet been received. Consequently, there is no t for which $(r,t) \models_T C^\epsilon\phi_1$. Therefore $\mathcal{S}_1$ is a system in which $C^C\phi_1$ is attained, but $C^\epsilon\phi_1$ is never attained.

Next, we demonstrate a system $\mathcal{S}_2$ and fact ϕ_2 for which $C^\epsilon\phi_2$ is attained but $C^C\phi_2$ is not. In system $\mathcal{S}_2$, there are three processes i, j, and k which communicate along reliable channels having transmission time bounded by ϵ. Process k can simultaneously send a message to both and i and j. At some point k does send a message m simultaneously to i and j. Also, i and j periodically send messages to each other at random times.

Let ϕ_2 be the fact "message m has been sent by k." $C^\epsilon\phi_2$ holds as soon as m is sent by k. However, $C^C\phi_2$ can never hold, as we shall demonstrate by contradiction. Suppose that $C^C\phi_2$ is attained in a run r of $\mathcal{S}_2$. By Theorem 3, i and j must learn $C^C\phi_2$ along a single consistent cut of r. Let $Learn(i)$ and $Learn(j)$ denote the states of i and j in that cut. $Learn(i)$ must be no earlier than immediately following i's reception of m, and similarly for j; otherwise, processor i (j) could not know $P_i\phi_2$ ($P_j\phi_2$). If $Learn(i)$ does immediately follow i's reception of m, it follows—since $Learn(j)$ is no earlier than j's reception of m and consequently could occur up to ϵ time units later than $Learn(i)$—that a message sent within ϵ time units of $Learn(i)$ could cause the cut to be inconsistent. The same argument is true if $Learn(i)$ occurs anywhere within the first ϵ time units after i receives m. Thus $Learn(i)$ must occur at least ϵ time units after i receives m. A symmetric argument holds for j; $Learn(j)$ must occur at least ϵ time units after j receives m. However, this again results in the situation where the minimum $Learn(i)$ and $Learn(j)$ are ϵ time units apart, as before. We can repeat the argument that $Learn(i)$ and $Learn(j)$ each must occur at least ϵ time units later, ad infinitum, with the result that $C^C\phi_2$ is never learned. ∎

6.2.3 Eventual Common Knowledge

Eventual common knowledge corresponds to agreement at some future global state of a run. We express it using standard definitions in the timed-runs model.

The eventuality operator, $\diamond$, is defined as follows.

$$(r,t) \models_T \diamond\phi \text{ iff } \exists\epsilon : \epsilon \geq 0 : (r, t+\epsilon) \models_T \phi$$

Then $E^{\diamond}$, "everyone eventually knows" is defined as expected.

$$E^{\diamond} \equiv \bigwedge_{i \in \mathcal{I}} \diamond K_i \phi$$

Note that, unless facts are stable, $E^{\diamond}\phi$ does not imply that $E\phi$ ever holds.

Analogously to other common knowledge variants, $C^{\diamond}\phi$ is defined as the greatest fixed point of $E^{\diamond}(\phi \wedge X)$. $C^{\diamond}\phi$ implies, but is strictly stronger than, $(E^{\diamond})^k\phi$ for all k:

$$C^{\diamond}\phi \Rightarrow \forall k : (E^{\diamond})^k\phi$$

We prove that this is not only incomparable to $C^C\phi$, but that there are systems and facts for which each is attained but the other is not.

Theorem 16. Eventual common knowledge is incomparable to concurrent common knowledge, i.e. $C^C\phi \not\Rightarrow C^{\diamond}\phi$ and $C^{\diamond}\phi \not\Rightarrow C^C\phi$. Moreover, there are systems and facts ϕ for which $C^C\phi$ is attained and $C^{\diamond}\phi$ is not, and vice versa.

Proof: First, we demonstrate a system and fact ϕ for which $C^C\phi$ is attained, but $C^{\diamond}\phi$ is not. Consider again system $\mathcal{S}_1$ from the proof of Theorem 15, containing only two processes i and j. Process i sends a single message m to j along a reliable channel with unbounded transmission time. Fact ϕ_1 is "message m has been received." As before, $C^C\phi_1$ is attained in any run of $\mathcal{S}_1$; however, i never *knows* ϕ_1. Since $C^{\diamond}\phi_1$ implies that eventually $K_i\phi_1$ holds, $C^{\diamond}\phi_1$ cannot be attained in any run of $\mathcal{S}_1$.

Next, we demonstrate a system and fact ϕ for which $C^{\diamond}\phi$ is attained, but $C^C\phi$ is not. Consider a system $\mathcal{S}_3$, again with only two processes, i and j. In this system, i sends some number of messages, at least one, to j along reliable, non-FIFO channels. We denote the messages $m_1, m_2, m_3 \ldots$

where message m_k is sent before message m_{k+1}. The sequence numbers are notational and not contained in the messages. Process j never sends a message to i. Let ϕ_3 be "i has sent at least one message to j." Consider any run r of $\mathcal{S}_3$. For any t after i sends the the first message to j, $(r,t) \models_T C^{\diamond}\phi$.

Let a be the asynchronous run such that $r \in \mathit{timing}(a)$. Suppose that $C^C\phi_3$ is attained in a. By Theorem 3, the states in which i and j learn $C^C\phi_3$ form a consistent cut of a. Suppose further that j learns $C^C\phi_3$ after receiving one message from j, and that i learns $C^C\phi_3$ after sending message m_k for some k. It is possible—due to the fact that the channels are non-FIFO—that message m_{k+1} is the first to arrive at process j. In this case the cut where $C^C\phi_3$ is learned is inconsistent, contradicting the assumption that j learns $C^C\phi_3$ after receiving one message.

Consider another run a' of $\mathcal{S}_3$ in which only one message is sent from i to j. Since j cannot distinguish a from a' until a second message is received—which could take arbitrarily long since message transmission is unbounded—in run a' process j never learns $C^C\phi_3$. Thus there is a run of $\mathcal{S}_3$ in which $C^C\phi_3$ is never attained, completing the proof. ∎

6.2.4 Timestamped Common Knowledge

Timestamped common knowledge [11,17] corresponds to agreement at local states having the same local clock value. It is sufficient for our purposes to use the asynchronous-runs model for comparison; we give the appropriate definitions for that model. Let $\mathit{clock}(a,c,i)$ be the value of i's local clock at cut c in run a. Then timestamped knowledge, K_i^T is defined as follows.

$$(a,c) \models_A K_i^T\phi \text{ iff } \forall c' : \mathit{clock}(a,c',i) = T : (a,c') \models_A K_i\phi$$

Note that the satisfaction of $K_i^T\phi$ is dependent only upon the run a, not upon the cut c. The definition of E^T follows the usual pattern.

$$E^T\phi \equiv \bigwedge_{i\in\mathcal{I}} K_i^T\phi$$

Now, timestamped common knowledge is defined as the greatest fixed point of $E^T(\phi \wedge X)$.

The strength of timestamped common knowledge depends upon characteristics of the local clocks used. For example [11]:

1. If it is common knowledge that all clocks always show identical times, then at T on any clock, $C^T\phi \Leftrightarrow C\phi$.

2. If ϕ is a stable fact and it is epsilon common knowledge that all clocks are within ϵ time units of each other, then at T on any clock, $C^T\phi \Rightarrow C^\epsilon\phi$.

3. If ϕ is a stable fact and it is eventual common knowledge that all local clocks read T eventually, then at time T on any clock, $C^T\phi \Rightarrow C^\diamond\phi$.

One type of local clock is termed a *logical clock* [15]. Logical clocks have the property that, if event a happens-before event b, then the local clock value at which a occurs is less than the local clock value at which b occurs. This implies that local states having the same clock values are incomparable with respect to happens-before, and consequently form a consistent cut. It might seem that if it is common knowledge that local clocks are logical clocks then $C^T\phi \Rightarrow C^C\phi$. However, there are two problems with this. One issue is that $C^T\phi$ alone does not guarantee that the clock value T is ever reached by any process. Consequently $C^T\phi$ may hold although not all processes know it; in particular, not those whose clocks never reach T. If T is never reached by any process, then $C^T(\mathit{false})$ holds. This is the case for $C^T\phi$ regardless of the local clock properties; it is also the reason that statements like "if it is eventual common knowledge that all local clocks read T eventually" are necessary in the three comparisons above. We will use $\mathit{Reached}(T)$ to denote that clock value T is reached by all processes in a run.

If it is common knowledge that all local clocks are logical clocks, then attainment of $C^T(\mathit{Reached}(T) \wedge \phi)$ does guarantee attainment of $C^C\phi$. Still, it is not the case that $C^T(\mathit{Reached}(T) \wedge \phi) \Rightarrow C^C\phi$; another issue is that $C^T\phi$ holds throughout the run, not just when clocks read T. Only when the clocks read T does $C^T\phi \Rightarrow C^C\phi$. We use the notation $\mathit{At}(T)$ as follows:

$$(a, c) \models_A \mathit{At}(T) \text{ iff } \forall i : \mathit{clock}(a, c, i) = T$$

Thus $\mathit{At}(T)$ implies both that T is reached by all local clocks and that, currently, all clocks have the value T. It is indeed the case that

$$C^T(\mathit{At}(T) \wedge \phi) \Rightarrow C^C\phi$$

as we will show in our next theorem.

It is not the case that with logical clocks $C^C\phi \Rightarrow C^T\phi$ for some T. The reasons for this are that (1) logical clocks do not guarantee that every consistent cut contains states having the same local clock value, and (2) as for eventual common knowledge, timestamped common knowledge implies that every process *knows* a fact, which is not guaranteed by concurrent common knowledge.

We formalize these observations regarding timestamped common knowledge with logical clocks in the next two theorems.

Theorem 17. Assume that it is common knowledge that local clock values are logical clocks. Then if $C^T(\mathit{Reached}(T) \wedge \phi)$ is attained by a system then $C^C\phi$ is attained also, and furthermore:

$$C^T(At(T) \wedge \phi) \Rightarrow C^C\phi$$

Proof: Suppose that $C^T(\mathit{Reached}(T) \wedge \phi)$ is attained in run a of a system $\mathcal{S}$. Consider any set of local states in which all clock values read T; there must be at least one set since $\mathit{Reached}(T)$ holds. From the definition of logical clocks, the set must form a consistent cut; call it c. From the definition of $C^T\phi$, $(a, c) \models_A \phi$. Cut c is distinguishable from the occurrence of the local clock value T and the knowledge of $\mathit{Reached}(T)$. Therefore, from Theorem 4, $(a, c) \models_A C^C\phi$. Thus attainment of $C^T(\mathit{Reached}(T) \wedge \phi)$ guarantees attainment of $C^C\phi$. Furthermore, since $C^T(At(T) \wedge \phi)$ implies the conditions on cut c above, $C^T(At(T) \wedge \phi) \Rightarrow C^C(\phi)$. ∎

Theorem 18. If it is common knowledge that local clock values are logical clocks, $C^C\phi \not\Rightarrow C^T\phi$ for any T. Moreover, there are systems and facts ϕ such that $C^C\phi$ is attained but $C^T\phi$ is not attained for any T.

Proof: Consider a system $\mathcal{S}_1'$ as follows. As in system $\mathcal{S}_1$ of Section 6.2.2, there are only two processes i and j. At some point process i sends a single message m to process j. In addition, i has a local clock which begins with the value 1, and increments to 2 immediately after i sends m. Process j has a local clock which begins at 2, and increments to 3 immediately after j receives m. It is simple to verify that the clock values indicated obey the conditions for logical clocks, since m is sent at local time 1 and received at local time 2. Let ϕ_1 be the fact "message m has been received." $C^C\phi_1$ is attained in this system exactly as in Section 6.2.2. However, there is no time T at which ϕ_1 holds when each process reaches T. Furthermore, i never *knows* ϕ_1, which is necessary by the definition of $C^T\phi$. By either argument, $C^T\phi_1$ is not attained for any T in this system. ∎

Finally, we consider general local clocks. Since attaining $C^C\phi$ does not guarantee attainment of $C^T\phi$ in the case of logical clocks, clearly it does not for the general case. Unlike for logical clocks, in the general case $C^T(\mathit{Reached}(T) \land \phi)$ does not guarantee attainment of $C^C\phi$.

Theorem 19. For general local clocks, there are systems in which $C^T(\mathit{Reached}(T) \land \phi)$ is attained, but $C^C\phi$ is not attained.

Proof: Consider system $\mathcal{S}_3'$ as follows: as in $\mathcal{S}_3$ in the proof of Theorem 16, there are two processors i and j, and i sends a series of messages—at least one—to j. Process j sends no messages to i. Also, in $\mathcal{S}_3'$, each local clock increments each time a message is sent or received. Recall fact ϕ_3, "i has sent at least one message to j." It is straightforward to see that $C^T(\mathit{Reached}(T) \land \phi_3)$ is attained in any run of this system, for $T = 1$. However, $C^C\phi_3$ is not attainable in all runs, exactly as in the proof of Theorem 16. ∎

Although $C^T(At(T) \land \phi) \Rightarrow C^C\phi$ for systems using logical clocks, this does not preclude the relevance of concurrent common knowledge for such systems. $C^T(At(T) \land \phi)$, as shown, is a stronger condition than $C^C\phi$. It implies a useful property for some distributed applications which $C^C\phi$ does not; namely, if two facts ϕ_1 and ϕ_2 become timestamped common knowledge with different timestamps, then all processors learn $C^{T_1}(At(T_1) \land \phi_1)$ and $C^{T_2}(At(T_2) \land \phi_2)$ in the same order. Known algorithms to achieve C^T with logical clocks [17] use two-phase algorithms such as Protocol 2. There do not appear to be low-latency protocols such as Protocol 1 for timestamped common knowledge. Furthermore, concurrent common knowledge does not require processes to keep a local clock value. Thus, for some applications $C^C\phi$ may be a more desirable, though weaker, form of knowledge than $C^T(At(T) \land \phi)$. One example in which both forms are useful is given in Section 5.3.

7 Conclusions

In this paper we have given a new knowledge-based definition of agreement that applies to asynchronous systems. We have defined the concept in terms of the causality relation between events, which is an appropriate replacement for the concept of time when one is discussing asynchronous systems [15]. We have defined concurrent common knowledge using a modal logic and semantics that are designed specifically to capture the causal structure relevant to asynchronous systems. We have shown that concurrent common knowledge is attainable by two simple and efficient algorithms and given several applications using it and other elements of the new logic. It is the attainability and broad applicability of concurrent common knowledge that makes it an important concept for asynchronous distributed computing.

One of the contributions of our work is that we have given a knowledge-theoretic definition that applies whenever one needs to reason about the global states of asynchronous systems. Thus we have pinned down the form of knowledge a protocol designer should try to attain when developing a protocol to reach agreement about some property of the global state. We have also used our logic to prove necessary and sufficient conditions for performing concurrent actions in asynchronous distributed systems.

There have been other proposals for weakened forms of common knowledge that are also attainable asynchronously, namely eventual common knowledge and timestamped common knowledge [11]. They use various temporal modalities in order to weaken the original definition of common knowledge. Concurrent common knowledge is strictly weaker than common knowledge but is, in general, incomparable, with the other forms above. In the case of timestamped common knowledge, if the clocks used in the definition are Lamport clocks as in [17] and the timestamp of interest is guaranteed to be reached by all processes, then timestamped common knowledge implies concurrent common knowledge when the appropriate local times are reached. However, in practice timestamped common knowledge involves longer latency and requires the suspension of system events for an interval. It also requires maintaining local clocks, which concurrent common knowledge does not. Thus achieving concurrent common knowledge may be more practical when it is sufficient for a particular application.

In the future, we hope to use our logic to understand better the communication requirements of a wider variety of asynchronous distributed protocols, and to aid in developing new and improved protocols. We also hope to extend the usefulness of our logic by addressing the issue of faulty environments.

Acknowledgements

We have benefited from discussions with Ken Birman, Ajei Gopal, Gil Neiger, Fred Schneider, Sam Toueg and especially Joe Halpern.

A Appendix

A.1 Proof of Consistent-Cut Existence

In this section we prove Lemma 1. This lemma says that, in any asynchronous run of any system, each local state of each process is included in some consistent cut of the system.

Proof of Lemma 1: Fix an asynchronous run, a, of a system. Consider any local state of any process, say state s_i^x on process i. Let e_i^x be the last event executed in s_i^x. We will iteratively construct a consistent cut c of a such that $(a,c)[i]$ is equal to s_i^x. Let $MinSends(k)$ for any k be the minimum (earliest) local state of process k in run a which includes the sending of all messages received by i from k in local state s_i^x. Initialize a vector of local states c as follows: $c[i] = s_i^x$ and for any $j \neq i$, $c[j] = MinSends(j))$. We refer to this as the *initial state vector.* Of course this vector is not necessarily a consistent cut. On each step of the iteration, find a message m' from any process j to any other process k whose reception is in c but whose sending is not (if such a message doesn't exist then we are finished). Set $c[j]$ to be the minimum local state in $a[j]$ which includes the sending of m'; note that the old $c[j]$ is a strict prefix of the original. In order to meet the conditions of the theorem, we must show that (1) the iterations terminate and that (2) the value of $c[i]$ never changes.

We first make an observation to be used extensively in the remainder of the proof: that, for any inconsistent message m' in the above construction, there is a message chain beginning with m' and ending with a message received within the initial state vector. Consider any message m' as above,

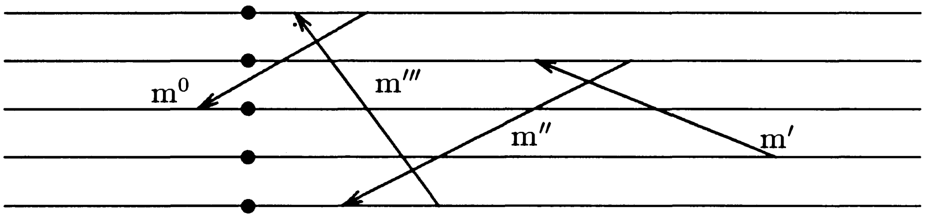

Figure 5: Proof of Lemma 1: Message chains during iterative construction.

from process j to process k. Then either $c[k]$ is a member of the initial state vector or the last event of $c[k]$ is the sending of a message m'', where m'' was the inconsistent message of some previous iteration. In the later case $receive(m') \rightarrow send(m'')$. This argument can be continued resulting in a message chain

$$send(m') \rightarrow receive(m') \rightarrow send(m'') \rightarrow receive(m'') \ldots \rightarrow receive(m^0)$$

where $receive(m^0)$ is in the initial state vector. (See Figure 5. The solid circles represent the initial state vector.)

We now show that the iteration terminates; suppose that it does not. At any iteration there are only finitely many inconsistent messages, since prefixes contain a finite number of receive events. Thus for the iteration to be non-terminating there must be an infinite message chain of the form described above. By the pigeonhole principle, in such a chain there must be two messages, say m_1 and m_2, that are sent by the same process. If $m_1 \rightarrow m_2$ in the chain then also $m_2 \rightarrow m_1$ because there is a local state which includes m_2 but not m_1. This cannot occur in any valid run. Therefore the iteration terminates.

Finally, we show that the local state of i is not changed, i.e. upon termination $c[i] = s_i^x$. Suppose that during the iteration state $c[i]$ is changed, due to a message m' sent by i. Again, there must be a chain of messages as above ending with $receive(m^0)$ in the original states and $send(m') \rightarrow receive(m^0)$. There are two cases depending on what process has $receive(m^0)$. (1) If it is process i then there is a circularity in $\rightarrow$ similar to the proof of termination above, as the message chain begins and ends with the same process. (2) Suppose that it is a process $j \neq i$. Recall that e_i^x is the last event of s_i^x and hence of the original $c[i]$. Recall also that, by the definitions of $MinSends$, the last event of j in the initial state vector is the sending of a message to i, call it m_j, which is received before or at event e_i^x. We thus have that m_j is sent after $receive(m^0)$ and received before or at e_i^x, so that $receive(m^0) \rightarrow e_i^x$. However, $e_i^x \rightarrow send(m')$ and $send(m') \rightarrow receive(m^0)$, resulting in an invalid circularity. Hence the final $c[i]$ is equal to s_i^x. ∎

References

[1] Ken Birman and Thomas Joseph. Reliable communication in the presence of failures. *A.C.M. Transactions on Computer Systems*, 5(1), February 1987.

[2] G. Bracha and S. Toueg. Distributed deadlock detection. *Distributed Computing*, 2(3):127–138, 1987.

[3] M. Chandy and L. Lamport. Finding global states of a distributed system. *A.C.M. Transactions on Computer Systems*, 3(1):63–75, 1985.

[4] M. Chandy and J. Misra. How processes learn. *Distributed Computing*, 1(1):40–52, 1986.

[5] E.J.H. Chang. Echo algorithms: depth parallel operations on graphs. *I.E.E.E. Transactions on Software Engineering*, SE-8(4):391–400, 1982.

[6] Carol Critchlow. On inhibition and atomicity in asynchronous consistent-cut protocols. Technical Report 89-1069, Cornell University Department of Computer Science, December 1989.

[7] Carol Critchlow and Kim Taylor. The inhibition spectrum and the achievement of causal consistency. Technical Report 90-1101, Cornell University Department of Computer Science, February 1990. To appear in *Proceedings of the ACM Symposium on Principles of Distributed Computing*, August 1990.

[8] C. Dwork and Y. Moses. Knowledge and common knowledge in a byzantine environment: The case of crash failures. In J. Halpern, editor, *Proceedings of the Conference on Theoretical Aspects of Reasoning*

About Knowledge, pages 149–170. Morgan Kaufmann, 1986. To appear in *Information and Computation*, 1990.

[9] N. Francez. Distributed termination. *A.C.M. Transactions on Programming Languages and Systems*, 2(1):42–55, January 1980.

[10] J. Y. Halpern and R. Fagin. Modelling knowledge and action in distributed systems. 1988. To appear in Distributed Computing.

[11] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. In *Proceedings of the Third A.C.M. Symposium on Principles of Distributed Computing*, pages 50–61, 1984. A revised version appears as IBM Research Report RJ 4421, Aug. 1987; to appear in J.A.C.M.

[12] Richard Koo and Sam Toueg. Checkpointing and rollback-recovery for distributed systems. *IEEE Transactions On Software Engineering*, SE-13(1):23–31, January 1987.

[13] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[14] L. Lamport. Paradigms for distributed computing. *Methods and tools for specification, an advanced course* (M. Paul and H.J. Siegert, eds.), Lecture Notes in Computer Science 190, pages 19–30,454–468, 1985.

[15] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the A.C.M.*, 21(7):558–565, 1977.

[16] Y. Moses and M. Tuttle. Programming simultaneous aactions using common knowledge. *Algorithmica*, 3:121–169, 1988. A preliminary version appeared in the 27th IEEE Symp. on Foundations of Computer Science, 1986.

[17] Gil Neiger and Sam Toueg. Substituting for real time and common knowledge in asynchronous distributed systems. In *Proceedings of the Sixth A.C.M. Symposium on Principles of Distributed Computing*, pages 281–293, 1987. To appear in J.A.C.M.

[18] R. Parikh and R. Ramanujam. Distributed processing and the logic of knowledge. In *Proceedings of the Brooklyn College Workshop on Logics of Programs*, pages 256–268, 1985.

[19] D. L. Russell. Process backup in producer-consumer systems. In *Proceedings of the A.C.M. Symposium on Operating Systems Principles*, November 1977.

[20] A. Tarski. A lattice-theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[21] Kim Taylor. The role of inhibition in asynchronous consistent-cut protocols. In *Lecture Notes in Computer Science 392: Distributed Algorithms* (Proceedings of the Third International Workshop on Distributed Algorithms, Nice, France, September 1989), J.-C. Bermond and M. Raynal, Editors, pages 280–291. Springer-Verlag, 1989.