*Ann. Rev. Comput. Sci. 1987. 2:37–68*






# USING REASONING ABOUT KNOWLEDGE TO ANALYZE DISTRIBUTED SYSTEMS

*Joseph Y. Halpern*

IBM Almaden Research Center, San Jose, CA 95120

## 1. INTRODUCTION

Designing, understanding, and reasoning about distributed systems can be complicated. The major complexities arise from the uncertainties inherent in the system, particularly with regard to message delivery and possible faulty or unexpected behavior of processors. A protocol must be designed (and proved!) to function correctly even if it is possible for messages to be lost, for messages to arrive out of order, or for some processor to fail. The difficulty of this task can be viewed as stemming from a lack of *global knowledge* in a system. To quote Gray (1979):

> The main problem ***unique*** to a distributed system is a lack of (global) knowledge. It is difficult (probably impossible) for one node to know everything about the rest of the network. Yet global knowledge seems to be required to answer questions such as "Where is the file *A*?", "Is there a deadlock?", [or] "What is the best way to answer the question . . . ?"

As this quotation suggests, analyzing distributed protocols often involves reasoning about processors' "states of knowledge" at various points in the protocol. One often hears informal arguments of the form "Once the sender receives the acknowledgment, it *knows* that the current packet has been delivered; it can then safely discard the current packet and send the next packet." Although notions of knowledge have frequently been used in informal descriptions, traditional formal analyses of distributed protocols avoided any explicit mention of knowledge. Recent work has shown that these informal arguments can be completely formalized, using ideas for formalizing knowledge that go back to work of philosophers in the late 1950s and early 1960s.

In this survey I explain how the notion of knowledge in distributed



8756–7016/87/1115–0037$02.00

systems can be formalized, give some examples from the literature showing that this formalization is indeed useful for analyzing distributed systems, discuss related current work, and present some important open problems. Although the focus of our discussion is mainly distributed systems of processors, the "processors" could also be people or robots. Thus the ideas presented apply perfectly well to the analysis of a system of communicating robots, a bargaining session, or a conversation. Not surprisingly, interest in knowledge has been rising recently in such areas as distributed AI, economics, and linguistics.



The model used by philosophers for capturing knowledge is called the *possible-worlds* model (cf Hintikka 1962). The intuition here is that besides the true state of affairs, an agent considers a number of other worlds or *states of affairs* possible. An agent is said to *know* a fact $\varphi$ if $\varphi$ is true at all the worlds he considers possible. In a situation such as a poker game, these possible worlds have a concrete interpretation: They are simply all the possible ways the cards could have been distributed among the players. Players may acquire additional information in the course of the play of the game. This additional information allows them to eliminate some of the worlds they consider possible. At some point Alice might know that Bob holds the ace of spades, since in all worlds (distributions of cards among players) that she currently considers possible, Bob holds the ace of spades. Note here that possibility is viewed as the dual of knowledge. Intuitively, the more worlds an agent considers possible, the greater her uncertainty, and the less she knows.

In distributed systems we can also give a concrete interpretation to the notion of possible worlds. We identify a system with a set of possible *runs*, where a run is a complete description of what happens in the system over time. For example, the fact that a given message may not arrive is captured by having one run in which it does arrive and another in which it does not. We define a *point* to be a pair $(r, t)$ consisting of a run $r$ and time $t$. We can view the points of a system as possible worlds. At any point, the system is in some *global state*, which we can just view as a tuple consisting of each processor's *local state*. At one point a processor will consider another point possible if the processor has the same *local state* in the global states corresponding to the two points. Thus processor $i$ is said to *know* a fact $\varphi$ at a point $(r, t)$ if $\varphi$ is true at all points $(r', t')$ that it cannot distinguish from $(r, t)$ (because it is in the same local state in the global states corresponding to these points). As Gray observes, a processor will in general not know everything about the global state of the system, but it will usually have some information about the global state.

This definition of knowledge in distributed systems is an *external* definition. The system designer ascribes knowledge to processors in each

global state. A processor does not compute this knowledge in any sense, nor can a processor necessarily answer questions based on its knowledge. As we shall show, this definition of knowledge turns out to be useful for many applications. However, there are clearly others for which it is important to take into account the difficulty of computing knowledge. (This issue is discussed in more detail in Section 7.)

In our analysis of distributed systems, certain states of "group knowledge" are shown to be relevant. We are often interested in situations in which everyone in a group (or every processor in a network) knows a certain fact. For example, a society certainly wants all drivers to know that a red light means "stop" and a green light means "go." But even if we assume that every driver knows this fact and follows the rules, this is not enough to ensure that traffic flows smoothly. A driver will not feel safe going when he sees a green light unless he knows that everyone else knows and follows the rules. Thus for traffic to flow smoothly it is also necessary that everyone know that everyone knows the rules. (Notice that a number of implicit assumptions about the relationship between knowledge and action are being made here.)

Even the state of knowledge in which everyone knows that everyone knows does not suffice for a number of applications. In some cases we also need to consider the state in which simultaneously everyone knows a fact $\varphi$, everyone knows that everyone knows $\varphi$, everyone knows that everyone knows that everyone knows $\varphi$, and so on. In this case we say that the group has *common knowledge* of $\varphi$. This key notion has been studied by philosophers (Lewis 1979), linguists (Clark & Marshall 1981; Perrault & Cohen 1981), economists (Aumann 1976; Milgrom 1981), researchers in AI (McCarthy et al 1978), and researchers in distributed systems (Halpern & Moses 1984; Lehmann 1984; Moses 1986; Dwork & Moses 1986; Moses & Tuttle 1986). It will be of particular interest to us here because, as is shown in Section 4, common knowledge is a prerequisite for simultaneous agreement and coordinated action.

At the other end of the spectrum from common knowledge is *implicit knowledge*. A group has implicit knowledge of a fact $\varphi$ if, by pooling their knowledge together, the members of the group could deduce $\varphi$, even though it may be the case that no member of the group individually knows $\varphi$. For example, if Alice knows that Bob is in love with either Carol or Susan, and Charlie knows that Bob is not in love with Carol, then together they implicitly know that Bob is in love with Susan, although neither Alice nor Charlie individually has this knowledge. Common knowledge and implicit knowledge turn out to be useful tools in helping us analyze complicated group situations.

The remainder of this survey is organized as follows. The possible-

worlds model for knowledge is reviewed in Section 2 and is related to distributed systems in Section 3. The following three sections give examples of using reasoning about knowledge to analyze distributed systems. In Section 4 there is a knowledge-based analysis of the *coordinated attack* problem (Gray 1978), taken from Halpern & Moses (1984). The importance of common knowledge and the relationship between common knowledge and agreement is brought out by this and related examples. Section 5 describes results from Chandy & Misra (1986) on how knowledge can be gained and lost in *asynchronous* systems. Such results can be used to provide lower bounds on the number of messages required in certain protocols. Section 6 describes results from Dwork & Moses (1986) and Moses & Tuttle (1986) on using knowledge to analyze *Simultaneous Byzantine Agreement* (Pease et al 1980). *Knowledge-based protocols*, ones in which the actions of a processor depend explicitly on tests for knowledge, turn out to be a useful tool in this analysis. In Section 7 there is a discussion of some attempts to take into account the difficulty of computing knowledge. The survey concludes in Section 8 with a discussion of related work and a number of problems deserving further investigation.



## 2. THE POSSIBLE-WORLDS MODEL

In order to reason formally about knowledge, we need a language that allows us to express notions of knowledge in a straightforward way. Suppose we have a group consisting of $n$ agents (or processors or robots), creatively named $1, \ldots, n$. For simplicity, we assume these agents wish to reason about a world that can be described in terms of a set $\Phi$ of primitive propositions, which we label $p, p', q, q', \ldots$. In distributed systems, these primitive propositions will typically represent statements such as "the value of variable $x$ is 0" or "processor 3 is faulty." In order to express a statement like "processor 1 knows that processor 3 is faulty," we augment the language by *modal* operators $K_1, \ldots, K_n$ (one for each agent). A statement like $K_1\varphi$ is then read "agent 1 knows $\varphi$."

Formally, we start with the primitive propositions in $\Phi$ and form more complicated formulas by closing off under negation, conjunction, and the modal operators $K_1, \ldots, K_n$. Thus, if $\varphi$ and $\psi$ are formulas, then so are $\sim\varphi$, $\varphi \wedge \psi$, and $K_i\varphi$, for $i = 1, \ldots, n$. We also use standard abbreviations from propositional logic, such as $\varphi \vee \psi$ for $\sim(\sim\varphi \wedge \sim\psi)$, $\varphi \Rightarrow \psi$ for $\sim(\varphi \wedge \sim\psi)$, and $\varphi \equiv \psi$ for $(\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$.

We can express quite complicated statements in a straightforward way using this language. For example, the formula $K_1K_2p \wedge \sim K_2K_1K_2p$ says that agent 1 knows that agent 2 knows $p$, but agent 2 does not know that agent 1 knows that agent 2 knows $p$. However, as it stands, the language

does not allow us to express the notions of common knowledge and implicit knowledge discussed in the introduction. In order to express these notions, we augment the language with modal operators $E_G$ ("everyone in the group $G$ knows"), $C_G$ ("it is common knowledge among the agents in $G$") and $I_G$ ("it is implicitly known among the agents in $G$") for every subset $G$ of $\{1, \ldots, n\}$. Thus, if $\varphi$ is a formula, then so are $E_G\varphi$, $C_G\varphi$, and $I_G\varphi$. We often omit the subscript $G$ if $G$ is clear from context. In this augmented language we can make statements like $K_3 \sim C_{\{1,2\}}p$ ("agent 3 knows that $p$ is not common knowledge among agents 1 and 2") and $Iq \wedge \sim Cq$ ("$q$ is implicitly known but is not common knowledge").

We next want to formalize the notion of possible worlds, as discussed in the introduction. One way of doing so is by using *Kripke structures* (Kripke 1963). A Kripke structure $M$ for $n$ agents is a tuple $(S, \pi, \mathcal{K}_1, \ldots, \mathcal{K}_n)$, where $S$ is a set of *states* or *possible worlds*, $\pi$ is an assignment of truth values to the primitive propositions of $\Phi$ for each state $s \in S$ (i.e. $\pi(s, p) \in \{\textbf{true}, \textbf{false}\}$ for each state $s \in S$ and each primitive proposition $p \in \Phi$), and $\mathcal{K}_i$ is an *equivalence relation* on $S$. [As usual, an equivalence relation $\mathcal{K}$ on $S$ is a binary relation that is *reflexive*, which means that for all $s \in S$, we have $(s, s) \in \mathcal{K}$, *symmetric*, which means that for all $s, t \in S$, we have $(s, t) \in \mathcal{K}$ if and only if $(t, s) \in \mathcal{K}$, and *transitive*, which means that for all $s, t, u \in S$, we have that if $(s, t) \in \mathcal{K}$ and $(t, u) \in \mathcal{K}$, then $(s, u) \in \mathcal{K}$.]

The truth assignment $\pi$ tell us, for each state $s$ and each primitive proposition $p$, whether $p$ is true or false at state $s$. Thus, if $p$ denotes the fact "processor 3 is faulty," then $\pi(s, p) = \textbf{true}$ captures the situation in which processor 3 is indeed faulty in the state $s$. $\mathcal{K}_i$ is intended to capture the possibility relation according to agent $i$; $(s, t) \in \mathcal{K}_i$ if agent $i$ cannot distinguish the possible worlds $s$ and $t$ in structure $M$.

We are now ready to assign truth values to formulas. A formula will be true or false at a possible world. It is quite possible that a formula will be true at one world but false at another. For example, in one world (global state of the system) processor 1 may know that processor 3 is faulty, in another it may not. To capture this, we define the notion "$\varphi$ *is true at* $(M, s)$" or "$(M, s)$ *satisfies* $\varphi$," written $(M, s) \models \varphi$, by induction on the structure of $\varphi$:

$(M, s) \models p$ (for a primitive proposition $p \in \Phi$) if $\pi(s, p) = \textbf{true}$

$(M, s) \models \varphi \wedge \psi$ if $(M, s) \models \varphi$ and $(M, s) \models \psi$

$(M, s) \models \sim \varphi$ if $(M, s) \not\models \varphi$

$(M, s) \models K_i\varphi$ if $(M, t) \models \varphi$ for all $t$ such that $(s, t) \in \mathcal{K}_i$.

The first three clauses in this definition are the same as the corresponding clauses for propositional logic. The last clause captures the intuition that agent $i$ knows $\varphi$ in world $s$ of structure $M$ exactly if $\varphi$ is true at all worlds that $i$ considers possible when in $s$.

These definitions are perhaps best illustrated by a simple example. One of the advantages of a Kripke structure is that it can be viewed as a labeled directed graph, where the nodes are the states of $S$, each node is labeled by the primitive propositions that are true and false at that state, and there is an edge between $s$ and $t$ labeled $i$ exactly if $(s,t) \in \mathcal{K}_i$. For example, suppose $\Phi = \{p\}$ and $n = 2$, so that our language only has one primitive proposition $p$ and there are only two agents. Further suppose that $M = (S, \pi, \mathcal{K}_1, \mathcal{K}_2)$, where $S = \{s,t,u\}$, $p$ is true at states $s$ and $u$, but false at $t$ (so that $\pi(s,p) = \pi(u,p) =$ **true** and $\pi(t,p) =$ **false**), agent 1 cannot distinguish $s$ and $t$ (so that $\mathcal{K}_1 = \{(s,s),$
2 cannot distinguish $s$ and $u$ (so that $\mathcal{K}_2 = \{(s,s),(s,u)(t,t)(u,s)(u,u)\}$). This situation can be captured by the graph in Figure 1. If we view $p$ as standing for "it is sunny in San Francisco," then in state $s$ it is sunny in San Francisco but agent 1 does not know it (since agent 1 considers both $s$ and $t$ possible). On the other hand, agent 1 does know that agent 2 knows whether or not it is sunny in San Francisco (since in both worlds agent 1 considers possible, agent 2 knows what the weather in San Francisco is). Agent 2 knows that it is sunny in San Francisco, but does not know that agent 1 does not know this fact (since agent 2 considers the world $u$ possible, and in this world agent 1 does know that it is sunny in San Francisco). Formally, we have

$$(M,s) \models p \wedge \sim K_1 p \wedge K_1(K_2 p \vee K_2 \sim p) \wedge K_2 p \wedge \sim K_2 \sim K_1 p.$$

Note that in both $s$ and $u$, the primitive proposition $p$ (the only primitive proposition in our language) gets the same truth value. One might think, therefore, that $s$ and $u$ are the same and that one of them can be eliminated. This is not true! A state is not completely characterized by the truth values that the primitive propositions get there. The possibility relation is also crucial. For example, in world $s$ agent 1 considers $t$ possible, while in $u$ he

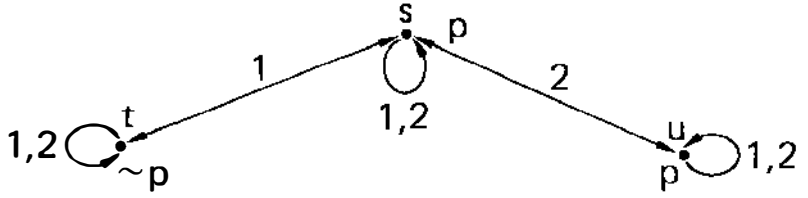


*Figure 1*

does not. As a consequence, agent 1 does not know $p$ in $s$, while in $u$ he does.

We can easily extend the definition of truth to handle common knowledge and implicit knowledge. Since $E_G\varphi$ is true exactly if everyone in the group $G$ knows $\varphi$, we have:

$(M, s) \models E_G\varphi$ if $(M, s) \models K_i\varphi$ for all $i \in G$.

$C_G\varphi$ is true if everyone in $G$ knows $\varphi$, everyone knows that everyone knows $\varphi$, etc. Let $E_G^1\varphi$ be an abbreviation for $E_G\varphi$, and let $E_G^{k+1}\varphi$ be an abbreviation for $E_G E_G^k\varphi$ for $k \geq 1$. Then we have

$(M, s) \models C_G\varphi$ if $(M, s) \models E_G^k\varphi$ for $k = 1, 2, \ldots$.

Our definition of common knowledge has an interesting graph-theoretical interpretation which is useful in many applications. Define a state $t$ to be *G-reachable from state s in k steps* ($k \geq 0$) if there exist states $s_0, s_1, \ldots, s_k$ such that $s_0 = s$, $s_k = t$, and, if $k \geq 1$, then for all $j$ with $0 \leq j \leq k-1$, there exists $i \in G$ such that $(s_j, s_{j+1}) \in \mathcal{K}_i$. We say $t$ is *G-reachable from s* if $t$ is $G$-reachable from $s$ in $k$ steps for some $k \geq 0$. Thus, $t$ is $G$-reachable from $s$ if there is a path in the graph from $s$ to $t$ all of whose edges are labeled by members of $G$.

**Lemma 2.1**

1. *$(M,s) \models E_G^k\varphi$ if and only if $(M,t) \models \varphi$ for all $t$ that are G-reachable from $s$ in at most $k$ steps.*
2. *$(M, s) \models C_G\varphi$ if and only if $(M, t) \models \varphi$ for all $t$ that are G-reachable from $s$.*

**Proof** Part (1) follows from a straightforward induction on $k$, while part (2) is immediate from part (1). ■

A group has implicit knowledge of $\varphi$ if someone who could "combine" the knowledge of the members of $G$ would know $\varphi$. How can we capture the idea of combining knowledge in our framework? In the Kripke structure illustrated above, in state $s$ agent 1 considers both $s$ and $t$ possible, but does not consider $u$ possible, while agent 2 considers $s$ and $u$ possible, but not $t$. Someone who could combine their knowledge would know that only $s$ was possible: agent 1 has enough knowledge to eliminate $u$ and agent 2 has enough knowledge to eliminate $t$. In general, we combine knowledge of the agents in a group by *intersecting* the sets of worlds that each of the agents in the group considers possible. Thus we have

$(M, s) \models I_G\varphi$ iff $(M, t) \models \varphi$ for all $t$ such that $(s, t) \in \cap_{\{i \in G\}} \mathcal{K}_i$.

The notion of knowledge defined here has a number of important properties. One way of characterizing its properties is by characterizing

the formulas that are always true. More formally, given a structure $M = (S, \pi, \mathcal{K}_1, \ldots, \mathcal{K}_n)$, we say a formula $\varphi$ is *valid in* $M$ if $(M, s) \models \varphi$ for every state $s \in S$; we say $\varphi$ is *valid* if it is valid in every structure $M$. It turns out that the validity problem for the logic we have presented is decidable, and its complexity has been well studied. It is known that if we have common knowledge in the language with $n \geq 2$ agents, then the problem of deciding whether a formula is valid is complete for exponential time; without common knowledge the validity problem is complete for polynomial space, and if we just consider the one-agent case, it is co-NP-complete; see Ladner (1977) and Halpern & Moses (1985). It is also known that we can find a sound and complete axiomatization that completely characterizes the valid formulas in this logic. [This logic is essentially a multi-agent version of the logic S5, which has long been studied by philosophers; our notion of knowledge inherits all the properties of S5. See Halpern & Moses (1985) for details.]

Rather than discuss the full complete axiomatization, I will focus here on three particular properties of our definition of knowledge. The first property says that agents know only true facts. Thus, all instances of the following axiom are valid:

$$K_i\varphi \Rightarrow \varphi.$$

This property has been taken by philosophers to be the major one distinguishing knowledge from *belief*. Although you may have false beliefs, you cannot know something that is false. This property holds because the actual world is always one of the worlds that an agent considers possible (i.e. $\mathcal{K}_i$ is reflexive). If $K_i\varphi$ holds at a particular world $s$ of structure $M$, then $\varphi$ must be true at all worlds that $i$ considers possible, so in particular it is true at $(M, s)$.

The second property says that each agent knows all the logical consequences of his knowledge. If an agent knows $\varphi$ and knows that $\varphi$ implies $\psi$, then he also knows $\psi$. This is so because if $\varphi$ and $\varphi \Rightarrow \psi$ are both true at all worlds the agent considers possible, then $\psi$ must also be true at all worlds the agent considers possible. Thus, all instances of the following axiom are valid:

$$(K_i\varphi \wedge K_i(\varphi \Rightarrow \psi)) \Rightarrow K_i\psi.$$

The last property says that an agent knows all valid formulas. If $\varphi$ is true at all the possible worlds of a structure $M$, then $\varphi$ must be true at all the worlds that an agent considers possible in any given world in $M$, so it must be the case that $K_i\varphi$ is true at all possible worlds of $M$. Thus we have

If $\varphi$ is valid in $M$, then so is $K_i\varphi$.

Note that this is a somewhat stronger statement than "If $\varphi$ is valid, then so is $K_i\varphi$." It is quite different from the formula $\varphi \Rightarrow K_i\varphi$, which says that if $\varphi$ is true then $i$ knows it. This formula is not valid. It is quite possible that Alice has the ace of spades but Bob does not know it.

The latter two properties emphasize the fact that our definition does not take into account the difficulty of computing knowledge. In some sense, it assumes that agents are *logically omniscient*: They know all the logical consequences of their knowledge and know all formulas that are valid in a given model. This observation has sparked many attempts to get notions of knowledge that do not have these properties. As we mentioned in the introduction, the way we ascribe knowledge to processors in distributed systems makes no attempt to take computation into account, so it is perhaps not surprising that it has these properties, nor should it be bothersome on philosophical grounds. The next few sections focus on this definition of knowledge and its applications to distributed systems; some attempts at modifying the model to get a definition of knowledge that does take computation into account are discussed in Section 7.



## 3. ASCRIBING KNOWLEDGE TO PROCESSORS IN A DISTRIBUTED SYSTEM

Recently there have been a number of papers that attempt to define the notion of knowledge in distributed systems (Halpern & Moses 1984; Parikh & Ramanujam 1985; Chandy & Misra 1985; Halpern & Fagin 1985; Dwork & Moses 1986; Fischer & Immerman 1986; Ladner & Reif 1986; Moses & Tuttle 1986). All of the definitions have essentially the same features, which we briefly outline below. Interestingly, Rosenschein and Kaelbling have used the same definition of knowledge to analyze synchronous digital machines, viewing the components of the machine as agents (Rosenschein & Kaelbling 1986).

A distributed system consists of a collection of processors, say $1, \ldots, n$, connected by a communication network. The processors communicate with each other over the links in the network. Intuitively, a *run* $r$ of a distributed system is a complete description of the execution of that system over time. Thus, a run includes a description of which messages a processor sent and received and when it received them; if processors have local clocks, it will also include the local time that each message was sent and received. If there are faulty processors, the run will include which processors were faulty, when they were faulty, and a description of their faulty behavior.

We identify a distributed system with a set of runs. Most often, we shall be considering the set of runs of a particular protocol implemented in an

environment with certain properties (for example, an environment in which communication is guaranteed, or one in which there are at most $f$ faulty processors). Various properties of the environment can be captured by taking a suitable set of runs. For example, we can capture the property of communication being guaranteed by taking the system to consist only of runs in which every message that is sent is later received. If we want to capture the assumption that at most $f$ processors ever fail, we take the system to consist only of runs in which at most $f$ processors fail. We can capture the property of all local clocks being synchronized by only considering runs in which the local clocks of processors have the same reading at all times. By viewing the set of runs as a measure space, we can capture probabilistic properties of runs, such as certain events being independent or having probability one.

At a time $t$ in a run $r$, each processor has some *history*, which intuitively consists of its initial state and the sequence of events observed by the processor in run $r$ up to time $t$. These events include the messages sent and received by the processor, perhaps some local internal events, and, if there are clocks in the system, the times at which these events took place.

In general a processor will not "remember" all of its history. Thus we can assign to each processor at every point in a run some *local state*, which will be a function of its history. This local state intuitively describes all that the processor remembers of what it has seen so far. If we consider "ideal" processors that have unbounded memory, then this local state might encode the processor's entire history, including all the messages it has sent and received and its internal transitions. In general, of course, the local state would encode only a portion of the processor's history. A *protocol* is simply a function from local states to actions. We focus attention here on deterministic protocols, although our definitions can be modified to deal with probabilistic and nondeterministic protocols.

Recall that we define a point to be a pair $(r, t)$ consisting of a run and a time. To capture these intuitions formally, we associate with each point $(r, t)$ a *global state* $g(r, t)$ of the system, which is simply an $n$-tuple $\langle l_1, \ldots, l_n \rangle$ describing each processor's local state at that point.[1] We say two points $(r, t)$ and $(r', t')$ are *indistinguishable* to processor $i$, and write $(r, t) \sim_i (r', t')$, if processor $i$'s local state is the same in both $g(r, t)$ and $g(r', t')$. The crucial point here is that since a processor's actions when

[1] In a more general model, we might also want one more component in the tuple to describe the *environment*, which intuitively consists of all the relevant features of the system not described by the processors' local states, such as messages in transit but not yet delivered, and so on (cf Fagin et al 1986). For ease of exposition, the environment component is omitted here.

it runs a protocol are a function of its local state, if two points are indistinguishable to processor $i$, then processor $i$ will perform the same actions at both these points.

We can now view a distributed system (i.e. a set of runs $R$, together with an assignment $g$ of global states to points) as a Kripke structure. The possible worlds are the points, and the indistinguishability relation $\sim_i$ defines the equivalence relation $\mathcal{K}_i$. We take the primitive propositions to be basic facts about the system (such as which processors are faulty and what the values are of certain variables), and assume that for each such basic fact $p$ and each point $(r, t)$, we have some way of deciding whether or not $p$ is true at $(r, t)$. All the definitions for knowledge given in the previous section apply immediately. In particular, processor $i$ knows a fact $\varphi$ at point $(r, t)$ in a set of runs $R$ if $\varphi$ is true at all the points that $i$ cannot distinguish from $(r, t)$. We write $(R, r, t) \models \varphi$ if $\varphi$ is true at the point $(r, t)$, where $r \in R$.

We might question whether this definition of knowledge is reasonable. One argument that it is reasonable is that it does capture one way the word "know" has been used informally. For example, when someone says "processor 2 does not know that processor 3 is faulty at the end of round 5 in this run," denoting "this run" by $r$, what is often meant is that there is a point indistinguishable to processor 2 from the point $(r, 5)$ where processor 3 is not faulty. More importantly, as we shall now see, this notion of knowledge is useful for analyzing protocols.

## 4. COMMON KNOWLEDGE AND THE "COORDINATED ATTACK" PROBLEM

A good example of using knowledge to analyze a protocol is provided by the *coordinated attack problem*, from the distributed systems folklore (Gray 1978). The version of the story given here, and its analysis, is taken from Halpern & Moses (1984):

> Two divisions of an army are camped on two hilltops overlooking a common valley. In the valley awaits the enemy. It is clear that if both divisions attack the enemy simultaneously they will win the battle, whereas if only one division attacks it will be defeated. The divisions do not initially have plans for launching an attack on the enemy, and the commanding general of the first division wishes to coordinate a simultaneous attack (at some time the next day). Neither general will decide to attack unless he is sure that the other will attack with him. The generals can only communicate by means of a messenger. Normally, it takes the messenger one hour to get from one encampment to the other. However, it is possible that he will get lost in the dark or, worse yet, be captured by the enemy. Fortunately, on this particular night, everything goes smoothly. How long will it take them to coordinate an attack?

Suppose the messenger sent by General $A$ makes it to General $B$ with a message saying "Attack at dawn." Will $B$ attack? No, since $A$ does not know $B$ got the message, and thus may not attack. So $B$ sends the messenger back with an acknowledgment. Suppose the messenger makes it. Will $A$ attack? No, because now $A$ is worried that $B$ does not know $A$ got the message, so that $B$ thinks $A$ may think that $B$ did not get the original message, and thus not attack. So $A$ sends the messenger back with an acknowledgment. But of course, this is not enough either.

In terms of knowledge, each time the messenger makes a transit, the *depth* of the generals' knowledge increases by one. If we let the primitive proposition $m$ stand for "a message saying 'Attack at dawn' was sent by $A$," then when $B$ gets the message, we have $K_B m$. When $A$ gets $B$'s acknowledgment, $A$ knows that $B$ knows $m$; i.e. $K_A K_B m$ holds. The next acknowledgment brings us to $K_B K_A K_B m$. Although more acknowledgments keep increasing the depth of knowledge, it is not hard to show that by following this protocol, the generals never attain common knowledge that the attack is to be held at dawn.

We can directly prove this result, but in fact it is a corollary of a more general result. Since the messenger may get captured or lost, communication between the generals is not guaranteed. We want to capture this in our formal model. Recall that we are identifying a system with a set of runs. Roughly speaking, we say that *communication is not guaranteed* in a system $R$ if for every run $r$ in $R$ and time $t$, there is another run $r'$ in $R$ that looks just like $r$ up to time $t$, except that an arbitrary subset of the messages delivered in $r$ at time $t$ is not delivered in run $r'$, and after time $t$ no messages are delivered in $r'$. This definition is meant to capture the intuition that it is always possible that no messages from a certain time on will arrive. We say a fact $\varphi$ is *initially undetermined* if for every run $r$ there is another run $r'$ such that the initial global states at $(r, 0)$ and $(r', 0)$ are identical (i.e. $g(r, 0) = g(r', 0)$) and $\varphi$ never holds in $r'$. For example, the fact "the generals are attacking" is initially undetermined in the coordinated attack problem, since if no messages are ever delivered the generals will never attack.

**Theorem 4.1** *Let $R$ be a system in which communication is not guaranteed. If $|G| \geq 2$ and $\varphi$ is initially undetermined, then for all runs $r \in R$ and all times $t$, we have $(R, r, t) \models \sim C_G \varphi$.*

This result tells us that in a system in which communication is not guaranteed, common knowledge of initially undetermined facts is not attainable *in any run of any protocol.* Now the description of the coordinated attack problem guarantees that the set of runs of any protocol run by the generals will be a system where communication is not guaranteed.

It thus follows that the generals cannot attain common knowledge of a fact such as "the generals are both attacking" in any run of any protocol.

It may seem to the reader that we have still not dealt with our original problem. The generals are not interested in attaining common knowledge; they are interested in coordinating an attack. However, as we are about to see, common knowledge is a prerequisite for agreement. In order to agree to attack, the generals must have common knowledge of the attack. For suppose the generals are running some protocol $P$ that achieves coordinated attack. Let $R$ be the set of runs of protocol $P$. As we noted above, $R$ is a set of runs in which communication is not guaranteed. Let $p$ be a formula that denotes "the generals are both attacking," and suppose $p$ holds at time $t$ in some run $r$ of $P$. Suppose $(r', t')$ is indistinguishable by $A$ from $(r, t)$ (see Figure 2). For example, it may be the case that in $(r', t')$ $A$'s last message to $B$ was not received, while in $(r, t)$ it was. (If the generals have access to a global clock, then we would have $t = t'$. Our analysis goes through whether or not we make this assumption.) Since $A$ attacks in $(r, t)$, and $A$ has the same local state at both $(r, t)$ and $(r', t')$, $A$ must also attack in $(r', t')$. Here we are using our crucial observation that an agent's or processor's actions can only depend on his local state. (Note that this intuitively says that a processor's actions can only depend on its knowledge.) Since protocol $P$ achieves coordinated attack, if $A$ attacks in $(r, t)$, then both generals attack at $(r', t')$; i.e. $p$ holds at $(r', t')$. Now suppose that $(r'', t'')$ is indistinguishable by $B$ from $(r', t')$. A similar argument shows that $B$ must attack in $(r'', t'')$, and hence both generals attack in $(r'', t'')$, and so $p$ holds at $(r'', t'')$ as well. An easy induction can be used to show that in fact $p$ holds at every point reachable from $(r, t)$. By Lemma 2.1, it follows that $p$ is common knowledge at $(r, t)$; i.e. $(R, r, t) \models Cp$. This is a formal statement of the fact that when the generals attack, it must be common knowledge that they are attacking. But $p$ is a fact that is initially undetermined. Thus, Theorem 4.1 implies that $p$ can never become common knowledge in $R$. We have now proved

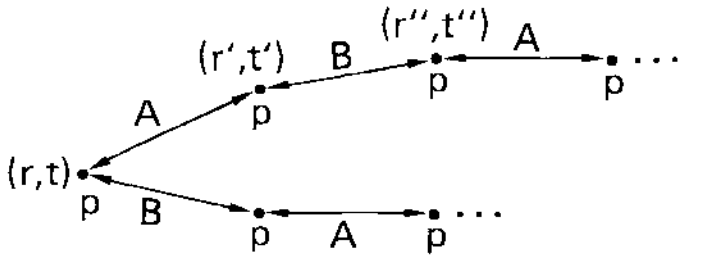


*Figure 2*

**Corollary 4.2** *Any protocol that guarantees that if one of the generals attacks then the other does so at the same time, is a protocol where necessarily neither general attacks.*

These results can be extended to show that not only is common knowledge not attainable in systems in which communication is not guaranteed, it is also not attainable in systems in which communication *is* guaranteed, as long as there is some uncertainty in message delivery time. Thus, in practical distributed systems, common knowledge is not attainable. This remark holds for systems involving interactions among people as well.

These results may seem paradoxical. We have seen that common knowledge is a prerequisite for coordinated action; similar arguments can be used to show that it is a prerequisite for agreement. We also showed that common knowledge is not attainable in practical distributed systems. Yet we often do seem to attain common knowledge, and tasks involving agreement and coordination certainly are carried out routinely in distributed systems.

This apparent paradox is examined carefully in Halpern & Moses (1984), where two approaches to resolving it are suggested. One approach is motivated by the observation that common knowledge *is* attainable in "idealized" models of reality where we assume, for example, that events can be guaranteed to happen simultaneously. There are times, both when carrying out a theoretical analysis of a situation and when considering what actions to perform, that it is useful to assume that this idealized model really does describe reality. Indeed, there are times that this assumption can never be contradicted by observations [see Halpern & Moses (1984) for details]. A second approach involves considering variants of common knowledge that are attainable under more reasonable assumptions. Such variants may prove a useful tool for specifying and analyzing different assumptions on communication in distributed systems.

Two variants that are of particular relevance in the analysis of the coordinated attack problem are *eventual common knowledge* and *likely common knowledge*. To understand how they are defined, it is best to first consider an alternative definition of common knowledge. $C\varphi$ can be viewed as the greatest fixpoint of the equation $X \equiv E(\varphi \wedge X)$. This equation says that the situation $X$ in which $C\varphi$ holds is exactly one where everyone knows both that $\varphi$ holds and that $X$ is the situation. Eventual common knowledge of $\varphi$, written $C^{\Diamond}\varphi$, is the greatest fixpoint of a slightly modified equation, $X \equiv \Diamond E(\varphi \wedge X)$, where $\Diamond\varphi$ says that $\varphi$ eventually holds [$\Diamond$ is the eventuality operator from temporal logic (cf Rescher & Urquhart 1971)]. Thus, $C^{\Diamond}\varphi$ implies that eventually everyone knows both that $\varphi$ holds and that eventual common knowledge of $\varphi$ holds. Eventual common

knowledge is attainable in systems in which messages are guaranteed to arrive eventually, even though there may be no upper bound on message delivery time. However, a result essentially analogous to Theorem 4.1 can be proved which says that in systems in which communication is not guaranteed, eventual common knowledge cannot be attained. From this result we can obtain a corollary analogous to Corollary 4.2, which says that the only protocol that guarantees that if one general attacks then the other general *eventually* attacks, is the protocol where necessarily neither general attacks.

Likely common knowledge of $\varphi$, written $C^L\varphi$, is the fixpoint of the equation $X \equiv LE(\varphi \wedge X)$, where $L\varphi$ says that $\varphi$ is likely to hold [$L$ is the likelihood operator from Halpern & Rabin (1983)]. Thus $\varphi$ is likely common knowledge if it is likely that everyone knows both that $\varphi$ holds and that $\varphi$ is likely common knowledge. Although eventual common knowledge cannot be attained if communication is not guaranteed, if any given message is likely to arrive, then likely common knowledge is attainable. Likely common knowledge and related notions of probabilistic common knowledge do seem to describe what is attained in practice in a situation such as the coordinated attack problem, and often are the prerequisites for action in such circumstances. The reader should refer to Halpern & Moses (1984) for more details on variants of common knowledge.

## 5. KNOWLEDGE IN ASYNCHRONOUS SYSTEMS[2]

In an asynchronous system, processors have no access to clocks. They can only keep track of events that have occurred, and have no way of measuring the time between events. As a result, knowledge in asynchronous systems has a number of additional properties. For simplicity in the following discussion, we restrict attention here to three types of events: a *send* (of a message by one processor to another), *receive* (of a message by one processor from another), or an *internal event*. Moreover, we assume that all events and messages are unique, and that time ranges over the natural numbers. At each time $n$ in a run, a (possibly empty) set of events occurs. Again for simplicity, we assume that there is no more than one event per unit of time. Processor $i$'s local state at the point $(r, n)$ is its history: its initial state, and the sequence consisting of the messages that $i$ sent and received and the internal events that took place locally, in the order they happened. Note that there is no "forgetting" here. As is already implicit

[2] The material in this section is taken from Chandy & Misra (1986), although it has been slightly modified to conform to the definitions used here.

in our intuition behind events, we assume that no event is in the history of more than one processor.

We restrict attention here to systems $R$ that can be characterized in the following way: For each processor $i$ there is a set $V_i$ of histories such that a run $r$ is in $R$ iff (1) at all times $n$ and for all processors $i$, we have that $i$'s local state at $(r,n)$ is in $V_i$, (2) for every receive event in $(r,n)$ of a message $\mathbf{m}$ by $i$ from $j$, there is a corresponding send event where $\mathbf{m}$ is sent to $i$ by $j$ that occurs earlier in the run, and (3) processor $i$'s history at $(r,0)$ is the empty sequence, and for all $n$, its history at $(r,n)$ is a (not necessarily strict) prefix of its history at $(r,n+1)$. Note that the first property can be viewed as a richness condition on the set of runs. It will allow us to create a new run by "pasting together" the histories of processors in a number of runs, as long as we can guarantee that the other properties are still satisfied. These properties are meant to capture (among other things) the intuition that time is meaningless in asynchronous systems. For example, suppose $r$ is a run in $R$ and $r'$ is the run in which the same event occurs at time $2n$ in $r'$ as occurs at time $n$ in $r$, and no event occurs at odd times in $r'$. It is easy to check that $r'$ has properties (1), (2), and (3) (since $r$ does), so $r'$ must also be in $R$. Similarly, any run that is like $r$ except that there are arbitrarily long "silent intervals" between the events of $r$ is also in $R$. It follows from these properties that an asynchronous system is one in which communication is not guaranteed; thus the results of the previous section on the unattainability of common knowledge apply immediately here.

Although time is meaningless in an asynchronous system, we can define a notion of *potential causality* between events, along the lines suggested in Lamport (1978). We want to capture the intuition that event $e$ may have caused event $e'$. For events $e$ and $e'$ in a run $r$, we write $e \rightarrow e'$ if either:

1. $e'$ is a receive and $e$ is the corresponding send,
2. for some processor $i$, events $e$ and $e'$ are both in $i$'s history at some point $(r,n)$ and $e$ precedes $e'$, or
3. for some event $e''$ we have $e \rightarrow e''$ and $e'' \rightarrow e'$.

A run $r$ has a *processor chain* $\langle i_1, \ldots, i_m \rangle$ if there exist events $e_1, \ldots, e_m$ in run $r$ such that event $e_j$ is in processor $i_j$'s history and $e_1 \rightarrow \cdots \rightarrow e_m$. Suppose that event $e_j$ takes place at time $n_j$ in run $r$. It is easy to see from the definition of $\rightarrow$ that if $i_j \neq i_{j+1}$, there must be a nonempty sequence of messages $\mathbf{m}_1, \ldots, \mathbf{m}_k$ in $r$ such that $\mathbf{m}_1$ is sent by $i_j$ at or after time $n_j$, $\mathbf{m}_{l+1}$ is sent by the processor that received $\mathbf{m}_l$ and is sent after $\mathbf{m}_l$ is received for $l = 1, \ldots, k-1$, and $\mathbf{m}_k$ is received by $i_{j+1}$ at or before $n_{j+1}$. In particular, this means that if there is a processor chain $\langle i_1, \ldots, i_m \rangle$ in $r$ with $i_j \neq i_{j+1}$ for $j = 1, \ldots, m-1$, then at least $m-1$ messages are sent in $r$.

We write $(r,n)[i_1,\ldots,i_m](r',n')$ if there exist points $(r_0,n_0),\ldots,(r_m,n_m)$ such that $(r,n)=(r_0,n_0)$, $(r',n')=(r_m,n_m)$, and for $j=1,\ldots,m$ we have $(r_{j-1},n_{j-1})\sim_{i_j}(r_j,n_j)$. Thus $(r,n)[i_1,\ldots,i_m](r',n')$ if at the point $(r,n)$ processor $i_1$ considers it possible that $i_2$ considers it possible... that $i_m$ considers it possible that $(r',n')$ is the case; i.e. $(r',n')$ is reachable from $(r,n)$ by a sequence of edges labeled $i_1,\ldots,i_m$.

There is a close relationship between processor chains and reachability, as shown in the following theorem. If $r$ is a run and $n<n'$, let $(r,n..n')$ denote the sequence of events in $r$ from $n+1$ to $n'$. The following theorem tells us that either $(r,n')$ is reachable from $(r,n)$ by a sequence of edges labeled $i_1,\ldots,i_m$, or there must be a causal sequence of events $e_1\rightarrow\cdots\rightarrow e_m$ such that $e_j$ is in processor $i_j$'s history.

**Lemma 5.1** *Let $r$ be a run and $n<n'$. Then either $(r,n)[i_1,\ldots,i_m](r,n')$ or there is a processor chain $\langle i_1,\ldots,i_m\rangle$ in $(r,n..n')$.*

Lemma 5.1 will allow us to relate message passing to knowledge in asynchronous systems. One direct consequence is stated in the following theorem, which essentially says that processors can only gain or lose knowledge by sending and receiving messages.

**Theorem 5.2** *Let $r$ be a run and $n<n'$.*

1. *If $(R,r,n)\models\sim K_{i_m}\varphi$ and $(R,r,n')\models K_{i_1}\ldots K_{i_m}\varphi$, then there is a processor chain $\langle i_m,\ldots,i_1\rangle$ in $(r,n..n')$.*
2. *If $(R,r,n)\models K_{i_1}\ldots K_{i_m}\varphi$ and $(R,r,n')\models\sim K_{i_m}\varphi$, then there is a processor chain $\langle i_1,\ldots,i_m\rangle$ in $(r,n..n')$.*

**Proof** We prove (2) here; the proof of (1) is similar. Suppose, in order to obtain a contradiction, that there is no processor chain $\langle i_1,\ldots,i_m\rangle$ in $(r,n..n')$. By Lemma 5.1, we have $(r,n)[i_1,\ldots,i_m](r,n')$. Thus, by definition, there exist points $(r_0,n_0),\ldots,(r_m,n_m)$ such that $(r,n)=(r_0,n_0)$, $(r,n')=(r_m,n_m)$, and for $j=1,\ldots,m$ we have $(r_{j-1},n_{j-1})\sim_{i_j}(r_j,n_j)$. We can now easily show, by a straightforward induction on $j$, that $(R,r_j,n_j)\models K_{i_j}\ldots K_{i_m}\varphi$ for $j=1,\ldots,m$. In particular, it follows that $(R,r,n')\models K_{i_m}\varphi$, a contradiction. ■

Using Theorem 5.2, we can prove a number of lower bounds on the number of messages required to solve certain problems. Consider the problem of *mutual exclusion*. Intuitively, the situation here is that from time to time a processor tries to access some shared resource that must only be accessed by one processor at a time. (For example, the processor may try to change the value of a shared variable.) We say the processor is in its *critical section* when it has access to the shared resource. A protocol solves the mutual exclusion problem if in every run of the protocol, no two processors are simultaneously in their critical sections. If $R$ is the set

of runs of a protocol that solves the mutual exclusion problem and $r$ is a run in $R$ in which $i_1, i_2, \ldots, i_m$ ($i_j \neq i_{j+1}$) enter their critical section in sequence, then it is easy to see that we must have a processor chain $\langle i_1, i_2, \ldots, i_m \rangle$ in $r$. For suppose that $i_j$ enters its critical section at time $n_j$ in $r, j = 1, \ldots, m$. Let $cs_j, j = 1, \ldots, m$, be primitive propositions denoting that $i_j$ is in its critical section. By the assumption that $R$ is the set of runs of a protocol solving the mutual exclusion problem, we must have that $cs_j \Rightarrow \sim cs_{j+1}$ is valid in $R$. Moreover, since the fact that $i_j$ is or is not in its critical section is determined by $i_j$'s local state, we must have that both $cs_j \Rightarrow K_{i_j}(cs_j)$ and $\sim cs_j \Rightarrow K_{i_j}(\sim cs_j)$ are valid in $R$. Since by assumption we have $(R, r, n_j) \models cs_j$, it follows from the previous observations and the general properties of knowledge discussed in Section 2 that

$$(R, r, n_j) \models K_{i_j} K_{i_{j+1}} \sim cs_{j+1} \quad \text{and} \quad (R, r, n_{j+1}) \models \sim K_{i_{j+1}} \sim cs_{j+1},$$

$$\text{for} \quad j = 1, \ldots, m-1.$$

Thus, by Theorem 5.2, there is a process chain $\langle i_j, i_{j+1} \rangle$ in $(r, n_j \ldots n_{j+1})$. It immediately follows that we have a processor chain $\langle i_1, i_2, \ldots, i_m \rangle$ in $r$. As we have already observed, the existence of this process chain in $r$ implies that at least $m-1$ messages are sent in $r$.

A protocol for *termination detection* runs "on top of" another protocol and detects that it has *terminated*, where we say a protocol has terminated at $(r, t)$ if no processor takes any more steps from that point on. Theorem 5.2 can also be used to show that a protocol for termination detection requires in general as many messages to do the detection as there are messages in the underlying computation. The reader is referred to Chandy & Misra (1986) for details of the proof.

## 6. SIMULTANEOUS BYZANTINE AGREEMENT

An important problem in distributed systems is that of reaching agreement in the presence of faults. This has been abstracted as the *Byzantine agreement* problem, which has been the focus of much study recently (see Fischer 1983). We consider here a variant of the Byzantine agreement problem called the *Simultaneous Byzantine Agreement* problem (SBA). Assume that each of the processors in a system starts out with a value, either 0 or 1. There may be some faulty processors in the system, although we do not know in advance which these are. The problem is to design a protocol that guarantees that all the nonfaulty processors agree on some value, again either 0 or 1, and do so at the same time. We assume for ease of exposition that we are working in a system in which communication proceeds in rounds and all messages sent in round $k$ are received by processors before

the beginning of round $k+1$. We further assume that the network is completely connected, so that each processor has a direct link to all other processors.

The formal problem statement is:

> Given $n$ processors, up to $f$ of which may be faulty, we want a protocol that guarantees that if each processor $i$ starts out with some initial value $x_i \in \{0, 1\}$, then
>
> 1. Each nonfaulty processor eventually "decides" on some value $y_i \in \{0, 1\}$.
> 2. The nonfaulty processors all decide on the same value ("agreement").
> 3. The nonfaulty processors all decide simultaneously (i.e. in the same round).
> 4. If all the initial bits $x_i$ are identical, all the nonfaulty processors decide $x_i$ ("validity").

The third clause is the one that guarantees simultaneous agreement. The fourth clause prevents the trivial solution where everyone always decides on 0 (or always decides on 1).

The problem is sensitive to the type of behavior we assume on the part of the faulty processors. The literature has concentrated on three failure modes:

1. *Crash failures*: a faulty processor may crash, after which it sends no messages.
2. *Omission failures*: a faulty processor may fail to send messages to some processors on any given round (but otherwise follows its protocol).
3. *Byzantine failures*: a faulty processor may exhibit arbitrary behavior. In particular, it may "lie," by sending messages that it was not supposed to send according to the protocol.

Note that crash failures are a special case of omission failures, where the faulty processor fails to send all messages to all processors after the round in which it first fails. Of course, omission failures are a special case of Byzantine failures.

There exist deterministic protocols that attain SBA for each of the three failure modes, although in the case of Byzantine failures, we require that $n \geq 3f+1$. [This is a tight bound. However, if we assume that processors can "sign" messages with "unforgeable signatures," there are protocols that attain SBA in the presence of arbitrarily many Byzantine failures. See Fischer (1983) or Strong & Dolev (1983) for details.] In all cases, the

known protocols require $f+1$ rounds to reach agreement in all runs (recall $f$ is an upper bound on the number of faulty processors). Moreover, in runs in which there are no faults, $f+1$ rounds are required to reach agreement, even if we consider only crash failures. Nevertheless, the question arises whether there are runs in which agreement can be reached in fewer than $f+1$ rounds. This question is investigated in Dwork & Moses (1986) for the case of crash failures and Moses & Tuttle (1986) for the case of omission failures. We now discuss these results.

An argument similar to that used in the case of coordinated attack can be used to show that when the nonfaulty processors decide on a value, it must be common knowledge among the nonfaulty processors that they are deciding on that value. However, we must be somewhat careful when defining the notion "it is common knowledge among the nonfaulty processors." The problem is that the set of nonfaulty processors is an *indexical* set: its members vary from point to point. We proceed as follows.

Let $N$ be the set of nonfaulty processors (formally, $N$ is a function from points to subsets of processors). We define $E_N\varphi$ to be $\bigwedge_{i\in N} K_i(i\in N \Rightarrow \varphi)$. Thus, $E_N\varphi$ says that every nonfaulty processor knows that if it is nonfaulty, then $\varphi$ holds.[3] We can now define $C_N\varphi$ as $E_N\varphi \wedge E_N E_N\varphi \wedge \ldots$. It is easy to check from these definitions that $C_N$ satisfies the fixpoint property:

$$C_N\varphi \equiv E_N(\varphi \wedge C_N\varphi).$$

Thus, a fact $\varphi$ is common knowledge among the nonfaulty processors if and only if every nonfaulty processor knows that if it is nonfaulty, then $\varphi$ holds and $\varphi$ is common knowledge among the nonfaulty processors.

It can now be shown:

**Theorem 6.1** *If $R$ is the set of runs of a protocol for SBA and a nonfaulty processor decides on the value $v$ at the point $(r, t)$, then*

1. $(R, r, t) \models C_N$ (all nonfaulty processors are deciding $v$) *and*
2. $(R, r, t) \models C_N$ (at least one processor had $v$ as its initial value).

Thus, if a nonfaulty processor decides $v$, then it must be common knowledge that all the nonfaulty processors decide $v$ at the same time, and that at least one of them had $v$ as an initial value.

This result establishes a tight relationship between SBA and common knowledge, which is exploited in the following protocol:

For round $l = 0, 1, 2, \ldots$ processor $i$ does the following:

[3] It turns out that this definition of $E_N\varphi$ is more useful than the more obvious $\bigwedge_{i\in N} K_i\varphi$. While these definitions are equivalent for nonindexical sets, they differ in situations where a nonfaulty processor does not know that it is nonfaulty. See Moses & Tuttle (1986) for further discussion.

**if** $K_i(i \in N \Rightarrow C_N$ (*some initial value was* 0))
**then** *decide* 0
**else if** $K_i(i \in N \Rightarrow C_N$ (*some initial value was* 1))
**then** *decide* 1
**else** *send complete history to every processor.*

This is an example of a *knowledge-based* protocol (Halpern & Fagin 1985): a processor's actions explicitly depend on tests for knowledge. A knowledge-based protocol may often be a very useful high-level way of describing the right actions to take in the case of SBA or other problems. Indeed, if we assume for now that this protocol terminates (so that the nonfaulty processors eventually attain common knowledge that some initial value was 0 or attain common knowledge that some initial value was 1), it is easy to see that it does attain SBA. When some nonfaulty processor knows that if it is not faulty then it is common knowledge among the nonfaulty processors that some initial value was 0 (respectively 1), then all the nonfaulty processors know this fact. Thus, all nonfaulty processors decide on the same value simultaneously.[4]

This protocol is in fact an *optimal* protocol for SBA in the case of crash failures and omission failures. In order to make this precise, for crash failures and omission failures, we define the *failure pattern* of a run to be a description of which processors fail, in what round they fail, and which processors they do not send messages to. (In the case of Byzantine failures, we can no longer characterize a processor's faulty behavior by simply describing which processors it fails to send to; faulty processors may send arbitrary messages at any round. This makes the analysis of Byzantine failures much more difficult.) Note that a run of a protocol is completely determined by the *initial configuration* (the initial states of the processors) and the failure pattern. A protocol $P$ is *optimal for SBA* if, for every initial configuration $C$ and failure pattern $F$, protocol $P$ achieves SBA in the run determined by $C$ and $F$ at least as soon as any other protocol $Q$ achieves SBA in the corresponding run of $Q$.

A *full-information* protocol is one where at each round, a processor sends its complete history to all the other processors. Note that the knowledge-based protocol described above is a full-information protocol. Intuitively, processors gain knowledge as quickly as possible using the full-information protocol. This intuition is made precise in Dwork & Moses (1986) and Moses & Tuttle (1986), where it is shown that in the case of crash failures and omission failures, full-information protocols are optimal

[4] As it stands, the protocol is biased towards 0, since if it is common knowledge both that some initial value was 0 and that some initial value was 1, the decision value will be 0. The protocol can easily be modified to be less biased.

for common knowledge; i.e. facts become common knowledge using a full-information protocol at least as soon as they become common knowledge using any other protocol. By putting this observation together with Theorem 6.1, we can see that out knowledge-based protocol is optimal for SBA, since the nonfaulty processors cannot decide on a value 0 (resp. 1) before it becomes common knowledge that some initial value was 0 (resp. 1), and the full-information protocol makes this fact common knowledge at least as soon as any other protocol.

Given that the knowledge-based protocol is optimal, we would like to convert it to a *standard* protocol by removing the tests for knowledge. It is not too hard to show that we can in fact do this. If we restrict our attention to crash failures or omission failures, there are only finitely many possible global states at any time, so that we can calculate the truth of a fact at any point simply by applying the definition of $\models$ given in Section 2. (Even if we allow Byzantine failures, we can again show that there are only finitely many "relevant" global states, so we can again compute what is common knowledge at a given point.)

Of course, we do not want to know only whether or not we can compute what is common knowledge. We would also like to know if this computation can be done *efficiently*. Moreover, we would like to characterize when facts become common knowledge. This will tell us, for example, how many rounds are required to reach SBA as a function of the failure pattern in the optimal protocol given above.

We first focus on the case of crash failures. Let $R$ be the set of runs defined by the full-information protocol in the case of crash failures. Let $F(r,k)$ be the number of processors that fail by round $k$ in run $r \in R$ and let $F'(r,k)$ be the maximum number of processors *implicitly* known by the nonfaulty processors to have failed by round $k$ in run $r$. Finally, define $W(r,l)$ to be the maximum of $F'(r,k)-k$ for $k \leq l$, and define $W(r)$, the *wastefulness* of run $r$, to be the maximum of $W(r,l)$ for $l \leq f+1$.

To understand these notions, consider a run $r$ in which no processor fails in round 1 and processors 1, 2, and 3 fail in round 2. Assume that the only processor to which processor 1 (resp. 2, 3) fails to send a message in round 2 is processor 4 (resp. 5, 1). Thus, processor 4 knows of processor 1's failure and processor 5 knows of processor 2's failure, but no nonfaulty processor knows of processor 3's failure. At the end of round 2, the nonfaulty processors implicitly know of two failures. Hence, $F(r,2) = 3$, but $F'(r,2) = 2$ and $W(r,2) = 0$.

Intuitively, we can imagine there is an "adversary" trying to prevent us from achieving Byzantine agreement. This adversary is initially provided with $f$ tokens (one for each fault). The adversary spends a token by making a processor faulty. As we shall see, the adversary's best strategy

is to spend as few tokens as possible, and in particular, not to spend more than $k$ tokens by round $k$. The wastefulness of a run measures how far off the optimal strategy the adversary has been in that run. The worst thing the adversary can do is make $f$ processors faulty in round 1. As soon as the nonfaulty processors discover the $f$ faults (which will happen by round 2 in the crash failure model), all their uncertainty about the run has been removed and they can quickly reach agreement. By limiting the number of faults, the adversary can increase the nonfaulty processors' uncertainty!

The following theorem characterizes when common knowledge of facts about the initial state (i.e. facts describing the processors' initial values) arises in terms of the wastefulness of the run. The key point is that if $l = f+1-W(r)$, then by round $l$ there must have been a *clean* round, one in which no processor fails. It turns out that the existence of a clean round is a prerequisite for common knowledge.

**Theorem 6.2** *Let $R$ consist of the runs of the full-information protocol in the case of crash failures. Initially undetermined facts about the initial state become common knowledge in run $r \in R$ at round $f+1-W(r)$ and no earlier. Moreover,*

1. *$l = f+1-W(r)$ iff $(R, r, l) \models C_N(l = f+1-W(\text{the current run}))$, and*
2. *If $l = f+1-W(r)$ and $\varphi$ is a fact about the initial state, then $(R, r, l) \models I_N\varphi$ iff $(R, r, l) \models C_N\varphi$.*

With this theorem in hand, it turns out we can eliminate the tests for common knowledge in the previous protocol as follows:

For round $l = 0, 1, 2, \ldots$, processor $i$ does the following:

**if** $K_i(l \geq f+1-W(\text{the current run}))$
**then if** $K_i(\textit{some initial value was } 0)$
**then** *decide* 0 **else** *decide* 1
**else** *send complete history to every processor.*

We must first show that this protocol attains SBA. Given a run $r$, suppose $l = f+1-W(r)$. From part 1 of Theorem 6.2, the fact that $l = f+1-W(r)$ will be common knowledge in round $l$ of run $r$, so that in particular every nonfaulty processor knows this fact. Thus all nonfaulty processors will decide (simultaneously) in round $l$ of run $r$. If some nonfaulty processor $i$ decides 0, then it must be because $i$ knew at round $l$ that some initial value was 0. Thus this fact is implicit knowledge among the nonfaulty processors at round $l$, and by part (2) of Theorem 6.2 (since it is a fact about the initial state), it is common knowledge among the nonfaulty processors. In particular, every nonfaulty processor knows that some initial value was 0, and so decides 0 at round $l$. Since we have already

shown that all processors decide at round $l$, it follows that they reach the same decision, and thus SBA is attained.

This is still a knowledge-based protocol, but now it is easy to get rid of the tests for knowledge. A processor knows that $l \geq f+1-W$(the current run) iff it knows at round $l$ that at some round $j \leq l$ there were at least $f+1-l+j$ faulty processors. It can obtain this information by observing which processors did not send it messages at round $j$ and by hearing from other processors about processors that did not send messages at round $j$. Similarly, it knows that some initial value was 0 iff either its own initial value was 0 or it received this information from some processor.

This protocol is an implementation of our original knowledge-based protocol, and it attains SBA in run $r$ in $f+1-W(r)$ rounds, which is optimal [see Dwork & Moses (1986) for more details]. Note that if $r$ is a run with no failures, then $W(r) = 0$, so it takes $f+1$ rounds to achieve agreement. On the other hand, if $r'$ is a run in which $f$ processors fail in round 1 and their failure is implicit knowledge among the nonfaulty processors in round 1, then $W(r') = f-1$, so this protocol attains SBA in 2 rounds in $r'$.

These results are extended in Moses & Tuttle (1986) to the case of omission failures. Our original knowledge-based protocol (with tests for common knowledge) is still optimal in this case. It is shown in Moses & Tuttle (1986) that we can again find a polynomial-time algorithm for computing what is common knowledge, and thus convert the knowledge-based protocol described above to an optimal standard protocol. We might also hope to extend these results to Byzantine failures. However, results of Moses & Tuttle (1986) suggest that it is unlikely that we will be able to do this. Define a *generalized omission failure* to be one where a processor may be faulty either because it fails to send a message or fails to receive one. Again it can be shown that our original knowledge-based protocol is optimal for SBA in the case of generalized omission failures, and a full-information protocol is optimal for attaining common knowledge. However, the following result is proved in Moses & Tuttle (1986):

**Theorem 6.3** *Let $R$ be the set of runs defined by the full-information protocol in the case of generalized omission failures. Computing whether an initially undetermined fact about the initial state is common knowledge at any given point in the network in NP-hard (in the size of the network).*

As a corollary to this theorem, it follows that NP-hard computations will be required in any optimal protocol for SBA in the case of generalized omission failures. Since generalized omission failures are a special case of Byzantine failures, this suggests that the same will be truc for Byzantine failures as well. (The result for Byzantine failures does not immediately

follow, since, for example, it may be that initially undetermined facts only become common knowledge at round $f+1$, at which time they are easy to compute.)

The analysis given here for SBA applies equally to other problems in which simultaneous actions are required. Such problems include the *firing squad* problem (Burns & Lynch 1985; Coan et al 1985), where processors must "fire" simultaneously at some point after the first nonfaulty processor receives a "start" message from an external source, and *weak Byzantine agreement* (see Fischer 1983), which is like Byzantine agreement except that agreement is only required if all the processors are nonfaulty. Since these problems also involve attaining common knowledge, lower bounds for attaining common knowledge immediately give lower bounds for the running time of protocols that solve these problems. Moreover, we can get optimal protocols for these problems along very much the same lines as the optimal protocol described for SBA [see Moses & Tuttle (1986) for details]. These observations show the generality and power of a knowledge-based approach to protocols.



## 7. TAKING COMPUTATION INTO ACCOUNT

As we observed in Section 2, our definition of knowledge does not take into account the difficulty in computing knowledge. In particular, it follows from our definition that agents know all logical consequences of their knowledge, and know all formulas valid in a given system. The analysis of Byzantine agreement done in the previous section shows both the strengths and weaknesses of this definition.

By reducing SBA to common knowledge we gain a deeper understanding of the problem and derive protocols that stop in an optimal number of rounds as a function of the failure pattern, at least in the case of crash failures and omission failures. However, computing common knowledge is likely to be very difficult in the case of generalized omission failures. While the knowledge-based protocol described in the previous section does give us some insight, it cannot be translated efficiently to a standard protocol in the case of generalized omission failures. Clearly, it would be useful to have a notion of knowledge that somehow took into account the difficulty of computation. Such a notion of knowledge might also be more appropriate for numerous other applications involving both human agents and computers.

Much effort has gone into dealing with the more general *logical omniscience* problem, and notions of knowledge have been defined for which an agent's knowledge is no longer closed under deduction and/or agents no longer know all valid formulas. Unfortunately, most of these

approaches do not directly deal with the problem of computing knowledge and none seems completely appropriate for the type of analysis required in distributed systems. We briefly review the major trends in this section.

One approach that has frequently been suggested is the syntactic approach: what an agent knows is simply represented by a set of formulas (cf Eberle 1974; Moore & Hendrix 1979). Of course, this set need not be constrained to be closed under logical consequence or to contain all instances of a given axiom scheme. While this does allow us to define a notion of knowledge that does not suffer from the logical omniscience problem, it is a notion that is difficult to analyze, and certainly does not approach the problem of computation at all.

A somewhat more sophisticated approach is taken by Konolige (1984), who considers starting with a set of base facts and then closing off under a (possibly incomplete) set of deduction rules. Similar ideas were recently proposed in Nguyen & Perry (1986), where, instead of starting off with a fixed set of base facts, the closure (under some possibly incomplete set of deduction rules) of the information that a processor has received via messages is considered.

The emphasis in these approaches is on limiting deduction in some way. The implicit assumption is that agents try to compute which formulas are valid (and may fail to do so due to limited resources). However, in many cases, a better paradigm than that of computing validity might be that of *model checking*: computing what formulas are true at a certain point in a given system. Techniques similar to those used in branching time temporal logic (Clarke et al 1986) can be used to show that the problem of checking whether a formula is true at a given point in the system is polynomial in the size of the system (i.e. the number of possible global states) and the size of the formula. Given a network of size $n$ and a protocol for all the processors, the number of possible global states can be very large and even infinite. In the case of the SBA protocols considered in the last section, it is not hard to show that in the case of crash failures, there are exponentially many ($\leq 2^{3n}$) possible global states at the end of round $k$. The analysis of Dwork & Moses (1986) shows that despite the exponential number of global states, it is still possible to determine whether a fact about the initial state is common knowledge in time polynomial in $n$. Moses & Tuttle (1986) show that this still holds true in the case of omission failures but fails in the case of generalized omission failures (unless P = NP). These observations suggest that a good theory of computational knowledge will have to have a large semantic component, rather than just taking a proof-theoretic approach and limiting deduction.

One semantic approach that has been taken is to augment the standard possible worlds by "impossible" worlds, where the customary rules of logic

do not hold (cf Cresswell 1973; Rantala 1982; Rescher & Urquhart 1971). Thus, in these models there may be *inconsistent* worlds, where a primitive proposition is both true and false, and *partial* worlds, where a primitive proposition may be neither true nor false. The semantics of conjunction, disjunction, and implication may also vary from the usual semantics of these operators in classical logic. In general, the construction of these models was not motivated by computational issues (in fact, the motivation was often only to find a model that satisfied a certain set of axioms).

One impossible-worlds model that is motivated by computational issues is that of Levesque (1984), who attempts to find a *tractable* (i.e. polynomial-time computable) notion of knowledge. He distinguishes between *explicit* and *implicit* knowledge, where explicit knowledge consists of those facts of which you are explicitly aware, while implicit knowledge consists, intuitively, of all the logical consequences of explicit knowledge. (This notion of implicit knowledge is different from the notion presented in Section 2, although they can be shown to be closely related.) Of course, an agent may not be aware of all his implicit knowledge. While implicit knowledge has the properties of logical omniscience, explicit knowledge does not.[5] As far as tractability goes, if we take $B\varphi$ to represent "the agent explicitly knows $\varphi$," then Levesque shows that for formulas of the form $B\varphi \Rightarrow B\psi$ where $\varphi$ and $\psi$ are propositional formulas in conjunctive normal form, the validity problem can be decided in polynomial time.

This result can still be viewed as describing a way of restricting the validity problem in order to make it tractable, so it does not seem applicable to the distributed systems problems we have been discussing. [The language is also restricted to the case of one agent and depth one formulas (so that formulas of the form $BB\varphi$ are not dealt with), which also limits its applicability to distributed systems problems.] Nevertheless, this particular type of tractability does suggest that the logic might be useful for providing semantics for knowledge-base queries. Suppose $\varphi$ is a description of (the facts known to) a knowledge base and $\psi$ is a query to the knowledge base; then $B\varphi \Rightarrow B\psi$ exactly if the knowledge base should answer yes to the query. It does not seem unreasonable to insist that $\varphi$ be in CNF; a knowledge base can be viewed as a collection (i.e. conjunction) of facts, each of which is a disjunction. While the query $\psi$ will in general not be in CNF, we would expect $\psi$ to be a small formula in comparison to $\varphi$, so that converting it to CNF (which might entail an exponential blowup in its size) would be relatively inexpensive.

We close this section with a discussion of the *logic of general awareness*

[5] Levesque actually considers belief rather than knowledge, but his results apply equally well to knowledge.

of Fagin & Halpern (1985). In this model, the standard Kripke structure for knowledge of Section 2 is augmented with an *awareness function*, which assigns to each agent at each possible world a set of formulas that the agent is "aware" of at that state. Implicit knowledge is defined just as knowledge was before. Explicit knowledge consists of implicit knowledge plus awareness. Thus an agent explicitly knows $\varphi$ in a given global state if (1) $\varphi$ is true at all the global states the agent considers possible and (2) $\varphi$ is in the agent's awareness set for that global state. If we take the formulas that a processor is aware of to be those that it can compute (at that global state) within some prespecified time or space bound, then explicit knowledge becomes a computational notion. Of course, this just pushes all the difficulty into the awareness function, but it does present a possibly useful framework for studying the problem.

## 8. RELATED WORK AND FUTURE DIRECTIONS

The idea of using knowledge to analyze distributed systems is fairly recent, but the area has seen a great deal of activity in the past two years. Many important open problems remain, some of which have already been mentioned in this survey. Some of these problems, and a brief account of the work already done on them, are listed below.

1. *Further analysis of protocols using knowledge*: In this survey we have seen how the coordinated attack problem and the simultaneous Byzantine agreement problem can be analyzed using knowledge. Recently, knowledge has also been used to analyze the atomic commit protocol (Hadzilacos 1987) and a family of data communication protocols (Halpern & Zuck 1987); in both cases the analysis seems to lead to a better understanding of the problem and the properties that any solution to it must have. Many other protocols should be amenable to this sort of analysis. It would be useful to have a larger body of examples on which we can test and develop our intuitions.
2. *Specifying, synthesizing, and verifying protocols*: Knowledge also seems to be a useful tool for specifying the properties a protocol should have. Once we specify a protocol's properties using the language of knowledge, we can try to synthesize a protocol having those properties. Temporal logic has already proved somewhat successful in this regard (cf Emerson & Clarke 1982; Manna & Wolper 1984). Preliminary work using knowledge in this way seems promising (cf Afrati et al 1986). Much work has already been done on developing proof techniques for verifying correctness of distributed protocols using temporal logic or other logics. It seems that many of these ideas can also be applied to

proving knowledge-based properties of programs, although Katz & Taubenfeld (1986) is the only work done thus far in this area.

3. *Programming with knowledge-based protocols*: As we have seen in the analysis of Byzantine agreement, knowledge-based protocols provide a high-level way to describe the relationship between knowledge and action. It may often be easier first to give a knowledge-based protocol that solves a problem, and then to convert it to a standard protocol. A more detailed understanding of the relationship between knowledge-based protocols and standard protocols would be useful. Ultimately we can imagine programming directly in a language that allows tests for knowledge, where the details of how that knowledge is computed are invisible to the programmer. Of course, we are currently a long way from that point.
4. *Resource-bounded reasoning*: We have already mentioned much of the work that has gone on in finding models of knowledge that capture the difficulty in computing knowledge. However, none of these models seems completely adequate; in particular, none of them gives a notion of knowledge appropriate for analyzing knowledge-based protocols that use only tests for computable knowledge. It would be particularly interesting to investigate the relationship between resource-bounded reasoning and the work of Goldwasser et al (1985), and others on *knowledge complexity*, which attempts to quantify the information released during an interaction in terms of computational complexity theory.
5. *Incorporating probability*: Dealing with probabilistic knowledge is clearly of interest, especially when analyzing randomized protocols. The work by economists on reasoning about knowledge incorporated probability right from the start (cf Aumann 1976). As was mentioned before, there is no difficulty in adding probability to the abstract model by viewing the set of possible worlds as a probability space. Can we usefully analyze randomized protocols by doing this?
6. *Properties of knowledge in distributed systems*: What are the intrinsic properties of our notion of knowledge in distributed systems? As was mentioned in Section 2, our definition of knowledge satisfies all the properties of the classical modal logic S5, but there may be some additional properties that arise when we restrict attention to distributed systems. Some work on this issue appears in Fagin & Vardi (1986) and Fagin et al (1986), where it is shown that the properties of knowledge depend heavily on assumptions we make about the distributed system, such as whether or not processors have unbounded memory. Precisely the same issues arise when we add time to the picture. The properties and the complexity of reasoning about knowledge and time may change

drastically depending on the assumptions we make about the system (see Halpern & Vardi 1986). It would be interesting to do a more thorough investigation of how properties of knowledge change with the communication medium. For example, one could try to characterize properties such as whether or not communication is guaranteed in terms of how the knowledge of processors in the system changes over time.

In summary, the recent progress in the field gives us reason to hope that reasoning about knowledge will prove to be an extremely useful tool in designing, analyzing, and understanding distributed systems. However, as the list above [and the list of problems given in Halpern (1986) in the general area of reasoning about knowledge] shows, there is much more work to be done.



## ACKNOWLEDGMENTS

Yoram Moses corrected some errors in a previous draft of this paper, and Moshe Vardi suggested the focus on computing knowledge rather than logical omniscience in Section 7. Yoram, Moshe, Cynthia Dwork, Ron Fagin, Vassos Hadzilacos, Jay Misra, and Ed Wimmers all made useful comments on previous drafts of this paper.

*Literature Cited*

Afrati, F., Papadimitriou, C. H., Papageorgiou, G. 1986. The synthesis of communication protocols. *Proc. Fifth Ann. ACM Symp. Principles of Distributed Computing*, pp. 263–71

Aho, A. V., Ullman, J. D., Yannakakis, M. 1979. Modelling communication protocols by automata. *Proc. 20th Ann. Symp. Foundations of Computer Science*, pp. 267–73

Aumann, R. J. 1976. Agreeing to disagree. *Ann. Stat.* 4(6): 1236–39

Burns, J., Lynch, N. A. 1985. *The Byzantine firing squad problem*, MIT Tech. Rep. MIT/LCS/TM-275

Chandy, M., Misra, J. 1986. How processes learn. *Distrib. Comput.* 1(1): 40–52; a preliminary version appears in *Proc. 4th ACM Symp. Principles of Distributed Computing*, 1985, pp. 204–14

Clark, H. H., Marshall, C. R. 1981. Definite reference and mutual knowledge. In *Elements of Discourse Understanding*, ed. A. K. Joshi, B. L. Webber, I. A. Sag. Cambridge: Cambridge Univ. Press

Clarke, E. M., Emerson, E. A., Sistla, A. P. 1983. Automatic verification of finite state concurrent programs: a practical approach. *Proc. 10th ACM Symp. Principles of Programming Languages*, pp. 117–26

Coan, B., Dolev, D., Dwork, C., Stockmeyer, L. 1985. The distributed firing squad problem. *Proc. 17th Ann. ACM Symp. Theory of Computing*, pp. 335–45

Cresswell, M. J. 1973. *Logics and Languages.* London: Methuen

Dwork, C., Moses, Y. 1986. Knowledge and common knowledge in a Byzantine environment I: crash failures. In *Theoretical Aspects of Reasoning About Knowledge: Proceedings of the 1986 Conference*, ed. J. Y. Halpern, pp. 149–70. Los Altos, Calif: Morgan Kaufmann. A revised version will appear in *Information and Control*

Eberle, R. A. 1974. A logic of believing, knowing and inferring. *Synthese* 26: 356–82

Emerson, E. A., Clarke, E. M. 1982. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.* 2: 241–66

Fagin, R., Halpern, J. Y. 1985. Belief, awareness, and limited reasoning. *Proc. Ninth Int. Joint Conf. Artif. Intell.*, pp. 491–501

Fagin, R., Halpern, J. Y., Vardi, M. Y.

1986. What can machines know? On the epistemic properties of machines. *Proc. AAAI-86*, pp. 428–34

Fagin, R., Vardi, M. Y. 1986. Knowledge and implicit knowledge in a distributed environment. In *Theoretical Aspects of Reasoning About Knowledge: Proceedings of the 1986 Conference*, ed. J. Y. Halpern, pp. 187–206. Los Altos, Calif: Morgan Kaufmann. A revised version will appear in *Artificial Intelligence*

Fischer, M. J. 1983. The consensus problem in unreliable distributed systems (a brief survey). Tech. Rep. 273, Yale Univ.

Fischer, M. J., Immerman, N. 1986. Foundations of knowledge for distributed systems. In *Theoretical Aspects of Reasoning About Knowledge: Proceedings of the 1986 Conference*, ed. J. Y. Halpern, pp. 171–85. Los Altos, Calif: Morgan Kaufmann

Goldwasser, S., Micali, S., Rackoff, C. 1985. The knowledge complexity of interactive proof-systems. *Proc. 17th Symp. Theory of Computing*, pp. 291–304

Gray, J. 1978. Notes on data base operating systems. In *Operating Systems: An Advanced Course. Springer Lect. Notes Comput. Sci. 60*, ed. R. Bayer, R. M. Graham, G. Seegmuller, pp. 393–481. New York: Springer-Verlag

Gray, J. 1979. A discussion of distributed systems. *Proc. Assoc. Ital. Calc. Automat.*, pp. 204–11

Hadzilacos, V. 1987. A knowledge-theoretic analysis of atomic commitment protocol. *Proc. Sixth ACM Symp. Principle of Database Systems*, pp. 129–34

Halpern, J. Y. 1986. Reasoning about knowledge: an overview. In *Theoretical Aspects of Reasoning About Knowledge: Proceedings of the 1986 Conference*, ed. J. Y. Halpern, pp. 1–17. Los Altos, Calif: Morgan Kaufmann

Halpern, J. Y., Fagin, R. 1985. A formal model of knowledge, action, and communication in distributed systems: preliminary report. *Proc. 4th ACM Symp. Principles of Distributed Computing*, pp. 224–36

Halpern, J. Y., Moses, Y. O. 1984. Knowledge and common knowledge in a distributed environment. *Proc. 3rd ACM Conf. Principles of Distributed Computing*, pp. 50–61. A revised version appears as *IBM Res. Rep. RJ4421*, 1986

Halpern, J. Y., Moses, Y. O. 1985. A guide to the modal logics of knowledge and belief: preliminary report. *Proc. Ninth Int. Joint Conf. Artif. Intell.*, pp. 480–90

Halpern, J. Y., Rabin, M. O. 1983. A logic to reason about likelihood. *Proc. Fifteenth Ann. ACM Symp. Theory of Computing*, pp. 310–19

Halpern, J. Y., Vardi, M. Y. 1986. The complexity of reasoning about knowledge and time. *Proc. Eighteenth Ann. ACM Symp. Theory of Computing*, pp. 304–15

Halpern, J. Y., Zuck, L. 1987. A little knowledge goes a long way: a simple knowledge-based analysis of a simple protocol. To appear in *Proc. Sixth Ann. ACM Symp. Principles of Distributed Computing*

Hintikka, J. 1962. *Knowledge and Belief*. Ithaca: Cornell Univ.

Katz, S., Taubenfeld, G. 1986. What processes know: definitions and proof methods. *Proc. Fifth Ann. ACM Symp. Principles of Distributed Computing*, pp. 249–62

Konolige, K. 1984. *Belief and Incompleteness*. SRI Artif. Intell. Note 319, SRI Int., Menlo Park, Calif.

Kripke, S. 1963. Semantical analysis of modal logic. *Z. Math. Logik Grundlagen Math.* 9: 67–96

Ladner, R. E. 1977. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comput.* 6(3): 467–80

Ladner, R. E., Reif, J. H. 1986. The logic of distributed protocols. In *Theoretical Aspects of Reasoning About Knowledge: Proceedings of the 1986 Conference*, ed. J. Y. Halpern, pp. 207–22. Los Altos, Calif: Morgan Kaufmann

Lamport, L. 1978. Time, clocks, and the orderings of events in a distributed system. *Commun. ACM* 21(7): 558–64

Lehmann, D. J. 1984. Knowledge, common knowledge, and related puzzles. *Proc. Third Ann. ACM Conf. Principles of Distributed Computing*, pp. 62–67

Levesque, H. J. 1984. A logic of implicit and explicit belief. *Proc. AAAI-84*, pp. 198–202. A revised version appears as FLAIR Tech. Rep. 32, 1984

Lewis, D. 1969. *Convention, A Philosophical Study*. Cambridge, Mass: Harvard Univ. Press

Manna, Z., Wolper, P. 1984. Synthesis of communicating processes from temporal logic specifications. *ACM Trans. Program. Lang. Syst.* 6(1): 68–93

McCarthy, J., Sato, M., Hayashi, T., Igarishi, S. 1978. On the model theory of knowledge. Stanford Comput. Sci. Dept. Rep. No. STAN-CS-78-657

Milgrom, P. 1981. An axiomatic characterization of common knowledge. *Econometrica* 49(1): 219–22

Moore, R. C., Hendrix, G. 1979. *Computational Models of Beliefs and the Semantics of Belief Sentences*. Tech. Note 187, SRI Int., Menlo Park, Calif.

Moses, Y. O. 1986. *Knowledge in a dis-*

*tributed environment*. PhD thesis. Stanford Univ.

Moses, Y. O., Tuttle, M. 1986. Programming simultaneous actions using common knowledge. *Proc. 27th Ann. Symp. Foundations of Computer Science*, pp. 208–21

Nguyen, V., Perry, K. 1986. Do we really know what knowledge is? IBM Res. Rep. RC 11830

Parikh, R., Ramanujam, R. 1985. Distributed processing and the logic of knowledge. *Proc. Brooklyn College Work. Logics of Programs*, ed. R. Parikh, pp. 256–68

Pease, M., Shostak, R., Lamport, L. 1980. Reaching agreement in the presence of faults. *J. ACM* 27(2): 228–34

Perrault, C. R., Cohen, P. R. 1981. It's for your own good: a note on inaccurate reference. In *Elements of Discourse Understanding*, ed. A. K. Joshi, B. L. Webber, I. A. Sag. NY: Cambridge Univ. Press

Rantala, V. 1982. Impossible worlds semantics and logical omniscience. *Acta Philos. Fenn.* 35: 106–15

Rescher, N., Brandom, R. 1979. *The Logic of Inconsistency*. Totowa, NJ: Rowman and Littlefield

Rescher, N., Urquhart, A. 1971. *Temporal Logic*. NY: Springer-Verlag

Rosenschein, S. J., Kaelbling, L. P. 1986. The synthesis of digital machines with provable epistemic properties. In *Theoretical Aspects of Reasoning About Knowledge: Proceedings of the 1986 Conference*, ed. J. Y. Halpern, pp. 83–98. Los Altos, Calif: Morgan Kaufmann

Strong, H. R., Dolev, D. 1983. Byzantine agreement. *COMPCON83: Digest of Papers*, pp. 77–82. San Francisco: IEEE

*Annual Review of Computer Science*
*Volume 2, 1987*

# CONTENTS