KNOWLEDGE CONSISTENCY: A USEFUL SUSPENSION OF DISBELIEF*

(preliminary report)

Gil Neiger
Department of Computer Science
Cornell University
Ithaca, New York 14853

ABSTRACT

The study of knowledge is of great use in distributed computer systems. It has led to better understanding of existing algorithms for such systems, as well as the development of new *knowledge-based* algorithms. The ability to achieve certain states of knowledge (e.g., common knowledge) provides a powerful tool for designing such algorithms. Unfortunately, it has been shown that for many systems it is impossible to achieve these states of knowledge. In this paper we consider alternative interpretations of knowledge under which these states can be achieved. We explore the notion of *consistent interpretations*, and show how they can be used to circumvent the known impossibility results in a number of cases. This may lead to greater applicability of knowledge-based algorithms.

*Partial support for this work was provided by the National Science Foundation under grant DCR86-01864.

1 INTRODUCTION

The study of knowledge in distributed computer systems is a rapidly growing field of research. It has been shown to be of great use in understanding existing systems, and in the development of new algorithms for these systems. In this paper we seek to expand the applicability of knowledge in distributed systems by examining notions of *knowledge consistency.*

Knowledge in distributed systems was first explored by Halpern and Moses [HM87]. They formalized the notion of ascribing knowledge to individual processors in such systems, and based upon this they defined a hierarchy of states of knowledge that a set of processors may possess. The highest of these is *common knowledge.* Intuitively, a fact is common knowledge if everyone knows it, and everyone knows that everyone knows it, and so on *ad infinitum.* They showed that in many systems of practical interest common knowledge cannot be achieved. Following this, they defined several modifications (weakenings) of common knowledge that *can* be achieved in practical systems. It was argued that in certain cases these weakenings of common knowledge might adequately substitute for true common knowledge.

Halpern and Fagin explored how knowledge can be used to determine actions taken by processors in distributed systems, and introduced *knowledge-based algorithms* [HF85, HF87]. These are algorithms that consider a processor's knowledge when determining what action it will take next. The ability to specify a processor's actions with a knowledge-based algorithm simplifies the development of solutions to many problems in distributed systems. Nevertheless, it is not always clear how to implement such algorithms.

Several researchers have examined the relationship between knowledge and the solutions to specific problems in distributed systems. Dwork and Moses showed that it is necessary to attain common knowledge in order to attain *Simultaneous Byzantine Agreement* [DM86]; Moses and Tuttle did the same for general simultaneous actions [MT86]. Halpern and Zuck showed that solutions to the *Sequence Transmission* problem require a specific level of knowledge (short of common knowledge) [HZ87]. Each of these arguments included a demonstration of a knowledge-based algorithm to solve the problem being considered. These results indicate that an understanding of the interaction between knowledge and action (through knowledge-based algorithms) may facilitate the solution and understanding of important problems in distributed systems.

The impossibility of achieving common knowledge in many distributed systems appears to limit its utility when developing algorithms for these systems. In spite of this, Neiger and Toueg showed that solutions to certain problems may be developed for systems in which common knowledge *can* be achieved, and then used correctly in systems in which common knowledge cannot be achieved [NT87]. They showed this by "interpreting" knowledge in a non-standard manner that did not alter the correctness of the algorithms under consideration. Processors can detect no inconsistency between this

interpretation and a "standard" one.

This is an application of what Halpern and Moses termed *internal knowledge consistency* [HM87]. In this paper we explore the notion of knowledge consistency more formally. We focus on the relationship between knowledge and action, and how this relationship may be preserved under different interpretations of processor knowledge. We consider applications of these ideas to different distributed systems.

In Section 2 we define distributed systems and their executions. In Section 3 we formally define processor knowledge. In Section 4 we discuss notions of knowledge consistency, and in Section 5 how these notions may be applied to distributed systems. Section 6 discusses future work and Section 7 contains conclusions.

2 Definitions, Assumptions, and Notation

In this section we define distributed systems and their executions.

2.1 Distributed Systems

We define a distributed system to be a set of processors connected by a message-passing medium. Let P be the set of processors.

The configuration of a distributed system is described by the *system state*; let $\mathcal{S}$ be the set of system states. This includes the configurations of the individual processors and that of the message-passing medium. Each processor perceives some part of this system state; this is its *view*; let $\mathcal{V}$ be the set of processor views. If the system is in state s then we denote processor p's view of this state by $v(p, s)$. A processor's view may or may not include the value of a local *clock*, whose relationship to real time is unspecified.

2.2 Events and Actions

Events that occur in the course of an execution change the system's state. Some events are the direct results of *actions* taken by the processors. Others are not explicitly controlled by the processors; these include the incrementing of a local clock, the delivery of a message by the message-passing medium, and low-level events that implement processor actions. Other events are the direct results of *actions* undertaken by the processors. Let $\mathcal{A}$ be the set of actions executable by processors. We assume that a processor has an opportunity to execute an action whenever its view changes. If a processor takes no action after a change of view then it executes the "null action", $a_\perp \in \mathcal{A}$. We also assume that one can determine from a processor's view the sequence of actions it took in this execution.

2.3. HISTORIES

A specific execution of a system is described by a *history*. A history consists of two functions that describe the execution with respect to real time.[1] The first is a *state history function*, S, which describes the states through which the system passes; $\mathrm{S}{:}\mathbf{N} \mapsto S$. The second is an *action history function*, A, which specifies the actions taken by processors; $\mathrm{A}{:}P \times \mathbf{N} \mapsto \mathcal{A}$; this is a partial function, since processors do not execute actions at every instant of real time. Note that if $\mathrm{S}(t) = s$ and $\mathrm{A}(p,t) = a$ then the system was in state s at time t and *then* processor p executed action a. Let global history $\mathrm{H} = \langle \mathrm{S}, \mathrm{A} \rangle$. The state changes experienced by the system (according to S) are compatible with the actions undertaken by the processors (according to A); for example, if $\mathrm{A}(p,t) =$ "send message m to q" then $\mathrm{S}(t+1)$ should reflect that the message has been sent.

Two histories H_1 and H_2 are *view-equivalent* (we write $\mathrm{H}_1 \| \mathrm{H}_2$) if each processor passes through the same sequence of views in H_1 that it does in H_2. If $\mathrm{H}_1 \| \mathrm{H}_2$ then no processor can distinguish H_1 from H_2.

A distributed system is identified by the set of histories that correspond to all executions of the system. We use capital letters to denote such sets. We usually use A to refer to the actual system being run.

A *point* is a pair $\langle \mathrm{H}, t \rangle$ that refers to the status of history H at time t. We say that for $p \in P$ points $\langle \mathrm{H}, t \rangle$ and $\langle \mathrm{H}', t' \rangle$ are *p-equivalent* if $v(p, \mathrm{S}(t)) = v(p, \mathrm{S}'(t'))$. In this case we write $\langle \mathrm{H}, t \rangle \sim_p \langle \mathrm{H}', t' \rangle$.

2.4 PROTOCOLS

Processor p runs a *local protocol* Π_p. Given the current view of processor p, Π_p specifies the next event (if any) to be executed by p; $\Pi_p{:}\mathcal{V} \mapsto \mathcal{A}$. A sequence of local protocols, $\Pi = (\Pi_p \mid p \in P)$, is a *global protocol*, or simply a *protocol*.

History H is an *execution of protocol* Π if the events executed by the processors in H are exactly those specified by Π, given the views through which the processors pass. That is,

$$\forall p \in P \, \forall t \in \mathbf{N} [\mathrm{A}(p,t) \text{ defined} \Rightarrow \mathrm{A}(p,t) = \Pi_p(v(p, \mathrm{S}(t)))].$$

Note that if $\mathrm{H}_1 \| \mathrm{H}_2$, and H_1 is an execution of Π, then so is H_2. This is because one can determine from a processor's view the sequence of actions it has taken.

2.5 PROBLEM SPECIFICATIONS

A distributed systems problem is specified by a predicate on histories. This is the problem's *specification*. For example, the *serializability* problem in distributed databases is specified by a predicate that is satisfied exactly by those histories of the database in

[1] We consider t, a real time, to be an element of N, the set of natural numbers.

which transactions are serializable. Protocol Π solves a problem with specification Σ in system S if whenever processors run Π in S, Σ is satisfied. Formally, Π *satisfies* Σ *in* S if every execution of Π in S satisfies Σ.

Many problems in distributed systems may be specified by predicates that only depend on the sequence of views through which the processors pass. Specification Σ is *view-based* if

$$\forall \mathrm{H}_1, \mathrm{H}_2[\mathrm{H}_1 \| \mathrm{H}_2 \Rightarrow (\Sigma(\mathrm{H}_1) \Leftrightarrow \Sigma(\mathrm{H}_2))];$$

that is, any two view-equivalent histories either both satisfy or both fail to satisfy Σ.[2] Many problems of interest in distributed systems have view-based specifications. Examples include transaction serializability, deadlock prevention, and stable-state detection.

3 Knowledge in Distributed Systems

In this section we discuss ways that knowledge is ascribed to processors in distributed systems. We consider different interpretations for such knowledge, and how such interpretations are used in knowledge-based protocols.

3.1 Definitions

In this section we formally ascribe knowledge to processors in a distributed system. To do so we give the language of a logical system to express such knowledge.

We assume that we have a language for expressing certain facts about the system, without referring to the knowledge of processors. These are *ground facts*, and express properties that may or may not be true of specific points in executions of the system. With each ground fact φ we associate a set of points, $\pi(\varphi)$, for which the fact is true. Ground facts are those such as "processor p sent message m to processor q", "the value of processor q's variable x is 5", and "processor r has halted". The truth of a ground fact is independent of the system being run, and depends only upon the the point in the execution.

Halpern and Moses introduced modality operators K_p (one for each $p \in P$) to extend a language of ground facts [HM87]; $\mathsf{K}_p\varphi$ denotes that processor p "knows" fact φ. How these knowledge operators are interpreted is a major concern of this paper. In general, however, we want such an interpretation to have properties that facilitate the design of knowledge-based protocols (see Section 3.3 below). Ideally, knowledge operators have the properties of the modal logic S5 [HM85]:

A1: the *knowledge axiom*: $\mathsf{K}_p\varphi \Rightarrow \varphi$;

A2: the *consequence closure axiom*: $\mathsf{K}_p\varphi \wedge \mathsf{K}_p(\varphi \Rightarrow \psi) \Rightarrow \mathsf{K}_p\psi$;

A3: the *positive introspection axiom*: $\mathsf{K}_p\varphi \Rightarrow \mathsf{K}_p\mathsf{K}_p\varphi$;

[2] These are similar to the logical specifications of Neiger and Toueg [NT87].

A4: the *negative introspection axiom*: $\neg \mathsf{K}_p\varphi \Rightarrow \mathsf{K}_p\neg \mathsf{K}_p\varphi$; and

R1: the *rule of necessitation*: if φ is valid then $\mathsf{K}_p\varphi$ is valid.

Higher levels of knowledge are defined based on the operators K_p. $\mathsf{E}\varphi$ (everyone knows φ) is $\bigwedge_{p\in P} \mathsf{K}_p\varphi$. If $\mathsf{E}^1\varphi \equiv \mathsf{E}\varphi$ and $\mathsf{E}^{m+1}\varphi \equiv \mathsf{E}(\mathsf{E}^m\varphi)$ then $\mathsf{C}\varphi$ (φ is *common knowledge*) is equivalent to $\bigwedge_{n\geq 1} \mathsf{E}^n\varphi$.[3]

3.2 Knowledge Interpretations

In this section we consider different interpretations of the knowledge operators K_p. A knowledge interpretation $\mathcal{I}$ is a function from points to truth valuations on formulas. $(\mathcal{I}, \mathrm{H}, t) \models \varphi$ indicates that φ holds at point $\langle \mathrm{H}, t\rangle$ under interpretation $\mathcal{I}$. We only consider interpretations $\mathcal{I}$ that satisfy the following:

1. if φ is a ground fact then $(\mathcal{I}, \mathrm{H}, t) \models \varphi$ if and only if $\langle \mathrm{H}, t\rangle \in \pi(\varphi)$;
2. $(\mathcal{I}, \mathrm{H}, t) \models \varphi \wedge \psi$ if and only if $(\mathcal{I}, \mathrm{H}, t) \models \varphi$ and $(\mathcal{I}, \mathrm{H}, t) \models \psi$;
3. $(\mathcal{I}, \mathrm{H}, t) \models \neg\varphi$ if and only if $(\mathcal{I}, \mathrm{H}, t) \not\models \varphi$; and
4. if $\langle \mathrm{H}, t\rangle \sim_p \langle \mathrm{H}', t'\rangle$ and $(\mathcal{I}, \mathrm{H}, t) \models \mathsf{K}_p\varphi$ then $(\mathcal{I}, \mathrm{H}', t') \models \mathsf{K}_p\varphi$.

(1) to (3) ensure that $\mathcal{I}$ is consistent with the meaning of ground facts, $\wedge$, and $\neg$. (4) states that a processor's knowledge must always be a function of its view. Thus we sometimes consider a knowledge interpretation as a function from views to a "knowledge valuation" of formulas.

We define *system-based interpretations* of the knowledge operators. If I is a system, let $\mathcal{I}_I$ be the knowledge interpretation based on system I (I need not equal A, the system being run). $\mathcal{I}_I$ is defined inductively on the structure of formulas: ground facts, conjunctions, and negations are given by (1) to (3) above. The operators K_p are interpreted as follows:

- $(\mathcal{I}_I, \mathrm{H}, t) \models \mathsf{K}_p\varphi$ if and only if

$$\forall \langle \mathrm{H}', t'\rangle \in I \times \mathbf{N} [\langle \mathrm{H}, t\rangle \sim_p \langle \mathrm{H}', t'\rangle \Rightarrow (\mathcal{I}_I, \mathrm{H}', t') \models \varphi].$$

We can extend this interpretation to the common knowledge operator C using the relations $\sim_p$, but this is beyond the extent of this paper.

Lemma 1: *Let A and I be two systems. Consider knowledge interpretation $\mathcal{I}_I$ when used in system A. Then*

[3] Halpern and Moses consider $\mathsf{C}\varphi$ to be the greatest solution to the fixed point equation $X \equiv \mathsf{E}(\varphi \wedge X)$ [HM87].

1. *the operators* K_p *satisfy axioms A2, A3, and A4 of S5;*
2. *if* $I \subseteq A$ *then rule R1 holds;*
3. *if* $I = A$ *then axiom A1 holds.*

The proof of Lemma 1 uses standard techniques and is omitted.

Axiom A1 may not be satisfied if $I \subset A$. For example, let A be some arbitrary distributed system, with no restrictions on the accuracy or synchronization of local clocks. Let I be a similar system, but in which clocks are always perfectly synchronized (note that $I \subseteq A$). Suppose that system A is being run, and let $\langle \mathrm{H}, t\rangle \in A \times \mathbf{N}$ be a point at which processor p's clock shows 5. If $\varphi =$ "processor q's clock shows 5" then $(\mathcal{I}_I, \mathrm{H}, t) \models \mathsf{K}_p\varphi$. This is because we are using system I, where clocks are synchronized, to interpret the operator K_p. Note that it may even be the case that q's clock does *not* show 5 at $\langle \mathrm{H}, t\rangle$, even though p "knows" that it does. This may be true if $\mathrm{H} \in A - I$.

3.3 Knowledge-Based Protocols

Halpern and Fagin introduced "knowledge-based protocols" for distributed systems [HF85,HF87]. We adapt their work to our definitions.

In Section 2.4 we defined protocols to map views to actions. These are what Halpern and Fagin call "simple protocols". A *knowledge-based protocol* also uses a processor's knowledge to determine the next event to be executed. Such protocols are written in an extended language, which allows the use of the knowledge operators K_p, E, and C to specify tests on the global state.

A processor's knowledge is the set of facts that it "knows" according to a chosen interpretation. Recall that by property (4) of knowledge interpretations the facts that a processor knows are dependent only upon its view and the chosen interpretation. Thus a knowledge-based protocol Π_p is a function that, given a view and a knowledge interpretation, determines the next event to be executed by p.

History H is an *$\mathcal{I}$-execution of knowledge-based protocol* Π if processors execute exactly the events specified by protocol Π, using knowledge interpretation $\mathcal{I}$. That is,

$$\forall p \in P\, \forall t \in \mathbf{N}[\mathrm{A}(p,t) \text{ defined} \Rightarrow \mathrm{A}(p,t) = \Pi_p(v(p, \mathrm{S}(t)), \mathcal{I})].$$

Note that if $\mathrm{H}_1 \| \mathrm{H}_2$, and H_1 is a $\mathcal{I}$-execution of Π, then so is H_2. This because one can determine from a processor's view the sequence of actions it has taken. If all $\mathcal{I}$-executions of Π in a system S satisfy specification Σ then we say that Π *$\mathcal{I}$-satisfies* Σ *in* S.

It may seem natural to execute a knowledge-based protocol using a knowledge interpretation based upon the system being run. There are occasions, however, when knowledge may be based upon other systems. These are discussed in more detail below.

4 Knowledge Consistency

Halpern and Moses defined a notion of "internal knowledge consistency" [HM87]. In this section we formalize various notions of knowledge consistency.

4.1 Two Definitions

Knowledge consistency is, in a sense, a measure of the appropriateness of a particular knowledge interpretation when running in a certain system. A knowledge interpretation is consistent for a system if the processors running a knowledge-based protocol in that system can never determine that the interpretation is not correct.

Specifically, we consider system-based interpretations as defined above. By Lemma 1, all such interpretations satisfy all of S5 with the exception of A1, the knowledge axiom ($K_p\varphi \Rightarrow \varphi$), and R1, the rule of necessitation. An interpretation is consistent with a system if R1 holds and processors in the system can never detect that the knowledge axiom does not hold.

We view knowledge consistency as a relation between two systems. One of these, A, is the system actually being run. The other, I, is the "ideal" system, used to interpret the knowledge operators. We consider the consistency of interpretation $\mathcal{I}_I$ with system A. We restrict ourselves to cases in which $I \subseteq A$; thus R1 holds (see Section 3.2).

$\mathcal{I}_I$ is *knowledge-consistent with system A* if the following holds:

$$\forall \langle \mathrm{H}, t\rangle \in A \times \mathbf{N}\, \forall p \in P\, \exists \langle \mathrm{H}_p, t_p\rangle \in I \times \mathbf{N} [\langle \mathrm{H}, t\rangle \sim_p \langle \mathrm{H}_p, t_p\rangle]$$

This is similar to the internally knowledge-consistent interpretations of Halpern and Moses [HM87].

Recall that when using $\mathcal{I}_I$ the knowledge axiom holds in system I. If $\mathcal{I}_I$ is knowledge-consistent with system A then a processor p running in A can never detect that it is not running in I (there is always a p-equivalent point in $I \times \mathbf{N}$). Thus no processor running in A can ever detect that the knowledge axiom (using $\mathcal{I}_I$) does not hold.

Note that for any point $\langle \mathrm{H}, t\rangle \in A \times \mathbf{N}$ each processor $p \in P$ may have a different point $\langle \mathrm{H}_p, t_p\rangle$ that it considers equivalent; there is no guarantee that $\mathrm{H}_p = \mathrm{H}_q$ for distinct p and q. Furthermore, if the definition gives $\langle \mathrm{H}, t_1\rangle \sim_p \langle \mathrm{H}_1, t_1'\rangle$ and $\langle \mathrm{H}, t_2\rangle \sim_p \langle \mathrm{H}_2, t_2'\rangle$, there is no guarantee that $\mathrm{H}_1 = \mathrm{H}_2$. The histories in I that p considers equivalent may change with time. There are cases when we want a stricter notion of knowledge consistency.

$\mathcal{I}_I$ is *uniformly knowledge-consistent with system A* if

$$\forall \mathrm{H} \in A\, \exists \mathrm{H}' \in I\, [\mathrm{H} \| \mathrm{H}'].$$

This is definition is strictly stronger than the previous. Thus our remarks regarding the knowledge axiom apply to it as well. This definition has the advantage of ensuring that there is some view-equivalent history that all processors consider "possible".

Consider the following example, drawn from the world of replicated databases. Let I be a system in which all transactions are performed atomically, and in the same order at each processor. Let A be a system that ensures *one-copy serializability* [BG86,BHG87]. (This essentially means that the result of running a set of transactions in A is always the same as would result if they had been run in the serial order at all sites.) If we assume that a processor's view includes only the transactions that it has initiated and any results returned by them from the database, then $\mathcal{I}_I$ is uniformly knowledge-consistent with A.

4.2 Using Knowledge Consistency

Knowledge-consistent interpretations allow us to develop knowledge-based protocols that are more powerful in the sense that they may make use of more knowledge. If system I is a subset of system A then, using $\mathcal{I}_I$ as a knowledge interpretation, a processor may "know" facts that it would not if it used interpretation $\mathcal{I}_A$.

For example, let A be a system with asynchronous message passing and let I be a system in which messages are delivered one second after they are sent. Suppose that $\mathcal{I}_I$ is knowledge-consistent with A. By using $\mathcal{I}_I$ processors can "know" facts that they would not if using $\mathcal{I}_A$: using $\mathcal{I}_I$ a processor knows, one second after sending a message, that it has been delivered; this is not true when using $\mathcal{I}_A$. Since $\mathcal{I}_I$ is knowledge-consistent with A the processor can never detect that its message was not promptly delivered.

Knowledge consistency guarantees us that at no point in any history can any processor detect an inconsistency. It may also be important that all actions of all processors, when considered together, solve a desired problem. The correctness of a knowledge-based protocol depends upon facts that cannot be discerned by any single processor at a given time; specifically, it depends upon the states through which the system passes in an execution of the protocol. For problems with view-based specifications this is addressed by considering uniform knowledge consistency:

Theorem 2: *Let I and A be two systems ($I \subseteq A$) such that $\mathcal{I}_I$ is uniformly knowledge-consistent with A and let Σ be a view-based specification. If knowledge-based protocol Π $\mathcal{I}_I$-satisfies Σ in I then Π also $\mathcal{I}_I$-satisfies Σ in A.*

Proof: Let $\mathrm{H} \in A$ be an $\mathcal{I}_I$-execution of Π. Since $\mathcal{I}_I$ is uniformly knowledge-consistent with A, there is an $\mathrm{H}' \in I$ such that $\mathrm{H}' \| \mathrm{H}$. H' is also an $\mathcal{I}_I$-execution of Π (see Section 3.3). Since Π $\mathcal{I}_I$-satisfies Σ in I, history H' satisfies Σ. Σ is view-based, so history H also satisfies Σ. Since H was chosen arbitrarily, Π must $\mathcal{I}_I$-satisfy Σ in A. □

Theorem 2 indicates that uniformly knowledge-consistent interpretations can be used when developing solutions to problems with view-based specifications. In fact, the programmer need only prove the program to be correct in the ideal system; Theorem 2 guarantees its correctness the actual system. Ideal systems generally provide a pro-

grammer with additional properties that simplify programming. Thus the use of uniform knowledge consistency can simplify the derivation of solutions to problems with view-based specifications.

Consider again our example of replicated databases from Section 4.1, where I is a system in which transactions are truly serial, and A, where transactions are serializable. In general, the correctness of a replicated database depends only on the views of the database that processors gain by executing transactions; this can be given with a view-based specification. Thus a replicated database written for system I (using knowledge-based protocols) will run correctly in system A, if $\mathcal{I}_I$ is used as the knowledge-interpretation. Since it is much simpler to program a replicated database system in which transactions are executed serially, this simplifies the design of such systems.

5 Applications of Knowledge Consistency

In this section we consider cases in which knowledge consistency can be applied in distributed systems.

5.1 Granularity of Perception

Often the granularity at which changes in a system occur may be different from that at which processors perceive these changes. For example, one action, as executed by a processor, may be implemented by several distinct events, each one of which changes the system state. If a processor cannot observe the system between these implementing events, then its view changes only after the last of these events. In such cases one may want to use an interpretation based upon a system in which each action is one atomic event.

Fischer and Immerman [FI86] consider two distributed systems.[4] In one of these, C (*coarse*), one processor action consists of sending and receiving messages to and from all other processors in the system, as well as changing the processor's local configuration. Processors execute these once per "tick" on a global clock. Their second system, F (*fine*), considers a finer granularity of execution. Each sending or receipt of a message is a separate action. Within this system, however, execution proceeds in *rounds*; that is, a processor does not send or receive messages pertaining to one round until it has received all messages from the previous round. Fischer and Immerman observe that in C it is relatively easy to obtain common knowledge. This is because of guarantees provided by the system: for example, after executing an event a processor *knows* that the messages it just sent have been received. No such guarantees are provided by F, and because of this common knowledge cannot be attained therein.

[4]The two systems described here correspond to their "protocols" $\mathcal{B}$ and $\mathcal{A}$.

Fischer and Immerman argue that these two systems are, in a sense, isomorphic, and that it is therefore counterintuitive that common knowledge can be achieved in one and not in the other. This problem is easily circumvented by considering consistent knowledge interpretations. Because of the nature of the protocols considered, one can show that interpretation $\mathcal{I}_C$ is uniformly knowledge-consistent with system F. Therefore one may write knowledge-based protocols for C and then run them in F, evaluating knowledge as if in C. Thus such protocols may operate as if common knowledge can indeed be achieved, and are still correct when used to solve problems with view-based specifications.[5]

5.2 Simulation of Restricted Systems

Often the system upon which a knowledge interpretation is based is not simply an "isomorphic" version of the system being run. Sometimes it is a system with different (usually more useful) properties. One may want to simulate an execution of such an "ideal" system while running a more "realistic" one.

Neiger and Toueg considered such cases for asynchronous systems [NT87]. *Asynchronous* systems are those for which there is no known bound on the delays that messages might experience while in transit. R was defined to be such a system with perfectly synchronized real-time clocks and L to be one with a form of logical clocks [L78]. $\mathcal{I}_R$ is uniformly knowledge-consistent with L, and thus can be used when solving view-based specifications in L.

Following this they considered *synchronous* systems, in which there are known bounds on message delays. R' is such a system with perfectly synchronized clocks, and L' one in which clocks are not perfectly synchronized, but in which communication is such that clocks have the properties of logical clocks. Once again, $\mathcal{I}_{R'}$ is uniformly knowledge-consistent with L', and can thus be used when solving view-based specifications in L'.

Neiger and Toueg defined a message passing primitive that, if implemented, would achieve common knowledge in the ideal systems R and R'. They implemented this primitive in L and L', and showed that these implementations can be used as if they achieve true common knowledge. These arguments can be formalized by considering the knowledge consistency of interpretations $\mathcal{I}_R$ and $\mathcal{I}_{R'}$ with systems L and L', respectively.

6 Future Work

The results given in this report are preliminary. Later versions of this paper will discuss more formally some of the topics outlined below.

[5] Fischer and Immerman do approximately the same thing by introducing the operators K_i^S (for $i \in P$) where S is the system upon which the interpretation of these operators is based.

6.1 Alternate Formulations of Knowledge Consistency

We have characterized consistent knowledge interpretations by the systems upon which they are based. There are cases, however, in which we would like to base a knowledge interpretation upon certain facts that should be "known" by processors in the system. Any *ground* fact whose negation cannot be known, *based on the system being run*, can itself be known in a consistent knowledge interpretation.[6] Thus the facts that cannot be known in a system may prove to be as useful as those that can.

We feel that there is a relationship between facts that can be consistently known and the notion of implicit knowledge [FV86]. We hope to formalize this relationship. This may allow us to determine exactly what facts can be consistently known when running a knowledge-based protocol, which may further simplify the design of such protocols. The protocol designer would then have think less in terms of the system being run, and more of the facts that processors may or may not know.

6.2 Using Weakenings of Common Knowledge

It has been shown that in systems with real-time clocks, *timestamped common knowledge* is identical to true common knowledge [HM87,NT87]. Let $\mathcal{I}_R$ be a knowledge interpretation based on such a system. If $\mathcal{I}_R$ is uniformly knowledge consistent with some practical system then achieving timestamped common knowledge in the practical system is as good as achieving true common knowledge when solving problems with view-based specifications.

There are other weakenings of common knowledge that can be achieved in practical distributed systems; in addition to those described by Halpern and Moses [HM87], there is also *concurrent common knowledge*, introduced by Panangaden and Taylor [PT86,PT87].

Theorem 2 indicates the strong relationship between view-based specifications and uniform knowledge consistency. Both concepts involve the absence of *real time*, and this relates them to the difference between true and timestamped knowledge. We hope to develop other notions of knowledge consistency (and specifications) that relate to the difference between true knowledge and its weakened forms.

7 Conclusions

The use of knowledge consistency promises to characterize more accurately the ways in which knowledge-based protocols can be used to solve problems in distributed systems. It unifies the formal semantics given to knowledge with its use in practical distributed systems.

In this paper we discussed different characterizations of knowledge consistency, and showed how one of them (uniform knowledge consistency) simplifies the design of

[6] This relationship is not as straightforward when one considers facts involving knowledge operators.

knowledge-based protocols. In addition, we gave examples of earlier results that can be understood in the context of knowledge consistency. These led to further applications of knowledge consistency, including a characterization of when weaker forms of knowledge might substitute for true common knowledge.

Acknowledgments

I would like to thank Micah Beck, Amitabh Shah, Pat Stephenson, and Sam Toueg for reading and commenting on earlier drafts of this paper. Micah Beck contributed the current subtitle.

References

[BG86] P. A. Bernstein and N. Goodman. Serializability theory for replicated databases. *Journal of Computer and System Sciences*, 31(3):355–374, December 1986.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addi ⌣n-Wesley, 1987.

[DM86] C. Dwork and Y. Moses. Knowledge and common knowledge in a Byzantine environment I: crash failures. In *Proceedings of the Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 149–169, Monterey, California, March 1986.

[FI86] M. J. Fischer and N. Immerman. Foundations of knowledge for distributed systems. In *Proceedings of the Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 171–185, Monterey, California, March 1986.

[FV86] R. Fagin and M. Y. Vardi. Knowledge and implicit knowledge in a distributed environment: Preliminary report. In *Proceedings of the Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 187–205, Monterey, California, March 1986. Also appears as IBM Technical Report RJ4990.

[HF85] J. Y. Halpern and R. Fagin. A formal model of knowledge, action, and communication in distributed systems: Preliminary report. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 224–236, Minaki, Ontario, August 1985.

[HF87] J. Y. Halpern and R. Fagin. Modelling knowledge and action in distributed systems. October 1987. Unpublished manuscript.

[HM85] J. Y. Halpern and Y. Moses. A guide to the modal logic of knowledge and belief. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 480–490, 1985.

[HM87] J. Y. Halpern and Y. Moses. *Knowledge and Common Knowledge in a Distributed Environment.* Technical Report RJ4421, IBM Research Laboratory, August 1987. An earlier version appeared in the 1984 Proceedings of the ACM Symposium on Principles of Distributed Computing.

[HZ87] J. Y. Halpern and L. D. Zuck. A little knowledge goes a long way: simple knowledge-based derivations and correctness proofs for a family of protocols. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 269–280, Vancouver, British Columbia, August 1987. A revised version appears as IBM Technical Report RJ5857.

[L78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[MT86] Y. Moses and M. R. Tuttle. Programming simultaneous actions using common knowledge. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, Toronto, Ontario, October 1986. An expanded version appears as MIT/LCS Technical Report 369, February 1987.

[NT87] G. Neiger and S. Toueg. Substituting for real time and common knowledge in asynchronous distributed systems. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 281–293, Vancouver, British Columbia, August 1987. An expanded version appears as Cornell University Technical Report 86-790.

[PT86] P. Panangaden and K. Taylor. *Concurrent Common Knowledge: a New Definition of Agreement for Asynchronous Systems.* Technical Report 86-802, Cornell University, December 1986.

[PT87] P. Panangaden and K. Taylor. December 1987. Personal communication.