Knowledge and Common Knowledge in a Distributed Environment

CS 294 Paper Review, April 2013

Colin Scott

In this paper I review Halpern and Moses' "Knowledge and Common Knowledge in a Distributed Environment" [1]. I begin by motivating the work (§1, §2) and providing an overview of the proofs (§3). I end by critiquing the work (§4) and providing suggestions for future research (§5).

1 Motivation

Reasoning about the behavior of distributed algorithms is a difficult task. Without a conceptual framework to encapsulate details, algorithm designers are often forced to resort to proof by cases or 'many-scenarios' reasoning to show correctness and impossibility results. As Yemini and Cohen put it, "proving protocols correct (or impossible) is a difficult and cumbersome art in the absence of proper formal tools to reason about protocols. Such backward-induction arguments as the one used in the impossibility proof should require less space and become more convincing with a proper set of tools." [4]

This paper seeks to provide that conceptual framework. By articulating the core of what needs to be proven (possession of knowledge), the authors hope to make the process of reasoning about distributed systems simpler and clearer. The authors' overall goal is to unify prior and future results in distributed systems under a single body of theory.

2 Solution

The authors observe that *knowledge* seems to lie at the core of many results in distributed systems. As they put it, "the vast majority of communication in a distributed system can be viewed as the act of improving the state of knowledge (in the sense of 'climbing up the hierarchy') of certain facts."

Yet many results in distributed systems only implicitly involve reasoning about knowledge. The proofs are consequently awkward, the reasoning is unclear, and results are disjoint and seemingly unrelated to each other.

The authors' thesis is that explicitly articulating the properties of knowledge will simplify reasoning and unify theoretical results. Beyond elucidating our understanding of knowledge, the authors go on to demonstrate (successfully, I think) through reworked and new proofs that reasoning is indeed simplified by considering knowledge.

The core of the discussion focuses on a hierarchy of knowledge states a group of nodes can obtain. For a given fact φ, Table 1 defines increasingly strong states of knowledge of φ, where higher levels of knowledge entail lower levels.

The authors show that these definitions satisfy several key axioms from modal logic, and later rely heavily on the axioms to prove further results. The easiest way to understand the axioms is to view knowledge states as a graph defined by equivalence relations: from the set of all possible runs for a given distributed algorithm, draw an edge labeled p_i between two points in the run(s) if node i has the same view (*i.e.* believes the same set of facts based on its history) at both points. We require that knowledge entails truth, so $K_i\varphi$

Table 1: Definitions of Knowledge States

Knowledge Class	Description
$K_i\varphi$	Node i knows φ
$D_G\varphi$	The group G as a whole has knowledge of φ
$S_G\varphi$	Someone in G knows φ
$E_G\varphi$	Everyone in G knows φ
$E_G^k\varphi$	$E_G^1\varphi = E_G\varphi$, $E_G^{k+1} = E_G^k E_G\varphi$
$C_G\varphi$	G has common knowledge of φ; $\forall_{k\in\mathbb{N}} E^k\varphi$

holds at a point iff φ holds at all points that share an edge labeled p_i. $D_G\varphi$ is defined similarly by the intersection of all the points we would consider for each node in G to individually know φ. $E_G\varphi$ holds at a point iff φ holds at all adjacent points for any node i in G. $E_G^k\varphi$ holds iff φ holds at all points k hops away, and $C_G\varphi$ holds iff φ holds at all points in the same connected component. The axioms from modal logic correspond to the structure of this graph.

After articulating a few classic distributed systems problems (the muddy children problem and the two generals problem) in terms of possession of knowledge, the bulk of the paper focuses on the last and strongest knowledge state: common knowledge. The authors show that common knowledge is important for problems requiring coordination. As we will see in the next section, common knowledge is also impossible to obtain in distributed systems.

Despite the impossibility of achieving common knowledge in practical distributed systems, real systems nonetheless achieve coordination! What these real systems are obtaining is in fact weaker than common knowledge. The last third of the paper articulates precisely the properties of weakened common knowledge states that are achievable in practice, including:

- *Epsilon Common Knowledge*: In synchronous systems, nodes can obtain common knowledge of a fact φ if we are willing to allow a time interval ϵ where not all nodes come to know $C_G\varphi$.
- *Eventual Common Knowledge*: Similarly for partially asynchronous systems, nodes can obtain common knowledge of a fact φ if we are fine with the weaker guarantee that eventually all nodes come to know $C_G\varphi$.
- *Timestamped Common Knowledge*: Rather than requiring strict simultaneity, nodes can obtain a weaker form of common knowledge if simultaneity is defined relative to the individual clocks of each node.
- *Internally Consistent Common Knowledge*: Rather than designing distributed algorithms with the assumption of a weaker knowledge guarantee, nodes can still act *as if* they had obtained strict common knowledge so long as they do not know other facts that contradict their belief. In restricting analysis to facts nodes can observe within a closed system, internally consistent common knowledge is similar to the serializability memory model.

The possibility of obtaining weakened common knowledge states does not discount the importance of strict common knowledge, as there are still a number of problems with the weakened knowledge states. The authors show that analogous impossibility proofs, such as impossibility of solving weakened versions of the two generals problem, also hold.

3 Proofs

In this section I provide insights for the main proofs of the paper.

3.1 Proposition 4: Coordinated Attack Requires Common Knowledge

This proof starts with an intuitive observation: the correctness condition of the coordinated attack problem requires that both generals attack at the same time; therefore if either general attacks at time t, it must know that the other general is also attacking at time t (otherwise it might incorrectly attack). Formally, let ψ be the ground fact "both generals are attacking". Then the correctness condition implies that $\psi \supset E_G(\psi)$.

The remainder of the proof simply invokes the induction rule (one of the modal logic axioms) to infer that $\psi \supset C_G(\psi)$. I found that the easiest way to understand this was to work out the proof for the induction rule myself:

Theorem 1. From $\varphi \supset E_G(\varphi)$ infer $\varphi \supset \forall_{k \in \mathbb{N}}.E_G^k(\varphi)$

Proof. Consider $k = 1$. Observe that $E_G(\varphi) = E_G^1(\varphi)$ by definition, so $\varphi \supset E_G^0(\varphi)$ holds. Now suppose inductively that the Theorem holds for $k > 1$. We show that the Theorem holds for $k + 1$:

$$\begin{aligned} E_G^k(\varphi) &= E_G^k(E(\varphi)) && \text{since } \varphi \supset E_G(\varphi) \\ &= E_G^{k+1}(\varphi) && \text{by definition} \end{aligned}$$

□

In words, if each node knows that $\psi \supset E_G(\psi)$, then each node can infer inductively whenever ψ holds that the other nodes must know that it knows that ψ.

Corollary. From $\varphi \supset E_G(\varphi)$ infer $\varphi \supset C_G(\varphi)$. This follows because $C_G \equiv E_G^\infty$.

3.2 Theorem 5: Common Knowledge In Networks That Do Not Guarantee Delivery

Theorem 5 forms the basis for the later proofs showing that common knowledge cannot be obtained. It states that common knowledge holds in runs where communication is successful iff common knowledge holds in suffixes of that run where no messages are delivered.

The proof is actually fairly intuitive; it relies on the fact that in networks where communication is not guaranteed, it is impossible to tell the difference between a slow node and a dead node.

Specifically, the proof proceeds by induction on the number of messages received in the run. In the inductive case, you can always transform a run where the last message is received (and common knowledge is obtained) into a run where the $k + 1_{st}$ message is delayed (yet the sender cannot tell whether the recipient has crashed), allowing us to invoke the inductive hypothesis.

3.3 Theorem 8: Common Knowledge In Synchronous Networks

The authors go on to show that common knowledge is unobtainable even in synchronous networks, where message delivery times are bounded. I found Theorem 8 to be the most surprising result of the paper.

The basic idea is that if there is *any* temporal uncertainty in message delays, then the sender of a message does not know exactly when the receiver has actually received the message (and vice versa). And in order to have common knowledge, the recipient must know that the sender knows that it received the message

(ad infinitum). In a strong sense, common knowledge requires absolute simultaneity; more precisely, if any node obtains common knowledge of a fact at time t, all other nodes must also obtain common knowledge at exactly time t, no earlier and no later.

As a concrete example of how temporal uncertainty renders common knowledge unobtainable, consider a sender s and a receiver d in a network where messages are either delivered instantaneously or are delivered after ϵ time units. We use ϕ to denote the ground fact "the message m has been sent". Let t_s denote the time the message is sent and t_d denote the time the message arrives. At t_d, $K_d\phi$ holds. But because there is uncertainty in whether the message took zero or ϵ time units to arrive, s cannot be sure that $K_d\phi$ holds until at least time $t_s + \epsilon$. Conversely, for all d knows, m might have been delivered instantaneously, so d does not know that s knows ϕ until at least time $t_d + \epsilon$. Since t_d might equal $t_s + \epsilon$, s must wait an additional ϵ time units until it knows that $t_d + \epsilon$ time has passed. Hence, $K_sK_dK_sK_d\phi$ cannot hold until at least $t_s + 2\epsilon$. This line of reasoning continues indefinitely, with each level of K_sK_d requiring an additional ϵ time units to be obtained.

4 Disadvantages

Although I was impressed by the strength of the authors' argument, the conceptual framework is certainly not without shortcomings. I outline a few of these shortcomings here.

My biggest concern is that while the conceptual framework seems to be very apt for showing impossibility results, it is not immediately clear to me that it is useful for showing *correctness* of existent algorithms. That is, how often can safety and liveness conditions be cleanly formulated in terms of knowledge? There seem to be many other seemingly unrelated properties we typically want prove, *e.g.* never deciding on a consensus value more than once.

Like the other distributed systems models we have seen, knowledge does not seem to capture a stronger notion of performance constraints than simple message complexity. This is not necessarily a fault of this paper though, simply because modeling performance constraints is highly complex.

Lastly, as the authors themselves discuss at length, strict common knowledge is not necessary for most practical applications, yet most of the results of the paper focus on its impossibility. I would be interested in a reading a paper that enumerates concrete examples of distributed systems problems that need to be addressed in practical systems, and relates them to these theoretical results. I believe this would give me better calibration of which results have immediate practical import.

5 Future Work

If the authors' thesis (that a theory of knowledge could serve to unify results in distributed systems) holds, then the most obvious avenue for future work is to carry out that unification. Beyond that though, I believe the paper incites a number of promising new research directions. The directions I present here are admittedly colored by my own interests.

My most immediate thought is that the proofs for the recent work on applying CAP theorem to networking [2] could be simplified substantially by reasoning about knowledge. The work currently proves impossibility results by relying on 'many-scenarios' arguments, similar to the original impossibility proof for the two generals problem. Reasoning about knowledge therefore appears to be a particularly apt proof method.

More generally, I think it would be enlightening to formulate network policies, which are effectively safety and liveness constraints on the behavior of the nodes of the network, in terms of knowledge states. Aurojit Panda and I are about to start a large research project on formulating and evaluating consistency

models appropriate for the networking domain. I believe formulating network policies in terms of knowledge may clarify our models.

Knowledge can also directly inform the way distributed systems are built. Specifically, declarative languages structure their control flow explicitly around predicates (knowledge states) and actions. Programs written in this way would be to straightforward to verify in comparison to imperative programs. Verification tools for declarative languages could leverage theoretical results of the kind presented in this paper.

Our own work on applying delta debugging to distributed systems [3] would benefit from the notion of knowledge. Our algorithm prunes inputs from a log of events in a buggy distributed system and replays the remaining events to check if a bug still exists at the end of the replay. The crux of the technical problem with this approach is that pruning inputs can perturb the remaining events in the history. In order to maintain causal relationships throughout this process, we manually define 'equivalence relationships' between events. For example, when comparing two message m_i from a run i and a message m_j from a modified run j, we consider m_i equivalent to m_j iff all fields of the messages except the sequence number are the same, and m_i occurs between the causal predecessor and successor of m_j. In this example we manually applied our domain knowledge about the fact that sequence numbers often are not causally related to the bugs we are interested in. Manually applying domain knowledge in this way is clearly ad-hoc and tedious. If we instead inferred information about the knowledge states of the distributed processes (*e.g.* by performing static analysis on the predicates of a declarative program), we could define the equivalence relations precisely with respect to what facts or knowledge states actually affect the outcome of the bugs we are interested in. Furthermore, knowledge may help us cleanly articulate a correctness proof of our algorithm, since it captures the core of what we are interested in, *viz.* what event information actually affects the overall outcome of the distributed algorithm's execution.

Finally, I wonder if a theory of justification would round out this work. In philosophy, the topic of justification is crucial to the study of epistemology. For the vast majority of our beliefs, we cannot provide a cogent proof that the belief is true, yet we still seem to have obtained knowledge of those facts. Instead of undeniable proofs, we rely on a system of internally consistent beliefs to justify further beliefs, which presumably provide a sufficient guarantee of truth to count as knowledge. In considering the knowledge of computer systems, this paper systems requires that knowledge strictly entails truth. It seems to me that nodes in a distributed systems might nevertheless know many facts that they cannot necessarily provide a cogent proof for. For example, nodes might correctly infer that other nodes are dead without being able to provide definite proof. Articulating the properties of such weaker forms of justification seems valuable. Incidentally, the authors touch on this briefly at the end of Section 6.

References

[1] J. Y. Halpern and Y. Moses. Knowledge and Common Knowledge in a Distributed Environment. JACM '90.

[2] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker. CAP for Networks. HotSDN '13.

[3] C. Scott, A. Wundsam, S. Whitlock, A. Or, E. Huang, K. Zarifis, and S. Shenker. How Did We Get Into This Mess? Isolating Fault-Inducing Inputs to SDN Control Software. In submission.

[4] Y. Yemini and D. Cohen. Some Issues in Distributed Process Communication. International Conference on Distributed Computing Systems '79.