Algorithmic Knowledge

Joseph Y. Halpern
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099
halpern@almaden.ibm.com

Yoram Moses
Weizmann Institute
Rehovot, ISRAEL
yoram@wisdom.weizmann.ac.il

Moshe Y. Vardi*
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099
vardi@almaden.ibm.com

Abstract: The standard model of knowledge in multi-agent systems suffers from what has been called the *logical omniscience problem*: agents know all tautologies, and know all the logical consequences of their knowledge. For many types of analysis, this turns out not to be a problem. Knowledge is viewed as being *ascribed* by the system designer to the agents; agents are not assumed to compute their knowledge in any way, nor is it assumed that they can necessarily answer questions based on their knowledge. Nevertheless, in many applications that we are interested in, agents need to *act* on their knowledge. In such applications, an externally ascribed notion of knowledge is insufficient: clearly an agent can base his actions only on what he *explicitly* knows. Furthermore, an agent that has to act on his knowledge has to be able to compute this knowledge; we do need to take into account the algorithms available to the agent, as well as the "effort" required to compute knowledge. In this paper, we show how the standard model can be modified in a natural way to take the computational aspects of knowledge into account.

*Current address: Dept. of Computer Science, Rice University, P.O. Box 1892, Houston, TX 77251-1892, vardi@cs.rice.edu

1 Introduction

Representing knowledge in terms of possible-world semantics has proved quite useful. One important application has been to analyzing complex multi-agent systems. In [FHMV94], a formal model of knowledge in multi-agent systems is proposed (which, in turn, is based on earlier models that appeared in [CM86, HF89, HM90, PR85, RK86]). This model of knowledge satisfies the axioms of S5. In particular, it suffers from what has been called the *logical omniscience problem* [Hin75]: agents know all tautologies, and know all the logical consequences of their knowledge. This was not viewed as a problem in the context of those papers, since knowledge is viewed as being *ascribed* by the system designer to the agents. Agents are not assumed to compute their knowledge in any way, nor is it assumed that they can necessarily answer questions based on their knowledge. Despite the fact that no notion of computation is involved, there are many examples to show that this notion of knowledge is useful when analyzing distributed systems [Hal87, Hal93].

Nevertheless, in many applications that we are interested in, agents need to *act* on their knowledge. In such applications, an externally ascribed notion of knowledge is insufficient: clearly an agent can base his action only on what he *explicitly* knows. Furthermore, an agent that has to act on his knowledge has to be able to compute this knowledge; we do need to take into account the algorithms available to the agent, as well as the "effort" required to compute knowledge. Computing knowledge demands that the agents have access to appropriate algorithms and to the computational resources required by these algorithms. In this paper, we show how the formal model proposed in [FHMV94] can be modified in a natural way to take the computational aspects of knowledge into account.

In the model of [FHMV94], each of the agents is assumed to be in some *local state*, which encapsulates all the information to which the agent has access. To take computation into account, we now assume that the agent has an algorithm to compute his knowledge, and that this algorithm is included in his local state. This simple, yet powerful, idea lets us model many situations in a natural way. For example, as we shall see, it lets us capture the distinction between "knowing how" and "knowing that". It also lets us capture the difference, for example, between an expert and a novice blackjack player: even when both have the same information, they use very different algorithms to compute their knowledge.

As expected, a model that takes computation into account does not suffer from the logical omniscience problem. Indeed, the solution we propose here is closely related to a solution proposed in [FH88] for dealing with the logical omniscience problem, namely, to include an awareness function. And, just like that solution, it has a syntactic component.

There have been a number of other papers that were concerned with the problem of computing knowledge. One of the first works in the AI literature relating knowledge and resource-bounded computation is that of Konolige [Kon86]. Under reasonable assumptions, Konolige's approach can be embedded in ours. The treatment of resource-bounded knowledge suggested by Moses [Mos88], and its extensions by Halpern, Moses and Tuttle [HMT88] to treat aspects of *interactive proofs* and *zero knowledge proofs* [GMR89], can be viewed as precursors of this work. Although our model is significantly different, it is in the spirit of [Mos88, HMT88]. There have also been many attempts in the game-theoretical literature to model resource-bounded agents, e.g., [Meg89, MW86, Ney85, Rub85] (see [Bin90](Chapters 5-6) for a foundational discussion); our formal model is much different from any proposed in this literature.

2 Knowledge in multi-agent systems

We briefly review the framework of [FHMV94] for modeling multi-agent systems. We assume that at each point in time, each agent is in some *local state*. Informally, this local state encodes the information the agent has observed thus far. In addition, there is also an *environment* state, that keeps track of everything relevant to the system not recorded in the agents' states. The way we split up the system into agents and environment depends on the system being analyzed.

A *global state* is a sequence $(s_e, s_1, \ldots, s_n)$ consisting of the environment state s_e and the local state s_i of each agent i. A *run* of the system is a function from time (which, for ease of exposition, we assume ranges over the natural numbers) to global states. Thus, if r is a run, then $r(0), r(1), \ldots$ is a sequence of global states that, roughly speaking, is a complete description of what happens over time in one possible execution of the system. We take a *system* to consist of a set of runs. Intuitively, these runs describe all the possible sequences of events that could occur in a system.

Given a system $\mathcal{R}$, we refer to a pair (r, m) consisting of a run $r \in \mathcal{R}$ and a time m as a *point*. If $r(m) = (s_e, s_1, \ldots, s_n)$, we define $r_i(m) = s_i$, for $i = 1, \ldots, n$; thus, $r_i(m)$ is process i's local state at the point (r, m). We say two points (r, m) and (r', m') are *indistinguishable* to agent i, and write $(r, m) \sim_i (r', m')$, if $r_i(m) = r'_i(m')$, i.e., if agent i has the same local state at both points. Finally, we define an *interpreted system* to be a pair $(\mathcal{R}, \pi)$ consisting of a system $\mathcal{R}$ together with a mapping π that associates a truth assignment to the primitive propositions with each point.

An interpreted system can be viewed as a Kripke structure: the points are the possible worlds, and $\sim_i$ plays the role of the accessibility relation. We give semantics to knowledge formulas in interpreted systems just as in Kripke structures: Given a point (r, m) in an interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$, we have $(\mathcal{I}, r, m) \models K_i\varphi$ if $(\mathcal{I}, r', m') \models \varphi$ for all points (r', m') such that $(r', m') \sim_i (r, m)$. Notice that under this interpretation, an agent knows φ if φ is true at all the situations the system could be in, given the agent's current information (as encoded by his local state). Since the $\sim_i$ is an equivalence relation, knowledge in this framework satisfies the S5 axioms. We view K_i as describing *implicit* knowledge, in contrast to the notion of explicit, algorithmic knowledge that we introduce in the next section.

The major application area of this framework has been in analyzing distributed protocols. For a given distributed protocol, it is often relatively straightforward to construct the system corresponding to the protocol. The local state of each process can typically be characterized by a number of internal variables (which, for example, describe the messages thus far received and the values of certain local variables), and there is a transition function that describes how the system changes from one global state to another. (See [FHMV94] for a detailed discussion of the modelling process, and for examples of its use.)

3 Algorithmic knowledge

Intuitively, we would like to say that the agent knows a fact φ if he can compute that he knows φ. As we mentioned in the introduction, we intend to model this by saying that the agent has an algorithm for deciding if he knows φ. To make this precise, we need to describe what it means for an algorithm to decide if agent i knows φ, and also what it means for the agent to "have" such an algorithm.

An agent's knowledge clearly depends on his local state. Thus, we might expect that an algorithm to decide if agent i knows φ is one that, given as input a local state ℓ and a formula φ, returns either "Yes" or "No", depending on whether agent i knows φ. Note that a "No" is not taken to mean that agent i knows that φ is false, but that he does not know that it is true. Determining the truth of all knowledge formulas at all points of the system may, however, be beyond an agent's computational abilities. In fact, one of the purposes of this approach is to come to grips with this limitation of computational agents. We may be happy with an algorithm that works in the prescribed manner only for certain formulas φ, or only on a subset of the points of the system. To deal with this, we allow an output of "?", in addition to "Yes" and "No"; a "?" output means that the agent is unable to compute whether he knows φ. Thus, we focus on algorithms that take as input a local state and a formula, and return as answer either "Yes", "No", or "?".

What does it mean to say that agent i "has" an algorithm for deciding φ? It is certainly not enough to say that there is some algorithm for i that gives the right answer on input φ. At a given point (r, m), either $K_i\varphi$ holds or it does not. Consider two trivial algorithms: the first is an algorithm that always says "Yes", and the other is the algorithm that always says "No". Clearly, one of them gives the right answer about whether $K_i\varphi$ holds at (r, m). Nevertheless, despite his being able to execute both of these algorithms, we would not say that agent i "has" an algorithm to compute whether he knows φ in this case, unless he knows which of the two algorithms to use when asked about φ. Part of "having" an algorithm is knowing when to use it.

We deal with this problem by assuming that the algorithm that agent i uses to compute his knowledge at a point (r, m) is part of his local state at (r, m). Thus, agent i "has" the algorithm in his local state. We do not mean to imply that the agent necessarily has the same algorithm at every point in the system. An agent may have a number of possible algorithms at his disposal, and the one he uses may depend on the information he receives. Alternatively, the information he receives may allow him to come up with a new algorithm.

Formally, we now model agent i's local state as a pair $\langle \mathtt{A}, \ell \rangle$, where $\mathtt{A}$ is an algorithm, and ℓ is the rest of his local state. We call $\mathtt{A}$ the agent's *local algorithm*, and we call ℓ the agent's *local data*. An interpreted system in which local states have this form is called an *algorithmic system*. When $r_i(m) = \langle \mathtt{A}, \ell \rangle$, we use $\mathtt{alg}_i(r, m)$ to denote the algorithm $\mathtt{A}$, and $\mathtt{data}_i(r, m)$ to denote i's local data ℓ. In local state $\langle \mathtt{A}, \ell \rangle$, the agent computes whether he knows φ by applying the local algorithm $\mathtt{A}$ to input (φ, ℓ).

The separation between an agent's local algorithm and his local data in algorithmic systems allows us to distinguish the agent's operational knowledge ("knowing how") from his factual knowledge about the world ("knowing that"). The operational knowledge is captured by the agent's local algorithm, while the factual knowledge is captured by his local data. Note that it is not always obvious how to partition a local state into local algorithm and local data (just as it is not always obvious how to partition a global state into local states). For example, as we discuss later, a cryptographic key can be viewed either as local data or as part of the decryption procedure, and therefore as part of the local algorithm. In general, there is more than one way to "cut the cake"; the appropriate choice is application dependent.

Algorithmic systems can model both our earlier notion of implicit, externally ascribed knowledge, and a notion of explicit knowledge that we call *algorithmic* knowledge. Implicit knowledge is denoted, as before, by the modal operator K_i, while algorithmic knowledge is denoted by the modal operator X_i, and is defined as follows:

$$(\mathcal{I}, r, m) \models X_i\varphi \text{ iff } \mathtt{A}(\varphi, \ell) = \text{“Yes”}, \text{ for } \mathtt{A} = \mathtt{alg}_i(r, m) \text{ and } \ell = \mathtt{data}_i(r, m).$$

Thus, agent i has algorithmic knowledge of φ at a given point if the agent's algorithm at that point outputs "Yes" when presented with φ and with the agent's local data. (Note that both the outputs "No" and "?" result in lack of algorithmic knowledge.) We say that a local algorithm *claims* (that agent i knows) a formula φ at a given point if it outputs "Yes" when presented with φ and i's local data at that point.

Notice that our definition makes clear that computing whether an agent knows φ has essentially nothing to do with computing whether φ is valid. The fact that checking validity is *PSPACE*-complete in multi-agent S5 [HM92] does not indicate that computing knowledge in any particular situation will necessarily be hard. On the other hand, as we shall see, there is a connection between computing knowledge and the *model-checking* problem, that is, the problem of checking whether a formula is true at a particular point in the system [HV91].

While the definition of algorithmic knowledge draws a direct connection between an agent's explicit knowledge and the need to compute this knowledge, $X_i\varphi$ as defined is a notion of belief. An algorithm could very well claim that agent i knows φ (i.e., output "Yes") whenever it chooses to, including at points where $K_i\varphi$ does not hold. This is not so unreasonable in practice; agents do make mistakes! The local algorithm of, say, a knowledge base (which we abbreviate as KB from now on) may very well occasionally give the wrong answer. In such cases, the answer given by the KB describes a belief, which may perhaps be modified as new information is added to the knowledge base.

Although algorithms that make mistakes are common, we are often interested in local algorithms that are correct. Formally, a local algorithm $\mathtt{A}$ is called *sound* for agent i in the system $\mathcal{I}$ if for all points (r, m) of $\mathcal{I}$ and formulas φ, if $\mathtt{alg}_i(r, m) = \mathtt{A}$ and $\mathtt{data}_i(r, m) = \ell$, then (a) $\mathtt{A}(\varphi, \ell) =$ "Yes" implies $(\mathcal{I}, r, m) \models K_i\varphi$, and (b) $\mathtt{A}(\varphi, \ell) =$ "No" implies $(\mathcal{I}, r, m) \models \neg K_i\varphi$. It follows that at a point where agent i uses a sound local algorithm, $X_i\varphi \Rightarrow K_i\varphi$ holds. Thus, if the local algorithms used by agent i in a given system $\mathcal{I}$ are sound, then X_i satisfies the Knowledge Axiom: $\mathcal{I} \models X_i\varphi \Rightarrow \varphi$. Soundness of local algorithms is clearly a desirable property, since acting based on knowledge is, in general, better than acting based on beliefs that may turn out to be wrong.

In a precise sense, when $X_i\varphi$ holds at a point where agent i uses a sound local algorithm, it is appropriate to say that i is able to compute that he knows φ. Soundness is a *safety* property; it is a guarantee that wrong answers are not given. Notice that an algorithm does not have to be very sophisticated in order to be sound. In fact, an algorithm that never outputs "Yes" and never outputs "No" (i.e., it always outputs "?") is clearly sound. We are often interested, in addition to soundness, in a guarantee that the algorithm always yields a definite ("Yes" or "No") answer. This motivates the following definition. A local algorithm $\mathtt{A}$ is called *complete* for agent i in the system $\mathcal{I}$ if for all points (r, m) of $\mathcal{I}$ and all formulas φ, if $\mathtt{alg}_i(r, m) = \mathtt{A}$ and $\mathtt{data}_i(r, m) = \ell$, then $\mathtt{A}(\varphi, \ell) \in \{$"Yes", "No"$\}$. It follows that at a point where agent i uses a sound and complete local algorithm, $X_i\varphi \Leftrightarrow K_i\varphi$ holds.

For many applications of interest, completeness is too strong a requirement. In [FHMV94], we introduce a notion of *knowledge-based programs*, in which there are explicit tests for knowledge, in order to help us explicitly capture the connection between knowledge and action. Such programs can contain statements of the form "**if asked 'does φ hold?' and $K_i\varphi$ do say 'Yes'**". (Knowledge-based programs are related in spirit to the *knowledge-based protocols* of [HF89], and to the notion of *agent-oriented programming* introduced by Shoham [Sho93].) Such a program contains only a finite number of knowledge tests. Thus, in order to implement such a program, we need an algorithm that is complete only with respect to the tests in the program. Moreover, the algorithm need not be able to compute the outcome of each test at every point; it suffices to be able to compute the outcome only at the local states where the test is encountered. This leads us to the following weakening of the completeness requirement. Given a set Ψ of formulas and a set L of local data for i in an algorithmic system $\mathcal{I}$, a local algorithm $\mathtt{A}$ is *complete with respect to* (Ψ, L) if for all points (r, m) in $\mathcal{I}$ such that $\mathtt{data}_i(r, m) \in L$ and all formulas $\psi \in \Psi$, it is the case that $\mathtt{A}(\psi, \mathtt{data}_i(r, m)) \in \{$"Yes", "No"$\}$. Thus, on the formulas and local data of interest, a complete algorithm always gives a definite answer. (A restricted version of soundness can be defined in an analogous manner.)

The soundness of an agent's local algorithm is related to the agent's rationality. A rational agent would not want to act based on incorrect information. Similarly, the completeness of agent's local algorithm is related to the agent's expertise. Intuitively, the less often an agent's local algorithm gives the answer "?", the more expert the agent is. Notice that if an agent has a sound algorithm for computing whether he knows φ at a particular local state, then he essentially has an algorithm for checking if $K_i\varphi$ holds at a point where he is in that local state; thus, he can do a limited form of model checking.

There is a subtlety in using the algorithmic framework to model the limited abilities of resource-bounded agents that has to do with the notion of time. We often classify computational complexity by analyzing the time requirement of the computation, e.g., polynomial time. Recall that a run is a function from time to global states. But what is the relationship between the notion of time when we speak of polynomial-time algorithms and the notion of time modeled explicitly in multi-agent systems? Our semantics for algorithmic knowledge suggests that the local algorithms always yield their result in one round regardless of their time complexity.

The reason for this apparent inconsistency is that we really have two notions of time in mind. Time in runs serves as a convenient way to partition the system behavior into rounds, where a round is a basic unit of activity. For example, in multi-agent systems that model communication in distributed systems, it is often convenient to take a round to be sufficiently long for a process to send a message to all other processes. Generally, our choice of the granularity of time is motivated by convenience of modeling, and time should not be thought of as "real time".

When we speak of time in the context of (polynomial-time) computations, we are not speaking of "real time" either. Rather, the notion of time in complexity is meant to measure the number of machine instructions performed by a computational device. In general, the precise relationship between the two notions of time is also application dependent. In modeling communication in distributed systems, a machine instruction could be "transmit one bit" and a round could include up to some polynomial number of such instructions. Thus, when we model an algorithmic system, the local algorithms have to be such that they can be run in one round. If one wishes to model algorithms with longer running time, then these algorithms should be split up among successive states.

As a consequence, our framework is more general than the *step-logics* considered by Elgot-Drapkin and Perlis [Elg91, EP90]. They essentially try to model the reasoning of agents over time; at each time step, the agent can carry out one step of reasoning. This can be viewed as a special case of our framework: We take the agent's local state to consist of a set of formulas (intuitively, those that it has deduced thus far). At each step, this set would be augmented by whatever new formulas are deduced. If we only allow one inference at each step, only one formula can be added at each step. The agent's local algorithm is now quite trivial: The agent explicitly knows φ only if it is one of the formulas that has been deduced thus far.

We close this section with a brief discussion of the properties of algorithmic knowledge, and its relationship to awareness. First of all, it is easy to see that algorithmic knowledge does not suffer from logical omniscience. There is no need for an agent to algorithmically know any tautologies, nor to know the logical consequences of his algorithmic knowledge. Indeed, if we put no constraints on the algorithms, there are no interesting properties of algorithmic knowledge in and of itself.

There is an important connection, however, between algorithmic knowledge and implicit knowledge. Recall that an agent's local algorithm is considered part of its local state. Moreover, the local algorithm operates only on the local data, which is also a component of the agent's local state. Thus, the agent's algorithmic knowledge depends only on its local state. As a result, an agent (implicitly) knows whether or not he has algorithmic knowledge. Thus, $X_i\varphi \Rightarrow K_iX_i\varphi$ and $\neg X_i\varphi \Rightarrow K_i\neg X_i\varphi$ are both valid in algorithmic systems.

As we already observed, $X_i\varphi \Rightarrow \varphi$ holds at all points agent i's local algorithm is sound, as does $X_i\varphi \Rightarrow K_i\varphi$, and $K_i\varphi \Leftrightarrow X_i\varphi$ holds at all points where i's local algorithm sound and complete. This means that at points where i's local algorithm is sound and complete, the properties of algorithmic knowledge and implicit knowledge coincide.

The behavior of a local algorithm clearly may depend on the syntax of φ, just as the notion of awareness in [FH88]. In fact, as we mentioned in the introduction, algorithmic knowledge is also closely related to the logic of awareness. In this logic, a new modal operator A_i is introduced, where $A_i\varphi$ can be read as "agent i is aware of φ". At every state, there is some (arbitrary) set of formulas that the agent is aware of. In the context of algorithmic knowledge, we can say that agent i is *aware* of φ, denoted $(\mathcal{I}, r, m) \models A_i\varphi$, in local state $r_i(m) = \langle \mathtt{A}, \ell \rangle$, if $\mathtt{A}(\varphi, \ell) \in \{\text{"Yes"}, \text{"No"}\}$. If $\mathtt{alg}_i(r,m)$ is a sound algorithm, then we have that $(\mathcal{I}, r, m) \models X_i\varphi \Leftrightarrow K_i\varphi \wedge A_i\varphi$. This is, in fact, the definition of explicit knowledge given in [FH88].

4 Examples

A framework is useful only if it can capture a wide variety of problems in a natural way. As we now show by example, algorithmic knowledge can indeed capture many situations of interest in which agents need to compute their knowledge.

Sophisticated vs. naive players: Consider the card game of blackjack. In this game, players of different skills vary in both their local data and local algorithms. The local data of a naive player often comprises just the information about the cards in his hand and the exposed cards on the table. Such a player would typically also use a rather simple local algorithm to decide whether or not to draw more cards. The local data of a more sophisticated player may also include information about cards that were exposed in previous rounds of the game (a practice called "card counting"). Such a player would typically also have a more sophisticated local algorithm at his disposal. It is well known that while the odds are against a naive player of blackjack, they favor a sufficiently sophisticated card counter [Tho61, Tho66]. Thus, a sophisticated player gets an advantage both from having more detailed local data and from using a more sophisticated local algorithm.

The example above can be extended to dealing with trading activity in financial markets such as the stock and futures market. Traders vary in both the raw information available to them (the local data) as well as the way they interpret this information (the local algorithm). Traders gain advantage by getting more information (perhaps even "inside information") and by using more sophisticated algorithms for estimating, for example, what the future value of a stock or commodity is likely to be.

Konolige's framework: Consider the reasoning systems discussed by Konolige [Kon86]. There an agent is assumed to have a base set B of formulas, and a formal system R of inference rules. Intuitively, the agent knows φ if she can deduce φ from her base formulas using her inference rules. When the base set B is finite and the formal system is decidable, there is an algorithm that can tell whether φ can be deduced from B using R. This case can be captured in our framework by having the base set of formulas be part of the agent's local data, while the formal system characterizes her local algorithm. We remark that we cannot deal with an undecidable system, although Konolige's framework would allow it, provided that it is axiomatizable. The reason is that we insisted that local algorithms be terminating (although they may return "?"). If we had allowed nonterminating algorithms, then we would have been able to deal with Konolige's framework in its full generality.

Levesque's logic of explicit belief: Levesque introduced a notion of *explicit belief*, captured by a modal operator B, in [Lev84], with the property that the validity of formulas of the form $B\kappa \Rightarrow B\varphi$, for κ and φ propositional formulas in conjunctive normal form (CNF) can be decided in polynomial time. Moreover, he proved that if $B\kappa \Rightarrow B\varphi$ is valid with respect to his semantics for explicit belief, then $\kappa \Rightarrow \varphi$ is a propositional tautology. As discussed in [FHMV94, HV91], KBs can be easily modeled in the framework of multi-agent systems. Now suppose that the KB is told only propositional formulas, so that its local state can be represented by a propositional formula κ describing what it has been told. When asked a query φ, we would like the KB to answer "Yes" exactly if $\kappa \Rightarrow \varphi$ is valid. Unfortunately, this is too much to expect from a KB, since testing propositional validity is co-NP-complete. Levesque suggested that by using his approach, we can at least get correct answers, with a comprehensible semantics.

We can recast this in terms of algorithmic knowledge. When asked a query φ in state κ, for κ and φ in CNF, the KB's local algorithm is to test whether $B\kappa \Rightarrow B\varphi$ is valid under Levesque's semantics. If it is, the algorithm outputs "Yes", otherwise it outputs "?". Thus, the KB has algorithmic knowledge of φ if its local state is κ iff $B\kappa \Rightarrow B\varphi$ is valid. By Levesque's results, this algorithm is sound for propositional formulas in CNF, and runs in polynomial time; however, it is not complete, even for formulas in CNF.

Cryptography: Another application that fits the framework of algorithmic systems well is modern-day cryptography. A popular approach in cryptography these days is to use computational difficulty to guarantee the security of encryption, via *public-key cryptography* (see, for example, [RSA78]). Messages are encrypted by applying a publicized function to the original text of the message. These functions are chosen in such a way that decrypting an encrypted message is easy for an agent who possesses a secret key (such as the factorization of a particular large number), while an agent with no access to such a key would have to use a great deal of computing power to decrypt the message. Thus, while all of the information about the original content of the message is encoded in the encrypted message, the content is inaccessible without the secret key. Suppose that i possesses the secret key, while j does not. Agent i then has at his disposal an efficient decryption algorithm, and hence will explicitly know the content of encrypted messages he receives. Agent j, not possessing the secret, will not explicitly know the content of such encrypted messages. On the other hand, j would have implicit knowledge of the content of each message he receives, although it would probably be of no practical use to him. This example points out another feature of our framework. Notice that the main source of i's uncertainty regarding j is not what j knows: i knows that j knows the encrypted message and does not know the secret. (Actually, the latter is probably belief rather than knowledge, but that is not the issue here.) Rather, it is uncertainty regarding j's algorithm. Such uncertainty is precisely the type of uncertainty we need to model in cryptographic settings, where the question of whether an opponent has an algorithm allowing him to break a code is crucial.

As we mentioned earlier, in [HMT88] a framework much in the spirit of the one presented here is used to analyze some aspects of cryptography, namely the work on zero-knowledge protocols. The framework of [HMT88] includes probability as well as knowledge. It would be straightforward to add probability to the framework discussed here, along the lines of what is done in [HMT88, HT93]. Once that is done, we can recast the analysis of zero-knowledge protocols done in [HMT88] in the framework of algorithmic knowledge.

Algorithmic programs: We mentioned above that knowledge-based programs can be used to describe the relationship between knowledge and action. Intuitively, knowledge-based programs prescribe what actions the agents should take as a function of their local state and their knowledge. In knowledge-based programs, however, an agent's actions depend on the agent's implicit knowledge. As a result, knowledge-based programs are not directly "runnable". We need a way of computing the agent's knowledge of the specific facts that appear in the program in order to be able to execute such a program. The concept of algorithmic knowledge suggests a solution to the "non-runnability" of knowledge-based programs; we can consider programs that prescribe what actions the agents should take as a function of their local state and their algorithmic knowledge. We call such programs *algorithmic* programs. The formal syntax and semantics of algorithmic programs will be described in [FHMV94]. In [FHMV94] we also provide conditions on when we can go from knowledge-based programs—which are often easier for the designer to think about—to algorithmic programs with tests for explicit knowledge, which are necessary for implementation.

In many applications, algorithmic programs capture our intuition of "action based on knowledge" better than knowledge-based programs. Designers of distributed programs often say things like: "once A knows B has received the message μ, then A will stop sending μ to B". Quite often, however, these designers are not necessarily referring to implicit knowledge. A program based on such an intuition will typically involve having B send A an acknowledgement when B receives μ, and having A stop sending μ once it receives such an acknowledgement. It may happen that before receiving this acknowledgement, A already has enough information to infer that B has received μ, e.g., A received a message from a third agent C that could not have been sent unless B had already received μ. In practice, A would usually not try to detect such knowledge (unless stopping to send μ as early as possible is absolutely essential); rather, it would continue to send μ until the particular test for the acknowledgement succeeds. We may conclude that the reference to knowledge in the designers' intuition above does not quite refer to implicit knowledge. Rather, it can be thought of as assuming that there are particular tests on which the knowledge is based. Clearly, these tests give rise to local algorithms for the processes to use in testing for knowledge about the relevant facts of interest. Thus, what the designers often have in mind is an algorithmic notion of knowledge rather than an information-based one, which is exactly what the notion of algorithmic programs is intended to capture.

This discussion suggests that reasoning about implicit knowledge will not suffice in the analysis of many complex algorithms; reasoning in terms of algorithmic knowledge may be necessary as well. One example where algorithmic knowledge has been found to be useful is in the design and description of protocols for Byzantine agreement [BGP89]. Although the notion of algorithmic knowledge was not explicitly used in [BGP89], the reasoning done there can be embedded directly in our framework.

5 Conclusions

We have presented a conceptually simple, yet quite powerful, framework in which computational properties of knowledge can be captured. We have shown that this framework enables us to model in a natural manner many applications where such computational properties are important.

Implicit knowledge has been shown to be an important tool for analyzing multi-agent systems. One of the advantages of implicit knowledge is that its simple semantics in terms of Kripke structures makes it easy to analyze. We believe that our simple model for algorithmic knowledge will be a useful tool for finer analysis of the connection between knowledge and action.

Acknowledgements

The first and second authors would like to thank Mark Tuttle for the insights obtained in their joint work on [HMT88], which had significant influence on this paper. We would also like to thank Ron Fagin for his collaboration on the book [FHMV94]; this collaboration nurtured the ideas in this paper. Finally, we thank Gerhard Lakemeyer for his useful comments on an earlier draft of the paper. The work of the second author was supported in part by a Helen and Milton A. Kimmelman Career Development Chair.

References

[BGP89] P. Berman, J. Garay, and K. J. Perry. Towards optimal distributed consensus. In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, pages 410–415, 1989.

[Bin90] K. Binmore. *Essays on the Foundations of Game Theory.* Basil Blackwell, Oxford, UK, 1990.

[CM86] K. M. Chandy and J. Misra. How processes learn. *Distributed Computing*, 1(1):40–52, 1986.

[Elg91] J. J. Elgot-Drapkin. Step-logic and the three-wise-men problem. In *Proc. National Conference on Artificial Intelligence (AAAI '91)*, 1991.

[EP90] J. J. Elgot-Drapkin and D. Perlis. Reasoning situation in time I: Basic concepts. *Journal of Experimental and Theoretical Artificial Intelligence*, 2(1):75–98, 1990.

[FH88] R. Fagin and J. Y. Halpern. Belief, awareness, and limited reasoning. *Artificial Intelligence*, 34:39–76, 1988.

[FHMV94] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge.* MIT Press, Cambridge, MA, To appear, 1994.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, February 1989.

[Hal87] J. Y. Halpern. Using reasoning about knowledge to analyze distributed systems. In J. F. Traub, B. J. Grosz, B. W. Lampson, and N. J. Nilsson, editors, *Annual Review of Computer Science, Vol. 2*, pages 37–68. Annual Reviews Inc., Palo Alto, CA, 1987.

[Hal93] J. Y. Halpern. Reasoning about knowledge: a survey circa 1991. In A. Kent and J. G. Williams, editors, *Encyclopedia of Computer Science and Technology, Volume 27 (Supplement 12).* Marcel Dekker, New York, 1993.

[HF89] J. Y. Halpern and R. Fagin. Modelling knowledge and action in distributed systems. *Distributed Computing*, 3(4):159–179, 1989.

[Hin75] J. Hintikka. Impossible possible worlds vindicated. *Journal of Philosophical Logic*, 4:475–484, 1975.

[HM90] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990. A preliminary version appeared in *Proc. 3rd ACM Symposium on Principles of Distributed Computing*, 1984.

[HM92] J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.

[HMT88] J. Y. Halpern, Y. Moses, and M. R. Tuttle. A knowledge-based analysis of zero knowledge. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 132–147, 1988.

[HT93] J. Y. Halpern and M. R. Tuttle. Knowledge, probability, and adversaries. *Journal of the ACM*, 40(4):917–962, 1993.

[HV91] J. Y. Halpern and M. Y. Vardi. Model checking vs. theorem proving: a manifesto. In J. A. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proc. Second International Conference (KR '91)*, pages 325–334. Morgan Kaufmann, San Mateo, CA, 1991.

[Kon86] K. Konolige. *A Deduction Model of Belief.* Morgan Kaufmann, San Mateo, CA, 1986.

[Lev84] H. J. Levesque. A logic of implicit and explicit belief. In *Proc. National Conference on Artificial Intelligence (AAAI '84)*, pages 198–202, 1984.

[Meg89] N. Megiddo. On computable beliefs of rational machines. *Games and Economical Behaviour*, 1:144–169, 1989.

[Mos88] Y. Moses. Resource-bounded knowledge. In M. Y. Vardi, editor, *Proc. Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 261–276. Morgan Kaufmann, San Mateo, CA, 1988.

[MW86] N. Megiddo and A. Wigderson. On play by means of computing machines. In J. Y. Halpern, editor, *Theoretical Aspects of Reasoning about Knowledge: Proc. 1986 Conference*, pages 259–274. Morgan Kaufmann, San Mateo, CA, 1986.

[Ney85] A. Neyman. Bounded complexity justifies cooperation in finitely repated prisoner's dilemma. *Economic Letters*, pages 227–229, 1985.

[PR85] R. Parikh and R. Ramanujam. Distributed processing and the logic of knowledge. In R. Parikh, editor, *Proc. Workshop on Logics of Programs*, pages 256–268, 1985.

[RK86] S. J. Rosenschein and L. P. Kaelbling. The synthesis of digital machines with provable epistemic properties. In J. Y. Halpern, editor, *Theoretical Aspects of Reasoning about Knowledge: Proc. 1986 Conference*, pages 83–97. Morgan Kaufmann, San Mateo, CA, 1986.

[RSA78] R. L. Rivest, A. Shamir, and L. Adelman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Rub85] A. Rubinstein. Finite automata play the repeated prisoner's dilemma. ST/ICERD Discussion Paper 85/109, London School of Economics, 1985.

[Sho93] Y. Shoham. Agent oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[Tho61] E. O. Thorp. A favorable strategy for twenty-one. *Proc. Natl. Acad. Sci.*, 47(1):110–112, 1961.

[Tho66] E. O. Thorp. *Beat the dealer.* Vintage, New York, 2nd edition, 1966.