Probabilistic Algorithmic Knowledge*

Joseph Y. Halpern
Department of Computer Science
Cornell University
Ithaca, NY 14853
halpern@cs.cornell.edu

Riccardo Pucella
Department of Computer Science
Cornell University
Ithaca, NY 14853
riccardo@cs.cornell.edu

Abstract

The framework of algorithmic knowledge assumes that agents use deterministic knowledge algorithms to compute the facts they explicitly know. We extend the framework to allow for randomized knowledge algorithms. We then characterize the information provided by a randomized knowledge algorithm when its answers have some probability of being incorrect. We formalize this information in terms of *evidence*; a randomized knowledge algorithm returning "Yes" to a query about a fact φ provides evidence for φ being true. Finally, we discuss the extent to which this evidence can be used as a basis for decisions.

1 Introduction

Under the standard possible-worlds interpretation of knowledge, which goes back to Hintikka [1962], an agent knows φ if φ is true at all the worlds the agent considers possible. This interpretation of knowledge has been found useful in capturing some important intuitions in the analysis of distributed protocols [Fagin, Halpern, Moses, and Vardi 1995]. However, its usefulness is somewhat limited by what Hintikka [1962] called the *logical omniscience problem*: agents know all tautologies and know all logical consequences of their knowledge. Many approaches have been developed to deal with the logical omniscience problem (see [Fagin, Halpern, Moses, and Vardi 1995, Chapter 10 and 11] for a discussion and survey). We focus on one approach here that has been called *algorithmic knowledge* [Halpern, Moses, and Vardi 1994]. The idea is simply to assume that agents are equipped with "knowledge algorithms" that they use to compute what they know. An agent *algorithmically knows* φ if its knowledge algorithm says "Yes" when asked φ.[1]

Algorithmic knowledge is a very general approach. For example, Berman, Garay, and Perry [1989] implicitly use a particular form of algorithmic knowledge in their analysis of Byzantine agreement. Roughly speaking they allow agents to perform limited tests based on the information they have; agents know only what follows from these limited tests. Ramanujam [1999] investigates

*Authors supported in part by NSF under grant CTC-0208535, by ONR under grants N00014-00-1-03-41 and N00014-01-10-511, by the DoD Multidisciplinary University Research Initiative (MURI) program administered by the ONR under grant N00014-01-1-0795, and by AFOSR under grant F49620-02-1-0101.

[1] We remark that what we are calling "knowledge algorithms" here are called "local algorithms" in [Fagin, Halpern, Moses, and Vardi 1995; Halpern, Moses, and Vardi 1994].

a particular form of algorithmic knowledge, where the knowledge algorithm is essentially a model-checking procedure for a standard logic of knowledge. More specifically, Ramanujam considers, at every state, the part of the model that a particular agent sees (for instance, an agent in a distributed system may be aware only of its immediate neighbors, the ones with whom he can communicate) and takes as knowledge algorithm the model-checking procedure for epistemic logic, applied to the submodel generated by the visible states. Halpern and Pucella [2002] have applied algorithmic knowledge to security to capture intruders who are resource bounded (and thus, for example, cannot factor the products of large primes that arise in the RSA cryptosystem [Rivest, Shamir, and Adelman 1978]).

All these examples use *sound* knowledge algorithms: although the algorithm may not give an answer under all circumstances, when it says "Yes" on input φ, the agent really does know φ in the standard possible-worlds sense. Although soundness is not required in the basic definition, it does seem to be useful in many applications.

Our interest in this paper is knowledge algorithms that may use some randomization. As we shall see, there are numerous examples of natural randomized knowledge algorithms. With randomization, whether or not the knowledge algorithm says "Yes" may depend on the outcome of coin tosses. This poses a slight difficulty in even giving semantics to algorithmic knowledge, since the standard semantics makes sense only for deterministic algorithms. To deal with this problem, we make the algorithms deterministic by supplying them an extra argument (intuitively, the outcome of a sequence of coin tosses) to "derandomize" them. We show that this approach provides a natural extension of the deterministic case.

To motivate the use of randomized knowledge algorithms, we consider a security example from Halpern and Pucella [2002]. The framework in that paper lets us reason about principals communicating in the presence of adversaries, using cryptographic protocols. The typical assumption made when analyzing security protocols is that adversaries can intercept all the messages exchanged by the principals, but cannot necessarily decrypt encrypted messages unless they have the appropriate decryption key. To capture precisely the capabilities of adversaries, we use knowledge algorithms. Roughly speaking, a knowledge algorithm for an adversary will specify what information the adversary can extract from intercepted messages. In this paper, we consider an adversary that further attempts to guess the cryptographic keys used by the principals in the protocol. We show how to capture the knowledge of such an adversary using a randomized knowledge algorithm.

Having defined the framework, we try to characterize the information obtained by getting a "Yes" answer to a query for φ. If the knowledge algorithm is sound, then a "Yes" answer guarantees that φ is true. However, the randomized algorithms of most interest to us give wrong answers with positive probability, so are not sound. Nevertheless, it certainly seems that if the probability that the algorithm gives the wrong answer is low, it provides very useful information when it says "Yes" to a query φ. This intuition appears in the randomized algorithms literature, where a "Yes" answer from a highly reliable randomized algorithm that is, one with a low probability of being wrong, is deemed "good enough". In what sense is this true? One contribution of our work is to provide a formal answer to that question. It may seem that a "Yes" answer to a query φ from a highly reliable randomized knowledge algorithm should make the probability that φ is true be high but, as we show, this is not necessarily true. Rather, the information should be viewed as *evidence* that φ is true; the probability that φ is true also depends in part on the prior probability of φ.

Evidence has been widely studied in the literature on inductive logic [Kyburg 1983]. Evidence for a particular hypothesis can be accumulated from different sources, and it is possible to combine the evidence provided by knowledge algorithms with other kind of evidence already accumulated for a formula, obtained for example by running other kinds of tests, and look at the combined evidence. We do not consider evidence in such a general setting in this paper; rather, we focus on the evidence contributed specifically by a randomized knowledge algorithm. Our companion paper [Halpern and Pucella 2003] provides a treatment of evidence in a more general setting.

2 Reasoning about knowledge and algorithmic knowledge

The aim is to be able to reason about properties of systems involving the knowledge of agents in the system. To formalize this type of reasoning, we first need a language. The syntax for a multiagent logic of knowledge is straightforward. Starting with a set Φ of primitive propositions, which we can think of as describing basic facts about the system, such as "the door is closed" or "agent A sent the message m to B", more complicated formulas are formed by closing off under negation, conjunction, and the modal operators $K_1, \ldots, K_n$ and $X_1, \ldots, X_n$. Thus, if φ and ψ are formulas, then so are $\neg\varphi$, $\varphi \wedge \psi$, $K_i\varphi$ (read "agent i knows φ"), and $X_i\varphi$ (read "agent i can compute φ").

The standard possible-worlds semantics for knowledge uses *Kripke structures* [Kripke 1963]. Formally, a Kripke structure is composed of a set S of states or possible worlds, an interpretation π which associates with each state in S a truth assignment to the primitive propositions (i.e., $\pi(s)(p) \in \{\mathbf{true}, \mathbf{false}\}$ for each state $s \in S$ and each primitive proposition p), and equivalence relations $\sim_i$ on S (recall that an equivalence relation is a binary relation which is reflexive, symmetric, and transitive). The relation $\sim_i$ is agent i's possibility relation. Intuitively, $s \sim_i t$ if agent i cannot distinguish state s from state t (so that if s is the actual state of the world, agent i would consider t a possible state of the world). For our purposes, the equivalence relations are obtained by taking a set $\mathcal{L}$ of *local states*, and giving each agent a *view* of the state, that is, a function $L_i : S \rightarrow \mathcal{L}$. We define $s \sim_i t$ iff $L_i(s) = L_i(t)$. In other words, agent i considers the states s and t indistinguishable if he has the same local state at both states.

To interpret explicit knowledge of the form $X_i\varphi$, we assign to each agent a *knowledge algorithm* that the agent can use to determine whether he knows a particular formula. A knowledge algorithm $\mathtt{A}$ takes as inputs a formula of the logic and a local state ℓ in $\mathcal{L}$. It is required to be deterministic and terminate on all inputs, with result "Yes", "No", or "?". An *algorithmic knowledge structure* M is a tuple $(S, \pi, L_1, \ldots, L_n, \mathtt{A}_1, \ldots, \mathtt{A}_n)$, where $L_1, \ldots, L_n$ are the view functions on the states, and $\mathtt{A}_1, \ldots, \mathtt{A}_n$ are knowledge algorithms.[2]

We define what it means for a formula φ to be true (or satisfied) at a state s in an algorithmic knowledge structure M, written $(M, s) \models \varphi$, inductively as follows:

$(M, s) \models p$ if $\pi(s)(p) = \mathbf{true}$

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

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

[2] Halpern *et al.* [1994] introduced algorithmic knowledge in the context of dynamic systems, that is, systems evolving in time. In addition, the knowledge algorithm is allowed to change at every state of the system. Since the issues that interest us do not involve time, we do not consider dynamic systems in this paper. We remark that what we are calling "algorithmic knowledge structures" here are called "algorithmic structures" in [Fagin, Halpern, Moses, and Vardi 1995; Halpern, Moses, and Vardi 1994]. The term "algorithmic knowledge structures" will be used in the second edition of [Fagin, Halpern, Moses, and Vardi 1995].

$(M, s) \models K_i\varphi$ if $(M, t) \models \varphi$ for all t with $s \sim_i t$

$(M, s) \models X_i\varphi$ if $\mathtt{A}_i(\varphi, L_i(s)) =$ "Yes".

The first clause shows how we use the π to define the semantics of the primitive propositions. The next two clauses, which define the semantics of $\neg$ and $\wedge$, are the standard clauses from propositional logic. The fourth clause is designed to capture the intuition that agent i knows φ exactly if φ is true in all the states that i considers possible. The final clause interprets $X_i\varphi$ via agent i's knowledge algorithm. Thus, agent i has algorithmic knowledge of φ at a given state if the agent's algorithm outputs "Yes" when presented with φ and with the agent's local state. (Both the outputs "No" and "?" result in lack of algorithmic knowledge.) As usual, we say that a formula φ is *valid in structure* M if $(M, s) \models \varphi$ for all states $s \in S$; φ is *valid* if it is valid in all structures.

We can think of K_i as representing *implicit knowledge*, facts that the agent implicitly knows, given its information. One can check that implicit knowledge is closed under implication, that is, $K_i\varphi \wedge K_i(\varphi \Rightarrow \psi) \Rightarrow K_i\psi$ is valid, and that an agent implicitly knows all valid formulas, so that if φ is valid, then $K_i\varphi$ is valid. These properties say that agents are very powerful reasoners. What is worse, while it is possible to change some properties of knowledge by changing the properties of the relation $\sim_i$, no matter how we change it, we still get closure under implication and knowledge of valid formulas as properties. They seem to be inescapable features of the possible-worlds approach. This suggests that the possible-worlds approach is appropriate only for "ideal knowers", ones that know all valid formulas as well as all logical consequences of their knowledge, and thus inappropriate for reasoning about agents that are computationally limited. In contrast, X_i represents *explicit knowledge*, facts whose truth the agent can compute explicitly. Since we put no a priori restrictions on the knowledge algorithms, an agent can explicitly know both φ and $\varphi \Rightarrow \psi$ without explicitly knowing ψ, for example.

As defined, there is no necessary connection between $X_i\varphi$ and $K_i\varphi$. An algorithm could very well claim that agent i knows φ (i.e., output "Yes") whenever it chooses to, including at states where $K_i\varphi$ does not hold. Although algorithms that make mistakes are common, we are often interested in knowledge algorithms that are correct. We say that a knowledge algorithm is *sound* for agent i in the structure M if for all states s of M and formulas φ, $\mathtt{A}_i(\varphi, L_i(s)) =$ "Yes" implies $(M, s) \models K_i\varphi$, and $\mathtt{A}_i(\varphi, L_i(s)) =$ "No" implies $(M, s) \models \neg K_i\varphi$. Thus, a knowledge algorithm is sound if its definite answers are correct. If we restrict attention to sound algorithms, then algorithmic knowledge can be viewed as an instance of awareness, as defined by Fagin and Halpern [1988].

3 Randomized knowledge algorithms

Randomized knowledge algorithms arise frequently in the literature (although they have typically not been viewed as knowledge algorithms). In order to deal with randomized algorithms in our framework, we need to address a technical question. Randomized algorithms are possibly nondeterministic; they may not yield the same result on every invocation with the same arguments. Since $X_i\varphi$ holds at a state s if the knowledge algorithm answers "Yes" at that state, this means that, with the semantics of the previous section, $X_i\varphi$ would not be well defined. Whether it holds at a given state depends on the outcome of random choices made by the algorithm. However, we expect the semantics to unambiguously declare a formula either true or false.

We deal with the problem by adding information to the semantic model to resolve the uncertainty. Observe that if the knowledge algorithm $\mathtt{A}$ is randomized, then the answer that $\mathtt{A}$ gives on input (φ, ℓ) will depend on the outcome of coin tosses (or whatever other randomizing device is used by $\mathtt{A}$). We thus turn the randomized algorithm into a deterministic algorithm by supplying it with an appropriate argument. For example, we supply an algorithm that makes random choices by tossing coins a sequence of outcomes of coin tosses. We can now interpret a knowledge algorithm answering "Yes" with probability α at a state by consider the probability of those sequences of coin tosses at the state that make the algorithm answer "Yes".

Formally, we start with an algorithmic knowledge structure $M = (S, \pi, L_1, \dots, L_n, \mathtt{A}_1, \dots, \mathtt{A}_n)$. For simplicity, assume that the randomness in the knowledge algorithms comes from tossing coins. A *derandomizer* is a tuple $v = (v_1, \dots, v_n)$ such that for every agent i, v_i is a sequence of outcomes of coin tosses (heads and tails). There is a separate sequence of coin tosses for each agent rather than just a single sequence of coin tosses, since we do not want to assume that all agents use the same coin. Let V be the set of all such derandomizers. To every randomized algorithm $\mathtt{A}$ we associate a derandomized algorithm $\mathtt{A}^d$ which takes as input not just the query φ and local state ℓ, but also the sequence v_i of i's coin tosses, taken from a derandomizer $(v_1, \dots, v_n)$. A *probabilistic algorithmic knowledge structure* extending $M = (S, \pi, L_1, \dots, L_n, \mathtt{A}_1, \dots, \mathtt{A}_n)$ is a tuple $M' = (S, \pi, L_1, \dots, L_n, \mathtt{A}_1^d, \dots, \mathtt{A}_n^d, \nu)$, where ν is a probability distribution on V and $\mathtt{A}_i^d$ is the derandomized version of $\mathtt{A}_i$. (Note that in a probabilistic algorithmic knowledge structure the knowledge algorithms are in fact deterministic.) Because we do not impose any restriction on the distribution ν, we do not require that the coin be fair or that the tosses be independent. We can in fact capture correlation between the agents' coins by using an appropriate distribution ν.

The truth of a formula is now determined relative to a pair (s, v) consisting of a state s and a derandomizer v. We abuse notation and continue to call these pairs states. The semantics of formulas in a probabilistic algorithmic knowledge structure is a straightforward extension of their semantics in algorithmic knowledge structures. The semantics of primitive propositions is given by π; conjunctions and negations are interpreted as usual; for knowledge and algorithmic knowledge, we have

$$(M', s, v) \models K_i\varphi \text{ if } (M', t, v') \models \varphi \text{ for all } v' \text{ and all } t \text{ such that } s \sim_i t$$

$$(M', s, v) \models X_i\varphi \text{ if } \mathtt{A}_i^d(\varphi, L_i(s), v_i) = \text{"Yes", where } v = (v_1, \dots, v_n).$$

Here, $\mathtt{A}_i^d$ gets v_i as part of its input. $\mathtt{A}_i^d(\varphi, \ell, v_i)$ is interpreted as the output of $\mathtt{A}_i^d$ given that v_i describes the results of the coin tosses. Having the sequence of coin tosses as part of the input allows us to talk about the probability that i's algorithm answers yes to the query φ at a local state ℓ. It is simply $\nu(\{v : \mathtt{A}_i^d(\varphi, \ell, v_i) = \text{"Yes"}\})$. To capture this in the language, we extend the language to allow formulas of the form $\Pr(\varphi) \geq \alpha$, read "the probability of φ is at least α". The semantics of such formulas is straightforward:

$$(M', s, v) \models \Pr(\varphi) \geq \alpha \text{ if } \nu(\{v' : (M, s, v') \models \varphi\}) \geq \alpha.$$

Note that the truth of $\Pr(\varphi) \geq \alpha$ at a state (s, v) is independent of v. Thus, we can abuse notation and write $(M', s) \models \Pr(\varphi) \geq \alpha$. In particular, $(M', s) \models \Pr(X_i\varphi) < \alpha$ (or, equivalently, $(M', s) \models \Pr(\neg X_i\varphi) \geq 1 - \alpha$) if the probability of the knowledge algorithm returning "Yes" on a query φ is less than α, given agent i's local state at state s.

If all the knowledge algorithms used are deterministic, then this semantics agrees with the semantics given in Section 2. To make this precise, note that if $\mathtt{A}$ is deterministic, then $\mathtt{A}^d(\varphi, \ell, v_i) = \mathtt{A}^d(\varphi, \ell, v_i')$ for all $v, v' \in V$. In this case, we abuse notation and write $\mathtt{A}(\varphi, \ell)$.

Proposition 3.1: *Let M' be a probabilistic algorithmic knowledge structure extending M. If all the knowledge algorithms used in M' are deterministic and there are no occurrences of* $\Pr$ *in φ, then $(M', s, v) \models \varphi$ iff $(M, s) \models \varphi$.*

As the next result shows, derandomizers only are necessary to interpret the $\Pr$ and X_i operators. Moreover, it is clear from the semantics that they are not needed to interpret the X_i operators if the knowledge algorithms are all deterministic.

Proposition 3.2: *Let M' be a probabilistic algorithmic knowledge structure extending M. If there are no occurrences of X_i and* $\Pr$ *in φ, then $(M', s, v) \models \varphi$ iff $(M, s) \models \varphi$.*

Propositions 3.1 and 3.2 justify the decision to "factor out" the randomization of the knowledge algorithms into semantic objects that are distinct from the states; the semantics of formulas that do not depend on the randomized choices do not in fact depend on those additional semantic objects.

4 An example from security

As we mentioned in the introduction, an important area of application for algorithmic knowledge is the analysis of cryptographic protocols. In previous work [Halpern and Pucella 2002], we showed how algorithmic knowledge can be used to model the resource limitations of an adversary. We briefly review the framework of that paper here.

Participants in a security protocol are viewed as exchanging messages in the free algebra generated by a set $\mathcal{P}$ of plaintexts and a set $\mathcal{K}$ of keys, over abstract operations $\cdot$ (concatenation) and $\{\!|\,|\!\}$ (encryption). The set $\mathcal{M}$ of messages is the smallest set that contains $\mathcal{K}$ and $\mathcal{P}$ and is closed under encryption and concatenation, so that if m_1 and m_2 are in $\mathcal{M}$ and $\mathsf{k} \in \mathcal{K}$, then $\mathsf{m}_1 \cdot \mathsf{m}_2$ and $\{\!|\mathsf{m}_1|\!\}_{\mathsf{k}}$ are in $\mathcal{M}$. We identify elements of $\mathcal{M}$ under the equivalence $\{\!|\{\!|\mathsf{m}|\!\}_{\mathsf{k}}|\!\}_{\mathsf{k}^{-1}} = \mathsf{m}$. We make the standard assumption that concatenation and encryption have enough redundancy to recognize that a term is in fact a concatenation $\mathsf{m}_1 \cdot \mathsf{m}_2$ or an encryption $\{\!|\mathsf{m}|\!\}_{\mathsf{k}}$.

In an *algorithmic security structure*, some of the agents are participants in the security protocol being modeled, while other agents are *adversaries* that do not participate in the protocol, but attempt to subvert it. The adversary is viewed as just another agent, whose local state contains all the messages it has intercepted, as well as the keys initially known to the adversary, such as the public keys of all the agents. We use $\mathit{initkey}(\ell)$ to denote the set of initial keys known by an agent with local state ℓ and write $\mathsf{recv}(\mathsf{m}) \in \ell$ if m is one of the messages received (or intercepted in the case of the adversary) by an agent with local state ℓ. We assume that the language includes a primitive proposition $\mathsf{has}_i(\mathsf{m})$ for every message m, essentially saying that message m is contained within a message that agent i has received. (We omit the formal definition of containment here; see [Halpern and Pucella 2002].)

Clearly, the adversary may not explicitly know that he has a given message if that message is encrypted using a key that the adversary does not know. To capture these restrictions, Dolev and Yao [1983] gave a now-standard description of capabilities of adversaries. Succinctly, a Dolev-Yao adversary can compose messages, replay them, or decipher them if he knows the right keys, but cannot otherwise "crack" encrypted messages. The Dolev-Yao model can be formalized by a relation $H \vdash_{DY} \mathsf{m}$ between a set H of messages and a message m. (Our formalization is equivalent to many other formalizations of Dolev-Yao in the literature, and is similar in spirit to that of Paulson [1998].) Intuitively, $H \vdash_{DY} \mathsf{m}$ means that an adversary can "extract" message m from a set of received messages and keys H, using the allowable operations. The derivation is defined using the following inference rules:

$$\frac{\mathsf{m} \in H}{H \vdash_{DY} \mathsf{m}} \qquad \frac{H \vdash_{DY} \{\!|\mathsf{m}|\!\}_{\mathsf{k}} \quad H \vdash_{DY} \mathsf{k}^{-1}}{H \vdash_{DY} \mathsf{m}} \qquad \frac{H \vdash_{DY} \mathsf{m}_1 \cdot \mathsf{m}_2}{H \vdash_{DY} \mathsf{m}_1} \qquad \frac{H \vdash_{DY} \mathsf{m}_1 \cdot \mathsf{m}_2}{H \vdash_{DY} \mathsf{m}_2}$$

where k^{-1} represents the key used to decrypt messages encrypted with k.

We can encode these capabilities via a knowledge algorithm $\mathtt{A}^{\mathrm{DY}}$ for the adversary as agent i. The details can be found in [Halpern and Pucella 2002], where it is also shown that this algorithm does indeed capture the Dolev-Yao adversary in the following sense:

Proposition 4.1: [Halpern and Pucella 2002] *If* $M = (S, \pi, L_1, \ldots, L_n, \mathtt{A}_1, \ldots, \mathtt{A}_n)$ *is an algorithmic security structure with an adversary as agent* i *and* $\mathtt{A}_i = \mathtt{A}_i^{\mathrm{DY}}$, *then* $(M, s) \models X_i(\mathsf{has}_i(\mathsf{m}))$ *iff* $\{\mathsf{m} : \mathsf{recv}(\mathsf{m}) \in L_i(s)\} \cup \mathit{initkeys}(\ell) \vdash_{DY} \mathsf{m}$. *Moreover, if* $(M, s) \models X_i(\mathsf{has}_i(\mathsf{m}))$ *then* $(M, s) \models \mathsf{has}_i(\mathsf{m})$.

Proposition 4.1 shows that $\mathtt{A}_i^{\mathrm{DY}}$ is a sound knowledge algorithm, and that it captures the Dolev-Yao adversary.

The Dolev-Yao algorithm is deterministic. It does not capture, for example, an adversary who guesses keys in an effort to crack an encryption. Assume that the key space consists of N keys, and let $\mathit{guesskeys}(n)$ return n of these, chosen uniformly at random. Let $\mathtt{A}_i^{\mathrm{DY}+\mathrm{rg}(n)}$ be the result of modifying $\mathtt{A}_i^{\mathrm{DY}}$ to take random guessing into account (the rg stands for *random guess*). Using $\mathtt{A}_i^{\mathrm{DY}+\mathrm{rg}(n)}$, the adversary gets to work with whatever keys it already had available, all the keys it can obtain using the standard Dolev-Yao algorithm, and the additional n randomly chosen keys returned by $\mathit{guesskeys}(n)$. We omit the straightforward formal details here.

Of course, if the total number N of keys is large relative to n, making n random guesses should not help much. Our framework lets us make this precise.

Proposition 4.2: *Suppose that* $M' = (S, \pi, L_1, \ldots, L_n, \mathtt{A}_1^d, \ldots, \mathtt{A}_n^d, \nu)$ *is a probabilistic algorithmic security structure with an adversary as agent* i *and that* $\mathtt{A}_i = \mathtt{A}_i^{\mathrm{DY}+\mathrm{rg}(n)}$ *Let* K *be the number of distinct keys used in the messages in the adversary's local state* ℓ *(that is, the number of keys used in the messages that the adversary has intercepted at a state* s *with* $L_i(s) = \ell$*). Suppose that* $K/N < 1/2$ *and that* ν *is the uniform distribution on sequences of coin tosses. If* $(M', s, v) \models \neg K_i X_i(\mathsf{has}_i(\mathsf{m}))$, *then* $(M', s, v) \models \Pr(X_i(\mathsf{has}_i(\mathsf{m}))) < 1 - e^{-2nK/N}$. *Moreover, if* $(M', s, v) \models X_i(\mathsf{has}_i(\mathsf{m}))$ *then* $(M', s, v) \models \mathsf{has}_i(\mathsf{m})$.

Proposition 4.2 says that what we expect to be true is in fact true: random guessing of keys is sound, but it does not help much (at least, if the number of keys guessed is a small fraction of the total numbers of keys). If it is possible that the adversary does not have algorithmic knowledge of m, then the probability that it has algorithmic knowledge is low. While this result just formalizes our intuitions, it does show that the probabilistic algorithmic knowledge framework has the resources to formalize these intuitions naturally.

5 Probabilistic algorithmic knowledge

While the "guessing" extension of the Dolev-Yao algorithm considered in the previous section is sound, we are often interested in randomized knowledge algorithms that may sometimes make mistakes. For example, suppose that Alice has in her local state a number $n > 2$. Let prime be a proposition true at state s if and only if the number n in Alice's local state is prime. Clearly, Alice either (implicitly) knows prime or knows $\neg$prime. However, this is implicit knowledge. Suppose that Alice uses Rabin's [1980] primality-testing algorithm to test if n is prime. That algorithm uses a (polynomial-time computable) predicate $P(n, a)$ with the following properties, for a natural number n and $1 \leq a \leq n - 1$:

(1) $P(n,a) \in \{0,1\}$,

(2) if n is composite, $P(n,a) = 1$ for at least $\frac{n}{2}$ choices of a,

(3) if n is prime, $P(n,a) = 0$ for all a.

Thus, Alice uses the following randomized knowledge algorithm $\mathrm{A}_{\mathrm{Alice}}$: when queried about prime, the algorithm picks a number a at random between 0 and the number n in Alice's local state; if $P(n,a) = 1$, it says "No" and if $P(n,a) = 0$, it says "Yes". (It is irrelevant for our purposes what the algorithm does on other queries.)

It is not hard to check that $\mathrm{A}_{\mathrm{Alice}}$ has the following properties: If the number n in Alice's local state is prime, then $\mathrm{A}_{\mathrm{Alice}}$ answers "Yes" to a query prime with probability 1 (and hence "No" to the same query with probability 0). If n is composite, $\mathrm{A}_{\mathrm{Alice}}$ answers "Yes" to a query prime with probability $\leq \frac{1}{2}$ and "No" with probability $\geq \frac{1}{2}$. Thus, if n is composite, there is a chance that $\mathrm{A}_{\mathrm{Alice}}$ will make a mistake, although we can make the probability of error arbitrarily small by applying the algorithm repeatedly.

Randomized knowledge algorithms like this are quite common in the literature. They are not sound, but are "almost sound". The question is what we can learn from such an "almost sound" algorithm. Note that we know the probability that $\mathrm{A}_{\mathrm{Alice}}$ says "Yes" given that n is prime; what we are interested in is the probability that n is prime given that $\mathrm{A}_{\mathrm{Alice}}$ says "Yes". (Of course, n is either prime or not. However, if Alice has to make decisions based on whether n is prime, it seems reasonable for her to ascribe a subjective probability to n's being prime. It is this subjective probability that we are referring to here.)

Bayes' rule tells us that

$$\Pr(n \text{ is prime} \mid \mathrm{A}_{\mathrm{Alice}} \text{ says "Yes"}) = \frac{\Pr(\mathrm{A}_{\mathrm{Alice}} \text{ says "Yes"} \mid n \text{ is prime})\Pr(n \text{ is prime})}{\Pr(\mathrm{A}_{\mathrm{Alice}} \text{ says "Yes"})}.$$

The only piece of information in this equation that we have is $\Pr(\mathrm{A}_{\mathrm{Alice}}$ says "Yes" $\mid n$ is prime$)$. If we had $\Pr(n$ is prime$)$, we could derive $\Pr(\mathrm{A}_{\mathrm{Alice}}$ says "Yes"$)$. However, we do not have that information, since we did not assume a probability distribution on the number in Alice's local state. Although we do not have the information needed to compute $\Pr(n$ is prime $\mid \mathrm{A}_{\mathrm{Alice}}$ says "Yes"$)$, there is still a strong intuition that if $X_i\varphi$ holds, this tells us something about whether the number is prime or not. How can this be formalized?

5.1 Evidence

Intuitively, the fact that $X_i\varphi$ holds provides "evidence" that φ holds. But what is evidence? There are a number of definitions in the literature. They all essentially give a way to assign a "weight" to different hypotheses based on an observation; they differ in exactly how they assign the weight (see

[Kyburg 1983] for a survey). Some of these approaches make sense only if there is a probability distribution on the hypotheses. Since this is typically not the case in the applications of interest to us (for example, in the primality example, we do not want to assume a probability on the input n), we use a definition of evidence given by Shafer [1982] and Walley [1987], which does not presume a probability on hypotheses.

For simplicity, we assume here that the set of hypotheses of interest has the form $\mathcal{H} = \{h_0, \overline{h_0}\}$. The hypothesis $\overline{h_0}$ is the negation of hypothesis h_0. For example, if h_0 is "n is prime", then $\overline{h_0}$ is "n is not prime". We are given a set $\mathcal{O}$ of *observations*, which can be understood as outcomes of experiments that we can make. Assume that for each hypotheses $h \in \mathcal{H}$ there is a probability space $(\mathcal{O}, 2^{\mathcal{O}}, \mu_h)$. Intuitively, $\mu_h(ob)$ is the probability of ob given that hypothesis h holds. While this looks like a conditional probability, notice that it does not require a probability on $\mathcal{H}$. Define an *evidence space* to be a tuple $\mathcal{E} = (\mathcal{H}, \mathcal{O}, \mu_{h_0}, \mu_{\overline{h_0}})$ where $\mathcal{H}$, $\mathcal{O}$, μ_{h_0}, and $\mu_{\overline{h_0}}$ are as above.

For an evidence space $\mathcal{E}$, the weight that the observation ob lends to hypothesis $h \in \mathcal{H}$, written $w_{\mathcal{E}}(ob, h)$ or simply $w(ob, h)$ when $\mathcal{E}$ is understood, is

$$w_{\mathcal{E}}(ob, h) \triangleq \frac{\mu_h(ob)}{\mu_{h_0}(ob) + \mu_{\overline{h_0}}(ob)}.$$

Observe that this measure always lies between 0 and 1, with 1 indicating that the full weight of the evidence goes for hypothesis h. While the weight of evidence w looks like a probability measure (and, in fact, for each fixed h, $w(\cdot, h)$ is a probability measure on $\mathcal{O}$), it should not be interpreted as a probability measure. It is simply a way to assign a weight to hypotheses given observations.

For the primality example, the set $\mathcal{H}$ of hypotheses is $\{\mathsf{prime}, \neg\mathsf{prime}\}$. The observations $\mathcal{O}$ are simply the possible outputs of the knowledge algorithm $\mathtt{A}_{\mathrm{Alice}}$ on the formula prime, namely, $\{\text{"Yes"}, \text{"No"}\}$. From the discussion following the description of the example, it follows that

$$\begin{array}{ll} \mu_{\mathsf{prime}}(\text{"Yes"}) = 1 & \mu_{\mathsf{prime}}(\text{"No"}) = 0 \\ \mu_{\neg\mathsf{prime}}(\text{"Yes"}) \leq \frac{1}{2} & \mu_{\neg\mathsf{prime}}(\text{"No"}) \geq \frac{1}{2}. \end{array}$$

Thus, $w(\text{"Yes"}, \mathsf{prime}) \geq \frac{2}{3}$ and $w(\text{"Yes"}, \neg\mathsf{prime}) \leq \frac{1}{3}$. Intuitively, a "Yes" answer to the query prime provides more evidence for the hypothesis prime than the hypothesis ¬prime. Similarly, $w(\text{"No"}, \mathsf{prime}) = 0$ and $w(\text{"No"}, \neg\mathsf{prime}) = 1$. Thus, an output of "No" to the query prime indicates that the hypothesis ¬prime must hold.

This notion of evidence can be generalized to more than two hypotheses. In fact, it is possible to interpret the weight function w as a prescription for how to update a prior probability on the hypotheses into a posterior probability on those hypotheses, after having considered the observations made. We do not focus on these aspects here; see [Halpern and Pucella 2003] for more details.

5.2 Reliable randomized algorithms

What does a "Yes" answer to a query φ given by an "almost sound" knowledge algorithm tell us about φ? To make this precise, we need to first characterize how reliable the knowledge algorithm is. A randomized knowledge algorithm $\mathtt{A}_i$ is (α, β)*-reliable for* φ *in* M' *(for agent* i*)* if for all states s and derandomizers v,

- $(M', s, v) \models \varphi$ implies $\nu(\{v' : \mathtt{A}_i^d(\varphi, L_i(s), v'_i) = \text{"Yes"}\}) \geq \alpha$, and

- $(M', s, v) \models \neg\varphi$ implies $\nu(\{v' : \mathtt{A}_i^d(\varphi, L_i(s), v_i') = \text{“Yes”}\}) \leq \beta$.

In other words, if φ is true at state s, then an (α, β)-reliable algorithm says “Yes” to φ at s with probability at least α (and hence is right when it answers “Yes” to query φ with probability at least α); on the other hand, if φ is false, it says “Yes” with probability at most β (and hence is wrong when it answer “Yes” to query φ with probability at most β). The primality testing knowledge algorithm is $(1, \frac{1}{2})$-reliable for prime.

We now associate an evidence space over the hypotheses $\{\varphi, \neg\varphi\}$ with a knowledge algorithm $\mathtt{A}_i$. Let $\mathcal{E}_{\mathtt{A}_i,\varphi} = (\{\varphi, \neg\varphi\}, \{\text{“Yes”}, \text{“No”}, \text{“?”}\}, \mu_\varphi, \mu_{\neg\varphi})$. Roughly speaking, $\mu_\varphi(\text{“Yes”})$ (resp., $\mu_\varphi(\text{“No”})$; $\mu_\varphi(\text{“?”})$) is the probability that $\mathtt{A}_i$ says “Yes” (resp., “No”; “?”) to a query φ in states where φ is true, while $\mu_{\neg\varphi}(\text{“Yes”})$ (resp., $\mu_{\neg\varphi}(\text{“No”})$; $\mu_{\neg\varphi}(\text{“?”})$) is the probability that $\mathtt{A}_i$ answers “Yes” (respectively, “No”; “?”) to the query φ in states where $\neg\varphi$ is true. This is actually what the definition reduces to when the output of the knowledge algorithm depends only on whether φ is true or false, and not on other details of the state. (This is in fact typically the case; in particular, it is true of the primality-testing algorithm.) However, in general, we take

$$\begin{aligned}
&\mu_\varphi(\text{“Yes”}) = \inf_{\{(s,v):(M,s,v)\models\varphi\}} \nu(\{v' : \mathtt{A}_i^d(\varphi, L_i(s), v_i') = \text{“Yes”}\}),\\
&\mu_\varphi(\text{“No”}) = \sup_{\{(s,v):(M,s,v)\models\varphi\}} \nu(\{v' : \mathtt{A}_i^d(\varphi, L_i(s), v_i') = \text{“No”}\}),\\
&\mu_\varphi(\text{“?”}) = 1 - \mu_\varphi(\text{“Yes”}) - \mu_\varphi(\text{“No”}).
\end{aligned}$$

Similarly,

$$\begin{aligned}
&\mu_{\neg\varphi}(\text{“Yes”}) = \sup_{\{(s,v):(M,s,v)\models\neg\varphi\}} \nu(\{v' : \mathtt{A}_i^d(\varphi, L_i(s), v_i') = \text{“Yes”}\}),\\
&\mu_{\neg\varphi}(\text{“No”}) = \inf_{\{(s,v):(M,s,v)\models\neg\varphi\}} \nu(\{v' : \mathtt{A}_i^d(\varphi, L_i(s), v_i') = \text{“No”}\}),\\
&\mu_{\neg\varphi}(\text{“?”}) = 1 - \mu_{\neg\varphi}(\text{“Yes”}) - \mu_{\neg\varphi}(\text{“No”}).
\end{aligned}$$

Intuitively, we are taking the worst-case probability of error in each case. Note that if $\mathtt{A}_i$ is (α, β)-reliable in M', then we have $\mu_\varphi(\text{“Yes”}) \geq \alpha$, $\mu_\varphi(\{\text{“No”}, \text{“?”}\}) \leq 1 - \alpha$, $\mu_{\neg\varphi}(\text{“Yes”}) \leq \beta$, and $\mu_{\neg\varphi}(\{\text{“No”}, \text{“?”}\}) \geq 1 - \beta$.

To be able to talk about evidence within the logic, we introduce an operator to capture the evidence provided by the knowledge algorithm of agent i, $\mathrm{Ev}_i(\varphi)$, read “the weight of evidence that i would get for φ if he were to query the knowledge algorithm about φ”. Define

$$(M', s, v) \models \mathrm{Ev}_i(\varphi) \geq \alpha \text{ if } w_{\mathcal{E}}(\mathtt{A}_i^d(\varphi, v_i, L_i(s)), \varphi) \geq \alpha.$$

We can similarly define $(M', s, v) \models \mathrm{Ev}_i(\varphi) \leq \alpha$. Thus, the Ev_i operator captures the evidence towards φ given by the answer of i's knowledge algorithm to a query φ.

We can now capture the relationship between reliable knowledge algorithms and evidence.

Proposition 5.1: *If $\mathtt{A}_i$ is (α, β)-reliable for φ in M', then the following formulas are valid in M':*

$$X_i\varphi \Rightarrow \mathrm{Ev}_i(\varphi) \geq \frac{\alpha}{\alpha + \beta},$$

$$\neg X_i\varphi \Rightarrow \mathrm{Ev}_i(\varphi) \leq \frac{1 - \alpha}{1 - \alpha + 1 - \beta}.$$

In other words, if a randomized knowledge algorithm says "Yes", then that provides evidence for φ being true. The weight of evidence depends on the reliability. Similarly, if the randomized knowledge algorithm says "No", there is less evidence in favor of φ being true.

Proposition 5.1 becomes interesting in the context of well-known classes of randomized algorithms [Motwani and Raghavan 1995]. An **RP** (random polynomial-time) algorithm is a polynomial-time randomized algorithm that is $(\frac{1}{2}, 0)$-reliable. It thus follows from Proposition 5.1 that if $\mathtt{A}_i$ is an **RP** algorithm, then $X_i\varphi \Rightarrow \mathrm{Ev}_i(\varphi) = 1$ and $\neg X_i\varphi \Rightarrow \mathrm{Ev}_i(\varphi) \leq \frac{1}{3}$ are both valid in M'. Under mild assumptions on the distribution ν (essentially, that it never assigns probability 0 to possible events), $\mathrm{Ev}_i(\varphi) = 1 \Rightarrow K_i\varphi$ is valid, so that in fact we have $X_i\varphi \Rightarrow K_i\varphi$, as expected. Similarly, a **BPP** (bounded-error probabilistic polynomial-time) algorithm is a polynomial-time randomized algorithm that is $(\frac{3}{4}, \frac{1}{4})$-reliable. Thus, by Proposition 5.1, if $\mathtt{A}_i$ is a **BPP** algorithm, then $X_i\varphi \Rightarrow \mathrm{Ev}_i(\varphi) \geq \frac{3}{4}$ and $\neg X_i\varphi \Rightarrow \mathrm{Ev}_i(\varphi) \leq \frac{1}{4}$ are both valid in M'.

Notice that the result of Proposition 5.1 talks about the evidence that the knowledge algorithm provides for φ. Intuitively, we might expect some kind of relationship between the evidence for φ and the evidence for $\neg\varphi$. A plausible relationship would be that high evidence for φ implies low evidence for $\neg\varphi$, and low evidence for φ implies high evidence for $\neg\varphi$. Unfortunately, given the definitions in this section, this is not the case. Evidence for φ is completely unrelated to evidence for $\neg\varphi$. Roughly speaking, this is because evidence for φ is measured by looking at the results of the knowledge algorithm when queried for φ, and evidence for $\neg\varphi$ is measured by looking at the results of the knowledge algorithm when queried for $\neg\varphi$. However, there is nothing in the definition of a knowledge algorithm that says that the answers of the knowledge algorithm to queries φ and $\neg\varphi$ need to be related in any way.

A relationship between evidence for φ and evidence for $\neg\varphi$ can be established by considering knowledge algorithms that are somewhat "well-behaved" with respect to negation. Say that a knowledge algorithm $\mathtt{A}$ *respects negation* if $\mathtt{A}(\neg\varphi, \ell) =$ "Yes" if and only if $\mathtt{A}(\varphi, \ell) =$ "No", and $\mathtt{A}(\neg\varphi, \ell) =$ "No" if and only if $\mathtt{A}(\varphi, \ell) =$ "Yes". (Similarly for randomized knowledge algorithms.) This is a natural way to define the behavior of a knowledge algorithm on negated formulas. Our first result shows that for knowledge algorithms that respect negation, reliability for φ is related to reliability for $\neg\varphi$:

Proposition 5.2: *If $\mathtt{A}_i$ respects negation, then $\mathtt{A}_i$ is (α, β)-reliable for φ in M' if and only if $\mathtt{A}_i$ is $(1-\beta, 1-\alpha)$-reliable for $\neg\varphi$ in M'.*

One can check, from the semantics of algorithmic knowledge, that if $\mathtt{A}_i$ respects negation, then $X_i\varphi \Rightarrow \neg X_i\neg\varphi$ and $X_i\neg\varphi \Rightarrow \neg X_i\varphi$ are valid formulas. Combined with Propositions 5.1 and 5.2, this yields the following result:

Proposition 5.3: *If $\mathtt{A}_i$ is (α, β)-reliable for φ in M', and $\mathtt{A}_i$ respects negation, then the following formulas are valid in M':*

$$X_i\varphi \Rightarrow \left(\mathrm{Ev}_i(\varphi) \geq \frac{\alpha}{\alpha+\beta} \wedge \mathrm{Ev}_i(\neg\varphi) \leq \frac{\beta}{\alpha+\beta}\right),$$

$$X_i\neg\varphi \Rightarrow \left(\mathrm{Ev}_i(\neg\varphi) \geq \frac{1-\beta}{1-\alpha+1-\beta} \wedge \mathrm{Ev}_i(\varphi) \leq \frac{1-\alpha}{1-\alpha+1-\beta}\right).$$

6 Conclusion

The first goal of this paper is to define the semantics of algorithmic knowledge in the presence of randomized knowledge algorithms. This is done by essentially derandomizing the knowledge algorithms, supplying them with an extra argument representing the random information (for instance, outcomes of coin tosses if the algorithm makes random choices by tossing coins).

Using Proposition 5.1, we can precisely characterize the contribution of algorithmic knowledge using randomized knowledge algorithms in terms of evidence. Note that this is the evidence provided by a single query to the knowledge algorithm. Of course, it is always possible to increase the reliability of a knowledge algorithm, by essentially iterating it. (Alternatively, we can equally well compute the cumulative evidence for φ provided by k queries to the knowledge algorithm, either directly by constructing an appropriate evidence space (the observations are now sequences of "Yes", "No", and "?" of length k) or by using *Dempster's Rule of Combination* [Shafer 1976]. See [Halpern and Pucella 2003] for a discussion of combining the evidence of a number of observations, as well combining evidence with prior probabilities.) But the question still remains: what does this evidence tell us? To make this point more definite, suppose that we have a $(.999, .001)$-reliable algorithm for φ that answers "Yes" to a query φ. What does this tell us about φ? This issue becomes important when φ is information on which the agent wants to base his actions. To take another example from security, consider an enforcement mechanism used to detect and react to intrusions in a system. Such an enforcement mechanism uses algorithms that analyze the behavior of users and attempt to recognize intruders. While the algorithms may sometimes be wrong, they are typically reliable, in our sense, with some associated probabilities. What actions should the system take based on a report that a user is an intruder?

If we have a probability on the hypotheses, evidence can be used to update this probability. More precisely, as shown in [Halpern and Fagin 1992], evidence can be viewed as a function from priors to posteriors. For example, if the (cumulative) evidence for n being a prime is α and the prior probability that n is prime is β, then the posterior probability of n being prime (that is, the probability of n being prime in light of the evidence) a straightforward application of Bayes' rule tells us that the posterior probability of n being prime is $(\alpha\beta)/(\alpha\beta+(1-\alpha)(1-\beta))$. Therefore, if we have a prior probability on the hypotheses, including the formula φ, then we can decide to perform an action when the posterior probability of φ is high enough. However, what can we do when there is no probability distribution on the hypotheses, as in the primality example at the beginning of this section? The probabilistic interpretation of evidence still lets us guide our decisions. As before, we assume that if the posterior probability of φ is high enough, we will act as if φ holds. The problem, of course, is that we do not have a prior probability. However, the evidence tells us what prior probabilities we must to be willing to assume for the posterior probability to be high enough. For example, a "Yes" from a $(.999, .001)$-reliable algorithm for φ says that as long as the prior probability of φ is at least .01, then the posterior is at least .9. This may be sufficient assurance for an agent to act.

Of course, it is also possible to treat evidence as primitive, and simply decide to go for the hypothesis for which there is more evidence, or for the hypothesis for which evidence is above a certain threshold. It would in fact be of independent interest to study the properties of a theory of decisions based on a primitive notion of evidence. We leave this to future work.

References

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

Dolev, D. and A. C. Yao (1983). On the security of public key protocols. *IEEE Transactions on Information Theory 29*(2), 198–208.

Fagin, R. and J. Y. Halpern (1988). Belief, awareness, and limited reasoning. *Artificial Intelligence 34*, 39–76.

Fagin, R., J. Y. Halpern, Y. Moses, and M. Y. Vardi (1995). *Reasoning about Knowledge*. The MIT Press.

Halpern, J. Y. and R. Fagin (1992). Two views of belief: belief as generalized probability and belief as evidence. *Artificial Intelligence 54*, 275–317.

Halpern, J. Y., Y. Moses, and M. Y. Vardi (1994). Algorithmic knowledge. In *Theoretical Aspects of Reasoning about Knowledge: Proc. Fifth Conference*, pp. 255–266.

Halpern, J. Y. and R. Pucella (2002). Modeling adversaries in a logic for reasoning about security protocols. In *Proceedings of FASec'02*. Available as Royal Holloway Department of Computer Science Technical Report CSD-TR-02-13. To appear in LNCS.

Halpern, J. Y. and R. Pucella (2003). A logic for reasoning about evidence. To appear in *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI'03)*.

Hintikka, J. (1962). *Knowledge and Belief.* Ithaca, N.Y.: Cornell University Press.

Kripke, S. (1963). A semantical analysis of modal logic I: normal modal propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 9*, 67–96. Announced in *Journal of Symbolic Logic 24*, 1959, p. 323.

Kyburg, H. (1983). Recent work in inductive logic. In T. Machan and K. Lucey (Eds.), *Recent Work in Philosophy*, pp. 87–150. Rowman & Allanheld.

Motwani, R. and P. Raghavan (1995). *Randomized Algorithms*. Cambridge University Press.

Paulson, L. C. (1998). The inductive approach to verifying cryptographic protocols. *Journal of Computer Security 6*(1/2), 85–128.

Rabin, M. O. (1980). Probabilistic algorithm for testing primality. *Journal of Number Theory 12*, 128–138.

Ramanujam, R. (1999). View-based explicit knowledge. *Annals of Pure and Applied Logic 96*(1–3), 343–368.

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

Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press.

Shafer, G. (1982). Belief functions and parametric models (with commentary). *Journal of the Royal Statistical Society, Series B 44*, 322–352.

Walley, P. (1987). Belief function representations of statistical evidence. *Annals of Statistics 18*(4), 1439–1465.