# Common Knowledge and Update in Finite Environments. I (Extended Abstract)

Ron van der Meyden

Information Sciences Laboratory
NTT Basic Research Laboratories
3-9-11 Midori-cho Musashino-shi
Tokyo 180 Japan

email: meyden@ntt-20.ntt.jp

**Abstract**

This paper studies the model checking problem for knowledge formulae in the $S5_n$ Kripke structures generated by finite state environments in which states determine an observation for each agent. Agents have perfect recall, and may operate synchronously or asynchronously. Common knowledge formulae are shown to be be intractable, but efficient incremental algorithms are developed for formulae containing only knowledge operators. Connections to knowledge updates and compilation of knowledge based protocols are discussed.

## 1 Introduction

The literature on reasoning about knowledge in distributed and multi-agent systems has dealt primarily with semantic issues [CM86, FHV91, HM90, HV89, PT92], the development of logics and decision procedures for their validity problem [FI86, HM92, HV88a, HV88b, LR86, PR85], and the use of modal languages with knowledge operators to analyze specific coordination tasks [DM90, Had87, Hal87, HZ92, Maz86]. There has been less work on the following question: given a description of a distributed system, how do we efficiently compute the answer to a query about the knowledge of the agents (processors) in a given state? Closely related is the question of update: how should an agent maintain its knowledge of the world, and its knowledge of other agents' knowledge, so that it may efficiently answer such queries.

In this paper we begin the investigation of these questions for a very simple abstract framework.[1] We suppose that some fixed number of agents inhabit an environment with a *finite* number of possible states. In each state every agent makes an observation, which will in general be insufficient to determine that state. The evolution of the state is constrained by a transition diagram,

[1]This paper is the first of a series, to be continued with [Mey93b].

which determines the *runs* of the system, i.e. the valid *finite* sequences of states. We suppose that both the transition diagram and the relationship between states and the observations of each agent are common knowledge to all agents. Though simple, this framework is sufficiently rich to represent many systems of interest, including finite state message passing systems, synchronous or asynchronous, with bounded buffers. The framework is also able to represent most games of imperfect information, including Bridge, Poker, Battleships, Stratego and Kriegspiel.

Since the observations do not determine the state, agents, in general, have only incomplete information about the current state. However, an agent potentially knows more than it learns from its most recent observation. For example, suppose I have been informed by an accomplice that my opponent in a Poker game holds exactly one ace. If I observe him discard one card (receiving another in its place) then even though I am not able to directly observe his hand, I may infer that he now holds at most two aces. The same sort of reasoning applies to knowledge about knowledge. If my opponent knew that I knew he had one ace (my accomplice is a double agent) then he knows that I now know he has at most two aces. Indeed, if it is common knowledge that I cheat, then it is common knowledge that my opponent has at most two aces.

In order to formalize this sort of reasoning, we need to ascribe a precise state of knowledge to the agents. There are a number of natural ways to do this. Following the standard development of knowledge in distributed systems [Hal87], in every run we are required to assign to each agent a local state. In order to exploit prior knowledge, an agent needs to recall its previous observations. We assume the strongest possible model of memory, namely that agents have *perfect recall* of all their prior observations. This, however, still leaves two possible ascriptions of local state, and we study both. The first of these assigns to an agent the sequence of its observations in all previous states. This corresponds to assuming that agents operate synchronously, i.e. with a global clock. The second ascription of local state we consider assigns to an agent its sequence of *distinct* observations. This corresponds to assuming that agents operate asynchronously.

We obtain from each of these local state ascriptions a Kripke structure which may be used to interpret languages with knowledge operators. The propositional modal language we consider in this paper contains an operator for the knowledge of each agent, as well as an operator for the common knowledge of each group of agents. The basic propositions describe the current state of the world. Thus the formulae of this language refer only to the current state, and to the mutual knowledge that agents have about the current state.

The problem we are interested in, then, is how to efficiently compute whether a given formula holds at a given world in this Kripke structure. The question of how to evaluate modal queries in a Kripke structure is the *model checking* problem, which is known to be in polynomial time for the modal language we consider [HM92]. However, this result is not applicable to our problem, since it assumes a *finite* Kripke structure, whereas the Kripke structure associated with an environment may have an infinite number of worlds, corresponding to the runs of the system. Nevertheless, it would appear at first blush that this complexity can be overcome. Given that there exist only a finite number of states of the world, and that our queries ask only about the current state, and agents' mutual knowledge of the current state, it seems reasonable to expect that the number of different states of knowledge an agent may attain in a given environment is also finite.

A surprisingly simple example presented in Section 2 shows that this is not the case. The states of knowledge attained by agents may require arbitrarily deep nestings of modal operators to describe. This suggests that the complexity of model checking knowledge formulae may be high.

Indeed, we show that for formulae involving common knowledge, the model checking problem in our framework is PSPACE complete in the synchronous case and *undecidable* in the asynchronous case. These results hold even with a fixed formula.

On the other hand, formulae not involving common knowledge are less complex. The reason for this is related to a slightly different perspective on our problem. Consider the point of view of an agent in one of our environments. As each new observation is made, the agent is interested in answering the question "Given what I have just seen, and all that I knew before, what do I know now?" That is, the agent is required to *update* its previous knowledge to reflect the new information. Now although the ascription of knowledge assumes that agents have perfect recall, this ascription is *external:* no assumption is made that agents actually maintain their sequence of observations. Any data structure that may be used to give correct answers to the knowledge queries of interest is an adequate representation of knowledge.

Given some constraints on formulae, it turns out that a data structure containing much less information than the agent's sequence of observations suffices as a knowledge representation. We show in Section 3 and Section 4 that provided the formulae we wish to check contain no occurrences of common knowledge operators, and only a bounded number of alternations of knowledge operators, there exists a constant time incremental algorithm for computing the agents' knowledge, which has the property that the data structure incrementally maintained has constant size. An incremental algorithm of this sort exists for both the synchronous semantics and the asynchronous semantics. In particular, these results imply that fixed formulae not involving common knowledge correspond to regular sets of runs, and may be implemented in linear time.

Our incremental algorithms take an approach to the update problem somewhat different from that usually adopted in the literature on knowledge base updates [KM91], which assumes that agents maintain their knowledge as a set of sentences of some language. Our methods are more closely related to the model checking approach to reasoning about knowledge recently advocated by Halpern and Vardi [HV91].

The results of this paper are a first step towards the construction of environments which facilitate the design and prototyping of protocols in distributed systems by providing automated knowledge analysis. In addition, our incremental algorithms may be viewed as a contribution to the issue of compilation of *knowledge based programming languages,* which involves the construction of extensional programs for the repeated evaluation of a fixed knowledge formula. We discuss this application further in Section 5.

The structure of the paper is as follows. Section 2 introduces the formal model and states some preliminary results. Section 3 deals with the synchronous case, and shows that certain classes of knowledge formulae may be tractably checked using an incrementally computed data structure. In Section 4 the asynchronous semantics is introduced, and it is shown how to generalize the results of Section 3 to this case. The relation of our work to compilation in knowledge based programming languages is discussed in Section 5, and Section 6 contains concluding remarks.

# 2 Definitions

Suppose that *Prop* is a set of propositional constants, and that $\mathcal{O}$ is a set of *observations.* An *environment* for $n$ agents with observations $\mathcal{O}$ is a tuple $E = \langle S, I, T, O_1, \ldots, O_n, V \rangle$ such that

1. $S$ is a finite set of *states,*

2. $I$ is a subset of $S$,

3. $T$ is a binary relation on $S$,

4. $O_i$ is a function from $S$ to $\mathcal{O}$, for each $i = 1 \ldots n$, and

5. $V$ is a *valuation* on $S$, i.e. a mapping from $S \times Prop$ to $\{0, 1\}$.

We say that $T$ is the *transition relation* of the environment $E$, and that the $O_i$ are the *observation functions* of $E$. The set $I$ represents the *initial states* of the environment.

If $s \in S$ is a state, the value $O_i(s)$ intuitively represents the information that agent $i$ is able to obtain in state $s$ by direct observation of the environment and its own current local state. The observation functions induce an equivalence relation $R_i$ on the set of states defined by $sR_is'$ when $O_i(s) = O_i(s')$. Intuitively, two states are equivalent with respect to $R_i$ when agent $i$ is unable to distinguish these states by immediate observation. The actual values of the observation functions will play no essential role in our framework, so we could equivalently have defined environments using the equivalence relations $R_i$ instead of the functions $O_i$. We will sometimes use this fact when presenting examples of environments.

**Example 2.1:** Suppose agent 1 sends a message to agent 2, in a system which guarantees that the message will be delivered either immediately, or with a delay of one time step. As agent 1 sends the message, it makes a permanent record of the fact that the message has been sent. Similarly, once the message has arrived, agent 1 makes a permanent record of this fact, which it is able to access at all times thereafter.

We may represent this situation as an environment $E$ as follows. Take the set of states to be $S = \{a, b, c\}$, where the state $a$ represents that the message has not yet been sent by agent 1, the state $b$ represents that the message is in transit, and the state $c$ represents that the message has been received by agent 2. The set of initial states $I$ of the environment consists of the state $a$ only. The transition relation $T$ is the set of tuples $\{\langle a, a\rangle, \langle a, b\rangle, \langle a, c\rangle, \langle b, c\rangle, \langle c, c\rangle\}$.

For the set of observations we choose $\mathcal{O} = \{sent, unsent, rcvd, unrcvd\}$, representing the values of the variables maintained by the two agents. Thus, the observation functions are given by $O_1(a) = unsent$, $O_1(b) = O_1(c) = sent$ for agent 1 and $O_2(a) = O_2(b) = unrcvd$, $O_2(c) = rcvd$ for agent 2. Taking $Prop = \{p\}$ to consist of a single proposition representing that the message has been received, we get the valuation $V$ with $V(x, p) = 1$ if and only if $x = c$. The environment $E$ is represented in Figure 1, where the arrows indicate transitions, and the ovals represent the equivalence classes associated with the observation relations.

Before describing how an environment determines a Kripke structure which may be used to answer queries about an agent's knowledge given a sequence of events, we first discuss Kripke structures in the abstract, and describe a standard construction on Kripke structures that helps to reduce the complexity of the computation of knowledge formulae.

We will deal with formulae in the propositional modal language $\mathcal{L}_n^C$ which has a modal operator $K_i$ for each agent $i$, as well as a modal operator $C_G$ for each set $G$ of two or more agents. Intuitively, the operator $K_i$ refers to the knowledge of the agent $i$, and the operator $C_G$ refers to the common knowledge of the group $G$. More precisely, the formulae of $\mathcal{L}_n^C$ are defined as follows:

1. Each propositional constant $p$ in $Prop$ is a formula.

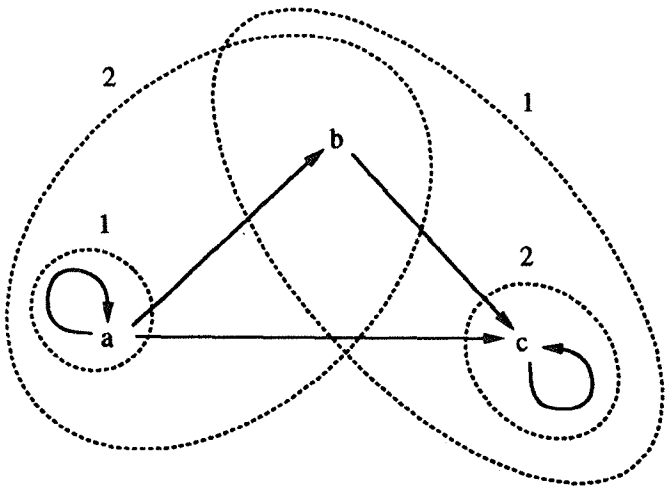


Figure 1: An environment

2. If $\varphi$ and $\psi$ are formulae then $\neg\varphi$ and $\varphi \wedge \psi$ are formulae.

3. If $\varphi$ is a formula then $K_i\varphi$ is a formula for each $i = 1 \ldots n$.

4. If $\varphi$ is a formula then $C_G\varphi$ is a formula for each subset $G$ of $\{1 \ldots n\}$ of cardinality two or more.

The sublanguage $\mathcal{L}_n$ consists of the formulae of $\mathcal{L}_n^C$ which do not contain the operators $C_G$.

The language $\mathcal{L}_n^C$ is interpreted in the following class of models. A *Kripke structure* for $n$ agents is a tuple $M = \langle W, \mathcal{K}_1, \ldots, \mathcal{K}_n, V \rangle$ where $W$ is a set of *worlds*, $\mathcal{K}_i$ is a binary relation on $W$ for each $i$, and $V : W \times Prop \rightarrow \{0,1\}$ is a valuation. If all the relations $\mathcal{K}_i$ are equivalence relations then we say that $M$ is an $S5_n$ structure. Most of the Kripke structures we deal with in this paper will be $S5_n$ structures. The semantics of $\mathcal{L}_n^C$ is given as usual, the only nonstandard point being the truth definition for the common knowledge operator. If $G$ is a set of agents then we say a world $u$ is $G$-accessible from a world $w$ if there exists a sequence of worlds $u_0, u_1, \ldots u_n$ such that $u_o = w$, $u_n = u$ and for all $i = 0 \ldots n-1$ there exists an agent $j \in G$ such that $u_i \mathcal{K}_j u_{i+1}$. The clauses of truth definition are the following.

1. For propositional constants $p \in Prop$, $M, w \models p$ if $V(w,p) = 1$.

2. $M, w \models \varphi_1 \wedge \varphi_2$ if $M, w \models \varphi_1$ and $M, w \models \varphi_2$.

3. $M, w \models \neg\varphi_1$ if not $M, w \models \varphi_1$.

4. $M, w \models K_i\varphi$ if $M, u \models \varphi$ for all worlds $u$ such that $w\mathcal{K}_i u$.

5. $M, w \models C_G\varphi$ if $M, u \models \varphi$ for all worlds $u$ which are $G$-accessible from $w$.

Note that if $M$ is an $S5_n$ structure and $G = \{i\}$ is a singleton set then $u$ is $G$-accessible from $w$ just when $w\mathcal{K}_i u$. Thus, in this case the semantics of $C_G$ collapses to the semantics of $K_i$, which is why we assumed above $G$ contains at least two agents.

The problems we will be concerned with in this paper involve what is known as the *model checking problem*: how to determine whether a formula $\varphi$ holds in a given a world $w$ in a Kripke structure $M$. The following result[2] describes the complexity of this problem when $M$ is a *finite* Kripke structure. In this case we write $||M||$ for the sum of the number of worlds in $M$ and the number of tuples in the relations $\mathcal{K}_i$.

**Proposition 2.2:** If $M$ is a finite $S5_n$ structure and $\varphi \in \mathcal{L}_n^C$ then $M, w \models \varphi$ can be decided in time $O(|\varphi| \cdot ||M||)$.

We now review a result that enables the complexity of model checking to be optimized. It is clear from the truth definition that only worlds $\{1, \ldots, n\}$-accessible from $w$ influence the truth of formulae at the world $w$. More formally, define the *submodel generated by a world* $w$ to be the structure $M' = \langle W', \mathcal{K}_1', \ldots, \mathcal{K}_n', V' \rangle$ with $W'$ equal to the set of worlds $\{1, \ldots, n\}$-accessible from $w$, and the $\mathcal{K}_i'$ and $V'$ the restrictions of the respective components of $M$ to the set $W'$. Then we have the following straightforward generalization of Segerberg's "Generation Theorem" [Che80]:

**Lemma 2.3:** If $\varphi$ is a formula of $\mathcal{L}_n^C$ and $M'$ is the submodel of $M$ generated by the world $w$ then $M, w \models \varphi$ if and only if $M', w \models \varphi$.

We are now ready to describe the Kripke structure associated with an environment. We consider two distinct such structures in this paper. In the present section, we will deal just with the *synchronous* interpretation. The Kripke structure associated with an environment in the asynchronous case will be introduced later, in Section 4.

Suppose $E = \langle S, I, T, O_1, \ldots, O_n, V \rangle$ is an environment. A *run* of the environment $E$ is a *finite* sequence of states $s_1 s_2 \ldots s_m$ such that $s_1 \in I$ and $s_j T s_{j+1}$ for each $j = 1 \ldots m-1$. We write $runs(E)$ for the set of runs of $E$. Note that under the assumption that $T$ contains a cycle, the set of runs is infinite. If $T$ is serial, then every run is the prefix of some strictly longer run. In other words, no run terminates. Environments with arbitrarily long runs will be of greatest interest to us. We adopt the following notational convention: constants $s, t, \ldots$ will always denote *states*; runs will be denoted by $r, r', r_1, \ldots$. If $r$ is a run we write $fin(r)$ for the final state of $r$. When we write $rs$, this will denote a run with final state $s$ and initial portion $r$.

For each agent $i$ we define an equivalence relation on the runs of an environment $E$. The *local state* of agent $i$ in a run $r = s_1 s_2 \ldots s_m$ of an environment $E$ is the sequence $\{r\}_i = O_i(s_1) O_i(s_2) \ldots O_i(s_m)$. That is, the local state of an agent is the sequence of observations made by the agent in the run. We say two runs $r$ and $r'$ are *indistinguishable to agent* $i$, and write $r \sim_i r'$, if $\{r\}_i = \{r'\}_i$. That is, two runs $r = s_1 s_2 \ldots s_m$ and $r' = s_1' s_2' \ldots s_{m'}'$ are indistinguishable to $i$ if $m = m'$ and for for each $l = 1 \ldots m$ we have $O_i(s_l) = O_i(s_l')$. In other words, we assume agents are *synchronous* and have *perfect recall* of their observations.

Now, we obtain for each environment $E$ for $n$ agents the Kripke structure

$$M_E = \langle W, \mathcal{K}_1, \ldots, \mathcal{K}_n, V_E \rangle$$

[2]For $\varphi \in \mathcal{L}_n$ this result is shown in [HM92]. The generalization to $\mathcal{L}_n^C$ is along similar lines. See [HU79] for an exposition of the measures of computational complexity we use in this paper.

with worlds $W = runs(E)$, access relations $\mathcal{K}_i = \sim_i$ for $i = 1 \ldots n$, and valuation $V_E : W \times Prop \rightarrow \{0,1\}$ given by $V_E(r,p) = V(fin(r),p)$. It is clear that $M_E$ is an $S5_n$ structure. If $r$ is a run then we will write $(E,r) \models \varphi$ instead of $M_E, r \models \varphi$. Note that for propositional constants $p$, the relation $(E,r) \models p$ depends only on the the final state of $r$. It follows from this that all formulae of $\mathcal{L}_n^C$ describe the current state, and the agents mutual knowledge of the current state. However, as Example 2.1 shows, environments may be constructed so that the current state contains a limited amount of information about the past.

**Example 2.4:** Consider the environment $E$ for two agents of Example 2.1. The runs of this environment are strings of the form $a^k b^l c^m$, where $k \geq 1$ and $0 \leq l \leq 1$ and $m \geq 0$ are natural numbers. Suppose $r = a^k b^l c^m$ and $r' = a^{k'} b^{l'} c^{m'}$ are runs of the same length. Note that $O_1(x) = O_1(c)$ if and only if $x = c$. Hence if $r \sim_1 r'$ then $m = m'$, and we find that there are precisely two runs $r'$ such that $r \sim_1 r'$, namely those with $k' = k + l - l'$, where $l'$ is either 0 or 1. (One of these runs is just $r$ itself.) A similar argument shows that there exist precisely two runs $r'$ such that $r \sim_2 r'$. Thus, the component generated by the runs of length $n$ of the Kripke structure $M_E$ is described by the sequence

$$ac^{n-1} \sim_1 abc^{n-2} \sim_2 a^2c^{n-2} \sim_1 a^2bc^{n-3} \sim_2 \ldots \sim_2 a^{n-1}c \sim_1 a^{n-1}b \sim_2 a^n$$

where we have omitted the relations holding between each run and itself. It follows that $(E, ac^{n-1}) \models (K_2K_1)^j p$ holds for all $j < n-1$, but not for $j \geq n-1$. In particular, we see that even if the message is delivered immediately, the fact of its delivery never becomes common knowledge[3].

This example illustrates a number of points about the Kripke structures $M_E$. First, the structure $M_E$ may have an infinite number of worlds. This means that Proposition 2.2 does not directly apply to the model checking problem in $M_E$ when $E$ is an environment in which the relation $T$ contains a cycle. On the other hand, note that the submodel of $M_E$ generated by a run $r$ contains only runs of the same length as $r$. Since the set $S$ of states is finite, there are only finitely many such runs, in fact at most $|S|^{length(r)}$. It follows directly from this that all formulae are decidable, and Proposition 2.2 may be used to obtain an exponential time algorithm. However, an improvement on this is possible. Rather than construct the complete submodel generated by a run, we can create runs only as they are needed, and reuse space after we are done with them. This yields the following upper bound.

**Proposition 2.5:** The set $\{ \langle E, r, \varphi \rangle \mid \varphi \in \mathcal{L}_n^C,\ r \in runs(E) \text{ and } (E,r) \models \varphi \}$ is decidable in polynomial space.

The following result shows that this upper bound is the best possible.

**Theorem 2.6:** There exists an environment $E$ for two agents and a propositional constant $p$ such that the set $\{r \mid (E,r) \models C_{\{1,2\}}p\}$ is PSPACE hard.

The proof of Theorem 2.6 is by an encoding of space bounded deterministic Turing machine computations. Consider the even numbered runs in the sequence of runs of Example 2.4. The symbol $b$ in these runs moves progressively from left to right, much like the head of a Turing

[3] A closely related example has previously been used by Halpern and Moses [HM90] to make this point.

machine. The hardness proof exploits this resemblance: runs correspond to configurations of a Turing machine, and a single step through the derivation relation of this machine corresponds to taking two steps in the Kripke structure, first to an intermediate configuration through the relation $\sim_1$ and then through the relation $\sim_2$.

# 3 Bounded Alternation Formulae

Theorem 2.6 shows that model checking common knowledge formulae may be highly complex. For certain formulae containing only knowledge operators the situation is better. Define the *alternation depth* of a formula in $\mathcal{L}_n$ to be the number of alternations of distinct knowledge operators in the formula. We will show in this section how to incrementally maintain a constant size data structure that makes it possible to efficiently model check formulae in $\mathcal{L}_n$ of *bounded alternation depth*. In particular, we will see that it is not necessary to maintain the history in this case.

The data structure we will use is defined as follows. Suppose $S$ is a fixed set of states equipped with a valuation function $V : S \times Prop \to \{0,1\}$. For numbers $k \geq 0$ and agents $i$ we define by mutual recursion the set $\mathcal{T}_k$ of *k-trees over* $S$, the set $\mathcal{T}_{k,i}$ of *i-objective k-trees over* $S$, and the set $\mathcal{F}_{k,i}$ of *i-objective forests over* $S$. Define $\mathcal{T}_0$ to be the set $S$, and also define $\mathcal{T}_{0,i} = S$ for each agent $i$. Once $\mathcal{T}_{k,i}$ has been defined, let $\mathcal{F}_{k,i}$ be the set of all subsets of $\mathcal{T}_{k,i}$. Now, define $\mathcal{T}_{k+1}$ to be the set of all tuples of the form $\langle s, U_1, \ldots, U_n \rangle$, where $s \in S$ and $U_i$ is in $\mathcal{F}_{k,i}$ for each $i = 1 \ldots n$. Finally, for each $i$ let $\mathcal{T}_{k+1,i}$ be the set of such tuples where $U_i$ is the empty set. Intuitively, the state $s$ represents the actual state of the world, and for each $i$ the set $U_i$ represents the knowledge of agent $i$. Note that each component $U_i$ in a 1-tree is simply a set of states, representing agent $i$'s knowledge about the world. For higher $k$ the set $U_i$ represents agent $i$'s knowledge both about the world and other agents' knowledge, up to alternation depth $k$.

The elements of $\mathcal{T}_k$ may be regarded as trees of height $k$, with the edges labelled by agents and vertices labelled by states in $S$. If $w \in \mathcal{T}_k$ we write $root(w)$ for the state $s$ occurring as the first component of the tuple $w$, or for $w$ itself in case $k = 0$. If $w = \langle s, U_1, \ldots, U_n \rangle$ and $v$ is an element of $U_i$, then we say that $v$ is an $i$-child of $w$. Note that no $i$-child of any vertex has an $i$-child itself. It is easily shown that if $exp(x,k)$ is the function defined by $exp(x,0) = x$ and $exp(x,k+1) = x2^{exp(x,k)}$ then $k$-trees over $S$ have size less than $C_k = exp(n \log(|S|), k)$. An element $U$ of $\mathcal{F}_{k,i}$ may be regarded as a forest of $i$-objective $k$-trees. We define the *i-objectification* function $obj_i$, that for each $k$ maps $k$-trees to $i$-objective $k$-trees, as follows. When $w \in \mathcal{T}_0$, we define $obj_i(w) = w$. When $k > 0$, $obj_i$ maps the $k$-tree $\langle s, U_1, \ldots, U_n \rangle$ to the $k$-tree obtained by replacing $U_i$ by the empty set.

By viewing a $k$-tree $w$ as tree-like Kripke structure, it is possible to define a truth relation $w \models \varphi$, meaningful when $\varphi$ is a formula of $\mathcal{L}_n$ of alternation depth less than or equal to $k$. The worlds of this Kripke structure are just the vertices of the tree $w$. If a vertex $v$ of $w$ is labelled by a state $s$, then we take the proposition $p$ to be true at $v$ just when $V(s,p) = 1$, where $V$ is the valuation on $S$. The appropriate accessibility relations are different from the $i$-child relations, however. We define the accessibility relation $\mathcal{K}_i$ to be the smallest relation such that each vertex may access its $i$-children, and all $i$-siblings of a vertex may access each other. This makes the accessibility relations *transitive* and *euclidean*, but not equivalence relations, since an $i$-child cannot access its parent.[4] We define $w \models \varphi$ to hold just when $\varphi$ is true at the root of this

[4]The Kripke structures we have defined here are very similar to the *knowledge trees* used by Halpern and Vardi

tree-like Kripke structure.

We now state a result showing that $k$-trees have enough structure to represent the set of all formulae of alternation depth $k$ or less which hold at a given world in an arbitrary $S5_n$ structure. Suppose $M = \langle W, \mathcal{K}_1, \ldots, \mathcal{K}_n, V_M \rangle$ is an $S5_n$ structure and let $f$ be a function mapping the worlds of $M$ to the set of states $S$. We assume $f$ preserves the valuation, i.e. for all worlds $w \in W$ and propositions $p$, we have $V_M(w,p) = V(f(w),p)$, where $V$ is the valuation on $S$. For $k \geq 0$ define the functions $F_k : W \to \mathcal{T}_k$ as follows. If $k = 0$, then define $F_k(w) = f(w)$. If $k > 0$, define $F_k(w) = \langle f(w), U_1, \ldots, U_n \rangle$, where $U_i = \{obj_i(F_{k-1}(v)) \mid w\mathcal{K}_i v\}$.

**Proposition 3.1:** If $\varphi$ is a formula of $\mathcal{L}_n$ of alternation depth less than or equal to $k$ then for all worlds $w$ of $M$ we have $M, w \models \varphi$ if and only if $F_k(w) \models \varphi$.[5]

Let us now specialize this result to the Kripke structure $M_E$ corresponding to an environment $E = \langle S, I, T, O_1, \ldots, O_n, V \rangle$ with observations $\mathcal{O}$. We first take the $k$-trees $\mathcal{T}_k$ to be constructed over the states $S$ of this environment, and the valuation on $S$ to be the valuation $V$ of $E$. Next, choose the function $f$ to be the the final state mapping *fin* from the set of worlds $runs(E)$ of $M_E$ to $S$. For the remainder of this section, we take $F_k$ to be the mappings that result from these choices. By the definition of $M_E$, the mapping *fin* preserves the valuation. Thus, we obtain the following corollary of Proposition 3.1.

**Corollary 3.2:** For every run $r$ of $E$ and sentence $\varphi$ of $\mathcal{L}_n$ of alternation depth less than or equal to $k$, we have $(E, r) \models \varphi$ if and only if $F_k(r) \models \varphi$.

Since the size of $F_k(r)$ is independent of the length of $r$, this result shows that for sufficiently long runs it suffices to compute in a structure which is smaller than the component of $M_E$ generated by the run. Of course, if it is necessary to first construct this component of $M_E$ in order to compute $F_k(r)$ this is no gain. However, there is a more efficient, incremental way to construct $F_k(r)$.

Define for each number $k \geq 0$ the function $G_k : \mathcal{T}_k \times S \to \mathcal{T}_k$, by induction on $k$. The definition of $G_k$ will be by mutual recursion with the functions $H_{k,i} : \mathcal{F}_{k,i} \times \mathcal{O} \to \mathcal{F}_{k,i}$, where $i$ is an agent and $k \geq 0$. We define $G_0(w,s) = s$. Once $G_k$ has been defined, we define for each $i$ the function $H_{k,i}$ by taking $H_{k,i}(U,o)$ to be the set of $k$-trees $G_k(v,s)$ where $v \in U$, there exists a transition of $E$ from $root(v)$ to $s$, and $O_i(s) = o$. Using the functions $H_{k,i}$, we may now define $G_{k+1}$ by the equation

$$G_{k+1}(\langle s, U_1, \ldots, U_n \rangle, s') = \langle s', H_{k,1}[U_1, O_1(s')], \ldots, H_{k,n}[U_n, O_n(s')] \rangle.$$

Note that if $U$ is the empty set then $H_{k,i}(U,o)$ is also empty. It follows from this that $G_k$ maps $k$-trees to $k$-trees, and also maps $i$-objective $k$-trees to $i$-objective $k$-trees, so these definitions are proper.

Intuitively, if agent $i$'s state of knowledge is (partially) represented by the the set of $k$-trees $U$, then $H_{k,i}(U,o)$ computes the agent's revised state of knowledge after it makes the observation $o$. Note that in the special case $k = 0$, we have that $U$ is a set of states and

$$H_{0,i}(U,o) = \{t \in S \mid \exists s \in U[sTt \text{ and } O_i(t) = o]\}. \tag{1}$$

[HV88a] to show decidability and completeness of logics for knowledge and synchronous time.

[5] It suffices that the access relations $\mathcal{K}_i$ be transitive and euclidean, so Proposition 3.1 also applies to $K45_n$, the logic of belief.

Thus the update computation represented by the mapping $H_{0,i}$ is closely related to (though not subsumed by) Winslett's [Win88] Possible Model Approach to the semantics of updates, and to Katsuno and Mendelzon's [KM91] generalization of this approach. Similar ideas occur in Rosenschein's discussion of knowledge in robotics [Ros85], and in the standard theory of state identification in finite state automata [Koh70]. The following result shows how $G_k$ may be used to compute $F_k$ incrementally.

**Lemma 3.3:** For every run $rs$ of $E$ and every number $k$, $F_k(rs) = G_k(F_k(r), s)$.

If $r$ is a run of length one then $F_k(r)$ may be straightforwardly computed: if $r$ consists of the single state $s$ then $F_k(r)$ is the $k$-tree with root $s$ and an $i$-child $F_{k-1}(s')$ for each state $s'$ with $O_i(s') = O_i(s)$. Combining this with Corollary 3.2, Lemma 3.3 and Proposition 2.2, we obtain the following result.

**Theorem 3.4:** For formulae $\varphi$ of alternation depth $k$ or less, $(E, r) \models \varphi$ can be decided in time $O(C_k \cdot (|r| + |\varphi|))$.

In particular, we see that to model check formulae of alternation depth bounded by a constant it suffices to maintain a data structure of constant size. Note that this result trades in the "exponential time" dependence on both formula size and length of run of Proposition 2.5 for a linear time dependence on both formula size and length of run, but at the cost of non-elementary dependence on alternation depth. Thus for high alternation depth formulae this approach is unlikely to be practicable without further optimization. When dealing with very deep formulae in short runs, the algorithm of Proposition 2.5 may well be more efficient.

A special case of Theorem 3.4 worth noting is when $n = 1$, i.e. there is just a single agent. Since all formulae of $\mathcal{L}_1$ are of alternation depth one or zero, it suffices to work with 1-trees, which are tuples of the form $\langle s, U \rangle$ where $s$ is a state and $U$ is a set of states. The set $U$ is updated according to equation (1). Thus the theory of this section may be viewed as a generalization of standard accounts of update in the single agent context. We note that our results make explicit the validity of this update computation under the assumption of perfect recall, which is not generally considered in this work.

Theorem 3.4 also subsumes the situation in which we wish to check a *fixed* formula of $\mathcal{L}_n$ in runs of increasing length, for which we obtain the following.

**Corollary 3.5:** Let $\varphi$ be a fixed formula of $\mathcal{L}_n$ and let $E$ be a fixed environment. Then the set $\{r \in \mathit{runs}(E) \mid (E, r) \models \varphi\}$ is regular.

In general, the finite state automaton obtained for a fixed formula will not be minimal, so further optimization is possible. We will discuss this possibility in more detail elsewhere.

# 4 Asynchronous Environments

We now consider an alternate semantics for environments, in which we do not assume that agents are synchronized. Instead of assuming that an agent is aware of every transition made, as before, we now assume that an agent notices transitions only when these lead to changes in its observation.

Define $\{r\}_i^a$, agent $i$'s *asynchronous local state* in the run $r$, to be the sequence of *distinct* observations of agent $i$. That is, if $r$ is a run consisting of a single state $s$ then we define $\{r\}_i^a = O_i(s)$, and for runs of the form $rs$ we define

$$\{rs\}_i^a = \{r\}_i^a \text{ when } O_i(s) = O_i(\mathit{fin}(r))$$

$$\{rs\}_i^a = \{r\}_i^a O_i(s) \text{ when } O_i(s) \neq O_i(\mathit{fin}(r)).$$

Using this function we may now define two runs $r, r'$ to be *asynchronously indistinguishable* to agent $i$, written $r \approx_i r'$, if $\{r\}_i^a = \{r'\}_i^a$. Notice that the agent still has *perfect recall* under this definition, but it is no longer assumed to be able to keep track of time. We write $M_E^a$ for the Kripke structure obtained from an environment $E$ by replacing the relations $\sim_i$ in $M_E$ by the relations $\approx_i$. We write $(E^a, r) \models \varphi$ when the formula $\varphi$ holds at run $r$ in $M_E^a$.

**Example 4.1:** We reinterpret the environment of Example 2.1 with respect to the asynchronous semantics. Consider first the asynchronous indistinguishability relation $\approx_1$ for agent 1. If $r$ is a run of the form $a^k$, then $\{r\}_1^a = \mathit{unsent}$. For all other runs $r$, of the form $a^k b^l c^m$ with $k \geq 1$, $0 \leq l \leq 1$ and $l + m \geq 1$, we have $\{r\}_1^a = \mathit{unsent} \cdot \mathit{sent}$. Thus, there are precisely two equivalence classes of runs with respect to the relation $\approx_1$. Similarly, it may be shown that the relation $\approx_2$ partitions the set of runs into precisely two classes: the runs of the form $a^k b^l$ and the runs of the form $a^k b^l c^m$ with $m \geq 1$. Note that none of these equivalence classes contains a run $a^k b^l c^m$ with $l > 1$: although the agents are unable to tell the time, they still know that the message is delayed by at most one time step.

We showed in Example 2.4 that in the synchronous case, the agents may acquire, with time, increasing levels of mutual knowledge about the proposition $p$: 'The message has been delivered.' In contrast, in the asynchronous interpretation, the formula $C_{\{1,2\}}\neg\mathcal{K}_1 p$ is true in all runs: it is always common knowledge that agent 1 does not know the message has been delivered.

In the synchronous case, we were able to establish the decidability of all formulae in $\mathcal{L}_n^C$ by noting the substructure of $M_E$ generated by any run is finite. As Example 4.1 shows, this is no longer the case when environments are interpreted asynchronously: in this example all runs generate the entire structure. Indeed, we may show the following:

**Theorem 4.2:** There exists an environment $F$ for two agents and a propositional constant $p$ such that it is undecidable, given a run $r$, to determine if $(F^a, r) \models C_{\{1,2\}} p$.

The proof of this result is by reduction from the Halting problem for Turing machines. It uses the fact that by modifying any environment $E$ by adding a one bit clock whose state alternates with every transition, and making this clock observable to all agents, one obtains an environment $E'$ for which the asynchronous structure $M_{E'}^a$ is isomorphic to the synchronous structure $M_E$. Part of the environment $F$ is the environment $E'$ so obtained from the environment $E$ constructed in the proof of Theorem 2.6, which simulates space bounded computations. The remainder of the environment $F$ uses asynchrony to guess the amount of space required by the computation of the Turing machine on the given input.

That common knowledge formulae are decidable in synchronous environments, but undecidable in asynchronous environments, might lead one to expect a similar jump in complexity for formulae

not containing common knowledge. In fact, the situation here is not much worse than in the synchronous case.

We will describe an incremental computation for formulae in $\mathcal{L}_n$ which generalizes the techniques of the previous section to the asynchronous case. The basic data structure we employ will again be $k$-trees. The sets $\mathcal{T}_k$ are defined as before. For each number $k \geq 0$ let the functions $F_k^a : runs(E) \longrightarrow \mathcal{T}_k$ be defined inductively by $F_0^a(r) = fin(r)$ and, for $k > 0$, $F_k^a(r) = \langle fin(r), U_1, \ldots, U_n \rangle$ where $U_i = \{obj_i(F_{k-1}^a(r')) \mid r \approx_i r'\}$. This definition is almost identical to the definition of the function $F_k$ in Section 3, the only difference being that we have replaced the accessibility relations $\sim_i$ of $M_E$ by the relations $\approx_i$ of $M_E^a$. Thus, we obtain the following consequence of Proposition 3.1, analogous to Corollary 3.2.

**Corollary 4.3:** For every run $r$ of $E$ and formula $\varphi$ of $\mathcal{L}_n$ of alternation depth less than or equal to $k$, we have $(E^a, r) \models \varphi$ if and only if $F_k^a(r) \models \varphi$.

This result shows that instead of checking the infinite structure $M_E^a$, we may check the finite structures $F_k^a(r)$. To complete the generalization of the results of the previous section, we need to show how to compute the functions $F_k^a$. In the synchronous case, the runs indistinguishable to agent $i$ from a run of the form $rs$ are all of the form $r's'$, where $r \sim_i r'$ and $O_i(s) = O_i(s')$. This is no longer the case under the asynchronous semantics, where we may have $rs \approx_i r's_1s_2 \ldots s_l$, with $r \approx_i r'$ and $O_i(s_1) = O_i(s_2) = \ldots = O_i(s_l)$. This means that a direct application of the previous incremental approach does not work - we may need to look further than the "next state" in computing $F_k^a(rs)$ from $F_k^a(r)$. Another complication is that the number $l$ is possibly unbounded. Nevertheless, an incremental characterization of the the functions $F_k^a$ is still possible. Briefly, what is required is to repeatedly apply the one-step incremental computation while the runs examined may be extended without changing the final observation. This process is terminated as soon as no new information is extracted.

We define for each $k \geq 0$ a function $G_k^a : \mathcal{T}_k \times S \longrightarrow \mathcal{T}_k$ which will play a role analogous to that of the function $G_k$ of the previous section. The definition is by means of a mutual recursion with the functions $H_{k,i}^a : \mathcal{F}_{k,i} \times \mathcal{O} \longrightarrow \mathcal{F}_{k,i}$, where $i$ is an agent and $k \geq 0$. Like the function $H_{k,i}$ used in the synchronous case, the function $H_{k,i}^a$ maps a state of knowledge $U$ of agent $i$ and an observation $o$ to the agent's new state of knowledge $H_{k,i}(U, o)$ upon making the observation $o$. Thus, as above we define $G_0^a(w, s) = s$, and

$$G_{k+1}^a(\langle s, U_1, \ldots, U_n \rangle, s') = \langle s', H_{k,1}^a[U_1, O_1(s')], \ldots, H_{k,n}^a[U_n, O_n(s')] \rangle.$$

To define the function $H_{k,i}^a$, we assume that $G_k^a$ has been defined. The definition makes use of an auxiliary function $J_{k,i,o} : \mathcal{F}_{k,i} \to \mathcal{F}_{k,i}$. Intuitively, this function computes the update agent $i$ would perform on making observation $o$ if it knew that either zero or one steps have been taken through the transition relation. Thus, if $U \in \mathcal{F}_{k,i}$, we put

$$\begin{aligned} J_{k,i,o}(U) \;=\; & \{w \in U \mid O_i(root(w)) = o\} \cup \\ & \{G_k^a(w, s) \mid w \in U \text{ and } root(w) T s \text{ and } O_i(s) = o\} \end{aligned}$$

The following result states some basic properties of $J_{k,i,o}$.

**Lemma 4.4:**

(1) The function $J_{k,i,o}$ is monotonic, i.e. if $U_1 \subseteq U_2$ then $J_{k,i,o}(U_1) \subseteq J_{k,i,o}(U_2)$.

(2) For all $U$, $J_{k,i,o}(U) \subseteq J^2_{k,i,o}(U)$.

It follows from Lemma 4.4 that for all $U$ we have $J_{k,i,o}(U) \subseteq J^2_{k,i,o}(U) \subseteq J^3_{k,i,o}(U) \subseteq \ldots$. Since these values are subsets of the finite set $\mathcal{T}_{k,i}$, this implies that there exists some $m$ for which $J^m_{k,i,o}(U) = J^{m+1}_{k,i,o}(U)$. We define $H^a_{k,i}(U,o)$ to be this fixpoint value $J^m_{k,i,o}(U)$. This completes the definition of the $H^a_{k,i}$ and $G^a_k$. We may now establish the following recursion relation, which gives the incremental way to compute $F^a_k$.

**Proposition 4.5:** For every run $rs$ we have $G^a_k(F^a_k(r), s) = F^a_k(rs)$.

We have now seen how to compute $F^a_k(rs)$ from $F^a_k(r)$. It remains to show how to calculate $F^a_k(r)$ when $r$ is a run of length one. In the synchronous case, this was straightforward, since it required only the examination of the states and observation relation. In the asynchronous case, this no longer suffices, and we need a fixpoint computation similar to that just described. For each number $k \geq 0$ define the function $g^a_k : S \longrightarrow \mathcal{T}_k$ by $g^a_0(s) = s$ and for $k > 0$ by $g^a_k(s) = G^a_k(u_k(s), s)$. Here $u_k(s)$ is the $k$-tree with root $s$ such that $v$ is an $i$-child of $u_k(s)$ if and only if $v = obj_i(g^a_{k-1}(s'))$ for some state $s'$ with $O_i(s') = O_i(s)$.

**Lemma 4.6:** For all $s \in S$ we have $g^a_k(s) = F^a_k(s)$.

Combining Corollary 4.3, Lemma 4.6, Proposition 4.5 and Proposition 2.2, we obtain the following results, analogous to Theorem 3.4 and Corollary 3.5 respectively.

**Theorem 4.7:** For each sentence $\varphi$ of $\mathcal{L}_n$ of alternation depth $k$ or less, $(E^a, r) \models \varphi$ can be decided in time $O(C_k \cdot (|r| + |\varphi|))$.

**Corollary 4.8:** Let $\varphi$ be a fixed formula of $\mathcal{L}_n$ and let $E$ be a fixed environment. Then the set $\{r \in runs(E) \mid (E^a, r) \models \varphi\}$ is regular.

# 5 Compiling Knowledge Based Programs

In a number of papers [DM90, Had87, HM90, HZ92, Maz86] it has been shown that knowledge formulae may provide necessary and sufficient conditions for the achievement of a variety of coordination tasks. This analysis enables the formulation of *knowledge based protocols* [HF89] for these tasks, in which knowledge formulae are used as preconditions for actions. Knowledge based protocols have been shown to provide perspicuous descriptions of standard protocols, and to elucidate the common ideas underlying apparently different protocols.

Given a precise description of the environment in which a knowledge based protocol runs, e.g. a specification of the assumptions concerning message delivery and loss, there exists an equivalent standard protocol, obtained by replacing the knowledge preconditions by appropriate standard computations based on the state of the processor. The determination of these standard computations has so far been carried out on a case by case basis. For example, Dwork and Moses [DM90] establish that in protocols solving a distributed agreement problem in the presence of processor failures, the truth of a particular formula asserting a type of common knowledge of a group of agents is a necessary and sufficient condition for agreement. They show how a test for the truth of this formula may be implemented, given a variety of assumptions on the nature of

failures. For omission failures it turns out that the test may be implemented in time polynomial in the number of processors, which number also bounds the length of the relevant runs. However, for arbitrary (Byzantine) failures, the test is shown to be NP hard, with an upper bound of PSPACE.

This type of analysis has shown the appropriateness of knowledge as a level of abstraction for the design of distributed systems. Consequently, the possibility of high level programming languages with explicit operators for knowledge is a current topic of investigation [MK93]. In order to permit the execution of knowledge based protocols, such a language would have to support the compilation of knowledge tests into standard functions of the state of the processor. Our work in this paper may be viewed as initial foray into the feasibility of this kind of compilation. Of particular interest in this regard are the results concerning evaluation of a fixed formula in runs of increasing length, which is precisely the type of operation required in knowledge based protocols. It is encouraging that fixed knowledge formulae may be implemented by finite state automata (Corollary 3.5, Corollary 4.8).

However, there remains a gap between our formalization and even the simplest type of knowledge based protocol. In knowledge based protocols, the set of successors of a given state depends not just on information local to that state, but also on the knowledge of the agents, since knowledge may constrain action. This means that the set of runs generated by a knowledge based protocol is generally smaller than the set of runs generated by the underlying transition system obtained from the agents' possible actions. Since the knowledge based protocol being run is assumed to be common knowledge, agents may take such interactions into account in their reasoning. Our model is unable to represent this interdependence of knowledge and action. Another limitation of our formulation is its finite state nature. The specification of even such simple protocols as the Alternating Bit Protocol requires an infinite input stream, hence an infinite state space. Nevertheless, given the simplicity of our framework, the results of this paper would seem to provide a lower bound on the general problem.

In this respect the high complexity we have found for common knowledge formulae (Theorem 2.6, Theorem 4.2) is somewhat disappointing. Common knowledge is known to be a prerequisite for simultaneous action, and variants of common knowledge have been found to play a central role in many other types of coordination [HM90, PT92]. Thus compilation of common knowledge formulae is likely to be a crucial requirement of knowledge based programming. The PSPACE hardness of common knowledge queries in synchronous environments shows that one cannot always expect to find efficient implementations for tests involving common knowledge. In the asynchronous case the situation is worse: here the undecidability result shows that compilation into even a terminating program is in general impossible.

However, some caution is required in interpreting these results. The environments constructed in the proofs of Theorem 2.6 and Theorem 4.2 are very unnatural. Although limited in some respects, the class of environments studied in this paper is too broad in other respects. It is not clear whether these results apply to more reasonable classes of environments, such as message passing systems. Further, in many cases common knowledge is not attainable, and various weakenings of common knowledge suffice for the coordination tasks of interest.

Related to the question of compilation of knowledge based programming formalisms is the work of Rosenschein and Kaelbling [Ros85, RK86], which deals with the verification of externally ascribed knowledge properties of situated machines. However, while the language of [RK86] permits the decomposition of a machine into a number of components, each of which is treated as an

agent, the focus is on agents' knowledge of the world rather than nestings of knowledge operators.

Another strand of work on knowledge based programming languages is Shoham's [Sho93] proposal to develop *Agent Oriented Programming Languages.* These are a specialization of message passing formulations of object oriented programming, in which objects take the form of agents, which have a variety of mental states, including beliefs and commitments. These mental states are revised as messages are sent and received. However, instead of the external ascription of knowledge we have studied in the present paper, belief in Shoham's model is *internally* ascribed: an agent believes just what is recorded in its database. Furthermore, update in Shoham's framework does not make use of any semantic knowledge, and default assumptions are applied: an agent maintains its beliefs about the world, and its beliefs about other agents beliefs, until it learns contradictory information [IS92]. We note that in our model no such persistence is assumed: indeed, in the synchronous case, an agents' knowledge may change with each tick of the clock, since it must be true in a non-deterministically changing world. It would be interesting to find semantic assumptions justifying belief persistence.

# 6 Conclusion

We have presented in this paper a model-theoretic approach to computing knowledge formulae in finite environments. An alternate approach would be to formalize an environment as a theory in a logic with modal operators for knowledge and time, and then compute knowledge by deduction in this logic. Complete axiomatizations of logics of knowledge and time in systems with perfect recall are known in both the synchronous case [HV86, HV88a] and the asynchronous case [Mey93a]. However, these results are for languages not containing common knowledge operators. For both synchronous and asynchronous systems, adding common knowledge to these logics makes the validity problem $\Pi^1_1$-complete [HV89], which implies that there can be no complete axiomatization. This precludes the direct application of logics and knowledge and time to our problems, since it is necessary to express that the environment is common knowledge to all agents. On the other hand, the results of this paper suggest that appropriate decidable and axiomatizable fragments of logics with common knowledge and time may exist, but this requires further work.

While the algorithms we developed in Section 3 and Section 4 are theoretically linear time for bounded alternation depth formulae, in many cases they will not be directly practicable, because the state spaces of most systems of interest are too large to be explicitly represented. However, recent work on model checking in temporal logic [Val90, BCM+90, ES93] is beginning to provide ways to deal with the state space explosion problem, and we expect that the techniques developed in this area can be applied to model checking knowledge also. We leave this for future work.

Finally, we remark that the two semantic ascriptions of knowledge we have studied in this paper are not the only ones possible. One reasonable alternative is to drop the perfect recall assumption and ascribe knowledge to an agent based on its current observation only. As shown by Fisher and Immerman [FI86], in systems of this form common knowledge formulae may be reduced to knowledge formulae.

# References


[BCM+90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: $10^{20}$ states and beyond. In *Proc. Symposium on Logic in Computer Science*, pages 428–439. IEEE, 1990.

[Che80] B. Chellas. *Modal Logic.* Cambridge University Press, Cambridge, 1980.

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

[DM90] C. Dwork and Y. Moses. Knowledge and common knowledge in a byzantine environment. *Information and Computation*, 88:156–186, 1990.

[ES93] E. A. Emerson and A. P. Sistla. Symmetry and model checking. In *Computer Aided Verification, Proc. 5th Int. Conf.*, pages 156–165. Springer LNCS No. 697, 1993.

[FHV91] R. Fagin, J. Halpern, and M. Y. Vardi. A model-theoretic analysis of knowledge. *Journal of the ACM*, 38(2):382–428, April 1991.

[FI86] M. J. Fischer and N. Immerman. Foundations of knowledge for distributed systems. In J.Y. Halpern, editor, *Theoretical Aspects of Reasoning about Knowledge: Proc. 1986 Conf.*, pages 171–186, San Mateo, California, 1986. Morgan Kaufmann.

[Had87] V. Hadzilacos. A knowledge theoretic analysis of atomic commitment protocols. In *Proc. 6th ACM SIGACT-SIGMOD-SIGART Conf. on Principles of Database Systems*, pages 129–134, 1987.

[Hal87] J. 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.* Annual Reviews Inc., Palo Alto, CA, 1987.

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

[HM90] J. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990.

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

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, Mass, 1979.

[HV86] J. Halpern and M. Vardi. The complexity of reasoning about knowledge and time: extended abstract. In *Proc. 18th Annual ACM Symposium on Theory of Computing*, pages 304–315, 1986.

[HV88a] J. Halpern and M. Y. Vardi. The complexity of reasoning about knowledge and time: Synchronous systems. Technical Report RJ 6097, IBM Almaden Research Center, 1988.

[HV88b] J. Halpern and M. Y. Vardi. Reasoning about knowledge and time in asynchronous systems. Technical Report RJ 6197, IBM Almaden Research Center, 1988.

[HV89] J. Halpern and M.Y. Vardi. The complexity of reasoning about knowledge and time, I: Lower bounds. *Journal of Computer and System Sciences*, 38(1):195–237, 1989.

[HV91] J. Halpern and M. Y. Vardi. Model checking vs. theorem proving: A manifesto. Technical Report RJ 7963, IBM Almaden Research Center, 1991. An extended version of a paper in *Proc. 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning, 1991.*

[HZ92] J. Halpern and L. Zuck. A little knowledge goes a long way: Simple knowledge based derivations and correctness proofs for a family of protocols. *Journal of the ACM*, 39(3):449–478, 1992.

[IS92] H. Isozaki and Y. Shoham. A mechanism for reasoning about time and belief. In *Proc. Int. Conf. on Fifth Generation Computer Systems*, pages 694–701, 1992.

[KM91] H. Katsuno and A. Mendelzon. On the difference between updating a knowledge base and revising it. In *Proc. 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 387–394, 1991.

[Koh70] Z. Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, New York, 1970.

[LR86] R. Ladner and J. Reif. The logic of distributed protocols. In J.Y. Halpern, editor, *Theoretical Aspects of Reasoning about Knowledge: Proc. 1986 Conf.*, pages 207–222, San Mateo, California, 1986. Morgan Kaufmann.

[Maz86] M. Mazer. A knowledge theoretic account of recovery in distributed systems: The case of negotiated commitment. In M.Y. Vardi, editor, *Proc. 2nd Conf. on Theoretical Aspects of Reasoning about Knowledge*, pages 207–222, San Mateo, California, 1986. Morgan Kaufmann.

[Mey93a] R. van der Meyden. Axioms for knowledge and time in distributed systems with perfect recall. Technical report, NTT Basic Research Labs, 1993.

[Mey93b] R. van der Meyden. Common knowledge and update in finite environments II: Simple environments. Technical report, NTT Basic Research Labs, 1993.

[MK93] Y. Moses and O. Kislev. Knowledge oriented programming. In *Proc. Symposium on Principles of Distributed Computing*, 1993.

[PR85] R. Parikh and R. Ramanujam. Distributed processing and the logic of knowledge. In *Proc., Workshop on Logics of Programs*, pages 256–268. Springer LNCS No. 193, 1985.

[PT92] P. Panangaden and K. Taylor. Concurrent common knowledge: defining agreement for asynchronous systems. *Distributed Computing*, 6:73–93, 1992.

[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 Conf.*, pages 83–98. Morgan Kaufman, Los Altos, CA, 1986.

[Ros85] S.J. Rosenschein. Formal properties of knowledge in AI and robotics. *New Generation Computing*, 3:345–357, 1985.

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

[Val90] A. Valmari. A stubborn attack on state explosion. In *Proc. 2nd Workshop on Computer Aided Verification*, pages 156–165. Springer LNCS No. 531, 1990.

[Win88] M. Winslett. Reasoning about action using a possible models approach. In *Proc. National Conf. on Artificial Intelligence*, pages 89–93, 1988.