Common Knowledge and Update in Finite Environments. I

Ron van der Meyden

Information Processing Principles Group
NTT Basic Research Laboratories, room A4S-220S
3-1 Wakamiya Morinosato
Atsugi-shi Kanagawa 243-01, Japan

email: meyden@ntt-20.ntt.jp
Phone: +81 462 40 3674
Fax: +81 462 40 4709

Submitted to: Information and Computation

An extended abstract of this paper appeared in *Proceedings of the 5th Conference on Theoretical Aspects of Reasoning about Knowledge*, (R. Fagin, ed.) pp. 225-242, Morgan Kaufmann 1994

June 2, 1994

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

In recent years there has been considerable activity in the application of logics of knowledge to distributed and multi-agent systems. Work in this area has dealt primarily with semantic issues [Chandy and Misra, 1986; Fagin *et al.*, 1991; Halpern and Moses, 1990; Halpern and Vardi, 1989; Panangaden and Taylor, 1992], the development of logics and decision procedures for their validity problem [Fischer and Immerman, 1986; Halpern and Moses, 1992; Halpern and Vardi, 1988a; Halpern and Vardi, 1988b; Ladner and Reif, 1986; Parikh and Ramanujam, 1985], and the use of modal languages with knowledge operators to analyze specific coordination tasks [Dwork and Moses, 1990; Hadzilacos, 1987; Halpern, 1987; Halpern and Zuck, 1992; Mazer, 1986]. It has been argued that the notion of knowledge provides a natural level of abstraction at which to reason about distributed systems.

This suggests that it may be worthwhile to consider the construction of automated tools that support the analysis and verification of distributed systems in terms of knowledge [Rosenschein and Kaelbling, 1986]. It has also been proposed that there should be *knowledge based programming languages,* containing primitive constructs at the knowledge level [Fagin *et al.*, 1994 forthcoming; Moses and Kislev, 1993; Shoham, 1993]. Both applications require answers to 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 determine whether it knows a particular fact.

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 of the environment 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, which determines the *runs* of the system, i.e. the possible sequences of states. We suppose that both the transition diagram and the relationship between states and the agents' observations are common knowledge to all agents.

[1] This paper is the first of a series, to be continued with [Meyden, 1994b].

Though simple, this framework is sufficiently rich to represent many systems of interest, including finite state shared variable systems, and message passing systems with bounded buffers. Both synchronous and asynchronous systems of these kinds fall within its scope. The framework is also able to represent most games of imperfect information, including Bridge, Poker, Battleships and Stratego. In Section 2 we show how it may model the "blindfold" chess game 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. Using its knowledge of the possible state transitions, an agent can derive from its knowledge about prior states of the world additional information about the current state, including additional information about other agents' knowledge. For example, suppose I have learnt from 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 cannot directly observe his current hand, I may use my knowledge of his previous hand to 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 cheated, then the fact that my opponent has at most two aces is also common knowledge.

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 usual definitions of knowledge in distributed systems [Halpern and Moses, 1990], in each global state of the system we are required to assign to each agent a local state. An agent i may then be said to *know* a fact φ at a global state in which the agent is in local state s just when φ is true in all global states where agent i is in local state s. Intuitively, an agent's local state encodes the information it has about the global state.

In our framework, an agent's local state is determined by its observations. In order to exploit prior knowledge in reasoning about the current global state, an agent needs to recall information obtained from its previous observations. We assume the strongest possible model of memory, namely that agents have *perfect recall* of all their observations. The perfect recall assumption is one that has frequently been made in the literature. Since it implies optimal use of information, perfect recall is the appropriate assumption to make in proving impossibility results and lower bounds on communication [Halpern and Moses, 1990; Chandy and Misra, 1986]. Perfect recall is also a convenient working assumption during the design of a protocol, that enables the definition of data structures to be deferred. Once it has been determined what information processors require, optimal data structures may be designed that encode just this relevant information [Dwork and Moses, 1990; Moses and Tuttle, 1988].

The perfect recall assumption, however, still leaves two possible ascriptions of local state, depending on whether the agents operate *synchronously* (i.e., with a global clock visible to all agents) or *asynchronously*. We study both cases. In the synchronous case we assign to an agent the sequence of its observations in all previous states. In the asynchronous case agents are not assumed to know the length of time for which they were making an observation, so we assign to an agent its sequence of observations with consecutive repetitions deleted.

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 sentences 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 formulae in a Kripke structure is the *model checking* problem. Model checking knowledge formulae in finite Kripke structures is known to be in polynomial time [Halpern and Moses, 1992]. Unfortunately, this result is not directly applicable to our problem, since the structures we consider may have an infinite number of worlds.

However, it would appear at first blush that this complexity can be overcome. The local state assignment is a semantic device for the external ascription of a state of knowledge to an agent. We do not assume that an agent actually maintains his sequence of observations: any data structure with the same informational content (with respect to the formulae of interest) is a permitted knowledge representation. Given that there exist only a finite number of states of the environment, and that our formulae refer only to the current state, and to agents' mutual knowledge of the current state, it is 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 in Section 6 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 for a fixed formula.

On the other hand, model checking formulae not involving common knowledge turns out to be 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.

Given some reasonable constraints on formulae, it turns out that a data structure more succinct than the agent's sequence of observations suffices as a knowledge representation. We show in Section 4 and Section 5 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 knowledge representation that may be updated in constant time and queried time linear in the size of the formula. This yields an incremental algorithm for model checking. Such an incremental algorithm 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 by means of finite state automata.

The knowledge representation used in these results is a special kind of Kripke structure. This is somewhat different from the approach usually taken in the literature on knowledge base updates [Katsuno and Mendelzon, 1991], in which it is assumed 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 in [Halpern and Vardi, 1991].

The structure of the paper is as follows. Section 2 introduces the formal model of environments and the semantics of knowledge in the synchronous case. Section 3 presents a detailed example of what the model may represent. Section 4 deals with the incremental approach to model checking bounded alternation formulae in synchronous environments. In Section 5 the asynchronous semantics is introduced, and it is shown how to generalize the results of Section 4 to this semantics. Section 6 gives lower bounds for common knowledge formulae. The relation of our work to compilation of knowledge based programming languages is discussed in Section 7, and Section 8 contains concluding remarks.

2 Definitions

We begin this section with a formal definition of environments, and describe how an environment determines a Kripke structure which may be used to ascribe knowledge to agents operating synchronously. We introduce the model checking problem, and give an upper bound on the complexity of this problem in synchronous environments.

We wish to model finite state systems in which each state determines an observation for the agents. Suppose that *Prop* is some 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 $\{0, 1\}$.

Intuitively, the elements of S represent the possible global states of the system being modelled, and may encode information such as the control state of components of the system, the values of variables maintained by the agents (either shared or local), messages in transit, etc. The set I represents the *initial states* of the environment. We say that T is the *transition relation* of the environment E, representing the possible state evolutions.

The functions O_i are called the *observation functions* of E. 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 wishes to send 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 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.

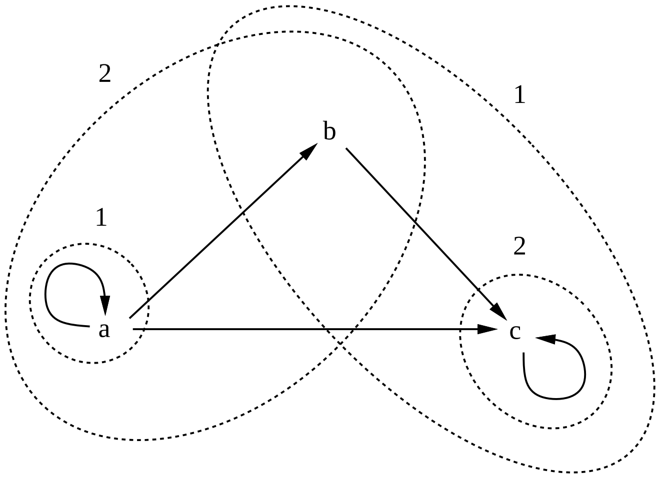

Figure 1: An environment

Before describing how an environment determines a particular Kripke structure that ascribes to an agent a state of knowledge after each possible sequence of events, we first discuss Kripke structures in the abstract, and describe a standard construction on Kripke structures, of use for the computation of knowledge formulae.

We will deal with queries about knowledge expressed in the propositional modal language $\mathcal{L}_n^C$ which has a modal operator K_i for each agent $i = 1 \dots n$, 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 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.
2. If φ and ψ are formulae then $\neg\varphi$ and $\varphi \wedge \psi$ are formulae.
3. If φ is a formula then $K_i\varphi$ is a formula for each $i = 1 \dots n$.
4. If φ is a formula then $C_G\varphi$ is a formula for each subset G of $\{1 \dots n\}$ 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

$$M = \langle W, \mathcal{K}_1, \dots, \mathcal{K}_n, V \rangle$$

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 $\mathrm{S5}_n$ structure. Most of the Kripke structures we deal with in this paper will be $\mathrm{S5}_n$ 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 $\mathrm{S5}_n$ 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 that 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 φ holds in a given a world w in a Kripke structure M. The following result 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 $\mathrm{S5}_n$ structure and $\varphi \in \mathcal{L}_n^C$ then $M, w \models \varphi$ can be decided in time $O(|\varphi| \cdot ||M||)$.

Proof: For $\varphi \in \mathcal{L}_n$ it is shown in [Halpern and Moses, 1992] how to compute, in time $O(|\varphi| \cdot ||M||)$, for all the worlds of M the truth values of all subformulae of φ. To extend this algorithm to $\mathcal{L}_n^C$, we show how to compute at all worlds of M, in time $O(||M||)$, the truth value of the proposition $C_G\varphi$, assuming that the truth value of φ has already been computed at all worlds. First, label all worlds at which $\neg\varphi$ holds with $\neg C_G\varphi$. Next, repeat the following operation until a fixpoint is reached: if a world u is labelled $\neg C_G\varphi$ and $u \sim_i v$ for some $i \in G$ then label v with $\neg C_G\varphi$. Once the fixpoint is reached, label with $C_G\varphi$ all the worlds not already labelled with $\neg C_G\varphi$. It is straightforward to check that this procedure can be implemented in time $O(||M||)$, and terminates with the property that each world w is labelled $C_G\varphi$ if and only if $M, w \models C_G\varphi$. $\Box$

We now review a known 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" [Chellas, 1980]:

Lemma 2.3: If φ 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 in order to assign states of knowledge to the agents. Our definitions will essentially be a special case of the framework introduced in [Halpern and Moses, 1990]. We consider in this paper two distinct ways to associate a Kripke structure with an environment. In the present section, we will deal just with the *synchronous* interpretation. The Kripke structure associated with an environment in the asynchronous interpretation will be introduced later, in Section 5.

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 if 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.

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 each $l = 1 \ldots m$ we have $O_i(s_l) = O_i(s'_l)$.

Informally, what agent i knows in a given run is which equivalence class of $\sim_i$ contains the actual run. Note that it is implicit in these definitions that an agent is able to "tell the time," since the number of states in the run is equal to the number of observations in the local state. Further, each distinct observation made by an agent is reflected in its local state. Thus, we may say these definitions characterize agents operating *synchronously* and with *perfect recall.* However, note that we do not assume that agents necessarily maintain their entire sequence of observations: the local state assignment is merely a semantical device for describing the agents' informational state.

We may now define for each environment E for n agents the Kripke structure $M_E = \langle W_E, \mathcal{K}_1, \ldots, \mathcal{K}_n, V_E \rangle$ used to interpret the language $\mathcal{L}_n^C$ in the environment E.

This structure has worlds $W_E = runs(E)$, access relations $\mathcal{K}_i = \sim_i$ for $i = 1 \ldots n$, and valuation $V_E : W_E \times Prop \rightarrow \{0,1\}$ given by $V_E(r,p) = V(\mathit{fin}(r), p)$. It is clear that M_E is an $S5_n$ structure.[2] 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 encodes 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 $O_1(c) = O_1(x)$ 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

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

where we have omitted the relations holding between each run and itself. It follows that $(E, ac^{n-1}) \models (K_2 K_1)^j p$ 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 model checking formulae in $\mathcal{L}_n^C$ is decidable, and Proposition 2.2 may be used to obtain an algorithm that runs in time and space $O(|\varphi| \cdot n \cdot |S|^{2 \cdot length(r)})$. However, an improvement on the space bound is possible. Rather than construct the entire structure, we can create runs only as they are needed, and reuse space after we are done with them. This yields

[2]In the terminology of [Halpern and Moses, 1990], a run is an *infinite* sequence of local state assignments, and the worlds of the Kripke structure associated with a distributed system are *points* of the form (ρ, n) where ρ is a run and n is an integer, describing a location in this run. We could have adopted this convention for serial environments, but because the language we consider contains no temporal operators, there is no essential difference.

[3]This example is closely related to one in [Halpern and Moses, 1990] that shows common knowledge is not attained when there is uncertainty about message transmission delays.

the following upper bound. We will show later that this bound is tight, but also that it can be improved in certain circumstances.

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 in PSPACE.

Proof: Note that we may write the truth conditions for the modal operators of $\mathcal{L}_n^C$ in the form

- $(E, r) \models K_i\varphi$ if for all for all runs r' such that r' has the same length as r and $r \sim_i r'$, we have $(E, r') \models \varphi$, and
- $(E, r) \models C_G\varphi$ if for all runs r' of the same length as r which are G-accessible from r we have $(E, r') \models \varphi$.

The bounded universal quantifiers in these expressions may be simulated by a Co-NP computation using space linear in $|r|$. It follows that it is possible to construct an alternating Turing machine [Chandra *et al.*, 1981] that decides whether a formula φ holds in a run r of environment E, using work space $S(n) = O(|r| + \log |\varphi|)$ and $A(n) = O(|\varphi|)$ alternations. The result now follows from a generalization of Savitch's Theorem proved in [Chandra *et al.*, 1981], which states that alternating computations using space $S(n)$ and $A(n)$ alternations may be simulated in deterministic space $A(n)S(n) + S(n)^2$. □

We remark that the improvement in the space bound obtained in this result comes at the cost of an increased computation time, since the algorithm described, when implemented deterministically, may perform much redundant computation.

Let us note one other point about Example 2.4. Given an environment E for n agents, define the mapping $\Phi_E : runs(E) \to \mathcal{P}(\mathcal{L}_n^C)$ by $\Phi_E(r) = \{ \varphi \in \mathcal{L}_n^C \mid (E, r) \models \varphi\}$. That is, $\Phi_E(r)$ is the set of formulae true in the run r, and captures all the information about the agent's state of knowledge in run r that is expressible in the language $\mathcal{L}_n^C$. Define an environment E to be *simple* if the image of the set of runs under the function Φ_E is finite. Informally, an environment is simple when, to the extent representable in the language $\mathcal{L}_n^C$, agents in that environment can attain only a finite number of distinct states of mutual knowledge. Even though the number of runs may be infinite, one might have expected from the fact that environments have a finite number of states, or from the regular manner in which the Kripke structure is generated from an environment, that all environments are simple. Example 2.4 shows that this is not the case, since the sets $\Phi_E(ac^n)$ are pairwise distinct, as n varies over the natural numbers. [4]

[4] [Meyden, 1994b] studies simple environments in greater detail.

3 Kriegspiel

We have seen in the previous section a simple example of an environment. To further illustrate the framework, we now present a more elaborate example. We will formulate Kriegspiel, the German game of double-blind chess, as an environment.

The moves of Kriegspiel[5] are like those of chess, but two boards are used, one bearing only white pieces, the other bearing only black pieces. Each player sees only the board with his own pieces. At his turn, a player proposes a move to an umpire. This proposed move is not revealed to his opponent. If the move is illegal according to the standard rules of chess, applied to the union of the positions on the two boards, the umpire announces to both players that an illegal move was proposed, and the same player is given the opportunity to propose a different move, until a legal move is found. Once he chooses a legal move, the umpire advises the opponent of any effects on his position. That is, if the successful move results in a piece being taken the umpire removes the appropriate piece from the the opponent's board. The player making the move is informed that a piece was taken, but the identity of this piece is not revealed. If the move results in a check (or checkmate) the umpire advises both players of this fact. The umpire having made the necessary announcements, play switches sides.

This game may be formalized as an environment as follows. For simplicity we ignore rules concerning castling and repeated positions. Let Pos be the set of all chess positions, i.e. arrangements of chess pieces on a chess board. Two subsets of Pos are of special interest: Pos_W and Pos_B, the positions containing only white pieces and, respectively, only black pieces. There exist two natural projections $\pi_W : Pos \longrightarrow Pos_W$ and $\pi_B : Pos \longrightarrow Pos_B$, which remove from a board position all the pieces of Black and White, respectively. Let $Move$ be the set of legal moves of chess pieces, e.g. "White Bishop on King 3 moves to Queen 4." The set Ump consists of the possible announcements of the umpire: either "play begins," "valid move," "invalid move," "piece taken," "check," "piece taken and check" or "checkmate."

The set S of states of the environment will be the set

$$Pos \times \mathcal{P}(Move) \times Ump \times \{B, W\}.$$

The meaning of a state $(b, M, u, p) \in S$ is intuitively that the current *combined* board position of the two players is b, it is player p's turn to move, the set of invalid moves already proposed by p is M, and the umpire's response to the last move proposed was u. Note that this means that White's board is in position $\pi_W(b)$ and Black's board is $\pi_B(b)$. There is a single initial state, namely $(b_0, \emptyset, \text{"play begins"}, W)$, where b_0 is the initial position of chess. The transition relation T corresponding to the rules above is given by $(b, M, u, p)T(b', M', u', p')$ just when either

1. there is a valid move m of player p in position b which is not in M, b' is the

[5] We follow the description of [von Nuemann and Morgenstern, 1953].

position that results from his making that move, $M' = \emptyset$, u' is the appropriate announcement of the umpire and p' is the opposite player to p, or

2. there is a move m of player p in position b which is not in M, m is invalid in position b, $M' = M \cup \{m\}$, $b' = b$, u' is "invalid move" and $p' = p$.

The observation functions are given as follows. The observation function O_W for White maps S to $Pos_W \times (\mathcal{P}(Move) \cup \{*\}) \times Ump$, where for all b, M and u, $O_W(b, M, u, W) = (\pi_W(b), M, u)$ and $O_W(b, M, u, B) = (\pi_W(b), *, u)$. Here $*$ corresponds to an unknown set of moves proposed by Black. The observation function for Black is given symmetrically.

The valuation of the environment will depend on the propositions of interest. A reasonable choice is to take *Prop* to consist of all assertions about locations of pieces, e.g. "a black pawn is on King 3," together with the proposition "White to move." The valuation V may then assign true to a proposition in a state (b, M, u, p) just in case it properly describes b, or in case $p = W$ when the proposition is "White to move."

We leave it to the reader to verify that with the observations we have described, the local state assignment of the previous section encodes all the information available to a player at each point of the game. Some of this information is available directly from the current observation, but sometimes the history of observations needs to be consulted. For example, if it is White's turn to move then White knows what moves he has already proposed directly from his current observation, but Black needs to consult his history to determine the number of invalid moves White has already proposed.

4 Bounded Alternation Formulae

In most applications the environment will be fixed, and we may expect the size of formulae of interest to be small. Thus, the greatest contribution to the complexity of model checking will come from the length of the run. The approach to model checking of Proposition 2.5 and the preceding discussion requires time exponential in the length of the run. For formulae not containing common knowledge operators, it is possible to do significantly better than this.

Define the *alternation depth* of a formula in $\mathcal{L}_n$ to be the number of alternations of distinct knowledge operators in the formula. For example, p is a formula of alternation depth 0 and $K_2K_1K_1K_2q$ has alternation depth 3. We will show in this section how to incrementally maintain a constant size data structure representing the agents' knowledge 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 for agents to maintain their sequence of observations in this case.

The data structure we will use is a certain type of finite tree in which the vertices are labelled by states of the environment and edges are labelled by agents. It is convenient to represent these trees as follows. Suppose S is a fixed set of states equipped with a valuation function $V : S \times Prop \rightarrow \{0,1\}$. 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 *forests of i-objective k-trees 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. 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 the number of k-trees over S is less than $C_k = exp(n|S|,k)$.

The elements of $\mathcal{T}_k$ correspond in an obvious way to 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. When $w \in \mathcal{T}_0$ the labelled tree corresponding to w consists of just the root, labelled w. The tree corresponding to $w \in \mathcal{T}_{k+1}$ has root labelled with $root(w)$, and for each i-child $v \in \mathcal{T}_k$ of w there is an i-labelled edge from the root to a vertex at which the labelled subtree is that corresponding to v.

The number of vertices of the labelled tree corresponding to a k-tree is at most C_{k-1}^c for some constant c. Note because an i-child of a $k+1$-tree is an i-objective k-tree, there does not exists in these labelled trees a pair of consecutive edges with the same label i. (Additional constraints follow from the fact that the U_i are sets, hence do not contain repetitions.) Using the correspondence between k-trees and labelled trees, an element U of $\mathcal{F}_{k,i}$ may be regarded as a forest of labelled trees.

By viewing a k-tree w as tree-like Kripke structure, it is possible to define a truth relation $w \models \varphi$, meaningful when φ 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 labelled tree corresponding to 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 not exactly those obtained from the tree edges, however. We define the accessibility relation $\mathcal{K}_i$ to be the relation such that each vertex may access its successors through edges labelled i, and all successors of a vertex through edges labelled i may access each other. This makes the accessibility relations *transitive* and *euclidean*, but not equivalence relations, since if there is an i-labelled edge from u to v then $u\mathcal{K}_i v$ but not $v\mathcal{K}_i u$.[6] We define $w \models \varphi$ to hold just

[6]The Kripke structures we have defined here are very similar to the *knowledge trees* used in [Halpern and Vardi, 1988a] to show decidability and completeness of logics for knowledge and syn-

when φ is true at the root of this 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. First, 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.

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 $k \geq 0$ and agents i define the functions $F_k : W \to \mathcal{T}_k$ and $U_{k,i} : W \to \mathcal{F}_{k,i}$ by mutual recursion as follows. If $k = 0$, then define $F_k(w) = f(w)$. For $k \geq 0$, define $F_{k+1}(w) = \langle f(w), U_{k,1}(w), \ldots, U_{k,n}(w) \rangle$, where $U_{k,i}(w) = \{obj_i(F_k(v)) \mid w\mathcal{K}_i v\}$. The following result states that $F_k(w)$ represents the formulae of alternation depth k or less holding at the world w.

Proposition 4.1: If φ 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 $F_k(w) \models \varphi$.[7]

Proof: For the proof it is convenient to give an explicit description of the truth relation $w \models \varphi$ for k-trees w, by means of a simultaneous recursion with a relation $w \models_{U,i} \varphi$, where U is in $\mathcal{F}_{k-1,i}$ and i is an agent. The appropriate clauses for the relation $\models$ are the following:

- If p is a propositional constant then $w \models p$ if $V(root(w), p) = 1$.
- $w \models \varphi_1 \wedge \varphi_2$ if $w \models \varphi_1$ and $w \models \varphi_2$
- $w \models \neg\varphi$ if not $w \models \varphi$.
- if $k \geq 1$ and $w = \langle v, U_1, \ldots, U_n \rangle$ then $w \models K_i\varphi$ if $w' \models_{U_i,i} \varphi$ for all $w' \in U_i$.

We will say that a formula φ is *k,i-compatible*, where $k \geq 0$ and i is an agent, if φ has alternation depth less than or equal to k and has no subformulae of the form $K_j\psi$ with $j \neq i$ and alternation depth equal to k. The relation $w \models_{U,i} \varphi$ will be meaningful when $k \geq 1$, w is a $k-1$-tree, the set U is in $\mathcal{F}_{k-1,i}$ and the formula φ is k,i-compatible. The definition of the relation $\models_{U,i}$ is as follows:

- If p is a propositional constant then $w \models_{U,i} p$ if $V(root(w), p) = 1$.
- $w \models_{U,i} \varphi_1 \wedge \varphi_2$ if $w \models_{U,i} \varphi_1$ and $w \models_{U,i} \varphi_2$
- $w \models_{U,i} \neg\varphi$ if not $w \models_{U,i} \varphi$.

---

chronous time.

[7] It suffices that the access relations $\mathcal{K}_i$ be transitive and euclidean, so Proposition 4.1 also applies to $K45_n$, the logic of belief.

- If $j = i$ then $w \models_{U,i} K_j\varphi$ if $w' \models_{U,i} \varphi$ for all $w' \in U$.
- If $j \neq i$ then $w \models_{U,i} K_j\varphi$ if $w' \models K_j\varphi$ for all $w' \in U$.

It is readily seen that this characterization of the relation $\models$ is equivalent to the definition above which views the elements of $\mathcal{T}_k$ as tree-like Kripke structures.

We now show by induction on $|\varphi| + k$ that if φ is a formula of $\mathcal{L}_n$ of alternation depth less than or equal to k, then for all worlds w of M,

(a) $F_k(w) \models \varphi$ if and only if $M, w \models \varphi$, and

(b) $obj_i(F_{k-1}(w)) \models_{U_{k-1,i}(w),i} \varphi$ if and only if $M, w \models \varphi$, provided φ is k,i-compatible.

The base cases, where φ is a propositional constant, are direct from the fact that f preserves the valuation. The cases where φ is formed using conjunction or negation are trivial. This leaves three cases.

Case 1: We show the inductive step for (a) and formulae of type $K_i\varphi$. Note that if $K_i\varphi$ has alternation depth less than or equal to k then φ is k,i-compatible. By the characterization of $\models$ and the definition of F_k, we have that $F_k(w) \models K_i\varphi$ just when $obj_i(F_{k-1}(u)) \models_{U_{k-1,i}(w),i} \varphi$ for all worlds u of M satisfying $w\mathcal{K}_iu$. Now because the relation $\mathcal{K}_i$ is an equivalence relation, we have that $U_{k-1,i}(w) = U_{k-1,i}(u)$ for each u satisfying $w\mathcal{K}_iu$. Thus, using part (b) of the induction hypothesis, we obtain that $F_k(w) \models K_i\varphi$ just when $M, u \models \varphi$ for all worlds u of M satisfying $w\mathcal{K}_iu$. But the latter holds precisely when $M, w \models K_i\varphi$. This proves (a).

Case 2: We show the inductive step for (b) for k,i-compatible formulae of type $K_i\varphi$. Note that φ is also k,i-compatible in this case. By the characterization of $\models$ and the definition of $U_{k,i}$, we have that $obj_i(F_{k-1}(w)) \models_{U_{k-1,i}(w),i} K_i\varphi$ just when $obj_i(F_{k-1}(v)) \models_{U_{k-1,i}(w),i} \varphi$ for all v with $w\mathcal{K}_iv$. Since $w\mathcal{K}_iv$ implies $U_{k-1,i}(w) = U_{k-1,i}(v)$, we obtain from part (b) of the induction hypothesis that $obj_i(F_{k-1}(w)) \models_{U_{k-1,i}(w),i} K_i\varphi$ just when $M, v \models \varphi$ for all worlds v satisfying wK_iv, which in turn holds precisely when $M, w \models K_i\varphi$. This proves (b).

Case 3: We show the inductive step for (b) for k,i-compatible formulae of type $K_j\varphi$, where $j \neq i$. Since $K_j\varphi$ is a subformula of itself, it must have alternation depth of at most $k-1$. By the characterization of $\models$, we have that $obj_i(F_{k-1}(w)) \models_{U_{k-1,i}(w),i} K_j\varphi$ just when $obj_i(F_{k-1}(w)) \models K_j\varphi$. Since $j \neq i$, $obj_i(F_{k-1}(w)) \models K_j\varphi$ if and only if $F_{k-1}(w) \models K_j\varphi$. It follows from the induction hypothesis for part (a) that the latter holds if and only if $M, w \models \varphi$, establishing the required equivalence. □

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 4.1.

Corollary 4.2: For every run r of E and sentence φ 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 it suffices to compute in a structure which for sufficiently long runs 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, as we now show, there is a more efficient, incremental way to construct $F_k(r)$.

Define for each number $k \geq 0$ the function $G_k : \mathcal{T}_k \times S \rightarrow \mathcal{T}_k$, by induction on k. The definition of G_k will be by mutual recursion with the functions $H_{k,i} : \mathcal{F}_{k,i} \times \mathcal{O} \rightarrow \mathcal{F}_{k,i}$, 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. Note also that $G_k(obj_i(w), s) = obj_i(G_k(w, s))$.

Intuitively, if agent i's state of knowledge (to level k) is represented by the the set of k-trees U, then $H_{k,i}(U, o)$ represents 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, representing the states considered possible by agent i, and

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

This equation expresses the agent's knowledge that a single step through the transition relation T has been taken from one of the states it previously considered possible, resulting in a state in which it observes o. The functions G_k recursively apply this computation to higher levels of knowledge. The following result shows how these functions may be used to compute F_k incrementally. We write rs for a run with final state s preceded by initial segment r.

Lemma 4.3: For every run rs of E and every number k, $F_k(rs) = G_k(F_k(r), s)$.

Proof: By induction on k. Note that for every run rs, we have that $F_0(rs) = s = G_0(F_0(r), s)$. Hence the claim holds for $k = 0$. Assume that it holds for k. Note first that $root(G_{k+1}(F_{k+1}(r), s)) = s = root(F_{k+1}(rs))$. We show that $G_{k+1}(F_{k+1}(r), s)$ and $F_{k+1}(rs)$ have the same i-children. By definition of G_{k+1} and F_{k+1}, the k-tree w is an i-child of $G_{k+1}(F_{k+1}(r), s)$ if and only if $w \in H_{k,i}(U_{k,i}(r), O_i(s))$. By definition of $H_{k,i}$, this is the case if and only if there exists a k-tree $v \in U_{k,i}(r)$ and a state

$s' \in S$ such that $w = G_k(v, s')$, there is a transition of E from $root(v)$ to s', and $O_i(s') = O_i(s)$.

By definition of $U_{k,i}$, the k-tree v is in $U_{k,i}(r)$ if and only if $v = obj_i(F_k(r'))$ for some run $r' \in runs(E)$ such that $r \sim_i r'$. In this case $root(v)$ is the final state of r', so for all states s' the sequence $r's'$ is a run of E just when there is a transition from $root(v)$ to s'. Further, from the induction hypothesis we have $G_k(F_k(r'), s') = F_k(r's')$, which implies $G_k(obj_i(F_k(r')), s') = obj_i(F_k(r's'))$ by the remark above. Finally, $rs \sim_i r's'$ just when $r \sim_i r'$ and $O_i(s) = O_i(s')$.

Putting these pieces together, we obtain that w is an i-child of $G_{k+1}(F_{k+1}(r), s)$ if and only if there exists a run $r's'$ of E with $r's' \sim_i rs$ and $w = obj_i(F_k(r's'))$. The latter holds just when w is an i-child of $F_{k+1}(rs)$. This establishes $G_{k+1}(F_{k+1}(r), s) = F_{k+1}(rs)$. □

If r is a run of length one then $F_k(r)$ may be straightforwardly computed: if r consists of the single initial state s then $F_0(r) = s$ and $F_{k+1}(r)$ is the $k+1$-tree with root s and an i-child $obj_i(F_k(s'))$ for each initial state s' with $O_i(s') = O_i(s)$. Combining this with Corollary 4.2, Lemma 4.3, Proposition 2.2, and the fact that k-trees have size C^c_{k-1}, we obtain the following result.

Theorem 4.4: For formulae φ of alternation depth k or less, $(E, r) \models \varphi$ can be decided in time $O(C^c_{k-1} \cdot (|r| + |\varphi|))$.

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 the length of run of Proposition 2.5 for a linear time dependence on the 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 4.4 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). This update computation is closely related to (though not subsumed by) the Possible Model Approach to the semantics of updates of [Winslett, 1988], and to the generalization of this approach in [Katsuno and Mendelzon, 1991]. Similar ideas occur the discussion of knowledge in robotics of [Rosenschein, 1985], and in the standard theory of state identification in finite state automata [Kohavi, 1970]. 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 literature.

Theorem 4.4 also subsumes the situation in which we wish to check a *fixed* formula of $\mathcal{L}_n$ in runs of increasing length. Because of the incremental nature of our construction, we obtain the following.

Corollary 4.5: Let φ 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.

Thus, fixed formulae in $\mathcal{L}_n$ can be implemented by means of finite state automata. In general, the finite state automaton obtained for a fixed formula by the methods of this section will not be minimal, so further optimization is possible. We will discuss this possibility in more detail elsewhere.

5 Asynchronous Environments

We now consider an alternate semantics for environments, in which we do not assume that agents are synchronized. We show that results similar to those of the previous section may be established for this semantics.

As before, an environment consists of a set of states, a valuation, a transition relation, and observation functions. However, we now associate a different Kripke structure with such an environment. The states of the Kripke structure are the runs of the environment, defined as above. What varies is the definition of agents' local states. Instead of assuming that an agent is aware of every transition made, we now assume that an agent notices transitions only when these lead to changes in its observation.

More formally, 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 φ holds at run r in M_E^a.

Example 5.1: 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 5.1 shows, this is no longer the case when environments are interpreted asynchronously: in this example all runs generate the entire structure. Nevertheless, formulae not containing common knowledge operators remain decidable.

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 $\mathcal{T}_k$ and $\mathcal{F}_{k,i}$ are defined as before. For each number $k \geq 0$ and agent i, let the functions $F_k^a : runs(E) \longrightarrow \mathcal{T}_k$ and $U_{k,i}^a : runs(E) \longrightarrow \mathcal{F}_{k,i}$ be defined mutually recursively by $F_0^a(r) = fin(r)$, and by $U_{k,i}^a(r) = \{obj_i(F_k^a(r')) \mid r \approx_i r'\}$ and $F_{k+1}^a(r) = \langle fin(r), U_{k,1}^a(r), \ldots, U_{k,n}^a(r)\rangle$ for $k \geq 0$. This definition is almost identical to the definition of the function F_k in Section 4, 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 4.1, analogous to Corollary 4.2.

Corollary 5.2: For every run r of E and formula φ 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 incrementally 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 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. However, 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 of the previous section 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 $H^a_{k,i} : \mathcal{F}_{k,i} \times \mathcal{O} \longrightarrow \mathcal{F}_{k,i}$, where i is an agent and $k \geq 0$. Like the function $H_{k,i}$ used in the synchronous case, the function $H^a_{k,i}$ maps a state of knowledge U of agent i and an observation o to the agent's new state of knowledge $H^a_{k,i}(U,o)$ upon making the observation o. Thus, as above we define $G^a_0(w,s) = s$, and

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

To define the function $H^a_{k,i}$, we assume that G^a_k has been defined. The definition makes use of an auxiliary function $J_{k,i,o} : \mathcal{F}_{k,i} \rightarrow \mathcal{F}_{k,i}$. 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

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

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

Lemma 5.3:

(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 5.3 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. Note that if U is the empty set then $H^a_{k,i}(U,o)$ is also empty. It follows from this that G^a_k maps k-trees to k-trees, and also maps i-objective k-trees to i-objective k-trees, so these definitions are proper. Note also that $G^a_k(obj_i(w), s) = obj_i(G^a_k(w,s))$.

We may now establish the following recursion relation, which gives the incremental way to compute F^a_k.

Proposition 5.4: For every run rs we have $G^a_k(F^a_k(r), s) = F^a_k(rs)$.

The proof of this result will be by induction on k. It is convenient to introduce the partial order on k-trees defined by $\langle s, U_1, \ldots, U_n\rangle \leq \langle s', U'_1, \ldots, U'_n\rangle$ when $s = s'$ and $U_i \subseteq U'_i$ for all $i = 1 \ldots n$. The base case $k = 0$ of Proposition 5.4 follows immediately from the definitions of G^a_0 and F^a_0. We assume that $G^a_k(F^a_k(r), s) = F^a_k(rs)$ for all runs rs, and show that $G^a_{k+1}(F^a_{k+1}(r), s) = F^a_{k+1}(rs)$, by establishing that $F^a_{k+1}(rs) \leq G^a_{k+1}(F^a_{k+1}(r), s)$ and $G^a_{k+1}(F^a_{k+1}(r), s) \leq F^a_{k+1}(rs)$. For the first of these inequalities, we make use of the following result.

Lemma 5.5:

(1) If rs and $r's'$ are runs with $r' \approx_i r$ and $O_i(s') = O_i(s)$ then $obj_i(F^a_k(r's'))$ is an i-child of $G^a_{k+1}(F^a_{k+1}(r), s)$.

(2) Suppose s is a state and let $r's'$ be a run with $O_i(s') = O_i(s)$. For all $w \in \mathcal{T}_k$, if $obj_i(F^a_k(r'))$ is an i-child of $G^a_{k+1}(w, s)$ then $obj_i(F^a_k(r's'))$ is an i-child of $G^a_{k+1}(w, s)$.

Proof: For (1), observe that if $r' \approx_i r$ then by the definition of $U^a_{k,i}$, we have that $obj_i(F^a_k(r'))$ is in $U^a_{k,i}(r)$. Let $o = O_i(s)$. Now $O_i(s') = o$ and there is a transition from $root(F^a_k(r')) = fin(r')$ to s', hence by the second part of the definition of $J_{k,i,o}$ we have that $G^a_k(obj_i(F^a_k(r')), s')$ is an element of $J_{k,i,o}(U^a_{k,i}(r))$. By definition of $H^a_{k,i}$, we have $J_{k,i,o}(U^a_{k,i}(r)) \subseteq H^a_{k,i}(U^a_{k,i}(r), o)$. By the the induction hypothesis and the remark above, $G^a_k(obj_i(F^a_k(r')), s') = obj_i(G^a_k(F^a_k(r'), s')) = obj_i(F^a_k(r's'))$. Thus $obj_i(F^a_k(r's'))$ is in $H^a_{k,i}(U^a_{k,i}(r), o)$, hence an i-child of $G^a_{k+1}(F^a_{k+1}(r), s)$.

For (2), suppose $G^a_{k+1}(w, s) = \langle s, U_1, \ldots, U_n\rangle$ and let $o = O_i(s)$. By the induction hypothesis, we have $obj_i(F^a_k(r's')) = G^a_k(obj_i(F^a_k(r')), s')$. Note that since $r's'$ is a run, there is a transition from $root(F^a_k(r')) = fin(r')$ to s'. In addition, $O_i(s') = o$. It follows using the second part of the definition of $J_{k,i,o}$ that if $obj_i(F^a_k(r'))$ is in U_i then $obj_i(F^a_k(r's'))$ is in $J_{k,i,o}(U_i)$. But by definition of G^a_k, we have that $U_i = H^a_{k,i}(V, o)$ for some forest V of k-trees. Since $J_{k,i,o}(H^a_{k,i}(V, o)) = H^a_{k,i}(V, o)$ for all V, we obtain that $obj_i(F^a_k(r's'))$ is in U_i. This establishes the result. $\Box$

We may now show that $F^a_{k+1}(rs) \leq G^a_{k+1}(F^a_{k+1}(r), s)$. It is clear that $root(F^a_{k+1}(rs)) = s = root(G^a_{k+1}(F^a_{k+1}(r), s))$. Suppose that v is an i-child of $F^a_{k+1}(rs)$. Then by definition of F^a_{k+1} there exists a run r' such that $r' \approx_i rs$ and $v = obj_i(F^a_k(r'))$. We consider two cases, and show that in both cases v is an i-child of $G^a_{k+1}(F^a_{k+1}(r), s)$.

First, suppose $O_i(fin(r)) = O_i(s) = o$. Then we have $r' \approx_i rs \approx_i r$, hence v is in $U^a_{k,i}(r)$, by definition of $U^a_{k,i}$. Since $O_i(root(v)) = O_i(fin(r')) = o$ it then follows from the first part of the definition of $J_{k,i,o}$ that v is in $J_{k,i,o}(U^a_{k,i}(r))$, and consequently in $H^a_{k,i}(U^a_{k,i}(r), s)$. This shows that v is an i-child of $G^a_{k+1}(F^a_{k+1}(r), s)$.

Second, suppose $O_i(fin(r)) \neq O_i(s)$. Then r' may be written as $r_1 s_1 \ldots s_m$ where $r_1 \approx_i r$ and $O_i(s_j) = O_i(s)$ for all $j = 1, \ldots, m$. Since $r_1 \approx_i r$, it follows from Lemma 5.5(1) that $obj_i(F^a_k(r_1 s_1))$ is an i-child of $G^a_{k+1}(F^a_{k+1}(r), s)$. Now, by repeated use of Lemma 5.5(2) it follows that $obj_i(F^a_k(r_1 s_1 \ldots s_m)) = v$ is an i-child of $G^a_{k+1}(F^a_{k+1}(r), s)$.

This completes the proof that $F^a_{k+1}(rs) \leq G^a_{k+1}(F^a_{k+1}(r), s)$. For the converse inequality, we need another lemma.

Lemma 5.6: For all runs rs and $o = O_i(s)$ we have

(1) $J_{k,i,o}(U^a_{k,i}(r)) \subseteq U^a_{k,i}(rs)$ and

(2) $J_{k,i,o}(U^a_{k,i}(rs)) \subseteq U^a_{k,i}(rs)$.

Proof: For (1), suppose that v is an i-child of $J_{k,i,o}(U^a_{k,i}(r))$. We consider the two clauses of the definition of $J_{k,i,o}$. First, if v arises from the first clause of the definition of $J_{k,i,o}$ then $v \in U^a_{k,i}(r)$ and $O_i(root(v)) = O_i(s)$. By the definition of $U^a_{k,i}$ there exists a run $r' \approx_i r$ such that $v = obj_i(F^a_k(r'))$. Thus $O_i(fin(r)) = O_i(fin(r')) =$

$O_i(root(v)) = O_i(s)$, which implies that $rs \approx_i r$. Hence $rs \approx_i r'$, and we obtain that $v = obj_i(F^a_k(r'))$ is in $U^a_{k,i}(rs)$.

Next, if v arises from the second clause then there exists $v' \in U^a_{k,i}(r)$, a state s' such that $O_i(s') = O_i(s)$ and $root(v')Ts'$, and we have $v = G^a_k(v', s')$. From the definition of $U^a_{k,i}$ we obtain that $v' = obj_i(F^a_k(r'))$ for some run $r' \approx_i r$. This run satisfies $fin(r') = root(v')$, so we have that $r's'$ is a run and $r's' \approx_i rs$. Thus, using the induction hypothesis, $v = G^a_k(v', s') = G^a_k(obj_i(F^a_k(r')), s') = obj_i(F^a_k(r's'))$. It therefore follows from the fact that $r's' \approx_i rs$ and the definition of $U^a_{k,i}$ that v is in $U^a_{k,i}(rs)$. This completes the proof of (1).

For (2), suppose v is in $J_{k,i,o}(U^a_{k,i}(rs))$. If v arises from the first clause of the definition of $J_{k,i,o}$, then it is immediate that v is in $U^a_{k,i}(rs)$. If v arises from the second clause of the definition of $J_{k,i,o}$ then there exists $v' \in U^a_{k,i}(rs)$, a state s' such that $O_i(s') = O_i(s)$ and $root(v')Ts'$, and we have $v = G^a_k(v', s')$. It follows using the definition of $U^a_{k,i}$ that there exists a run $r' \approx_i rs$ such that $v' = obj_i(F^a_k(r'))$. Since $fin(r') = root(v')Ts'$, we obtain that $r's'$ is a run. Because $O_i(s') = O_i(s)$ we also have that $r's' \approx_i rs$. Now, using the induction hypothesis, we have that $v = G^a_k(v', s') = G^a_k(obj_i(F^a_k(r')), s') = obj_i(F^a_k(r's'))$. Hence v is in $U^a_{k,i}(rs)$ by the definition of $U^a_{k,i}$. □

The inequality $G^a_{k+1}(F^a_{k+1}(r), s) \leq F^a_{k+1}(rs)$ follows directly from Lemma 5.6. To see this, note that Lemma 5.6(1), together with the monotonicity of $J_{k,i,o}$, implies that $(J_{k,i,o})^2(U^a_{k,i}(r)) \subseteq J_{k,i,o}(U^a_{k,i}(rs))$. From this, using Lemma 5.6(2), we obtain $(J_{k,i,o})^2(U^a_{k,i}(rs))) \subseteq U^a_{k,i}(rs)$. Repeating this argument, we see that $(J_{k,i,o})^m(U^a_{k,i}(r)) \subseteq U^a_{k,i}(rs)$ for all $m \geq 1$. By definition of G^a_{k+1}, this implies that $G^a_{k+1}(F^a_{k+1}(r), s)) \leq F^a_{k+1}(rs)$. This completes the proof of Proposition 5.4.

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 initial 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 $g^a_k : I \longrightarrow \mathcal{T}_k$ by $g^a_0(s) = s$ and for $k > 0$ by $g^a_k(s) = G^a_k(f_k(s), s)$, where $f_k(s)$ is the k-tree with root s such that v is an i-child of $f_k(s)$ if and only if $v = obj_i(g^a_{k-1}(s'))$ for some initial state $s' \in I$ with $O_i(s') = O_i(s)$.

Lemma 5.7: For all $s \in I$ we have $g^a_k(s) = F^a_k(s)$.

Proof: By induction on k. For $k = 0$ the result is immediate from the definitions. We suppose that $g^a_{k-1}(s) = F^a_{k-1}(s)$ for all $s \in S$, and show that $g^a_k(s) = F^a_k(s)$ by establishing $g^a_k(s) \leq F^a_k(s)$ and $F^a_k(s) \leq g^a_k(s)$. Clearly $root(F^a_k(s)) = s = root(g^a_k(s))$.

First, we show that $g^a_k(s) \leq F^a_k(s)$. By the induction hypothesis and the definitions of f_k and F^a_k, we have $f_k(s) \leq F^a_k(s)$. The monotonicity of $J_{k,i,o}$ implies that G^a_k is monotonic in its first argument, so we obtain that $g^a_k(s) = G^a_k(f_k(s), s) \leq G^a_k(F^a_k(s), s)$. By Lemma 5.6(2), $G^a_k(F^a_k(s), s) \leq F^a_k(s)$, hence $g^a_k(s) \leq F^a_k(s)$.

Next, we show that $F^a_k(s) \leq g^a_k(s)$. Suppose v is an i-child of $F^a_k(s)$. Then there exists a run r with $s \approx_i r$ and $v = obj_i(F^a_{k-1}(r))$. We show by induction on the length of r that v is an i-child of $g^a_k(s)$. For the basis, suppose that $r = s'$, where s' is an initial state with $O_i(s') = O_i(s) = o$. By the (outer) induction hypothesis, $F^a_{k-1}(s') = g^a_{k-1}(s')$, so v is an i-child of $f_k(s)$. Since $O_i(root(v)) = O_i(s') = o$, it follows by means of the first clause of the definition of $J_{k,i,o}$ that v is an i-child of $G^a_k(f_k(s), s) = g^a_k(s)$. This establishes the basis. For the inductive step, we need to show that if $r's'$ is a run with $O_i(s') = O_i(s)$ and $obj_i(F^a_{k-1}(r'))$ is an i-child of $G^a_k(f_k(s), s)$ then $obj_i(F^a_{k-1}(r's'))$ is an i-child of $G^a_k(f_k(s), s)$. This is immediate from Lemma 5.5(2). $\square$

Combining Corollary 5.2, Lemma 5.7, Proposition 5.4 and Proposition 2.2, we obtain the following results, analogous to Theorem 4.4 and Corollary 4.5 respectively.

Theorem 5.8: For each sentence φ of $\mathcal{L}_n$ of alternation depth k or less, $(E^a, r) \models \varphi$ can be decided in time $O(C^c_{k-1} \cdot (|r| + |\varphi|))$.

Corollary 5.9: Let φ 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.

6 Common Knowledge

The results of Section 4 and Section 5 show that for both the synchronous and asynchronous semantics, it is possible to do model checking by means of an incremental algorithm, provided formulae do not involve the common knowledge operator. In particular, the set of runs satisfying a given formula of this kind is in both cases regular, hence linear time decidable. Does the same hold for formulae involving common knowledge? Unfortunately, it does not, in a very strong sense. In this section we establish lower bounds for model checking common knowledge formulae. These bounds strongly suggest that, in general, no efficient incremental approach, nor indeed any polynomial time algorithm, exists for such formulae. This negative result is obtained for both the synchronous and asynchronous semantics. However, whereas formulae in $\mathcal{L}_n$ are essentially of the same order of complexity with respect to these two semantics, common knowledge formulae turn out to be considerably more complex in the asynchronous case. We begin with the synchronous semantics.

Theorem 6.1: There exists an environment E for two agents and a propositional constant p such that the set $\{r \in runs(E) \mid (E, r) \models C_{\{1,2\}}p\}$ is PSPACE hard (under LOGSPACE reductions).

Note that this lower bound is tight, by Proposition 2.5. It follows from Theorem 6.1 that the set of runs corresponding to a common knowledge formula is generally not regular (else PSPACE equals LOGSPACE, which is known to be false). Thus Corollary 4.5 does not generalize to common knowledge formulae. Moreover, it is very unlikely that even a polynomial time algorithm exists for model checking such formulae, for this would imply that PSPACE is equal to PTIME.

We will prove Theorem 6.1 by showing how to encode 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 machine. Our hardness proof will exploit this resemblance: runs will correspond to configurations of a Turing machine, and a single step through the derivation relation of this machine will correspond to taking two steps in the Kripke structure, first to an intermediate configuration through the relation $\sim_1$ and then through the relation $\sim_2$.

Let M be a Turing machine solving a PSPACE complete problem, with tape symbols Σ and states Q. We take M to consist of a set of quintuples $\langle s, q, s', d, q' \rangle$, representing that if the head is scanning a cell containing symbol $s \in \Sigma$ while the machine is in state $q \in Q$, then in the next step of the computation the machine writes symbol $s' \in \Sigma$ in the current location, moves the head in direction d, which is either L or R, (representing left or right, respectively) and assumes the new state $q' \in Q$. We assume without loss of generality that M moves the head to the right hand side of the allotted space before halting, and halts only when it accepts. We also assume that Σ contains the blank symbol and a special set of end of tape symbols used by M to ensure that the head does not move out of the allotted space during any computation. However, it is convenient to introduce two additional markers ϵ_l and ϵ_r, representing positions to the left and right of the ends of the tape, which the head may never occupy. A *configuration* of the machine may be represented as a string r of the form

$$\epsilon_l s_1 s_2 \dots s_k \langle s_{k+1}, q \rangle s_{k+2} \dots s_n \epsilon_r.$$

For $i = 1 \dots n$ the symbol $s_i \in \Sigma$ represents the contents of the i-th cell of the tape. The pair $\langle s_{k+1}, q \rangle \in \Sigma \times Q$ represents that the machine is scanning cell $k+1$ of the tape and is in state q. If r' is the configuration reached in one step from the configuration r, and these configurations have the same length, then we write $r \Rightarrow r'$. We write $r \Rightarrow^* r'$ for the transitive closure of this relation. It is convenient to define Δ_L to be the set of pairs $\langle s, q \rangle$ such that M contains a quintuple of the form $\langle s, q, t, L, q' \rangle$. Intuitively, Δ_L encodes the situations in which the head moves to the left. Similarly, we define Δ_R (representing cases in which the machine moves to the right) to be the set of pairs $\langle s, q \rangle$ such that M contains a quintuple $\langle s, q, t, R, q' \rangle$.

We now describe the environment $E = \langle S, I, T, O_1, \dots, O_n, V \rangle$ used to encode the Turing machine. This environment will have a set S of states consisting of the the markers ϵ_l, and ϵ_r, all symbols in Σ, all pairs in $\Sigma \times Q$, a special state $*$, as well as

1. for each $h \in \Delta_L$ and $s \in \Sigma$, states $L_{s,h}$ and $H_{s,h}$, and
2. for each $h \in \Delta_R$ and $s \in \Sigma$, states $H_{h,s}$ and $R_{h,s}$.

The symbols $L_{s,h}$, $H_{s,h}$, $H_{h,s}$ and $R_{s,h}$ will play the role of special markers in the runs intermediate between configurations. Their precise meaning will be explained below. The set of initial states will be $I = \{\epsilon_l\}$. We take *Prop* to consist if the single propositional constant p, which holds only at the state ϵ_r. For the transition relation T, it is convenient to write $s \to t$ instead of sTt. The set of transitions of E will consist of the following:

1. $s \to s'$, for each $s, s' \in \Sigma$,
2. $s \to h$ and $h \to s$ for each $s \in \Sigma$ and $h \in \Sigma \times Q$,
3. $s \to L_{s',h} \to H_{s',h} \to s''$ for all $s, s', s'' \in \Sigma$ and $h \in \Delta_L$,
4. $s \to H_{h,s'} \to R_{h,s'} \to s''$ for all $s, s', s'' \in \Sigma$ and $h \in \Delta_R$,
5. $\epsilon_l \to s$ and $\epsilon_l \to h$ and $\epsilon_l \to H_{h,s}$ and $\epsilon_l \to L_{s,h'}$ for all $s \in \Sigma$, $h \in \Delta_R$ and $h' \in \Delta_L$,
6. $s \to \epsilon_r$ and $h \to \epsilon_r$ and $R_{h,s} \to \epsilon_r$ and $H_{s,h'} \to \epsilon_r$ for all $s \in \Sigma$, $h \in \Delta_R$ and $h' \in \Delta_L$,
7. $\langle s, q\rangle \to *$, where q is the final state of M.

In particular, we note that each configuration c of the form described above is a run of E. There exist other runs of E, but we will be primarily interested in the runs which correspond to configurations, and in another set of runs to be introduced shortly.

We now specify the observation functions of the environment. Since the actual values taken by these functions are of no interest, it suffices to list the equivalence classes of the equivalence relations R_i defined by sR_it when $O_i(s) = O_i(t)$. We will write $[s]_i$ for the equivalence class of R_i containing state s. If $s \in \Sigma$ and $h \in \Sigma \times Q$ then

$$\begin{aligned}[s]_1 &= \{s\} \cup \{L_{s,h} \mid h \in \Delta_L\} \cup \{R_{h,s} \mid h \in \Delta_R\} \\ [h]_1 &= \{h\} \cup \{H_{s,h} \mid s \in \Sigma\} \cup \{H_{h,s} \mid s \in \Sigma\}.\end{aligned}$$

The only other non-trivial equivalence class of R_1 is $\{\epsilon_r, *\}$. Next, if $s' \in \Sigma$ then

$$\begin{aligned}[s']_2 = \{s'\} &\cup \{H_{t,\langle s,q\rangle} \mid \langle s, q, s', L, q'\rangle \in M,\ t \in \Sigma\} \\ &\cup \{H_{\langle s,q\rangle,t} \mid \langle s, q, s', R, q'\rangle \in M,\ t \in \Sigma\}\end{aligned}$$

and if $\langle t, q'\rangle \in \Sigma \times Q$ then

$$\begin{aligned}[\langle t, q'\rangle]_2 = \{\langle t, q'\rangle\} &\cup \{L_{t,\langle s,q\rangle} \mid \langle s, q, s', L, q'\rangle \in M,\ t \in \Sigma\} \\ &\cup \{R_{\langle s,q\rangle,t} \mid \langle s, q, s', R, q'\rangle \in M,\ t \in \Sigma\}\end{aligned}$$

All other equivalence classes of R_2 are singletons. Note that for distinct $s, s' \in \Sigma$ and distinct $h, h' \in \Sigma \times Q$, we have for each agent $i = 1, 2$ that $[s]_i \cap [s']_i = \emptyset$, $[s]_i \cap [h]_i = \emptyset$ and $[h]_i \cap [h']_i = \emptyset$, so the sets described above do indeed partition the state space into disjoint classes. This completes the description of the environment.

We now introduce a class of runs that will play the role of intermediate points in the simulation of $\Rightarrow$ as $\sim_1$ followed by $\sim_2$. If $r = \epsilon_l s_1 s_2 \ldots s_k \langle s_{k+1}, q\rangle s_{k+2} \ldots s_n \epsilon_r$ is a configuration, then define the sequence $\hat{r}$ by

$$\hat{r} = \epsilon_l s_1 s_2 \ldots s_{k-1} L_{s_k,\langle s_{k+1},q\rangle} H_{s_k,\langle s_{k+1},q\rangle} s_{k+2} \ldots s_n \epsilon_r$$

when $\langle s_{k+1}, q\rangle \in \Delta_L$, and

$$\hat{r} = \epsilon_l s_1 s_2 \ldots s_k H_{\langle s_{k+1},q\rangle,s_{k+2}} R_{\langle s_{k+1},q\rangle,s_{k+2}} s_{k+3} \ldots s_n \epsilon_r$$

when $\langle s_{k+1}, q\rangle \in \Delta_R$. In both cases the sequence $\hat{r}$ is a run of E. In the first case, note that the information that the k-th tape cell will contain the head in the configuration to follow r is not locally available from the symbol s_k occupying that cell. Intuitively, the role of the symbol $L_{s_k,\langle s_{k+1},q\rangle}$ is to make this information locally available. Indeed, note that the next contents of the k-th cell may be directly computed from the symbol $L_{s_k,\langle s_{k+1},q\rangle}$. Intuitively, the symbol $H_{s_k,\langle s_{k+1},q\rangle}$ encodes information about the cell currently occupied by the head, together with the contents of the cell to the left to which the head will move next. Again, this information suffices to compute the next contents of this cell. The intuitive meaning of the symbols $H_{\langle s_{k+1},q\rangle,s_{k+2}}$ and $R_{\langle s_{k+1},q\rangle,s_{k+2}}$ is analogous. The next two lemmas formalize these intuitions.

Lemma 6.2: If r is a configuration in which the Turing machine state is not final, then there exist precisely two runs r' with $r \sim_1 r'$, namely r itself and $\hat{r}$. If r is a final (accepting) configuration then $r \sim_1 r'$ for some run r' with final state $*$.

Proof: We consider just the case that the Turing machine state q is not the final accepting state: the remainder of the result requires only a trivial verification. Let $r = \epsilon_l s_1 s_2 \ldots s_k \langle s_{k+1}, q\rangle s_{k+2} \ldots s_n \epsilon_r$ and suppose $r \sim_1 r' = x_0 x_1 \ldots x_{n+1}$, where each x_i is in S. Suppose that xyz are any three consecutive states in r not containing the head symbol $\langle s_{k+1}, q\rangle$ and that $x'y'z'$ is a sequence of states with $xyz \sim_1 x'y'z'$. Note that state y must be a tape symbol in Σ. By definition of the transition relation, if $y' = L_{h,y}$ for some $h \in \Sigma \times Q$ then necessarily $z' = H_{h,y}$. But since z is either ϵ_r or in Σ, we cannot then have $z R_1 z'$, so this is a contradiction. It follows that y' is not of the form $L_{h,y}$. A similar argument shows that y' is not of the form $R_{y,h}$. The only other state in $[y]_1$ is y itself, so we obtain that $y' = y$. Since this argument applies to any substring of r of length three, (and clearly $x_0 = \epsilon_l$ and $x_{n+1} = \epsilon_r$) we see that r' is equal to r, except for possibly the three positions x_k, x_{k+1} and x_{k+2}.

Suppose that $\langle s_{k+1}, q\rangle \in \Delta_L$. Now if $x_{k+1} = H_{y,\langle s_{k+1},q\rangle}$ for some y, then it follows from the definition of T and R_1 that $y = s_k$ and $x_k = L_{s_k,\langle s_{k+1},q\rangle}$. Further, we must have $x_{k+1} \in \Sigma \cup \{\epsilon_r\}$. It follows that x_{k+1} is identical to the $k+1$-th element of r, and consequently $r' = \hat{r}$. By symmetry, we also obtain $r' = \hat{r}$ when $\langle s_{k+1}, q\rangle \in \Delta_R$. Finally, if $x_{k+1} = s_{k+1}$ then straightforward considerations show $r' = r$. $\Box$

Lemma 6.3: If r is a configuration, then $r \sim_2 r'$ if and only if $r' = r$ or $r' = \hat{c}$ for some configuration c such that $c \Rightarrow r$

Proof: The implication from right to left is readily checked, so we prove just the implication from left to right. Let $r = \epsilon_l s_1 s_2 \ldots s_k \langle s_{k+1}, q\rangle s_{k+2} \ldots s_n \epsilon_r$ and suppose $r \sim_2 r' = x_0 x_1 \ldots x_{n+1}$. Arguments similar to those in Lemma 6.2 show that r' is identical to r except for possibly x_k, x_{k+1} and x_{k+2}. Suppose first $x_{k+1} = L_{s_{k+1},\langle s,q'\rangle}$. Then by definition of T we must have $x_k \in \Sigma \cup \{\epsilon_l\}$, hence x_k is equal to the k-th element of r. In addition, we must have $x_{k+2} = H_{s_{k+1},\langle s,q'\rangle}$. Since $x_{k+2} R_2 s_{k+2}$, it follows that $\langle s, q', s_{k+2}, L, q\rangle \in M$. This shows that $r' = \hat{c}$ for a configuration c such that $c \Rightarrow r$. A similar argument applies if $x_{k+1} = R_{\langle s,q'\rangle, s_{k+2}}$. Finally, if $x_{k+1} = \langle s_{k+1}, q\rangle$ then it is easy to show that $r' = r$. □

Putting together Lemma 6.2 and Lemma 6.3, we see that if r_0 is an initial configuration containing sufficient tape cells for the computation of M and if

$$r_0 \Rightarrow r_1 \Rightarrow r_2 \Rightarrow \ldots \Rightarrow r_{n-1} \Rightarrow r_n$$

is an initial segment of the computation of M, then we have

$$r_0 \ \sim_1 \ \hat{r}_0 \ \sim_2 \ r_1 \ \sim_1 \ \hat{r}_1 \ \sim_2 \ r_2 \ \sim_1 \ldots \ \sim_2 \ r_{n-1} \ \sim_1 \ \hat{r}_{n-1} \ \sim_2 \ r_n.$$

If r_n is an accepting configuration, then we have in addition that $r_n \sim_1 r$ for some run r with final state $*$, for which $(E, r) \models \neg p$. It follows that if r_0 describes an initial configuration for an input on which M eventually accepts, as well as sufficient space for the computation, then $(E, r_0) \models \neg C_{\{1,2\}} p$.

Suppose now that M does not accept (or halt) when started in configuration r_0. We would like to show that $(E, r_0) \models C_{\{1,2\}} p$. This does not follow directly, since the linear structure above does not completely describe the substructure of M_E generated by the run r_0. For each r_{i+1}, there may exist configurations r other than r_i such that $r \Rightarrow r_{i+1}$, and hence $r \sim_1 \hat{r} \sim_2 r_{i+1}$. By alternating the relations $\sim_1$ and $\sim_2$, we may simulate the computation of M *backwards* as well as forwards, and the backward steps need not be deterministic.[8] However, the forward steps are deterministic, so backward steps cannot take us to configuration from which a forward computation would halt. We see that $r_0 \sim_{\{1,2\}} r$ if and only $r = r_1$ or $r = \hat{r}_1$ for some configuration r_1 such that there exists a (non-halting) configuration r' with $r_0 \Rightarrow^* r'$ and $r_1 \Rightarrow^* r'$. Since the final state of any run r_1 of the form of a configuration is ϵ_r, as is the final state of $\hat{r}_1$, we have $(E, r) \models p$ for all such runs r. Thus, we obtain $(E, r_0) \models C_{\{1,2\}} p$. This shows that $(E, r_0) \models C_{\{1,2\}} p$ if and only if the computation of M from the initial configuration r_0 does not halt, and completes the proof of Theorem 6.1.

We remark that this proof is closely related to tiling problems [Lewis, 1979], finite versions of which have been studied as an alternative foundation for the theory of computational complexity [Lewis and Papadimitriou, 1981; Chlebus, 1986; Gradel,

[8] Actually, it is possible to make the backward steps deterministic also, but we do not need this fact. A Turing machine is said to be *reversible* if for every configuration c there exists at most one configuration c' such that $c' \Rightarrow c$. It is known [Bennet, 1989] that every Turing machine running in space $f(n)$ can be simulated by a reversible Turing machine running in space $f(n)^2$.

1990]. A tiling problem corresponding to PSPACE may be formulated as follows [Gradel, 1990]. An instance of the problem consists of a finite set D of dominoes and two binary relations H, V on D, together with a number n determining an infinite lattice $L = \{0 \ldots n\} \times \{0, 1, 2, \ldots\}$. The problem is to decide the existence of a tiling of L by D, that is, a mapping $\tau : L \to D$ such that for all $(x, y) \in L$, $\tau(x, y) H \tau(x+1, y)$ and $\tau(x, y) V \tau(x, y+1)$, provided these values are defined. Here the horizontal relation H may be thought of as corresponding to our transition relation T, and the vertical relation V may be thought of as corresponding to our $\sim_i$. The principle difference between tiling and the reachability problem for common knowledge is that we restrict to vertical relations which are equivalence relations: this difference accounts for the extra complexity in our proof.

We now turn to a lower bound for common knowledge formulae in the asynchronous interpretation of environments. We first note that the asynchronous semantics is able to simulate the synchronous semantics by means of a straightforward construction. In order to turn an asynchronous system into a synchronous system, all we need to do is add a clock, visible to all agents. Because we assume perfect recall, this clock need not have an infinite number of states: two alternating states suffice.

To make this precise in our framework, let $E = \langle S, I, T, O_1, \ldots, O_n, V \rangle$ be an environment. Define the *doubled* environment

$$D(E) = \langle S', I', T', O'_1, \ldots, O'_n, V' \rangle$$

as follows. We take the states S' to be the set $S \times \{0, 1\}$. We will write these states in the format $s \cdot j$ where $s \in S$ and $j \in \{0, 1\}$. Intuitively, s is the state of the environment and j is the state of the clock. The valuation V' is defined by $V'(s \cdot j, p) = V(s, p)$. The transitions T' will be precisely those of T, but with the requirement that the clock state changes at each step, i.e.

$$T' = \{(s \cdot j,\ t \cdot k) \mid (s, t) \in T,\ j, k \in \{0, 1\} \text{ and } j \neq k\}.$$

The observation functions of E' are defined precisely as in E, but we make the clock visible to all agents: for each agent i we let $O'_i(s \cdot j) = O_i(s) \cdot j$. In terms of the corresponding equivalence relations R'_i on states, we have $(s \cdot j) R'_i (s' \cdot j')$ just when $s R_i s'$ and $j = j'$. Finally, we take the initial states I' to consist of all states $s \cdot 1$ where $s \in I$.

With these definitions, there exists a one-to-one correspondence between the runs of E and the runs of $D(E)$. This correspondence is established by the *doubling function* d, which maps a run $s_1 \ldots s_n$ of E to the run $s_1 \cdot 1$, $s_2 \cdot 0$, $s_3 \cdot 1$, $s_4 \cdot 0, \ldots, s_n \cdot (n \ mod \ 2)$ of $D(E)$. Clearly, this function preserves the valuation. Furthermore, for all runs r, r' of E, we have $d(r) \approx_i d(r')$ if and only if $r \sim_i r'$. This is because each agent is able to detect every state transition in $D(E)$. Thus, even under the asynchronous interpretation, $D(E)$ behaves like a synchronous environment, and we have the following result.

Proposition 6.4: For every environment E, the asynchronous Kripke structure $M^a_{D(E)}$ is isomorphic to the synchronous Kripke structure M_E.

This result implies that the asynchronous interpretation of environments makes knowledge formulae at least as complex as the synchronous interpretation, so by Theorem 6.1 we cannot expect to generalize the results of Section 5 to common knowledge formulae. In fact, the asynchronous interpretation is much more complex than this argument suggest, as the following result shows.

Theorem 6.5: 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 $(F^a, r) \models C_{\{1,2\}}p$.

Proof: By reduction from the Halting problem for Turing machines. Let M be a universal Turing Machine. We construct an environment F which implements M in such a way that whenever x is an input of the machine M, there exists a run $r(x)$ such that $(F^a, r(x)) \models C_{\{1,2\}}p$ if and only if M halts on input x. Intuitively, the simulation works in two parts. First, exploiting the fact that asynchrony may make a run indistinguishable from arbitrarily long runs, we guess the amount of space required for the simulation of M on x. Next, we use a *synchronous* environment to perform the computation of M on x, restricted to the amount of space guessed in the first part. This is achieved by means of the doubled environment $D(E)$, where E is an environment simulating M under the synchronous semantics. Note that we no longer assume that M is space bounded, but we do assume M requires new tape cells only to the right, i.e. M never tries to move to the left of the first cell in a configuration. We also assume that M never writes the blank symbol, so that all configurations consist of a sequence of non-blank symbols followed by some number of blanks.

Formally, the environment F is constructed as follows. We write Γ for the union of the set $\Sigma \setminus \{\beta\}$ with the set $\Sigma \times Q$, where Σ is the set of tape symbols of M, Q is the set of states of M and β is the blank tape symbol. Let E be the environment for M described in the proof of Theorem 6.1. The states of the environment F will consist of all states of the environment $D(E)$, together with a state $\overline{a} \cdot j$ for each element a of $\Gamma \cup \{\beta, \epsilon_l, \epsilon_r\}$ and $j \in \{0, 1\}$. These additional states we will call *bar-states.* The valuation of the environment F makes the proposition p true at $\epsilon_r \cdot j$ and $\overline{\epsilon_r} \cdot j$ for $j = 0, 1$, but false everywhere else.

We take the set of transitions of the environment F to consist of all the transitions of the environment $D(E)$, plus the following:

1. $\overline{\epsilon_l} \cdot 1 \to \overline{a} \cdot 0$ for each $a \in \Gamma$
2. $\overline{a} \cdot j \to \overline{b} \cdot k$ for all $a, b \in \Gamma$ and bits $j \neq k$,
3. $\overline{a} \cdot j \to \overline{\beta} \cdot k$ for all $a \in \Gamma$ and bits $j \neq k$,
4. $\overline{\beta} \cdot j \to \overline{\beta} \cdot k$ for all bits $j \neq k$, and

5. $\overline{\beta} \cdot j \to \overline{\epsilon_r} \cdot k$ for all bits $j \neq k$.

If $r = s_1 s_2 \ldots s_m$ is a sequence of states of E, write $\overline{r}$ for the sequence $\overline{s_1} \ldots \overline{s_m}$. If σ is a string over Γ and $k > 0$ is a natural number, then define the run

$$r(\sigma, k) = d(\overline{\epsilon_l}\, \overline{\sigma}\, \overline{\beta}^k\, \overline{\epsilon_r}).$$

When σ represents an *instantaneous description* of the machine M (i.e. a configuration without the end markers ϵ_l, ϵ_r), we may interpret $r(\sigma, k)$ as a configuration consisting of σ, plus an allotment of k tape cells for the computation. Note that if a run contains a bar-state, then all states in that run are bar-states. Thus, the runs of F may be divided into two classes: the class of runs of the form $d(r)$, where r is a run of $D(E)$, and the class of runs which are prefixes of $r(\sigma, k)$ for some $\sigma \in \Gamma^*$ and $k \geq 0$.

Finally, we describe the observation functions of the environment F by listing the equivalence classes of the corresponding relations R_i. We begin with the equivalence classes of R_1. For each $a \in \Gamma \cup \{\epsilon_l, \epsilon_r\}$ and $j \in \{0, 1\}$, the bar-state $\overline{a} \cdot j$ will comprise a singleton equivalence class. The set $\{\overline{\beta} \cdot 0, \overline{\beta} \cdot 1\}$ also forms an R_1 equivalence class. The remaining equivalence classes of R_1 are exactly as in $D(E)$, i.e. $s \cdot j \in [t \cdot k]_1$ just when $j = k$ and sR_1t according to the description of R_1 in the proof of Theorem 6.1. To obtain the equivalence classes of the relation R_2 in F, we modify the equivalence classes of the environment $D(E)$ by adding $\overline{a} \cdot j$ to $[a \cdot j]_2$ for each $a \in \Gamma \cup \{\epsilon_l, \epsilon_r, \beta\}$ and bit j. Note that this definition is coherent, because each equivalence class of R_2 in $D(E)$ contains at most one state of the form $s \cdot j$ with $s \in \Gamma \cup \{\epsilon_l, \epsilon_r, \beta\}$.

Let us now see how these observation relations determine the asynchronous indistinguishability relations on runs. First, noting that no equivalence class of R_1 contains both a bar state and a state of $D(E)$, we see that no run of $D(E)$ is $\approx_1$-equivalent to any run consisting of bar states. In particular, we obtain the following:

- If r_1 is a run of E then $d(r_1) \approx_1 r$ iff $r = d(r_2)$ for some run r_2 of E with $r_1 \sim_1 r_2$ with respect to the environment E. Thus, if r_1 is a non-final configuration of M, then by Lemma 6.2 the run r_2 is either r_1 or $\hat{r}_1$. Additionally, if r_1 is a final configuration with the head in the rightmost tape cell, then we have $d(r_1) \approx_1 d(r')$ for some r' with final state $*$.

- For runs of F of the form $r(\sigma, k)$, where σ is a string over Γ and $k > 0$, we have $r(\sigma, k) \approx_1 r$ iff $r = r(\sigma, l)$ for some $l > 0$.

Next, note that the equivalence relation R_2 preserves the clock bit, hence $\approx_2$-equivalent runs are of the same length. Also, each equivalence class of R_2 contains at most one bar-state, so no two distinct runs comprised of bar-states are $\approx_2$-equivalent. Thus, to describe the relation $\approx_2$ it suffices to consider runs of the form $d(r_1)$ with r_1 a run of E. In case r_1 is a configuration of M we find $d(r_1) \approx_2 r$ if and only if either (i) $r = d(r_2)$ for some run r_2 of E with $r_1 \sim_2 r_2$ or (ii) $r = d(\overline{r_1}) = r(\sigma, k)$ for some σ and

k. By Lemma 6.3, case (i) holds just in when $r_2 = r_1$ or $r_2 = \hat{r}$ for some configuration r such that $r \Rightarrow r_1$. (Recall that $\Rightarrow$ refers to the derivation relation of M *restricted to configurations of the same length.*)

Suppose the machine M eventually halts when started with instantaneous description σ, and the computation uses precisely k additional tape cells, halting at the rightmost of these cells. Then there exists a sequence of configurations

$$r_0 \Rightarrow r_1 \Rightarrow r_2 \Rightarrow \ldots \Rightarrow r_{n-1} \Rightarrow r_n$$

where $d(\overline{r_0}) = r(\sigma, k)$ and r_n is a final configuration. By the considerations above we obtain

$$d(r_0) \approx_1 d(\hat{r}_0) \approx_2 d(r_1) \approx_1 d(\hat{r}_1) \approx_2 d(r_2) \approx_1 \ldots \approx_2 d(r_{n-1}) \approx_1 d(\hat{r}_{n-1}) \approx_2 d(r_n)$$

Furthermore, $r(\sigma, 1) \approx_1 r(\sigma, k) \approx_2 d(r_0)$ and $d(r_n) \approx_1 d(r)$ for some run r of E with final state $*$. Since $(F, d(r)) \models \neg p$, this shows that if σ is an instantaneous description from which M eventually accepts, then $(F^a, r(\sigma, 1)) \models \neg C_{\{1,2\}} p$.

Next, we need to show that if σ is an instantaneous description of M from which the computation of M does not halt, then all runs $\{1, 2\}$-accessible from $r(\sigma, 1)$ satisfy p. The situation here is like that in the proof of Theorem 6.1, where we noted that backward steps in the derivation relation of M do not lead to configurations with different termination properties. Things are complicated slightly by the fact that the $\approx_{\{1,2\}}$ access sequence may now contain steps of the form $r(\tau, k) \approx_1 r(\tau, l)$, but all these do is add or delete blank cells, which cannot turn a configuration from which M fails to halt (given sufficient space) into one from which M does eventually halt. Thus, if σ is an instantaneous description from which M does not halt, then $r(\sigma, 1) \approx_{\{0,1\}} r_1$ implies that either $r_1 = r(\tau, k)$ for some τ and k, $r_1 = d(r)$, or $r_1 = d(\hat{r})$ for some non-final configuration r. In every case the final state of r_1 is of the form $\epsilon_r \cdot j$ or $\overline{\epsilon_r} \cdot j$, and we have $(F^a, r_1) \models p$. This establishes that if M does not halt from instantaneous description σ, then $(F^a, r(\sigma, 1)) \models C_{\{1,2\}} p$. $\square$

Somewhat related to the results of this section are a number of papers [Chandra and Stockmeyer, 1976; Peterson and Reif, 1979; Reif, 1984; Reif and Peterson, 1980], which study the complexity of games of incomplete information and develop applications to distributed processing. The framework of these papers is related to ours, but the complexity results developed are different because they deal with the problem of determining the existence of a winning strategy, rather than the problem of computing players' knowledge.

7 Compiling Knowledge Based Programs

In a number of papers [Dwork and Moses, 1990; Hadzilacos, 1987; Halpern and Moses, 1990; Halpern and Zuck, 1992; Mazer, 1986] 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* [Halpern and Fagin, 1989] 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, a knowledge based protocol may be implemented by a 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, 1990] establishes 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 [Moses and Kislev, 1993]. In order to permit the execution of knowledge based protocols, such a language would have to support the compilation of knowledge formulae 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 using finite state automata.

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 of the environment 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 of 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, this in turn influences the agents' knowledge. The model of the present paper 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 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 [Halpern and Moses, 1990; Panangaden and Taylor, 1992]. Thus compilation of common knowledge formulae is likely to be a crucial requirement of knowledge based programming. The PSPACE hardness of common knowledge formulae in synchronous environments shows that one cannot always expect to find efficient implementations for protocols involving tests for 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. Note that the lower bound results state that there *exist* environments in which common knowledge formulae are hard, not that these formulae are hard in *all* environments. The environments constructed in the proofs of Theorem 6.1 and Theorem 6.5 are very unnatural. Although limited, the class of environments studied in this paper is in some respects too broad. It is not clear to what extent these results will persist in more reasonable classes of environments. One class worth investigating is the class of asynchronous message passing systems, in which common knowledge is known to be constant, 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, which deals with the verification of externally ascribed knowledge properties of situated machines [Rosenschein, 1985; Rosenschein and Kaelbling, 1986]. However, while the language of [Rosenschein and Kaelbling, 1986] 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 proposal to develop *Agent Oriented Programming Languages* [Shoham, 1993]. 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 [Isozaki and Shoham, 1992]. 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.

8 Conclusion

We have presented in this paper a model-theoretic approach to computing knowledge 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 perform deduction in this logic. Complete axiomatizations of logics of knowledge and time in systems with perfect recall are known in both the synchronous case [Halpern and Vardi, 1986; Halpern and Vardi, 1988a] and the asynchronous case [Meyden, 1994a]. 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 Π^1_1-complete [Halpern and Vardi, 1989], which implies that there can be no complete axiomatization. This precludes the direct application of logics of 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 4 and Section 5 are theoretically linear time, 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 [Valmari, 1990; Burch *et al.*, 1990; Emerson and Sistla, 1993] 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 generalized to knowledge formulae. 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 in [Fischer and Immerman, 1986], in systems of this form common knowledge formulae may be reduced to knowledge formulae.

Acknowledgments

Thanks to Hirofumi Katsuno for comments on earlier versions of this paper.

References

[Bennet, 1989] C. H. Bennet. Time space trade-offs for reversible computation. *SIAM Journal of Computing*, 18(4):766–776, 1989.

[Burch *et al.*, 1990] 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.

[Chandra and Stockmeyer, 1976] A.K. Chandra and L.J. Stockmeyer. Alternation. In *Proc. 17th Annual ACM Symposium on Theory of Computing*, pages 98–108, 1976.

[Chandra *et al.*, 1981] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

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

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

[Chlebus, 1986] B. Chlebus. Domino-tiling games. *Journal of Computer and System Sciences*, 32:374–392, 1986.

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

[Emerson and Sistla, 1993] 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.

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

[Fagin *et al.*, 1994 forthcoming] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about knowledge.* MIT Press, Cambridge, MA, 1994, forthcoming.

[Fischer and Immerman, 1986] 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.

[Gradel, 1990] E. Gradel. Domino games and complexity. *SIAM Journal of Computing*, 19(5):787–804, 1990.

[Hadzilacos, 1987] 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.

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

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

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

[Halpern and Vardi, 1986] 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.

[Halpern and Vardi, 1988a] 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.

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

[Halpern and Vardi, 1989] 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.

[Halpern and Vardi, 1991] 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.*

[Halpern and Zuck, 1992] 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.

[Halpern, 1987] 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.

[Isozaki and Shoham, 1992] 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.

[Katsuno and Mendelzon, 1991] 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.

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

[Ladner and Reif, 1986] 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.

[Lewis and Papadimitriou, 1981] H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Lewis, 1979] H. R. Lewis. *Unsolvable Classes of Quantificational Formulas.* Addison Wesley, Reading, MA, 1979.

[Mazer, 1986] 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.

[Meyden, 1994a] R. van der Meyden. Axioms for knowledge and time in distributed systems with perfect recall. In *Proc. IEEE Symp. on Logic in Computer Science*, 1994. to appear.

[Meyden, 1994b] R. van der Meyden. Common knowledge and update in finite environments II: Simple environments. Technical report, NTT Basic Research Labs, 1994. in preparation.

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

[Moses and Tuttle, 1988] Y. Moses and M. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3:121–169, 1988.

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

[Parikh and Ramanujam, 1985] 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.

[Peterson and Reif, 1979] G. L. Peterson and J. H. Reif. Multi-person alternation. In *Proc. Symposium on Foundations of Computer Science*, pages 348–363, 1979.

[Reif and Peterson, 1980] J. H. Reif and G. L. Peterson. A dynamic logic of multiprocessing with incomplete information. In *Proc. Symposium on Principles of Programming Languages*, pages 193– 202, 1980.

[Reif, 1984] J. H. Reif. The complexity of two-player games with incomplete information. *Journal of Computer and System Sciences*, 29:274–301, 1984.

[Rosenschein and Kaelbling, 1986] 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.

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

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

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

[von Nuemann and Morgenstern, 1953] J. von Nuemann and O. Morgenstern. *Theory of Games and Economic Behaviour.* Princeton University Press, Princeton, NJ, third edition, 1953.

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