Common Knowledge and Update in Finite Environments

Ron van der Meyden

Computing Science
University of Technology, Sydney
PO Box 123, Broadway NSW 2007
Australia

email: ron@socs.uts.edu.au
fax: +61 2 330 1807

To appear: 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

August 25, 1997

Abstract

Logics of knowledge have been shown to provide a useful approach to the high level specification and analysis of distributed systems. It has been proposed that such systems can be developed using knowledge-based protocols, in which agents' actions have preconditions that test their state of knowledge. Both computer-assisted analysis of the knowledge properties of systems and automated compilation of knowledge-based protocols require the development of algorithms for the computation of states of knowledge. This paper studies one of the computational problems of interest, 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 are assumed to have perfect recall, and may operate synchronously or asynchronously. It is shown that, in this setting, model checking of common knowledge formulae is 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.

Keywords: logic of knowledge, epistemic logic, modal logic, model checking, distributed systems, multi-agent systems, concurrency, knowledge-based protocols, algorithms, complexity

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, 1988]. 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, e.g., [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.*, 1995a; 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 investigate these questions for a very simple abstract framework. 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 *traces* 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.

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, Stratego and Kriegspiel.

Since the observations do not determine the state, agents, in general, have only incomplete information about the current state. However, an agent potentially knows more than it learns from its most recent observation. 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 [Fagin *et al.*, 1995b], 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 an $S5_n$ *Kripke structure*, a semantic 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 the Kripke structure generated by an environment. The question of how to evaluate modal formulae in a Kripke structure is the *model checking* problem. Model checking knowledge formulae in finite Kripke $S5_n$ 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 its 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 5 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. 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 that provided the formulae we wish to check contain no occurrences of common knowledge operators, and k or fewer alternations of knowledge operators, there exists a knowledge representation, called k-trees, that may be updated in constant time and queried in 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, it is shown that fixed formulae not involving common knowledge correspond to regular sets of traces, so that model checking may be implemented by means of finite state automata in this case.

These results contribute to the goal of providing automated support for knowledge-theoretic analysis of distributed systems by beginning to delineate the feasible from the infeasible problems in this methodology. In particular, the results of Section 4 show that for bounded alternation depth formulae, it is possible to finitely represent and compute states of knowledge with respect to the perfect recall semantics. These results may be applied to show that certain problems concerning the verification of knowledge-based programs are, in principle, feasible, as is discussed in Section 6.

The knowledge representation, k-trees, used in the results of Section 4, may be of independent interest. Our use of such structures 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 advocated in [Halpern and Vardi, 1991].

The structure of the paper is as follows. Section 2 recalls the standard semantics of knowledge, introduces our formal model of distributed environments, and defines the problems we study in this paper. Section 3 establishes an upper bound for model checking common knowledge formulae in synchronous environments. Section 4 deals with bounded alternation depth formulae, showing that there exists an efficient incremental approach to model checking such formulae in both synchronous and asynchronous environments. Section 5 concerns lower bounds for model checking common knowledge formulae in the synchronous and the asynchronous cases. The relation of our work to compilation of knowledge-based programming languages is discussed in Section 6, and Section 7 contains concluding remarks.

2 Definitions

In this section we describe the formal model of distributed systems and the ascription of states of knowledge to agents. We begin in Section 2.1 by recalling the general semantics of knowledge using Kripke structures, as well as a number of standard constructions on such structures. In Section 2.2 we define finite state environments, and give two ways of associating a Kripke structure with an environment. In Section 2.3 we define the main problem studied in this paper, the model checking problem in these Kripke structures, and describe a number of ways of measuring its complexity.

2.1 Kripke Structures

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 $i = 1 \ldots n$, as well as a modal operator C_G for each set G of two or more agents [Halpern and Moses, 1992]. 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 $i = 1 \ldots n$.
4. If φ is a formula then $C_G\varphi$ is a formula, for each subset G of $\{1 \ldots n\}$ of cardinality two or more.

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

The language $\mathcal{L}_n^C$ is interpreted in the following class of models. A *Kripke structure* for n agents is a tuple

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

where W is a set of *worlds*, $\mathcal{K}_i$ is a binary relation on W for each i, and $V : W \times \mathit{Prop} \to \{0, 1\}$ is a valuation. If all the relations $\mathcal{K}_i$ are equivalence relations then we say that

M is an $S5_n$ structure. Most of the Kripke structures we deal with in this paper will be $S5_n$ structures. The semantics of $\mathcal{L}_n^C$ is given as usual, the only nonstandard point being the truth definition for the common knowledge operator. If G is a set of agents then we say a world u is G-accessible from a world w if there exists a sequence of worlds $u_0, u_1, \ldots u_n$ such that $u_o = w$, $u_n = u$ and for all $i = 0 \ldots n-1$ there exists an agent $j \in G$ such that $u_i \mathcal{K}_j u_{i+1}$. The following clauses give an inductive definition of the relation $M, w \models \varphi$, representing that formula φ is *true* at world w in the Kripke structure M:

1. For propositional constants $p \in Prop$, $M, w \models p$ if $V(w, p) = 1$.
2. $M, w \models \varphi_1 \wedge \varphi_2$ if $M, w \models \varphi_1$ and $M, w \models \varphi_2$.
3. $M, w \models \neg\varphi_1$ if not $M, w \models \varphi_1$.
4. $M, w \models K_i\varphi$ if $M, u \models \varphi$ for all worlds u such that $w\mathcal{K}_i u$.
5. $M, w \models C_G\varphi$ if $M, u \models \varphi$ for all worlds u which are G-accessible from w.

Note that if M is an $S5_n$ structure and $G = \{i\}$ is a singleton set then u is G-accessible from w just when $w\mathcal{K}_i u$. Thus, in this case the semantics of C_G collapses to the semantics of K_i, which is why we assumed above 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 at a given 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 2.1: If M is a finite $S5_n$ structure and $\varphi \in \mathcal{L}_n^C$ then $M, w \models \varphi$ can be decided in time $O(|\varphi| \cdot ||M||)$.

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 $M, w \models C_G\varphi$. □

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 2.2: 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 $M', w \models \varphi$.

2.2 Environments

We now define *environments*, our formal model of distributed systems, and describe two ways in which an environment generates a Kripke structure for the interpretation of the agents' knowledge, depending on whether agents are taken to run synchronously or asynchronously.

We wish to model distributed systems in which each state determines an observation for each agent. 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 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 $S \times \mathit{Prop}$ to $\{0, 1\}$.

An environment is *finite-state* if the set S is finite. We consider only finite state environments in this paper.

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. In general, state transitions will be non-deterministic, i.e., for a given state s there may be many states t such that sTt.

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 2.3: 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 $\mathcal{O} = \{\mathit{sent}, \mathit{unsent}, \mathit{rcvd}, \mathit{unrcvd}\}$, representing the values of the variables maintained by the two agents. Thus, the observation functions are given by $O_1(a) = \mathit{unsent}$, $O_1(b) = O_1(c) = \mathit{sent}$ for agent 1 and $O_2(a) = O_2(b) = \mathit{unrcvd}$, $O_2(c) = \mathit{rcvd}$ for agent 2. Taking $\mathit{Prop} = \{p\}$ 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.

Environments may also be used to represent games of incomplete information, such as Bridge, Poker, Battleships and Kriegspiel. In this case, the states represent the possible game positions that may be reached during play, the transition relation captures the legal moves, and the observation functions model the aspects of the position visible to the players. For example, in card games each player typically observes only their own hand and the cards that have been revealed by other players.

Suppose $E = \langle S, I, T, O_1, \ldots, O_n, V\rangle$ is an environment. A *trace* 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 $\mathit{traces}(E)$ for the set of traces of E. Note that if T contains a cycle, the set of traces is infinite. If T is serial, then every trace is the prefix of some strictly longer trace. In other words, no trace terminates. Environments with arbitrarily long traces will be of greatest interest to us. We adopt the following notational convention: constants $s, t, \ldots$ will always denote *states*; traces will be denoted by $r, r', r_1, \ldots$. If r is a trace we write $\mathit{fin}(r)$ for the final state of r. Intuitively, a trace represents a possible history of the system, and the final state of a trace represents

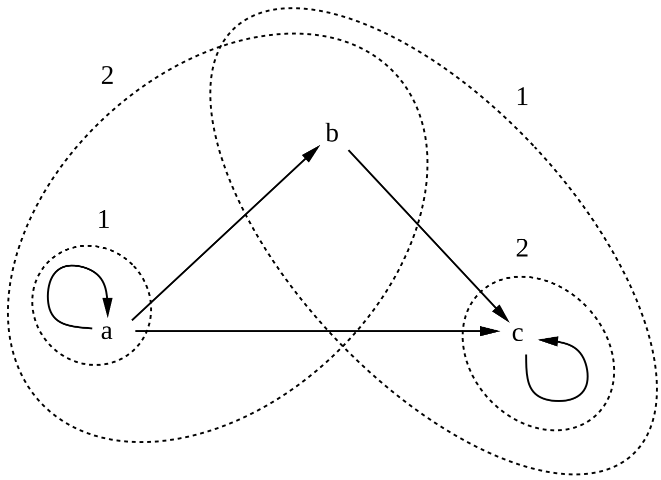

Figure 1: An environment

the "current" state.

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 be essentially a special case of the framework introduced in [Halpern and Moses, 1990], modulo some minor differences that are discussed in Appendix A. We consider in this paper two distinct ways to associate a Kripke structure with an environment, depending on whether agents operate *synchronously* or *asynchronously*. In both cases, agents will have *perfect recall*. (See [Fagin *et al.*, 1995b] for a more extensive motivation of these concepts.) We treat the synchronous case first.

For each agent i we define an equivalence relation on the traces of an environment E. The *synchronous local state* of agent i in a trace $r = s_1 s_2 \ldots s_m$ 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 trace. We say two traces r and r' are *indistinguishable to agent* i, and write $r \sim_i r'$, if $\{r\}_i = \{r'\}_i$. That is, two traces $r = s_1 s_2 \ldots s_m$ 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 trace is which equivalence class of $\sim_i$ contains the actual trace. Note that it is implicit in these definitions that an agent is able to "tell the time," since the number of states in the trace 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 $W_E = \mathit{traces}(E)$, accessibility relations $\mathcal{K}_i = \sim_i$ for $i = 1 \ldots n$, and valuation $V_E : W_E \times \mathit{Prop} \to \{0, 1\}$ given by $V_E(r, p) = V(\mathit{fin}(r), p)$. It is clear that M_E is an $\mathrm{S5}_n$ structure. If r is a trace 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 final state, and the agents' mutual knowledge of the final state. However, as Example 2.3 shows, environments may be constructed so that the final state encodes a limited amount of information about the past.

Example 2.4: Consider the environment E for two agents of Example 2.3. The traces 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 traces of the same length. Note that $O_2(c) = O_2(x)$ if and only if $x = c$. Hence if $r \sim_2 r'$ then $m = m'$, and we find that there are at most two traces r' such that $r \sim_2 r'$, namely those with $k' = k + l - l'$, where l' is either 0 or 1. (One of these traces is just r itself.) A similar argument shows that there exist at most two traces r' such that $r \sim_1 r'$. Thus, the component generated by the traces 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 trace and itself. It follows that $(E, ac^{n-1}) \models (K_2K_1)^j p$ holds for all $j < n-1$, but not for $j \geq n-1$. In particular, we see that even if the message is delivered immediately, the fact of its delivery never becomes common knowledge.[1]

We now define a second semantics for knowledge in environments, in which we do not assume that agents are synchronized. What varies in this second semantics is the definition of agents' local states. Instead of assuming that an agent is aware of every transition made, we 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 trace r, to be the sequence of *distinct* observations of agent i. That is, if r is a trace consisting of a single state s then we define $\{r\}_i^a = O_i(s)$, and for traces of the form rs we define

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

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

[1]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.

Using this function, define two traces 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 trace r in M_E^a.

Example 2.5: We reinterpret the environment of Example 2.3 with respect to the asynchronous semantics. Consider first the asynchronous indistinguishability relation $\approx_1$ for agent 1. If r is a trace of the form a^k, then $\{r\}_1^a = unsent$. For all other traces 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 $\{r\}_1^a = unsent \cdot sent$. Thus, there are precisely two equivalence classes of traces with respect to the relation $\approx_1$. Similarly, it may be shown that the relation $\approx_2$ partitions the set of traces into precisely two classes: the traces of the form $a^k b^l$ and the traces of the form $a^k b^l c^m$ with $m \geq 1$. Note that none of these equivalence classes contains a trace $a^k b^l c^m$ 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 $C_{\{1,2\}} \neg K_1 p$ is true in all traces: it is always common knowledge that agent 1 does not know the message has been delivered.

2.3 Problems Studied

We can now give a precise statement of the problems we study in this paper. At the most general level, we consider the model checking problem in the structures M_E and M_E^a. That is, the problem of deciding whether $(E, r) \models \varphi$ or $(E^a, r) \models \varphi$ hold, given a finite environment E, a trace r of E, and a formula φ in $\mathcal{L}_n^C$. We seek efficient algorithms for these decision problems, or a classification of their complexity if intractable.

Since the number of traces of an environment may be infinite, Proposition 2.1 does not directly apply to model checking in the structures M_E and M_E^a. Indeed, it is not even immediately apparent that the model checking problem in these structures is decidable. In the next section we will establish decidability in the synchronous case. The story is a little more complex in the asynchronous case. For formulae involving common knowledge, the problem turns out to be undecidable in this case — this will be shown in Section 5. However, as will be shown in Section 4.3, it is decidable for formulae in $\mathcal{L}_n$.

A variety of restrictions of these problems are of interest, as are a number of ways of measuring their complexity. At the most general level, one can measure the complexity of model checking when all three of the parameters E, r and φ are varied, with domain the set of all possible finite environments, traces, and formulae in $\mathcal{L}_n^C$ respectively. This also gives an upper bound on the complexity of all restrictions of the model checking problem.

The main motivation for the study of restricted versions of the model checking problem arises from the problem of implementing knowledge-based programs [Fagin *et al.*, 1995a]. We will discuss the relation of the knowledge-based program implementation problem to the model checking problem at greater length in Section 6. For our present purposes, it suffices to note that in knowledge-based programs, an agent must compute at each step of its operation the answer to a given set of queries concerning its knowledge, in order to determine its next action. This motivates three changes of perspective on the general model checking problem.

First, the formulae relevant to an implementation of a knowledge-based program are not all possible formulae in $\mathcal{L}_n^C$, just those that actually occur in the agents' programs. In case of agent i's program, these queries can be assumed to be of the form $K_i\varphi$ for some $\varphi \in \mathcal{L}_n^C$. Thus, we are led to the investigation of the model checking problem for restricted sets Φ of queries of this form.

A second restriction of the general model checking problem concerns the presentation of the input. The model checking problem is stated above from the point of view of an external observer of the system, since it assumes complete information about the trace. However, one can also pose the model checking problem from the point of view of the agents: how can an agent compute the answer to a query about its knowledge, given the information to which it is exposed over a given trace? To formalize this question, observe first that the truth value of the assertion $(E, r) \models K_i\varphi$ depends only upon the local state $\{r\}_i$, rather than upon the trace r as a whole. Thus one can study the *agent-centric* problem of deciding $(E, r) \models K_i\varphi$ for formulae $\varphi \in \Phi$, given $\{r\}_i$ for some trace r of E. Similarly, in the asynchronous case we have the problem of deciding $(E^a, r) \models K_i\varphi$ for formulae $\varphi \in \Phi$ given $\{r\}_i^a$ for some trace r of E.

The final modification to the general model checking problem motivated by the relation to knowledge-based programs concerns the structure of the model checking computation. As we have already noted, the local state of an agent serves only for the purpose of externally ascribing to the agent a state of knowledge. We do not require that the agent actually maintains its local state: any representation that suffices to correctly answer the queries of interest is admissible.

To model an "incremental" notion of implementation of model checking that is related to the notions of implementations of knowledge-based programs studied in [Fagin *et al.*, 1995a; Meyden, 1996a], we introduce a class of automata. Define an automaton with inputs Σ and properties Π to be a tuple of the form $A = \langle Q, q, \mu, \pi \rangle$, where

1. Q is a set of states,

2. $q \in Q$ is an initial state,

3. $\mu : Q \times \Sigma \rightarrow Q$ is a function used to update a state given an input, and

4. $\pi : Q \times \Pi \to \{\mathbf{true}, \mathbf{false}\}$ is a function assigning a truth value to each property in each state of the automaton.

The first three components are standard; the final component simply serves to provide a set of outputs at each state of the automaton. As usual, we may define inductively the state $A(\sigma)$ of the automaton reached on the input of a sequence σ of inputs from Σ by $A(\epsilon) = q$ (where ϵ is the null string), and $A(\sigma \cdot x) = \mu(A(\sigma), x)$.

These automata provide a class of implementations of model checking as follows. If Φ is a set of formulae of $\mathcal{L}_n^C$, we say *A is an implementation in E (or E^a) of model checking queries Φ over traces* when A is an automaton with inputs $\Sigma = S$, the set of states of E, and properties Φ, such that for all traces r of E and queries $\varphi \in \Phi$ we have $\pi(A(r), \varphi) = \mathbf{true}$ if and only if $(E, r) \models \varphi$ (respectively, $(E^a, r) \models \varphi$).

Similarly, to define implementations of the agent-centric version of model checking, suppose that Φ_i is a set of formulae of $\mathcal{L}_n^C$ of the form $K_i\varphi$. Say that the automaton A is an *implementation in E (or E^a) of model checking queries Φ_i over observations of agent i* if A is an automaton with inputs $\mathcal{O}$, the set of observations in E, and properties Φ_i, such that for all traces r of E and queries $\varphi \in \Phi_i$ we have $\pi(A(\{r\}_i), \varphi) = \mathbf{true}$ if and only if $(E, r) \models \varphi$ (respectively, $\pi(A(\{r\}_i^a), \varphi) = \mathbf{true}$ if and only if $(E^a, r) \models \varphi$).

The external and agent-centric versions of the model checking problem are easily seen to be inter-reducible. Given a procedure solving the external model checking problem, we obtain a procedure for the agent-centric model checking problem simply by restriction. In the other direction, suppose we are given for each agent i a procedure $P_i(l, K_i\varphi)$ solving the agent-centric model checking problem for agent i given the local state l and formulae of the form $K_i\varphi$. Note that the formula $C_G\varphi \equiv K_iC_G\varphi$ is valid for any agent i in G. It follows that any formula in $\mathcal{L}_n^C$ can be rewritten as a boolean combination of atomic propositions and formulae of the form $K_i\varphi$. Thus we obtain a procedure for the general model checking problem that uses the final state of the trace r to determine the truth value of atomic propositions at r and makes the call $P_i(\{r\}_i, K_i\varphi)$ (in the asynchronous case, $P_i(\{r\}_i^a, K_i\varphi)$) for subformulae of the form $K_i\varphi$.

Viewing model checking as implemented by automata provides a way to measure its inherent complexity in terms of space requirements. Clearly, maintaining just the local state itself would suffice for the agent to be able to compute queries concerning its knowledge (when this is a decidable problem). This requires an amount of memory growing at least linearly with the length of the trace. Does any knowledge representation more compact than the local state suffice? We will show in Section 4 that for sets of queries Φ of bounded alternation depth, the answer to this question is positive — there exists a procedure that given Φ and an environment E, produces finite state implementations of both agent-centric and external model checking. Section 4.2 deals with the synchronous case, and Section 4.3 with the asynchronous case. A consequence of the work in the latter section is that model checking formulae in $\mathcal{L}_n$ is in fact decidable in the asynchronous case.

3 A General Upper Bound for Synchronous Environments

We now provide an upper bound on the complexity, in the synchronous case, of the most general form of the model checking problem, in which all three of the parameters: the environment, the trace, and the formula, are varied. This also yields an upper bound on the more restricted problems we consider.

The decidability of model checking in the synchronous structure M_E is not too hard to see. Note that because $r \sim_i r'$ implies that r and r' have the same length, the submodel of M_E generated by a trace r contains only traces of the same length as r. Since the set S of states is finite, there are only finitely many such traces, in fact at most $|S|^{|r|}$. Thus the generated submodels are all finite. Using Lemma 2.2 it follows that model checking formulae in $\mathcal{L}_n^C$ is decidable, and the approach of Proposition 2.1 may be used to obtain an algorithm that decides $(E,r) \models \varphi$ in time and space $O(|\varphi| \cdot n \cdot |S|^{2 \cdot |r|})$.

This algorithm is exponential in both space and time. We now show that it is possible to improve the space bound. The observation underlying the algorithm with improved space complexity is that rather than construct the entire structure generated by a trace, we can create traces only as they are needed by the model checking computation, and reuse the space after we are done with them. This yields the following upper bound.

Proposition 3.1: The set $\{ \langle E, r, \varphi \rangle \mid \varphi \in \mathcal{L}_n^C,\ r \in \mathit{traces}(E) \text{ and } (E,r) \models \varphi \}$ is in PSPACE.

Proof: The most convenient way to describe the computation is by means of a standard reduction from alternating computations [Chandra *et al.*, 1981]. An *Alternating Turing machine* resembles a nondeterministic Turing machine inasmuch as each configuration derives up to two successors. However, in addition, each non-final state of the machine is labelled either "universal" or "existential". A computation tree generated by such a machine is *accepting* if either

- it consists of a single leaf node, and the state of the machine in that node is a final accepting state, or
- the root node is labelled by a configuration with a universal state, and there are two subtrees, both of which are accepting, or
- the root node is labelled by a configuration with an existential state, and there are two subtrees, at least one of which is accepting.

The machine accepts an input if it generates an accepting computation tree when started on an initial configuration for that input.

We show how to implement model checking by means of an alternating computation, which we describe as a recursive procedure $P(E, r, \varphi)$, in which the principal recursion is on the structure of the formula φ. The treatment of atomic formulae and the boolean operators is straightforward, so we discuss only the modal operators K_i and C_G of $\mathcal{L}_n^C$. Note that we may write the truth conditions for these in the form

- $(E, r) \models K_i\varphi$ if for all for all traces r' such that r' has the same length as r, if $r \sim_i r'$ then $(E, r') \models \varphi$, and
- $(E, r) \models C_G\varphi$ if for all traces r' of the same length as r, if r' is G-accessible from r then $(E, r') \models \varphi$.

The first of these conditions shows that $P(E, r, K_i\varphi)$ can be determined by an alternating computation that uses $|r|$ steps of universal branching to generate a trace r' of the same length as r, then verifies that $r \sim_i r'$ and recursively calls $P(E, r', \varphi)$.

The second condition requires determination of the G-accessible traces. This may be done efficiently by means of a standard trick for space-efficient graph search in alternating computations. Define the predicate $REACH(r, r', m, G)$ to hold when there exists a path of length at most m from r to r' through the relations $\sim_i$, for $i \in G$. This predicate may be computed recursively by noting that $REACH(r, r', m, G)$ holds for $m > 0$ just when there exists a trace r'' such that $REACH(r, r'', \lceil m/2 \rceil, G)$ and $REACH(r'', r', \lfloor m/2 \rfloor, G)$. This recursive step may be implemented by using $|r|$ steps of existential branching to guess r'', deterministically verifying that r'' is a trace, then universally branching to verify the two recursive calls. Note that r' is G-accessible from r if and only if $REACH(r, r', |S|^{|r|}, G)$ holds, where S is the set of states of E. Thus the call $REACH(r, r', m, G)$ may be implemented by means of an alternating computation requiring space $|E| + 2|r|$ to represent the environment and the traces, and space $\log(|S|^{|r|})$ for the counter m.

Using the routine $REACH$, we can now implement $P(E, r, C_G\varphi)$ by universally branching $|r|$ steps to guess the trace r', then universally branching to make the two calls $REACH(r, r', |S|^{|r|}, G)$ and $P(E, r', \varphi)$.

This shows that it is possible to construct an alternating Turing machine that decides whether a formula φ holds in a trace r of environment E, using work space $O(|E| + |r| \cdot \log(|E|) + |\varphi|)$ and $O(|\varphi|)$ 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 $A(n)S(n) + S(n)^2$. $\square$

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, repeatedly considering the same trace. In the current state of knowledge about the relationship between deterministic space and time, it is likely to be difficult to avoid this cost.

We will show later (Theorem 5.1) that the above bound is tight. The following section shows that this bound can be improved for formulae not containing common knowledge.

4 Bounded Alternation Formulae

In this section we treat a special case of the model checking problem, in which the formulae checked are in $\mathcal{L}_n$, i.e., do not contain common knowledge operators. We will show that if the formulae of interest are of bounded alternation depth, then it is possible to construct finite state implementations for both external and agent-centric model checking. Consequently, it is not necessary for agents to maintain their sequence of observations in this case. This result applies to both the synchronous case and the asynchronous case. A consequence of these results is that model checking formulae in $\mathcal{L}_n$ is in fact decidable in the asynchronous case.

The idea underlying these results is that for the purpose of providing semantics for formulae of alternation depth at most k, Kripke structures contain in general a considerable amount of redundancy, and the relevant information can be compressed into a class of much smaller structures. These structures, called k-trees, are introduced in Section 4.1. In order to exploit these structures in implementations of model checking, it is necessary to show that the compression operation commutes with an operation that updates the compressed representations. This is done for the synchronous case in Section 4.2, and for the asynchronous case in Section 4.3.

4.1 Trees

The structures we will use for the compact representation of the information expressed by a world in a Kripke structure will be a type of finite trees in which vertices are labelled by states of the environment and edges are labelled by agents. It is convenient to represent these trees as follows.

We assume throughout this section that we have been given a fixed finite set S of states equipped with a valuation function $V : S \times \mathit{Prop} \to \{0, 1\}$. In applications, S will be the set of states of an environment and V the associated valuation.

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 of tuples of the form $\langle s, \emptyset, \ldots, \emptyset \rangle$ where $s \in S$ and the number of copies of the empty set $\emptyset$ is equal to n, the number of agents. For each agent i we also define $\mathcal{T}_{0,i}$ to be equal to $\mathcal{T}_0$. 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 union of $\mathcal{T}_k$ with 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 tuples $\langle s, U_1, \ldots, U_n \rangle$ in $\mathcal{T}_k$ where U_i is the empty set.

Intuitively, in the tuple $\langle s, U_1, \ldots, U_n \rangle$, the state s represents the actual state of the world, and for each i the set U_i represents the knowledge of agent i. Identifying a 0-tree $\langle s, \emptyset, \ldots, \emptyset \rangle$ with the state s, 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.

> **Lemma 4.1:** The number of k-trees over S is less than $C_k = exp(2n|S|, k)/n$, where $exp(x, k)$ is the function defined by $exp(x, 0) = x$ and $exp(x, k+1) = x2^{exp(x,k)}$.

The elements of $\mathcal{T}_k$ correspond in an obvious way to trees of height k, with edges labelled by agents and vertices labelled by states in S. If $w = \langle s, U_1, \ldots, U_n \rangle$ we define $root(w)$ to be the state s. If 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 $root(w)$. 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 $O((C_{k-1})^2)$. Note that because an i-child of a $k+1$-tree is an i-objective k-tree, there does not exist in these labelled trees a pair of consecutive edges with the same label i. An additional constraint follows from the fact that the U_i are sets, hence do not contain repetitions: the subtrees at two distinct i-children of any vertex must be non-isomorphic. 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.

We now show how formulae may be interpreted in k-trees, by defining a relation $w \models \varphi$ between formulae φ in $\mathcal{L}_n$ and k-trees w. First, if $w = \langle s, U_1, \ldots, U_n \rangle$ is a k-tree and V is a forest of i-objective k'-trees, define $w *_i V$ to be the tuple $\langle s, U_1, \ldots, U_{i-1}, V, U_{i+1}, \ldots, U_n \rangle$. Note that this is a $max(k, 1+k')$-tree. Now define the satisfaction relation $w \models \varphi$ by the following recursion on the structure of φ:

1. $w \models p$ for propositional constants p if $V(root(w), p) = 1$ (where V is the valuation on the set S),
2. $w \models \neg\varphi$ if not $w \models \varphi$,
3. $w \models \varphi_1 \wedge \varphi_2$ if $w \models \varphi_1$ and $w \models \varphi_2$,
4. $w \models K_i\varphi$ if $v *_i U_i \models \varphi$ for all $v \in U_i$, where $w = \langle s, U_1, \ldots, U_n \rangle$.

We remark that that this definition may be shown to be equivalent to one obtained by constructing a tree-like Kripke structure from w, and interpreting formulae at the root of this structure. 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 $V(s, p) = 1$. 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

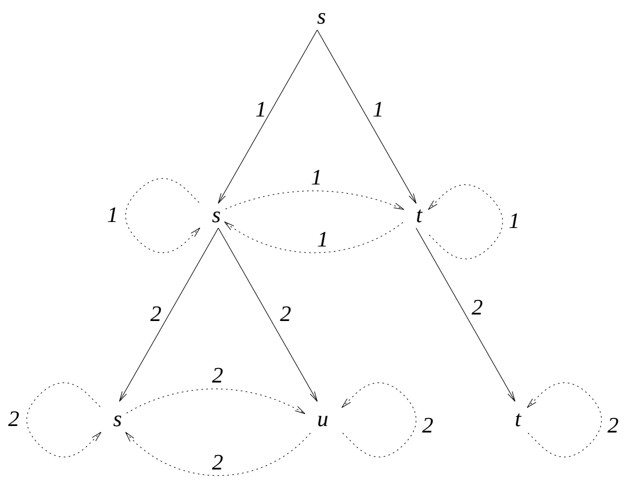

Figure 2: A 2-tree

Euclidean, but not equivalence relations, since if there is an i-labelled edge from u to v then $u\mathcal{K}_iv$ but not $v\mathcal{K}_iu$.[2]

Example 4.2: Figure 2 depicts a 2-tree for two agents. The vertices of this tree are labelled by the states s, t, u, v, and the edges are labelled by agents. Thus, the subtree at the left 1-child of the root represents the tuple $x = \langle s, \emptyset, \{\langle s, \emptyset, \emptyset\rangle, \langle u, \emptyset, \emptyset\rangle\}\rangle$, the subtree at the right 1-child of the root represents the tuple $y = \langle t, \emptyset, \{\langle t, \emptyset, \emptyset\rangle\}\rangle$, and the diagram as a whole represents the 2-tree $\langle s, \{x, y\}, \emptyset\rangle$. The tree edges are represented by solid lines, and the additional edges required to interpret the 2-tree as a Kripke structure are indicated by means of broken lines.

We now show that when dealing with formulae of alternation depth at most k, it suffices to work with k-trees rather than arbitrary Kripke structures. Suppose $M = \langle W, \mathcal{K}_1, \ldots, \mathcal{K}_n, V_M\rangle$ is an $\mathrm{S5}_n$ 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 $r \in W$ and propositions p, we have $V_M(r, p) = V(f(r), p)$, where V is the valuation on S. (In our applications of the following results, M will be the Kripke structure generated by an environment with states S and f will be the final state mapping on traces. Hence we use the symbol r to denote worlds of M.) We will show that there exists a k-tree that represents the set of all formulae of alternation depth k or less which hold at a given world in M. Intuitively, the appropriate k-tree is that obtained by unfolding the Kripke structure into an infinite tree, pruning this tree to height k, and identifying sibling vertices that generate isomorphic subtrees.

[2]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 synchronous time.

First, define the *i-objectification* function obj_i, that for each k maps k-trees to i-objective k-trees, by $obj_i(w) = w *_i \emptyset$. 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 $F_k(r) = \langle f(r), \emptyset, \ldots, \emptyset \rangle$. For $k \geq 0$, define $F_{k+1}(r) = \langle f(r), U_{k,1}(r), \ldots, U_{k,n}(r) \rangle$, where $U_{k,i}(r) = \{ obj_i(F_k(r')) \mid r\mathcal{K}_i r' \}$.

Example 4.3: Consider the $S5_2$ Kripke structure M with five worlds $r_1 \ldots r_5$, whose accessibility relations have the equivalence classes $\{r_1, r_2, r_3\}$ and $\{r_4, r_5\}$ for agent 1, and $\{r_1, r_3, r_4, r_5\}$ and $\{r_2\}$ for agent 2. Suppose that the state mapping f is given by $f(r_1) = f(r_3) = f(r_5) = s$, $f(r_2) = t$ and $f(r_4) = u$. Then the 2-tree $obj_2(F_2(r_1))$ is that depicted in Figure 2. Note that $obj_1(F_1(r_1)) = obj_1(F_1(r_3)) = \langle s, \emptyset, \{\langle s, \emptyset, \emptyset\rangle, \langle u, \emptyset, \emptyset\rangle\}\rangle$. Thus, although there are three worlds x such that $r_1 \sim_1 x$, the fact that $U_{1,1}(r_1)$ is a *set* of 1-trees means that the images of two of these worlds are identified.

The following result shows that $F_k(r)$ represents the formulae of alternation depth k or less holding at the world r.

Proposition 4.4: If φ is a formula of $\mathcal{L}_n$ of alternation depth less than or equal to k then for all worlds r of M we have $M, r \models \varphi$ if and only if $F_k(r) \models \varphi$.

Proof: The proof requires establishing an additional property of the mapping F_k. 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. Note that if $K_i\varphi$ is a formula of alternation depth k then φ is a k,i-compatible formula. In addition to the main claim we prove that if φ is k,i-compatible then $M, r \models \varphi$ if and only if $obj_i(F_k(r)) *_i U_{k,i}(r) \models \varphi$. (Note that $obj_i(F_k(r)) *_i U_{k,i}(r)$ is a $k+1$-tree.) The proof is by simultaneous induction on the structure of φ.

The base case follows directly from the fact that $root(F_k(r)) = fin(r)$. The cases involving the boolean operators are straightforward. We prove the main claim in the case that φ is a formula of the form $K_i\varphi'$ of alternation depth k. The argument for the additional property in this case is similar. By definition, $F_k(r) \models K_i\varphi'$ if and only if $u *_i U_{k-1,i}(r) \models \varphi'$ for all $u \in U_{k-1,i}(r)$. But the trees in $U_{k-1,i}(r)$ are precisely the $k-1$-trees $obj_i(F_{k-1}(r'))$ for $r' \sim_i r$. Moreover, if $r' \sim_i r$ then $U_{k-1,i}(r) = U_{k-1,i}(r')$. Using the definition of the relation $\models$ in k-trees, it follows that $F_k(r) \models K_i\varphi'$ if and only if $obj_i(F_{k-1}(r')) *_i U_{k-1,i}(r') \models \varphi'$ for all $r' \sim_i r$. Since φ' is k,i-compatible, we obtain using the inductive hypothesis and the semantics of K_i that $F_k(r) \models K_i\varphi'$ if and only if $M, r \models K_i\varphi'$. $\square$

We remark that it suffices that the accessibility relations $\mathcal{K}_i$ be transitive and Euclidean, so Proposition 4.4 also applies to $K45_n$, the logic of belief [Chellas, 1980; Halpern and Moses, 1992].

The k-trees, as defined above, are closely related to the k-worlds defined in [Fagin *et al.*, 1991], and Proposition 4.4 resembles a result implicit in Theorem 3.1 of that paper. However, there are a number of significant technical differences. Firstly, k-trees are more succinct than the corresponding k-worlds, which have many components that redundantly model the same information, and consequently require a number of constraints to ensure that these components are consistent.

Secondly, the definition of the satisfaction relation $\models$ for k-worlds applies only for formulae with k levels of nesting of knowledge operators (where, e.g., the formula $K_1 K_1 p$ has two levels of nesting), whereas Proposition 4.4 works for the larger class of formulae of alternation depth k. This difference is not an essential one — it can be shown that the k-world corresponding to a world of a K45 (or S5) structure does contain the appropriate information for dealing with formulae of alternation depth k, so the definition of satisfaction in k-worlds could be modified along the lines of our definition for k-trees to yield a result similar to Proposition 4.4. We note however, that our definition would not work for logics weaker than K45.

Finally, whereas k-worlds are constructed over a basis of truth assignments, k-trees are built over a basis of a set of states S equipped with a valuation. The reason for this difference is that we are interested in the dynamic behaviour of knowledge in the Kripke structures generated by an environment, so we need to maintain more information that just the truth assignment associated with a world. This will become clearer in the next section, where we show how to exploit this extra information.

4.2 Synchronous Systems

We now consider model checking of formulae in $\mathcal{L}_n$ for the synchronous case. We will show in this section how k-trees can be used to provide implementations of both external and agent-centric model checking. It follows from the result of the previous section that there exists a k-tree that provides a compact representation of the formulae of alternation depth k holding at a trace. The main contribution of this section is to show that this k-tree can be computed in a space-efficient manner.

As a first step, let us specialize Proposition 4.4 to the Kripke structure M_E corresponding to an environment $E = \langle S, I, T, O_1, \ldots, O_n, V \rangle$ with observations $\mathcal{O}$. 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 final state mapping *fin* from the set of worlds $traces(E)$ 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 4.4.

Corollary 4.5: For every trace 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 $O((C_k)^2)$, which is independent of the length of r, this result shows that model checking at a trace r may be carried out by a computation in a structure which, for sufficiently long traces, is smaller than the component of M_E

generated by the trace. 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 $F_k(r)$: if rs is a trace consisting of a prefix r and final state s, then the k-tree $F_k(rs)$ can be computed from the k-tree $F_k(r)$ and the state s.

To motivate the definitions to follow, consider first the depth one case. Let $F_1(r)$ be the 1-tree $\langle t, U_1, \ldots, U_n \rangle$, where t is the final state of r. For each agent i the set U_i may be viewed as the set of states agent i considers possible in the trace r, by identifying each 0-tree $\langle x, \emptyset, \ldots, \emptyset \rangle$ with the state x. When the trace is extended by adding the new state s, we obtain a new 1-tree $F_1(rs)$, which must be of the form $\langle s, U'_1, \ldots, U'_n \rangle$ for some sets U'_i, representing the states agent i considers possible in the trace rs. Thus, the main question is how to determine these new sets U'_i from the old sets U_i and the fact that the actual transition being made in the environment is one from state t to state s.

Let $o = O_i(s)$ be the new observation agent i makes when the state changes from t to s. This observation may be the same as the agent's observation in the state t, but since we are assuming that the system is synchronous, the agent is aware that a transition has taken place, and knows that the new state gives rise to observation o. Thus, if a state s' is not reachable in a single step from a state in U_i, or it is not the case that $O_i(s') = o$, then the agent should not consider s' to be a possible state in the trace rs. However, if the agent considered a state $t' \in U_i$ to be possible in the trace r, and a state s' with $O_i(s') = o$ is reachable in one step of the environment from t', then the above considerations do not suffice for the agent to exclude s' from its set of possibilities in the trace rs. This suggests that

$$U'_i = \{s' \in S \mid \exists t' \in U_i[t'Ts' \text{ and } O_i(s') = o]\}. \quad (1)$$

The calculation of the right hand side of this equation is depicted in Figure 3, in which the points represent states of the environment and the letters represent the observations made by agent i at these states. Note that if the agent has perfect recall, and complete knowledge of the possible transitions of the environment, it can calculate this set from its old knowledge set U_i and its new observation o.

This discussion has suggested how an agent may update its knowledge about the state, and indicated how we can compute $F_1(rs)$ given $F_1(r)$ and s. For higher levels of knowledge, i.e., including agents' knowledge about other agents' knowledge, represented in $k+1$-trees for $k > 1$, we apply this intuition recursively. Suppose we have already determined a function G_k mapping a k-tree w and a state s to a k-tree $G_k(w, s)$ that represents the result of updating w when the state of the environment changes to s. We wish to determine how to compute from a $k+1$-tree $\langle t, U_1, \ldots, U_n \rangle$ a new $k+1$-tree $\langle s, U'_1, \ldots, U'_n \rangle$ representing the agents' knowledge when the state of the environment changes from t to s.

Note that the root of a k-tree represents the state of the environment. Arguing as above, this suggests that agent i will consider possible in the trace rs the set of states

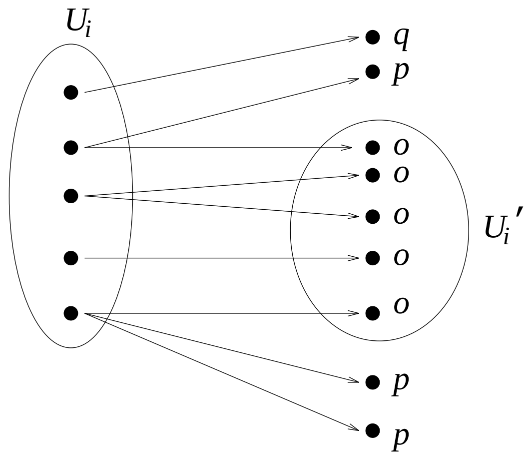

Figure 3: Updating an agent's knowledge set

s' such that $O_i(s') = O_i(s)$ and for some k-tree w in U_i there exists a transition of the environment from $root(w)$ to s'. However, a state s' is not quite what we seek for the set U_i' — we need instead a k-tree encoding information about the other agents' states of knowledge. Since the system is synchronous, the agent is aware that other agents are also updating their knowledge. If it considers w a possible situation in the trace r, and it knows that this update process may be represented by the function G_k, then in the trace rs it should consider the k-tree $G_k(w, s')$ to be a possible representation of the actual situation.

This motivates the following definitions. Define for each number $k \geq 0$ the function $G_k : \mathcal{T}_k \times S \to \mathcal{T}_k$, by induction on k. The definition of G_k will be by mutual recursion with the functions $H_{k,i} : \mathcal{F}_{k,i} \times \mathcal{O} \to \mathcal{F}_{k,i}$, where i is an agent and $k \geq 0$. 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.

We define $G_0(w, s) = \langle s, \emptyset, \ldots, \emptyset \rangle$. 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 $G_k(obj_i(w), s) = obj_i(G_k(w, s))$. The following result shows how these functions may be used to compute F_k incrementally.

Lemma 4.6: For every trace 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 trace rs, we have that $F_0(rs) = \langle s, \emptyset, \ldots, \emptyset \rangle = G_0(F_0(r), s)$. 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 trace $r' \in traces(E)$ 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 trace 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 trace $r's'$ 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 trace of length one then $F_k(r)$ may be straightforwardly computed: if r consists of the single initial state s then $F_0(r) = \langle s, \emptyset, \ldots, \emptyset \rangle$ 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 $O_i(s') = O_i(s)$. Together with the above result, this yields the following characterization of the complexity of implementations of model checking in M_E.

Theorem 4.7: In every finite state environment E there exists

1. a finite state implementation in E over traces of model checking the set of formulae Φ_k of alternation depth at most k, and
2. a finite state implementation in E over observations of agent i of model checking the set of formulae in Φ_k of the form $K_i\varphi$.

In both cases, for formulae φ of alternation depth k or less, $(E, r) \models \varphi$ is decided in time $O((C_{k-1})^2 \cdot (|r| + |\varphi|))$.

Proof: We describe the automata $A = \langle Q, q_0, \mu, \pi \rangle$ in each case. The fact that they provide implementations is direct from Corollary 4.5, and Lemma 4.6. The upper bound follows from the observation that the satisfaction relation in k-trees corresponds to model checking in a Kripke structure generated by the tree, Proposition 2.1, and the fact that k-trees have size $O((C_{k-1})^2)$.

In the external case, take $Q = \mathcal{T}_k \cup \{q_0\}$, where q_0 is some new object that will serve as the initial state of the automaton. The update function $\mu : Q \times S \to Q$ is given by $\mu(w, s) = F_k(s)$ when $w = q_0$ and $\mu(w, s) = G_k(w, s)$ otherwise. The function π is given by $\pi(w, \varphi) = \mathbf{true}$ if and only if $w \models \varphi$.

In the agent-centric case, take $Q = \mathcal{F}_{k-1,i} \cup \{q_0\}$, where q_0 is some new object. The update function $\mu : Q \times \mathcal{O} \to Q$ is given by $\mu(U, o) = \{obj_i(F_{k-1}(s)) \mid s \in$

I and $O_i(s) = o\}$ when $w = q_0$ and $\mu(U, o) = H_{k-1,i}(U, o)$ otherwise. Finally, the function π is given by $\pi(U, K_i\varphi) = \mathbf{true}$ if and only if $w *_i U \models \varphi$ for all $w \in U$. □

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 trace of Proposition 3.1 for a linear time dependence on the length of trace, 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 traces, the algorithm of Proposition 3.1 may well be more efficient.

A special case of Theorem 4.7 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 database updates of [Winslett, 1988], and to the generalization of this approach in [Katsuno and Mendelzon, 1991]. Similar ideas occur in 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 generalizing certain aspects 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 4.7 clearly subsumes the situation in which we wish to check a *fixed* formula of $\mathcal{L}_n$ in traces of increasing length, giving the following characterization, which we will later use as a basis for comparison with common knowledge formulae.

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

In general, the finite state automata obtained by the methods of this section will not be minimal, so further optimization may be possible.

4.3 Asynchronous Environments

In the synchronous case, we were able to establish the decidability of model checking all formulae in $\mathcal{L}_n^C$ by noting that the substructure of M_E generated by any trace is finite. As shown by Example 2.5, this is no longer the case when environments are interpreted asynchronously: in this example the substructure generated by any trace is the entire structure. We will show in Section 5 that this results in formulae involving common knowledge having an undecidable model checking problem in M_E^a. Nevertheless, formulae not containing common knowledge operators remain decidable. We will obtain this result in the present section as a consequence of an incremental computation for model checking formulae in $\mathcal{L}_n$ in the structure M_E^a, which generalizes the techniques of the previous section.

The basic data structure we employ will again be k-trees. For the remainder of this section, we assume that these trees have been constructed over the states S of the environment E, and that the mappings F_k and $U_{k,i}$ of Section 4.1 are defined with respect to the structure $M = M_E^a$. We take the function f required by the definition to be the final state mapping on traces. Thus, we obtain the following consequence of Proposition 4.4.

Corollary 4.9: For every trace 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 $F_k(r) \models \varphi$.

This result shows that instead of checking the infinite structure M_E^a, we may check the finite structures $F_k(r)$. To complete the generalization of the results of the previous section, we need to show how to incrementally compute the functions F_k. In the synchronous case, the traces indistinguishable to agent i from a trace 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 $F_k(rs)$ from $F_k(r)$.

To understand this in the depth one case, suppose that $F_1(r) = \langle t, U_1, \ldots, U_n \rangle$ and $F_1(rs) = \langle s, U_1', \ldots, U_n' \rangle$. Identifying a 0-tree with the state at its root as usual, the sets U_i and U_i' are sets of states of the environment. We seek to express U_i' in terms of U_i and the observation $o = O_i(s)$ made by agent i when the state changes to s.

Let us first note a property of the sets U_i and U_i' — these sets are closed under transitions that preserve agent i's observation. That is, if state x is in U_i and there is a transition of the environment from x to a state y with $O_i(x) = O_i(y)$ then y also is in U_i. This is direct from the definitions of F_1 and asynchronous local states. In particular, we see that if the observation o is the same as the observation of agent i in the final state of r then the agent's local state does not change, so in this case we must have $U_i' = U_i$.

Next, suppose that the observation o is different from the observation o' made by the agent in the final state of r. Then the agent is aware that its observation has changed from o' to o. If previously it considered the state t' possible, and there is a transition from t' to a state s' with $O_i(s') = o$ then, as we argued in the synchronous case, the agent should consider the state s' possible, so we should have $s' \in U_i'$. However, as we noted above, the set U_i' should be closed under transitions preserving the agent's observation o. This suggests that U_i' is the smallest set V such that

1. $t' \in V$ for all t' such that $O_i(t') = o$ and there exists an $s' \in U_i$ with $s'Tt'$, and

2. V is closed under transitions preserving agent i's observation.

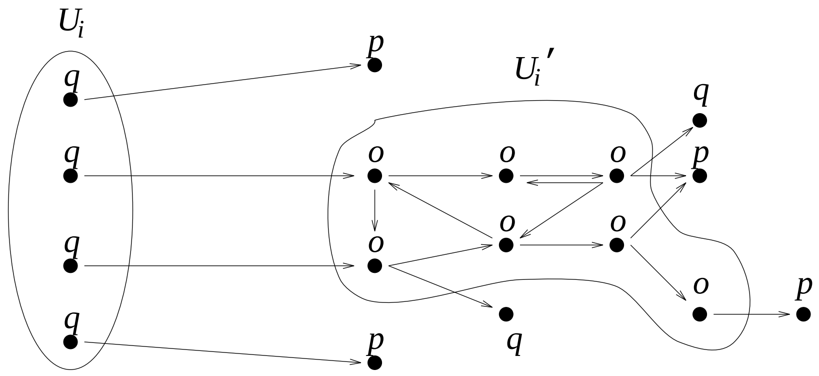

Figure 4: Updating an agent's asynchronous knowledge set

This set is depicted in Figure 4.

Note that because of cycles in the transition relation, there may exist arbitrarily large l such that $rs \approx_i r's_1s_2 \ldots s_l$ and $O_i(s_j) = o$ for $j = 1 \ldots l$. In the depth one case, we are interested only in the *states* that an agent considers possible, so it suffices to restrict attention to traces in which the s_i do not repeat. However, once we consider what an agent knows about other agents' knowledge, it may be the case that the other agents have different states of knowledge in the traces $r's_1$ and $r's_1s_2 \ldots s_l$ even when $s_1 = s_l$. Thus, in order to reach a trace at which a certain state of knowledge of the other agents obtains, it may be necessary to traverse a cycle many times. The main observation underlying the decidability result of this section is that if we are interested in knowledge only up to alternation level k, then the number of possible states of knowledge of the other agents is finite. Thus, we need only traverse cycles, collecting k-trees representing these states of knowledge, until we reach a "fixpoint" where all further transitions fail to produce new possibilities.

To formalize this intuition, we define for each $k \geq 0$ a function $G^a_k : \mathcal{T}_k \times S \longrightarrow \mathcal{T}_k$ 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 $G^a_0(w, s) = \langle s, \emptyset, \ldots, \emptyset \rangle$, 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

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

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

Lemma 4.10:

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

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

It follows from Lemma 4.10 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 $J^m_{k,i,o}(U)$. Observe that in the case that $k = 0$ and the agent's observation has changed this fixpoint value corresponds to the set U'_i depicted in Figure 4.

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 now set out to establish the following recursion relation, which gives the incremental way to compute F_k.

Proposition 4.11: For every trace rs we have $G^a_k(F_k(r),s) = F_k(rs)$.

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 4.11 follows immediately from the definitions of G^a_0 and F_0. We assume that $G^a_k(F_k(r),s) = F_k(rs)$ for all traces rs, and show that $G^a_{k+1}(F_{k+1}(r),s) = F_{k+1}(rs)$, by establishing that $F_{k+1}(rs) \leq G^a_{k+1}(F_{k+1}(r),s)$ and $G^a_{k+1}(F_{k+1}(r),s) \leq F_{k+1}(rs)$. For the first of these inequalities, we make use of the following result.

Lemma 4.12:

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

(2) Suppose s is a state and let $r's'$ be a trace with $O_i(s') = O_i(s)$. For all $w \in \mathcal{T}_k$, if $obj_i(F_k(r'))$ is an i-child of $G^a_{k+1}(w,s)$ then $obj_i(F_k(r's'))$ 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 $obj_i(F_k(r'))$ is in $U^a_{k,i}(r)$. Let $o = O_i(s)$. Now $O_i(s') = o$ and there is a transition from $root(F_k(r')) = fin(r')$ to s', hence by the second part of the definition of $J_{k,i,o}$ we have that $G^a_k(obj_i(F_k(r')),s')$ 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, $G^a_k(obj_i(F_k(r')), s') = obj_i(G^a_k(F_k(r'), s')) = obj_i(F_k(r's'))$. Thus $obj_i(F_k(r's'))$ is in $H^a_{k,i}(U^a_{k,i}(r), o)$, hence an i-child of $G^a_{k+1}(F_{k+1}(r), s)$.

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 $obj_i(F_k(r's')) = G^a_k(obj_i(F_k(r')), s')$. Note that since $r's'$ is a trace, there is a transition from $root(F_k(r')) = fin(r')$ to s'. In addition, $O_i(s') = o$. It follows using the second part of the definition of $J_{k,i,o}$ that if $obj_i(F_k(r'))$ is in U_i then $obj_i(F_k(r's'))$ 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 $obj_i(F_k(r's'))$ is in U_i. This establishes the result. □

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

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 $G^a_{k+1}(F_{k+1}(r), s)$.

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 4.12(1) that $obj_i(F_k(r_1 s_1))$ is an i-child of $G^a_{k+1}(F_{k+1}(r), s)$. Now, by repeated use of Lemma 4.12(2) it follows that $obj_i(F_k(r_1 s_1 \ldots s_m)) = v$ is an i-child of $G^a_{k+1}(F_{k+1}(r), s)$.

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

Lemma 4.13: For all traces 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 in $J_{k,i,o}(U^a_{k,i}(r))$. 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 trace $r' \approx_i r$ such that $v = obj_i(F_k(r'))$. 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 $v = obj_i(F_k(r'))$ 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 $v' = obj_i(F_k(r'))$ for some trace $r' \approx_i r$. This trace satisfies $fin(r') = root(v')$, so we have that $r's'$ is a trace and $r's' \approx_i rs$. Thus, using the induction hypothesis, $v = G^a_k(v', s') = G^a_k(obj_i(F_k(r')), s') = obj_i(F_k(r's'))$. 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 trace $r' \approx_i rs$ such that $v' = obj_i(F_k(r'))$. Since $fin(r') = root(v')Ts'$, we obtain that $r's'$ is a trace. Because $O_i(s') = O_i(s)$ we also have that $r's' \approx_i rs$. Now, using the induction hypothesis, we have that $v = G^a_k(v', s') = G^a_k(obj_i(F_k(r')), s') = obj_i(F_k(r's'))$. Hence v is in $U^a_{k,i}(rs)$ by the definition of $U^a_{k,i}$. □

The inequality $G^a_{k+1}(F_{k+1}(r), s) \leq F_{k+1}(rs)$ follows directly from Lemma 4.13. To see this, note that Lemma 4.13(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 4.13(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 $G^a_{k+1}(F_{k+1}(r), s)) \leq F_{k+1}(rs)$. This completes the proof of Proposition 4.11.

We have now seen how to compute $F_k(rs)$ from $F_k(r)$. It remains to show how to calculate $F_k(r)$ when r is a trace 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 $g^a_0(s) = \langle s, \emptyset, \ldots, \emptyset \rangle$ 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 4.14: For all $s \in I$ we have $g^a_k(s) = F_k(s)$.

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

First, we show that $g^a_k(s) \leq F_k(s)$. By the induction hypothesis and the definitions of f_k and F_k, we have $f_k(s) \leq F_k(s)$. The monotonicity of $J_{k,i,o}$ implies that G^a_k is monotonic in its first argument, so we obtain that $g^a_k(s) = G^a_k(f_k(s), s) \leq G^a_k(F_k(s), s)$. By Lemma 4.13(2), $G^a_k(F_k(s), s) \leq F_k(s)$, hence $g^a_k(s) \leq F_k(s)$.

Next, we show that $F_k(s) \leq g^a_k(s)$. Suppose v is an i-child of $F_k(s)$. Then there exists a trace r with $s \approx_i r$ and $v = obj_i(F_{k-1}(r))$. 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, $F_{k-1}(s') = g^a_{k-1}(s')$, 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 trace with $O_i(s') = O_i(s)$ and $obj_i(F_{k-1}(r'))$ is an i-child of $G^a_k(f_k(s), s)$ then $obj_i(F_{k-1}(r's'))$ is an i-child of $G^a_k(f_k(s), s)$. This is immediate from Lemma 4.12(2). □

Combining Corollary 4.9, Lemma 4.14, Proposition 4.11 and Proposition 2.1, we obtain the following results, analogous to Theorem 4.7. and Corollary 4.8 respectively.

Theorem 4.15: In every finite state environment E there exists

1. a finite state implementation in E^a of model checking the set of formulae Φ_k of alternation depth at most k over traces, and
2. a finite state implementation in E^a of model checking the set of formulae in Φ_k of the form $K_i\varphi$ over observations of agent i.

In both cases, for formulae φ of alternation depth k or less, $(E^a, r) \models \varphi$ is decided in time $O(C_k \cdot (|r| + |\varphi|))$.

Proof: The automata $A = \langle Q, q_0, \mu, \pi \rangle$ are similar to those for the synchronous case. For external model checking, take $Q = \mathcal{T}_k \cup \{q_0\}$, where q_0 is some new object that will serve as the initial state of the automaton. The update function $\mu : Q \times S \to Q$ is given by $\mu(w, s) = g_k^a(s)$ when $w = q_0$ and $\mu(w, s) = G_k^a(w, s)$ otherwise. The function π is given by $\pi(w, \varphi) = \mathbf{true}$ if and only if $w \models \varphi$.

In the agent-centric case, take $Q = \mathcal{F}_{k-1,i} \cup \{q_0\}$, where q_0 is some new object that will serve as the initial state of the automaton. The update function $\mu : Q \times \mathcal{O} \to Q$ is given by $\mu(U, o) = \{obj_i(F_{k-1}(s)) \mid s \in I \text{ and } O_i(s) = o\}$ when $w = q_0$ and $\mu(U, o) = H^a_{k-1,i}(U, o)$ otherwise. The function π is given by $\pi(U, K_i\varphi) = \mathbf{true}$ if and only if $w *_i U \models \varphi$ for all $w \in U$.

For the complexity bound, we argue as follows. Write $time(f)$ for the time required to compute the value of the function f. Note that $time(H^a_{k,i}) = O(|S| \cdot C_k \cdot time(G^a_k))$, since computing $H^a_{k,i}(U)$ involves computing $G^a_k(w, s)$ for up to C_k k-trees w and $|S|$ states s. Hence $time(G^a_k) = O(n \cdot time(H^a_{k-1,i})) = O(n \cdot |S| \cdot C_{k-1} \cdot time(G^a_{k-1}))$. It follows that $time(G^a_k) = O(C_k)$. $\Box$

Corollary 4.16: Let φ be a fixed formula of $\mathcal{L}_n$ and let E be a fixed environment. Then the set $\{r \in traces(E) \mid (E, r) \models \varphi\}$ is regular.

In particular, we see from Theorem 4.15 that model checking formulae in $\mathcal{L}_n$ in the structure M^a_E is in fact decidable.

5 Common Knowledge

The results of Section 4.2 and Section 4.3 show that for both the synchronous and asynchronous semantics, there exist finite state implementations of model checking, provided formulae do not involve the common knowledge operator. In particular, the set of traces satisfying a given formula of this kind is in both cases regular (Corollary 4.8 and Corollary 4.16), 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 5.1: There exists an environment E for two agents and a propositional constant p such that the set $\{r \in \mathit{traces}(E) \mid (E, r) \models C_{\{1,2\}}p\}$ is PSPACE hard (under LOGSPACE reductions).

Note that this lower bound is tight, by Proposition 3.1. It follows from Theorem 5.1 that the set of traces corresponding to a common knowledge formula is generally not regular (else PSPACE equals LOGSPACE, which is known to be false). Thus Corollary 4.8 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 5.1 by showing how to encode space bounded deterministic Turing machine computations. Consider the even numbered traces in the sequence of traces of Example 2.4. The symbol b in these traces moves progressively from left to right, much like the head of a Turing machine. Our hardness proof will exploit this resemblance: traces 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 c of the form

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

For $i = 1 \ldots n$ 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 c' is the configuration reached in one step from the configuration c, and these configurations have the same length, then we write $c \Rightarrow c'$. We write $c \Rightarrow^* c'$ 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 $E = \langle S, I, T, O_1, \ldots, O_n, V\rangle$ used to encode the Turing machine. This environment will have a set S of states consisting of 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 traces 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 of 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 trace of E. There exist other traces of E, but we will be primarily interested in the traces which correspond to configurations, and in another set of traces 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

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

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

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

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

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

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 traces 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 trace 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 5.2: If r is a configuration in which the Turing machine state is not final, then there exist precisely two traces 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 trace 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 zR_1z', 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 $r' = r$. □

Lemma 5.3: 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 5.2 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 5.2 and Lemma 5.3, 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 trace 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 trace 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.[3] However, the forward steps are deterministic, so backward steps cannot take us to a 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 trace 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 traces 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 5.1.

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, 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 $\tau : L \rightarrow D$ 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 principal 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

[3]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 $f(n)^2$.

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 sR_is' 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 traces of E and the traces of $D(E)$. This correspondence is established by the *doubling function* d, which maps a trace $s_1 \ldots s_n$ of E to the trace $s_1 \cdot 1,\ s_2 \cdot 0,\ s_3 \cdot 1,\ s_4 \cdot 0, \ldots,\ s_n \cdot (n \ mod\ 2)$ of $D(E)$. Clearly, this function preserves the valuation. Furthermore, for all traces 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 5.4: 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 5.1 we cannot expect to generalize the results of Section 4.3 to common knowledge formulae. In fact, the asynchronous interpretation is much more complex than this argument suggests, as the following result shows.

Theorem 5.5: There exists an environment F for two agents and a propositional constant p such that it is undecidable, given a trace 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 trace $r(x)$ 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 trace indistinguishable from arbitrarily long traces, 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 5.1. 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 trace

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

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 trace contains a bar-state, then all states in that trace are bar-states. Thus, the traces of F may be divided into two classes: the class of traces of the form $d(r)$, where r is a trace of $D(E)$, and the class of traces 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 5.1. 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 traces. First, noting that no equivalence class of R_1 contains both a bar state and a state of $D(E)$, we see that no trace of $D(E)$ is $\approx_1$-equivalent to any trace consisting of bar states. In particular, we obtain the following:

- If r_1 is a trace of E then $d(r_1) \approx_1 r$ iff $r = d(r_2)$ for some trace 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 5.2 the trace 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 traces 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 traces are of the same length. Also, each equivalence class of R_2 contains at most one bar-state, so no two distinct traces comprised of bar-states are $\approx_2$-equivalent. Thus, to describe the relation $\approx_2$ it suffices to consider traces of the form $d(r_1)$ with r_1 a trace 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 trace 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 5.3, 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 trace 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 traces $\{1, 2\}$-accessible from $r(\sigma, 1)$ satisfy p. The situation here is like that in the proof of Theorem 5.1, 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 into one from which M does eventually halt (given sufficient space). 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 $(F^a, r(\sigma, 1)) \models C_{\{1,2\}}p$. $\Box$

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.

6 Knowledge-Based Programs

In a number of papers [Dwork and Moses, 1990; Hadzilacos, 1987; Halpern and Moses, 1990; Halpern and Zuck, 1992; Mazer, 1988] it has been shown that knowledge formulae may provide necessary and sufficient conditions for the achievement of a variety of coordination tasks. This enables the formulation of *knowledge-based programs* [Halpern and Fagin, 1989; Fagin *et al.*, 1995a; Fagin *et al.*, 1995b] for these tasks, in which knowledge formulae are used as preconditions for actions. Knowledge-based programs 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 program runs, e.g. a specification of the assumptions concerning message delivery and loss, a knowledge-based program 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; Moses and Tuttle, 1988] 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. It is shown how a test for the truth of this formula may be implemented, given a variety of assumptions on the nature of failures. For crash and sending omission failures it turns out that the test may be implemented in time polynomial in the number of processors (this number also bounds the length of the relevant traces). However, for general omission 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; Fagin *et al.*, 1995a; Fagin *et al.*, 1995b]. In order to permit the execution of knowledge-based programs, 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 an initial foray into the feasibility of automating this kind of compilation when knowledge is interpreted with respect to a perfect recall semantics. Of particular interest in this regard are the results concerning evaluation of fixed sets of formula in traces of increasing length, which is closely related to the type of operation required in knowledge-based programs. It is encouraging that model checking of fixed knowledge formulae may be implemented using finite state automata. In subsequent work [Meyden, 1996a], we show that the results of Section 4 may be applied to automate the *verification* of implementations of knowledge-based programs with respect to the perfect recall semantics. (Results of [Vardi, 1996] concern the verification of implementations of knowledge-based programs under a more restricted semantics for knowledge.)

However, our results fall short of enabling an automated procedure for *compilation* of even the simplest type of knowledge-based program. In knowledge-based programs, 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 traces of a knowledge-based program is generally smaller than the set of traces generated by the underlying transition system obtained from the agents' possible actions. Since the knowledge-based program 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. We consider a generalization of the framework of the present paper that explicitly models knowledge-based programs in [Meyden, 1996b; Meyden, 1996a; Meyden, 1997].

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. Some of the examples in the knowledge literature also use *full-information protocols* [Moses and Tuttle, 1988; Dwork and Moses, 1990], in which agents send messages containing their entire history. In general, these works deal only with traces of a finitely bounded length, so this could be modelled using finite state environments. However, it would be interesting to study full information protocols with arbitrarily long traces, which would require a more expressive framework. Nevertheless, given the simplicity of our framework, the results of this paper would seem to provide lower bounds applicable to more general frameworks.

In this respect the high complexity we have found for model checking 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, model checking of common knowledge formulae is likely to be a crucial requirement for some applications 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 programs 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 5.1 and Theorem 5.5 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 the hardness results will persist in more reasonable classes of environments. We have shown recently that in environments in which all communication is by synchronous broadcast, model checking of even common knowledge formulae is tractable [Meyden, 1996a; Meyden, 1997], and knowledge-based programs in such environments have finite state implementations. Another class of environments 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 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.

7 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, 1994]. 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 4.2 and Section 4.3 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 Umberto Bonollo, Hirofumi Katsuno, Yoram Moses and the anonymous referees for helpful 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.*, 1995a] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. Knowledge-based programs. In *Proc. ACM Symposium on Principles of Distributed Computing*, 1995.

[Fagin *et al.*, 1995b] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about knowledge.* MIT Press, Cambridge, MA, 1995.

[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, 1988] 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, 1988. Morgan Kaufmann.

[Meyden, 1994] R. van der Meyden. Axioms for knowledge and time in distributed systems with perfect recall. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 448–457. IEEE Press, 1994.

[Meyden, 1996a] R. van der Meyden. Finite state implementations of knowledge-based programs. In *Proceedings of the Conference on Foundations of Software Technology and Theoretical Computer Science*, Springer LNCS No. 1180, pages 262–273, Hyderabad, India, December 1996.

[Meyden, 1996b] R. van der Meyden. Knowledge-based programs: On the complexity of perfect recall in finite environments. In *Proc. Conf. on Theoretical Aspects of Rationality and Knowledge*, pages 31–50, 1996.

[Meyden, 1997] R. van der Meyden. Constructing finite state implementations of knowledge-based programs with perfect recall. In *Intelligent Agent Systems: Theoretical and Practical Issues (Based on a Workshop Held at PRICAI'96, Cairns, Aust., Aug 1996)*, Springer LNAI No. 1209, pages 135–151, 1997.

[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.

[Vardi, 1996] M. Vardi. Implementing knowledge-based programs. In *Proc. of the Conf. on Theoretical Aspects of Rationality and Knowledge*, pages 15–30. Morgan Kaufmann, 1996.

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

A An Interpreted System Semantics

The semantics of knowledge in environments defined in this paper has been chosen to provide a parsimonious treatment of the particular problems we consider. It is possible to give a slightly more general definition of the semantics in terms of the *interpreted systems* of [Fagin *et al.*, 1995b]. We sketch this alternative semantics here and show that it is essentially equivalent to ours. For brevity, we assume familiarity with the approach of [Fagin *et al.*, 1995b].

Suppose that $E = \langle S, I, T, O_1, \ldots, O_n \rangle$ is an environment with observations $\mathcal{O}$. The definitions to follow make the most sense under the assumption that the transition relation T is serial, i.e., for each state $s \in S$ there exists a state $t \in S$ with sTt. In most applications this assumption can be guaranteed by adding one or more new states representing that the system has terminated.

The framework of [Fagin *et al.*, 1995b] is based on *runs*, which are functions ρ mapping the natural numbers to tuples in $L_e \times L_1 \times \ldots \times L_n$, where L_e is a set of states of the environment, and for each $i = 1 \ldots m$, L_i is a set of *local states* of agent i. We take $L_e = S$, and each $L_i = \mathcal{O}^*$, the set of finite sequences of observations.

Define an *infinite trace* τ of an environment E to be an infinite sequence $(s_0, s_1, \ldots)$ of states of E such that $s_0 \in I$ and $s_m T s_{m+1}$ for all $m \geq 0$. Each infinite trace gives rise to two runs, depending on whether we take the synchronous or asynchronous interpretation. In either case, we have $\rho_e(m) = s_m$. In the synchronous case we define the local states of agent i by $\rho_i(m) = \{s_0 \ldots s_m\}_i$, the (synchronous) local state of agent i in the trace consisting of the finite prefix of τ of length $m+1$. Define $\mathcal{R}$ to be the set of runs so obtained from all infinite traces of E. Similarly, in the asynchronous case we may define the local states of agent i by $\rho_i(m) = \{s_0 \ldots s_m\}_i^a$, using the asynchronous local state assignment from Section 2, and take $\mathcal{R}^a$ to be the set of runs obtained using this definition from all infinite traces of E.

A *point* is a pair (ρ, m) consisting of a run ρ and a number $m \geq 0$. When the environment states L_e in runs are states of the environment E, an interpretation function π mapping points to valuations on the set of basic propositions may be defined by $\pi(\rho, m)(p) = V(\rho_e(m), p)$.

With these definitions, we obtain the two interpreted systems $\mathcal{I} = (\mathcal{R}, \pi)$ and $\mathcal{I}^a = (\mathcal{R}^a, \pi)$. We may interpret formulae of $\mathcal{L}_n^C$ at points in these systems in the usual way, following the definitions of [Fagin *et al.*, 1995b]. The following result shows that that these interpretations correspond closely to the semantics of Section 2.

Proposition A.1: Let E be an environment with serial transition relation T. Given an infinite trace τ and number $m \geq 0$, let r be the finite trace of E consisting of the first $m+1$ states of τ. Then for all formulae $\varphi \in \mathcal{L}_n^C$ we have the following.

1. If ρ is the run obtained from τ according to the synchronous interpretation, then $(E, r) \models \varphi$ if and only if $\mathcal{I}, (\rho, m) \models \varphi$.

2. If ρ' is the run obtained from τ according to the asynchronous interpretation, then $(E^a, r) \models \varphi$ if and only if $\mathcal{I}^a, (\rho', m) \models \varphi$.

Proof: Straightforward induction on the complexity of φ. $\Box$

In effect, this result states that the language $\mathcal{L}_n^C$ is too weak to distinguish runs having the same history r up to the point (ρ, m). This justifies basing the study of model checking on the finite objects r rather than the infinite objects (ρ, m).