Finite State Implementations of Knowledge-Based Programs

(Extended Abstract)

Ron van der Meyden*

Computing Science
University of Technology, Sydney
PO Box 123, Broadway NSW 2007
Australia
email: ron@socs.uts.edu.au

Abstract. Knowledge-based programs have been proposed as an abstract formalism for the design of distributed systems, based on the idea that an agent's actions are a function of its state of knowledge. We identify two natural cases in which finite state implementations of atemporal knowledge-based programs in finite environments may be automatically constructed. The first concerns an interpretation of knowledge in which agents are aware only of their current observation and the current time. The second concerns the perfect recall interpretation of knowledge, in environments in which all communication is by synchronous broadcast. These results contribute towards a theory of automated synthesis of protocols from knowledge-based specifications.

1 Introduction

Viewing agents as having states of *knowledge* has been found to be a useful abstraction for the design and analysis of distributed systems [HM90, DM90, Had87, HZ92]. Central to this approach is a focus on the relationship between an agent's actions and its state of knowledge. This has lead to the proposal of knowledge-based programming formalisms [FHMV95a], in which agents' actions have as preconditions formulae expressing properties of their knowledge. Knowledge-based programs allow for highly intuitive descriptions of protocols, abstracting both from the agents' data structures and the environment in which they run. Thus, these programs provide a promising approach to the *synthesis* of protocols from high level descriptions.

* To appear, Proc. Conf. on Foundations of Software Technology and Theoretical Computer Science, LNCS No. 1180, Dec 1996, pp. 262-273. Work begun while the author was with the Information Sciences Laboratory, NTT Basic Research Laboratories, Kanagawa, Japan, and continued at the Department of Applied Math and Computer Science, Weizmann Institute of Science. Thanks to Yoram Moses and Moshe Vardi for helpful discussions.

There is an impediment to this goal, however: knowledge-based programs cannot be directly executed — rather, it is necessary to *implement* a knowledge-based program by translating it to a standard program in which the preconditions for actions are tests on the agents' concrete states. In a knowledge-based program an agent's state of knowledge depends on a number of factors, including the behaviour of the environment in which the program runs, and the actions being performed by the agents in this environment. These actions in turn depend on the agents' states of knowledge. This means that the relationship between a knowledge-based program and its implementations is subtle, involving a non-trivial fixpoint definition.

Consequently, a knowledge-based program may, in general, have many behaviourally distinct implementations, or none. In this paper we focus on the class of *atemporal* programs and *synchronous* interpretations of knowledge, for which this problem does not arise, and for which there always exists an implementation, unique up to behavioural equivalence. The main issue then concerns the representation and construction of this implementation. We consider the simplest possible type of representation — *finite state protocols* — and ask "when does a knowledge-based program have a finite state implementation, and how can it be computed?"

In previous work [Mey96], we have shown that with respect to a perfect recall interpretation of knowledge, finite state implementations of atemporal programs do not always exist, even when the environment is finite state. The contribution of the present paper is a general condition sufficient for the existence of finite state implementations. Using this condition, we identify two cases in which such implementations may be automatically constructed. One of these cases involves interpretations of knowledge in which agents are forgetful, being aware only of their current observation and the present time — this assumption on its own suffices for the existence of finite state implementations. The second concerns the perfect recall interpretation of knowledge, but requires a restricted class of finite environments: *broadcast environments*, in which all communication between agents is by synchronous broadcast.

The structure of the paper is as follows. Sections 2 to 4 define knowledge-based programs and their semantic interpretation. Section 5 states the sufficient condition for the existence of finite state implementations. Section 6 applies this condition to an interpretation of knowledge in which agents are aware only of their current observation and the time. The application to perfect recall in broadcast environments is presented in Section 7. Section 8 discusses related work and concludes.

2 Knowledge-Based Programs

To make the paper self-contained, we recall in the next three sections the definitions from [Mey96] of knowledge-based programs, the environments in which they run, and their implementations. These definitions are a variant of the definitions in [FHMV95a], to which the reader is referred for motivation. For a description of how our framework differs from that of [FHMV95a], and motivation for the changes, see [Mey96].

To describe the agents' states of knowledge we work with the propositional multi-modal language for knowledge $\mathcal{L}_n^C$ generated from some set of basic propositions $Prop$ by means of the usual boolean operators, the monadic modal operators K_i, where $i = 1 \ldots n$ is an agent, and the monadic operators C_G, where G is a set of two or more agents. Intuitively, $K_i\varphi$ expresses that agent i knows φ, and $C_G\varphi$ expresses that φ is common knowledge amongst the group of agents G.

This language is interpreted in the standard way [FHMV95b] in a class of $\mathrm{S5}_n$ Kripke structures of the form $M = \langle W, \sim_1, \ldots, \sim_n, V\rangle$, where W is a set of worlds, for each $i = 1 \ldots n$ the accessibility relation $\sim_i$ is an equivalence relation on W, and $V : W \times Prop \to \{0, 1\}$ is a valuation on W. If G is a group of agents then we may define the equivalence relation $\sim_G$ on W by $u \sim_G v$ if there exists a sequence $u_0, \ldots, u_k$ of worlds such that $u_0 = u$, $u_k = v$ and for each $j = 0 \ldots k-1$ there exists an agent $i \in G$ with $u_j \sim_i u_{j+1}$. The crucial clauses of the definition of satisfaction of a formula at a world u are given by

1. $(M, u) \models p$, where p is an atomic proposition, if $V(u, p) = 1$.
2. $(M, u) \models K_i\varphi$ if $M, v \models \varphi$ for all worlds v with $u \sim_i v$.
3. $(M, u) \models C_G\varphi$ if $M, v \models \varphi$ for all worlds v with $u \sim_G v$.

We now describe the structure of knowledge-based programs. For each agent $i = 1 \ldots n$ let ACT_i be the set of actions that may be performed by agent i. Similarly, let ACT_e be the set of actions that may be performed by the environment in which the agents operate. If $a_e \in ACT_e$ and $a_i \in ACT_i$ for each $i = 1 \ldots n$, we say that the tuple $\mathbf{a} = \langle a_e, a_1, \ldots, a_n\rangle$ is a *joint action,* and we write ACT for the set of all joint actions.

Call a formula of $\mathcal{L}_n^C$ *i-subjective* if it is a boolean combination of formulae of the form $K_i\varphi$. An *atemporal knowledge-based program* for agent i is a finite statement $\mathbf{Pg}_i$ of the form

$$\begin{array}{l} \textbf{case of} \\ \quad \textbf{if } \varphi_1 \textbf{ do } a_1 \\ \qquad \vdots \\ \quad \textbf{if } \varphi_m \textbf{ do } a_m \\ \textbf{end case} \end{array}$$

where the φ_j are i-subjective formulae of $\mathcal{L}_n^C$ and the $a_j \in ACT_i$ are actions of agent i. Intuitively, a program of this form is executed by repeatedly evaluating the case statement, ad infinitum. At each step of the computation, the agent determines which of the formulae φ_j accurately describe its current state of knowledge. It non-deterministically executes one of the actions a_j for which the formula φ_j is true, updates its state of knowledge according to any new input it receives, and then repeats the process. We will give a more precise semantics below.

To describe the behaviour of the world in which agents operate, we define a *finite interpreted environment* to be a tuple of the form $E = \langle S_e, I_e, P_e, \tau, O, V_e \rangle$ where the components are as follows:

1. S_e is a finite set of *states of the environment.* Intuitively, states of the environment may encode such information as messages in transit, failure of components, etc. and possibly the values of certain local variables maintained by the agents.
2. I_e is a subset of S_e, representing the possible *initial states* of the environment.
3. $P_e : S_e \rightarrow \mathcal{P}(ACT_e)$ is a function, called the *protocol of the environment,* mapping states to subsets of the set ACT_e of actions performable by the environment. Intuitively, $P_e(s)$ represents the set of actions that may be performed by the environment when the system is in state s.
4. τ is a function mapping joint actions $\mathbf{a} \in ACT$ to state transition functions $\tau(\mathbf{a}) : S_e \rightarrow S_e$. Intuitively, when the joint action $\mathbf{a}$ is performed in the state s, the resulting state of the environment is $\tau(\mathbf{a})(s)$.
5. The component O is a function mapping the set of states S_e to $\mathcal{O}^n$, where $\mathcal{O}$ is some set of *observations.* If s is a global state then the i-th component $O_i(s)$ of $O(s)$ will be called the *observation* of agent i in the state s.
6. $V_e : S_e \times Prop \rightarrow \{0,1\}$ is a valuation, assigning a truth value $V(s,p)$ in each state s to each atomic proposition $p \in Prop$.

A *trace* of an environment E is a *finite* sequence $s_0 \ldots s_m$ of states such that for all $i = 0 \ldots m-1$ there exists a joint action $\mathbf{a} = \langle a_e, a_1, \ldots, a_n \rangle$ such that $s_{i+1} = \tau(\mathbf{a})(s_i)$ and $a_e \in P_e(s_i)$. We write $init(r)$ and $fin(r)$ for the initial and final states of a trace r, respectively.

3 Protocols

Next, we introduce the standard protocols which will be used as implementations of knowledge-based programs, and describe the set of traces produced by executing such a protocol. Define a *protocol* for agent i to be a tuple $P_i = \langle S_i, q_i, \alpha_i, \mu_i \rangle$ consisting of (1) a set S_i of *protocol states,* (2) an element $q_i \in S_i$, the protocol's *initial state,* (3) a function $\alpha_i : S_i \times \mathcal{O} \rightarrow \mathcal{P}(ACT_i)$, such that $\alpha_i(s,o)$ is a nonempty set representing the possible next actions that agent i may take when it is in state s and is making observation o, and (4) a function $\mu_i : S_i \times \mathcal{O} \rightarrow S_i$, such that $\mu_i(s,o)$ represents the next protocol state the agent assumes after it has been in state s making observation o. In general, we allow the set of protocol states to be infinite. A *joint protocol* is a tuple $\mathbf{P} = \langle P_1, \ldots, P_n \rangle$ such that each P_i is a protocol of agent i.

Intuitively, protocol states represent the memory that agents maintain about their sequence of observations for the purpose of implementing their knowledge-based programs. For each trace r of an environment E, a protocol P_i determines a protocol state $P_i(r)$ for agent i by means of the following recursive definition. If r is a trace of length one, consisting of an initial state of E, then $P_i(r) = q_i$, the initial state of the protocol. If r is the trace $r's$, where r' is a trace and s

is a state of the environment, then $P_i(r) = \mu_i(P_i(r'), O_i(\mathit{fin}(r')))$, where $\mathit{fin}(r')$ is the final state of r'. Thus $P_i(r)$ represents the agent's memory of its *past* observations in the trace r, excluding the current observation.

In executing a protocol, an agent's next action in a given trace is selected from a set determined by its memory of past observations together with its current observation. Define the *set of actions of agent i enabled by the protocol P_i at a trace r* with final state s to be the set $act_i(P_i, r) = \alpha_i(P_i(r), O_i(s))$. Similarly, the set of joint actions $\mathbf{act}(\mathbf{P}, r)$ enabled at r by a joint protocol $\mathbf{P}$ contains precisely the joint actions $\langle a_e, a_1, \ldots, a_n\rangle$ such that $a_e \in P_e(s_e)$ and $a_i \in act_i(P_i, r)$ for each $i = 1 \ldots n$.

We may now describe the set of traces that result when a particular protocol is executed in a given environment. Given a joint protocol $\mathbf{P}$ and an environment E, we define the *set of traces generated by* $\mathbf{P}$ *and* E, to be the smallest set of traces $\mathcal{R}$ such that (1) for each initial state $s \in I_e$ of the environment $\mathcal{R}$ contains the trace of length one composed just of the state s, and (2) if r is a trace in $\mathcal{R}$ with final state s, and if $\mathbf{a} \in \mathbf{act}(\mathbf{P}, r)$ is a joint action enabled by $\mathbf{P}$ at r then the trace $r \cdot \tau(\mathbf{a})(s)$ is in $\mathcal{R}$. Intuitively, this is the set of traces generated when the agents incrementally maintain their memory using the update functions μ_i, and select their next action at each step using the functions α_i.

4 Implementation of Knowledge-Based Programs

Clearly, in order to execute a knowledge-based program according to the informal description above, we need some means to interpret the knowledge formulae it contains. To do so, we introduce a particular class of Kripke structures.

These structures will be obtained from a set of traces by means of *views*, a generalisation of observations. Intuitively, a view captures the information available to the agents in each trace as the result of a sequence of observations. Define a *joint view* of an environment E to be a function $\{\cdot\}$ mapping traces of E to X^n for some set X. We write $\{r\}_i$ for the i-th component of the result of applying $\{\cdot\}$ to the trace r.

We will focus in this paper on two views, the *synchronous perfect recall view* $\{\cdot\}^{pr}$ and the *clock view* $\{\cdot\}^{clock}$. The synchronous perfect recall view provides the agent with a complete record of all the observations it has made in a trace. If r is the trace $s_1 \ldots s_k$ then this view is defined by $\{r\}_i^{pr} = O_i(s_1) \ldots O_i(s_k)$. The clock view provides the agent with information only about its current observation and the current time. This view is defined by $\{r\}_i^{clock} = (|r|, O_i(\mathit{fin}(r)))$, where $|r|$ is the length of the trace r.

We will assume that an agent's view of a trace always includes the agent's current observation. That is, for all traces r and r' and all agents i we have that if $\{r\}_i = \{r'\}_i$ then $O_i(\mathit{fin}(r)) = O_i(\mathit{fin}(r'))$. Secondly, we assume that the observations are the agent's only source of information about a trace. Thus all views $\{\cdot\}$ are required to be a function of the perfect recall view. That is, for each agent i there must exist a function f such that for all traces r we have $\{r\}_i = f(\{r\}_i^{pr})$.

A joint view $\{\cdot\} : S \rightarrow X^n$ of an environment E is *incremental* if for each agent i there exists a function $\nu : X \times \mathcal{O} \rightarrow X$ such that for all $rs \in \mathit{traces}(E)$ we have $\{rs\}_i = \nu(\{r\}_i, O_i(s))$. A view is *synchronous* when for all agents i, if $\{r\}_i = \{r'\}_i$ then $|r| = |r'|$. Intuitively, this means that the current time can be determined from the view. Both the perfect recall view and the clock view are synchronous and incremental. Given the assumptions above, we may consider the perfect recall view to be the maximal synchronous view, and the clock view to be the minimal synchronous view.

Given a set $\mathcal{R}$ of traces of environment E with valuation V_e, and a joint view $\{\cdot\}$ of E, we may define the Kripke structure $M(\mathcal{R}, \{\cdot\}, E) = (W, \sim_1, \ldots, \sim_n, V)$, where

- the set of worlds $W = \mathcal{R}$, and
- for all $r, r' \in \mathcal{R}$ we have $r \sim_i r'$ iff $\{r\}_i = \{r'\}_i$, and
- the valuation $V : \mathcal{R} \times \mathit{Prop} \rightarrow \{0, 1\}$ is defined by $V(r, p) = V_e(\mathit{fin}(r), p)$.

That is, the accessibility relations are derived from the view, and truth of basic propositions at traces is determined from their final states according to the valuation V_e provided by the environment. Intuitively, $r \sim_i r'$ if, based on the information provided by the view $\{\cdot\}_i$, the agent cannot distinguish between the traces r and r'. We will call $M(\mathcal{R}, \{\cdot\}, E)$ the *interpreted system* obtained from $\mathcal{R}$ and $\{\cdot\}$ in E .

We may now define the *set of actions enabled by the program* $\mathbf{Pg}_i$ *at a trace* r *of system* M to be the set $\mathbf{Pg}_i(M, r)$ consisting of all the a_i such that $\mathbf{Pg}_i$ contains a line "**if** φ **do** a_i" where $(M, r) \models \varphi$. Similarly, the set of joint actions $\mathbf{Pg}(M, r)$ enabled by a joint knowledge-based program in a trace r with final state s contains precisely the joint actions $\langle a_e, a_1, \ldots, a_n \rangle$ such that $a_e \in P_e(s)$ and $a_i \in \mathbf{Pg}_i(M, r)$ for each $i = 1 \ldots n$.

This definition leaves open the question of what system we are to use when executing a knowledge-based program. The answer to this is that we should use a system that is itself generated by running the program. Say that *a joint protocol* $\mathbf{P}$ *implements a knowledge-based program* $\mathbf{Pg}$ *in an environment* E *with respect to the joint view* $\{\cdot\}$ if for all r in the set of traces $\mathcal{R}(\mathbf{P}, E)$ generated by $\mathbf{P}$ and E, we have $\mathbf{Pg}(M, r) = \mathbf{act}(\mathbf{P}, r)$, where $M = M(\mathcal{R}(\mathbf{P}, E), \{\cdot\}, E)$ is the system obtained from $\mathcal{R}(\mathbf{P}, E)$ and $\{\cdot\}$ in E. That is, the joint actions prescribed by the knowledge-based program, when the tests for knowledge are interpreted according to the system M, are precisely those prescribed by the standard protocol $\mathbf{P}$.

Note that this definition (unlike that of [FHMV95a]) refers only to the behaviour of the protocol, not to its internal states. More precisely, say that two joint protocols $\mathbf{P}$ and $\mathbf{P}'$ are *behaviourally equivalent* with respect to an environment E if they generate precisely the same set of traces by means of the same sets of actions at each step. That is, $\mathbf{P}$ and $\mathbf{P}'$ are behaviourally equivalent when $\mathcal{R}(\mathbf{P}, E) = \mathcal{R}(\mathbf{P}', E)$, and for all $r \in \mathcal{R}(\mathbf{P}, E)$ we have $\mathbf{act}(\mathbf{P}, r) = \mathbf{act}(\mathbf{P}', r)$. When this relation holds, $\mathbf{P}$ implements the knowledge-based program $\mathbf{Pg}$ with respect to the view $\{\cdot\}$ if and only if $\mathbf{P}'$ also implements $\mathbf{Pg}$ with respect to the view $\{\cdot\}$.

In the framework of [FHMV95a] knowledge-based programs, even atemporal ones, can have no, exactly one, or many implementations, which may be behaviourally different. Our notion of implementation is more liberal, so this is also the case for our definitions. However, the notion of implementation is better behaved when we consider implementations with respect to synchronous views.

Proposition 1. *For every environment E and atemporal joint knowledge-based program* $\mathbf{Pg}$*, implementations of* $\mathbf{Pg}$ *in E with respect to a synchronous view $\{\cdot\}$ always exist. Moreover, all such implementations are behaviourally equivalent.*

Although our formulation is slightly different, this result can be viewed as a special case of Theorem 7.2.4 of [FHMV95b]. If $\mathcal{R}$ is the unique set of all traces generated by any implementation of $\mathbf{Pg}$ in E with respect to a synchronous view $\{\cdot\}$, we refer to $M(\mathcal{R}, \{\cdot\}, E)$ as the *system generated by* $\mathbf{Pg}$ *in E with respect to* $\{\cdot\}$, denoted by $M(\mathbf{Pg}, \{\cdot\}, E)$.

The implementation guaranteed to exist by Proposition 1 uses the values of the synchronous perfect recall view $\{\cdot\}_i^{pr}$ as the protocol states for agent i. In case the view $\{\cdot\}$ is incremental, we may use the values of $\{\cdot\}_i$ instead, and update these using the incremental update function ν promised by the incrementality of the view. In general, both these implementations are very inefficient in their usage of space. For example, the values of the synchronous perfect recall view grow as a linear function of the length of the trace; in the case of the clock view the values grow as the log of the length of the trace. However, our notion of implementation does not require that an agent maintain its view of a trace, only that it behave as if it did. This brings us to the central question we study in this paper: under what conditions do there exist *finite state* implementations? We will present some natural conditions under which this is the case.

5 A Sufficient Condition for Finite State Implementations

In this section we present a sufficient condition for the existence of a finite state implementation of a knowledge-based program. We will give two applications of this result in the following sections, where we identify a variety of circumstances where the condition applies.

Define a *simulation* from an $S5_n$ structure $M = \langle W, \mathcal{K}_1, \ldots, \mathcal{K}_n, V\rangle$ to an $S5_n$ structure $M' = \langle W', \mathcal{K}_1', \ldots, \mathcal{K}_n', V'\rangle$ to be a function $\kappa : W \to W'$ such that

1. for all worlds $w \in W$ and propositions p, $V(w,p) = V'(\kappa(w), p)$,
2. for all $w, w' \in W$ and agents i, if $w\mathcal{K}_i w'$ then $\kappa(w)\mathcal{K}_i'\kappa(w')$,
3. for all $w \in W$, $w' \in W'$ and agents i, if $\kappa(w)\mathcal{K}_i' w'$ then there exists $v \in W$ with $w\mathcal{K}_i v$ and $\kappa(v) = w'$.

An easy induction on the construction of a formula $\varphi \in \mathcal{L}_n^C$ shows that if κ is a simulation from M to M' then $M, w \models \varphi$ iff $M', \kappa(w) \models \varphi$.

Theorem 2. *Let* $\mathbf{Pg}$ *be an atemporal knowledge-based program and suppose that the joint view* $\{\cdot\}$ *of environment* E *is synchronous and incremental. If there exists a simulation from* $M(\mathbf{Pg}, \{\cdot\}, E)$ *to a finite* $S5_n$ *structure* M' *then there exists a finite state implementation of* $\mathbf{Pg}$ *in* E *with respect to* $\{\cdot\}$.

Perhaps somewhat surprisingly, the preconditions of this result refer only to the knowledge relations, and not to the relationship between agents' states of knowledge from one instance to the next. Potentially, this relationship could be highly irregular and non-computable. The proof involves showing that this cannot be the case. The finite state implementation uses the equivalence classes of the relations $\mathcal{K}'_i$ of M' as the states of the protocol for agent i.

6 The Clock View

As a first application of Theorem 2, we show that finite state implementations are guaranteed to exist with respect to the clock view. Intuitively, this view provides the agents with two pieces of information: the current time and the current observation. Since all agents have access to the current time, the information derived from this is common knowledge. As we will show, the additional information an agent derives from its current observation is "orthogonal" to this common knowledge.

To make this precise, define the finite $\mathrm{S5}_n$ structure $M' = \langle W', \mathcal{K}_1, \ldots, \mathcal{K}_n, V' \rangle$ as follows. The worlds in W' are the pairs (X, s) where X is a subset of the set of states S of the environment E and the state s of E is an element of X. The accessibility relations $\mathcal{K}_i$ are defined by $(X, s)\mathcal{K}_i(X', s')$ when $X = X'$ and $O_i(s) = O_i(s')$. The valuation V' is defined by $V((X, s), p) = V_e(s, p)$, where V_e is the valuation of the environment E.

Proposition 3. *Suppose* $\mathcal{R}$ *is any subset of the set of traces of environment* E*. Define the mapping* $\kappa : \mathcal{R} \to W'$ *by* $\kappa(r) = (X(r), \mathit{fin}(r))$*, where* $X(r) = \{\mathit{fin}(r') \mid r' \in \mathcal{R} \text{ and } |r'| = |r|\}$*. Then* κ *is a simulation from* $M(\mathcal{R}, \{\cdot\}^{clock}, E)$ *to* M'.

The following is an immediate corollary of Proposition 3 and Theorem 2.

Theorem 4. *Let* E *be a finite environment and* $\mathbf{Pg}$ *an atemporal knowledge-based program. Then* $\mathbf{Pg}$ *has a finite state implementation in* E *with respect to the clock view.*

7 Perfect Recall in Broadcast Environments

In the previous section we showed that knowledge-based programs always have finite state implementations with respect to the clock view. As is shown in [Mey96], this is not the case for knowledge-based programs interpreted with respect to the perfect recall view. Indeed, the complexity of computing an agent's next action from its sequence of observations may be as high as PSPACE complete. By adapting the proof of this result, we may establish the following.

Theorem 5. *Given an atemporal knowledge-based program* $\mathbf{Pg}$ *and a finite environment* E*, the following are undecidable: (i) determining whether* $\mathbf{Pg}$ *has a finite state implementation in* E *with respect to* $\{\cdot\}^{pr}$*, and (ii) determining if there exists a simulation from* $M(\mathbf{Pg}, \{\cdot\}^{pr}, E)$ *to a finite* $S5_n$ *structure.*

While Theorem 5 states that determining the existence of finite state implementations is in general undecidable, it does not prevent the existence of classes of environments in which knowledge-based programs *always* have finite state implementations with respect to the perfect recall view. In this section we present a natural class of such environments. This class, called *broadcast environments*, models situations in which agents may maintain private information, but in which the only means this information can be communicated is by synchronous simultaneous broadcast to all agents. This sort of situation arises in systems in which components communicate by means of a shared bus, and protocols that operate in a sequence of synchronised *rounds* with broadcasts between rounds may also be modelled in this way. Broadcast environments are able to represent a variety of games of incomplete information, including Battleships and Bridge.

Formally, a broadcast environment is an environment $E = \langle S_e, I_e, P_e, \tau, O, V_e \rangle$ of a specific structure we now describe. The states of the environment will indicate a private state for each of the agents, a state for the remainder of the system, and a record of the public part of the most recent joint action performed. For each agent $i = 1 \ldots n$, we assume that there exists a set S_i of *instantaneous private states*, intuitively representing the state of the private objects maintained by the agent. The private states S_i will be observable and modifiable by agent i only. We also assume a set of *shared* states S_0, representing the remainder of the system, which is under the shared control of the agents.

The actions performed by agent i in a broadcast environment will be of the form $a_i || b_i$ where a_i is in a set A_i of *external* actions and b_i is in a set B_i of internal actions of agent i. The internal component b_i will affect only the agent's private state, and will be unobservable to the other agents. On the other hand, the external component a_i will be observable to all agents, but it will affect only the shared state. The actions performed by the environment itself (which we treat as agent 0) will similarly be comprised of two components, an internal component unobservable to agents $1 \ldots n$, and an observable external component. However, in this case, both components affect only the shared state.

The set of states S_e of a broadcast environment is composed from the above: these states are tuples of the form $\langle a_0, \ldots, a_n; p_0, \ldots, p_n \rangle$, where for each $i = 0 \ldots n$, the component $p_i \in S_i$ is a private state of agent i and the component $a_i \in A_i$ is an external action of agent i. Intuitively, a tuple $\langle a_0, \ldots, a_n; p_0, \ldots, p_n \rangle$ models a situation in which the shared component of the system is in state p_0, in which agents i's private objects are in state p_i, and in which the most recent set of external actions performed is given by the a_i. To distinguish them from the private and shared states, we refer to the elements of S_e as *global* states. In initial states, there is no most recent external action, so here we require that $a_i = \epsilon$ for all $i = 0 \ldots n$, where ϵ is a special null action. We allow the set I_e of initial states to be any nonempty set of states of this form.

The action interpretation function τ is required to be some function such that the effect of executing a joint action $\langle a_0 || b_0, a_1 || b_1, \ldots, a_n || b_n \rangle$ in a global state $\langle a'_0, \ldots, a'_n; p'_0, \ldots, p'_n \rangle$ is to produce a global state $\langle a_0, \ldots, a_n; p_0, \ldots, p_n \rangle$ in which the action part records the external portion of the joint action, in which the shared state p_0 is computed as a function of p'_0 and the component $\langle a_0, b_0, a_1, a_2, \ldots, a_n \rangle$ affecting the shared state, and in which each private state p_i is computed as a function of p'_i and the private action b_i.

In a broadcast environment, agents are able to observe their own private state, as well as some aspects of the shared state. However, we assume that all agents make the *same* observation of the shared state. Formally, we suppose that there exists a "common" observation function O_c mapping each shared state in S_0 to some observation. Using this, the agent's observation functions are defined by $O_i(\langle a_0, \ldots, a_n; p_0, \ldots, p_n \rangle) = (a_0, \ldots, a_n, O_c(p_0), p_i)$. That is, in each global state, agent i observes the most recent set of external actions performed, the common observation of the shared state, and its own private state. We will comment below on the requirement that the external actions be observable.

Finally, we require that the protocol P_e of the environment depend only upon the shared state and the most recent external action. We allow the finite set of propositions *Prop* to describe any property of the states S_e, captured semantically by the valuation $V_e : S_e \times \mathit{Prop} \to \{0, 1\}$. We now define a *broadcast environment* to be any environment of the form described above.

Suppose we are given a finite broadcast environment E. We will use Theorem 2 to demonstrate that all knowledge-based programs have finite state implementations in E with respect to the perfect recall view. To define the finite Kripke structure required by Theorem 2, we first need a few definitions.

If $s = \langle a_0, \ldots, a_n; p_0, \ldots, p_n \rangle$ is a global state as above then we define the *joint external action at* s, denoted $\mathbf{ext}(s)$, to be the tuple $\langle a_0, \ldots, a_n \rangle$. If $r = s_0 s_1 \ldots s_n$ is a trace of a broadcast environment, we will write $\mathbf{ext}(r)$ for the sequence $\mathbf{ext}(s_0) \ldots \mathbf{ext}(s_n)$. If $\mathbf{e}$ is a sequence of joint external actions and $\mathcal{R}$ is a set of traces then we define the function $f^{\mathcal{R}}_{\mathbf{e}} : I_e \to \mathcal{P}(S_e)$, as follows. If $s \in I_e$ is an initial state then $f^{\mathcal{R}}_{\mathbf{e}}(s)$ is the set of states $t \in S_e$ such that there exists a trace $r \in \mathcal{R}$ with $\mathit{init}(r) = s$ and $\mathbf{ext}(r) = \mathbf{e}$ and $\mathit{fin}(r) = t$. Intuitively, $f^{\mathcal{R}}_{\mathbf{e}}(s)$ represents the set of states that would be considered possible by an agent who knows the trace to be in $\mathcal{R}$, knows that the initial state is s and observes only the sequence of external actions $\mathbf{e}$.

Of course, the agents in a broadcast environment also have access to their own private states, so they have extra information that enables them to eliminate certain states from the set $f^{\mathcal{R}}_{\mathbf{e}}(s)$. However, we will show that this information is orthogonal to the information derived from the sequence of external actions. Define the finite Kripke structure $M' = \langle W', \mathcal{K}_1, \ldots \mathcal{K}_n, V' \rangle$ to have

1. set of worlds W' equal to the set of triples (s, f, t) where $s \in I_e$, f is a function from I_e to $\mathcal{P}(S_e)$ and $t \in f(s)$;
2. $(s, f, t)\mathcal{K}_i(s', f', t')$ if and only if $O_i(s) = O_i(s')$ and $f = f'$ and $O_i(t) = O_i(t')$;
3. $V'((s, f, t), p) = V_e(t, p)$, where V_e is the valuation on the global states of E.

Given a set of traces $\mathcal{R}$ of the broadcast environment E, define the mapping $\kappa : \mathcal{R} \to W'$ by $\kappa(r) = (init(r), f^{\mathcal{R}}_{\mathbf{ext}(r)}, fin(r))$. Note that whenever r is a trace in $\mathcal{R}$ we have $fin(r) \in f^{\mathcal{R}}_{\mathbf{ext}(r)}(init(r))$, so this mapping is well-defined.

Lemma 6. *If $\mathcal{R}$ is the set of traces of the system M generated in a broadcast environment by a knowledge-based program with respect to the perfect recall view, then κ is a simulation from M to M'.*

Using Theorem 2 this immediately gives the following result.

Theorem 7. *If E is a finite broadcast environment then all knowledge-based programs have finite state implementations in E with respect to the perfect recall view.*

It is possible to slightly generalise this result, by modifying the definition of broadcast environment to allow agents to make distinct observations of the shared state, provided they gain no information from this observation that they would not have been able to derive from their observation of the external actions. This helps to extend the result to cover games like stratego and puzzles like the muddy children puzzle [FHMV95b].

However, we remark that Theorem 7 breaks down under a generalisation in which agents still make the same observation of the shared state, but in which the external actions are unobservable. We will show in the full version of the paper that if we take agent i's observation in state $\langle a_0, \ldots a_n; p_0, \ldots p_n \rangle$ to be $(O_c(p_0), p_i)$, then a finite state implementation may no longer exist.

8 Conclusion

The computational problems involved in the implementation of knowledge-based programs have thus far received only limited attention. Results in [FHMV95a] can be interpreted in our framework as dealing with the complexity of determining the existence and uniqueness of implementations of knowledge-based programs containing temporal operators with respect to the "observational" view defined by $\{r\}^{obs}_i = O_i(fin(r))$, in which an agent is aware only of its current observation. Implementations are not guaranteed to exist with respect to this view, but when they do they can be assumed to be finite state. Vardi [Var96] deals with implementations with respect to $\{\cdot\}^{obs}$ of atemporal programs, and also considers the problem of determining whether a given protocol implements a knowledge-based program.

Because of space limitations, we have suppressed in this extended abstract a number of issues that will be covered in the full version of the paper. We will show there that the results may be generalised to deal with programs in which the tests contain not just operators for knowledge, but also *past-time* temporal operators. We also deal more carefully with the constructive nature of the proofs that finite state implementations exist, showing how to compute these implementations. Finally, we treat there the problem of deciding whether a given finite state protocol implements a knowledge-based program with respect to the views $\{\cdot\}^{clock}$ and $\{\cdot\}^{pr}$.

These results begin to map out the decidable problems concerning the implementation of knowledge-based programs, but the feasibility of this approach for automated protocol synthesis remains to be seen. One issue that will clearly need to be addressed is the state space explosion problem, known from the literature on temporal logic model checking. However, we note that the constructions of the present paper appear to be amenable to the symbolic techniques that have recently been used with some success in this area [BCM+90].

References

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

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

[FHMV95a] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. Knowledge-based programs. In *Proc. ACM Symposium on Principles of Distributed Computing*, 1995.

[FHMV95b] R. Fagin, J. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, MA, 1995.

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

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

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

[Mey96] R. van der Meyden. Knowledge-based programs: on the complexity of perfect recall in finite environments. In *Proc. of the Conf. on Theoretical Aspects of Rationality and Knowledge*, pages 31–50. Morgan Kaufmann, 1996.

[Var96] M. Vardi. Implementing knowledge-based programs. In *Proc. of the Conf. on Theoretical Aspects of Rationality and Knowledge*, pages 15–30. Morgan Kaufmann, 1996.

This article was processed using the LaTeX macro package with LLNCS style