Constructing Finite State Implementations of Knowledge-Based Programs with Perfect Recall* (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 multi-agent protocols, based on the idea that an agent's actions are a function of its state of knowledge. The key questions in this approach concern the relationship between knowledge-based programs and their concrete implementations, in which the actions are a function of the agents' local states. In previous work we have shown that with respect to a perfect recall semantics for knowledge, finite state implementations of knowledge-based programs do not always exist. Indeed, determining the existence of such an implementation is undecidable. However, we also identified a sufficient condition under which the existence of a finite state implementation is guaranteed, although this sufficient condition is also undecidable. We show in this paper that there nevertheless exists an approach to the optimization of implementations that results in a finite state implementation just when the sufficient condition holds. These results contribute towards a theory of automated synthesis of multi-agent 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, multi-agent and situated systems [HM90, RK86]. The central insight of this approach is that an agent's actions should depend on its epistemic state. 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.

* To appear, Proc. of PRICAI Workshop on Theoretical and Practical Foundations of Intelligent Agents, Cairns Aug 1996, Springer LNAI. 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 for helpful discussions and to Lawrence Cavedon for his comments on the manuscript.

Knowledge-based programs allow for extremely intuitive descriptions of agent behaviour, abstracting both from the agents' data structures and the environment in which they run.

In knowledge-based programs, agents' concrete states can be arbitrary data structures, and the relationship between knowledge formulae and agents' concrete states is defined semantically. An agent *knows* a fact P when in a local state s if P is true in all global states of the system in which the agent is in local state s. Since this definition of knowledge is non-operational, a knowledge-based program cannot be directly executed – it is first necessary to *implement* it by translating it to a standard program in which the preconditions for actions are tests on the agents' concrete states.[2]

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 a *synchronous perfect recall* interpretation 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 is then the representation and construction of this implementation. *Finite state protocols* are the simplest possible type of representation. In previous work [Mey96b], we have shown that with respect to a perfect recall interpretation of knowledge, finite state implementations of atemporal programs may not exist, even when the environment is finite state. However, we have also identified a general condition sufficient for the existence of finite state implementations [Mey96a], and shown that it applies for some natural classes of systems. It turns out that, like the problem of deciding the existence of a finite state implementation of a given knowledge-based program, the question of whether the sufficient condition applies is undecidable. This still allows a finite state implementation to be automatically discovered when one exists, since the problem of determining whether a given finite state protocol is an implementation of a given knowledge-based program is decidable. However, this involves an exhaustive search through all finite state implementations.

In the present paper, we show that a much more systematic procedure, involving *no* search, is able to construct finite state implementations when the sufficient condition applies. In particular, we show that there exists an imple-

---

[2] In this respect knowledge-based programs differ from proposals for *agent oriented programming* [Sho93] and BDI architectures [RG92] in which agents' local states are logical theories, updated by means of procedures provided by the designer. The epistemic level description of agents in these proposals remains operational, and there is no guarantee that knowledge states are updated in a way that conforms to their semantic interpretation.

mentation with the property that it runs in constant space just when the sufficient condition holds. This result contributes towards a theory of automated synthesis of multi-agent protocols from knowledge-based specifications.

The structure of the paper is as follows. Sections 2 to 4 define knowledge-based programs and their semantic interpretation. Section 5 describes a canonical implementation for a knowledge-based program. Section 6 states a sufficient condition for the existence of finite state implementations. This sufficient condition is undecidable, but Section 7 shows that a systematic procedure (based on an optimization of the canonical implementation) is able to detect that it applies, and construct a finite state implementation as a result. Section 8 concludes.

2 Knowledge-Based Programs

To make the paper self-contained, we recall in the next three sections the definitions from [Mey96b] 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 [Mey96b].

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 $M = \langle W, \mathcal{K}_1, \ldots, \mathcal{K}_n, V\rangle$, where W is a set of worlds, for each $i = 1 \ldots n$ the accessibility relation $\mathcal{K}_i$ 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 $u_j \mathcal{K}_i u_{j+1}$. The crucial clauses of the truth definition 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 $u\mathcal{K}_i v$.
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

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

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. $V_e : S_e \times \mathit{Prop} \rightarrow \{0, 1\}$ is a valuation, assigning a truth value $V(s,p)$ in each state s to each atomic proposition $p \in \mathit{Prop}$.

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 $\mathit{fin}(r)$ and $\mathit{init}(r)$ for the final and initial states of a trace r, respectively.

Example 1. We will use as a running example a robot motion planning problem introduced in [BLMS]. The discrete variant we consider here is due to [FHMV95b]. Consider a robot that moves along a fixed track with a discrete set of positions $\{0..4\}$. Initially the robot is at the leftmost position 0. The robot moves under the control of the environment, and all moves are to the right. The only action the robot can perform is to apply the brakes, which prevents any motion. The robot has a sensor that gives a reading in $\{0..4\}$ of its position, but this sensor is subject to some noise, so that the reading could be the actual position, or one position to the left or right. The robot has a goal of halting within the region $\{1, 2, 3\}$.

To express this situation as an environment, define the set of states S_e to be the set of pairs (p, ρ), where

1. $p \in \{0..4\}$ represents the actual position of the robot,
2. $\rho \in \{0..4\}$ represents the sensor reading of the robot, required to satisfy $|p - \rho| \leq 1$.

Since the robot starts in position 0, but the sensor reading has some uncertainty, the set of initial states $I_e = \{(0, 0),\ (0, 1)\}$.

For the robot (agent 1), we use two actions *halt* and *noop*, representing the robot applying the brakes and performing no action, respectively. The environment's actions are of the form (m, n), where $m \in \{0, 1\}$ represents velocity of the robot, and $n \in \{-1, 0, 1\}$ represents the noise in the sensor. The environment's protocol allows any action, except that if the robot is at position 4 it must have velocity 0, and if the robot is at either endpoint then the noise can't be such as to give a reading out of the range $\{0..4\}$. That is,

$$P_e((p, \rho)) = \begin{cases} \{0, 1\} \times \{0, 1\} & \text{if } p = 0, \\ \{0\} \times \{-1, 0\} & \text{if } p = 4, \\ \{0, 1\} \times \{-1, 0, 1\} & \text{otherwise.} \end{cases}$$

To describe the effect of joint actions on the states, we define the transition function τ by

$$\tau(\langle (m, n), a_1 \rangle)[(p, \rho)] = (p', p' + n)$$

where $p' = p + m$ if $a_1 = \mathit{noop}$ and $p' = p$ if $a_1 = \mathit{halt}$. In any state, the robot observes its sensor reading, so we take the set of observations $\mathcal{O}$ to be $\{0..4\}$, and define the observation function O_1 by $O_1[(p, \rho)] = \rho$. We will use a single propositional constant g, representing that the robot is in the goal region, so we define the valuation function by $V_e(g, (p, \rho)) = 1$ iff $p \in \{1, 2, 3\}$.

To halt in the goal region, the robot can apply the following strategy: wait until it knows that it is in the goal region, then apply the brakes (and keep applying them thereafter.) We may represent this strategy as the following knowledge-based program $\mathbf{Pg}_1$:

case of
  if $K_1(g)$ **do** *halt*
  if $\neg K_1(g)$ **do** *noop*
end case

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 each $a_i \in act_i(P_i, r)$.

We may now describe the set of traces that results 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 $s_e \in I_e$ of the environment, $\mathcal{R}$ contains the trace of length one composed just of the state s_e, and (2) if r is a trace in $\mathcal{R}$ with final state s, and $\mathbf{a} \in \mathbf{act}(\mathbf{P}, r)$ 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.

Example 2. Define the protocol P_1 for agent 1 in the environment of Example 1 as follows. There are two states $S_1 = \{q_1, h\}$, where q_1 is the initial state. The update function is given by

$$\mu_1(q_1, \rho) = \begin{cases} h & \text{if } \rho \geq 2 \\ q_1 & \text{otherwise} \end{cases}$$

and $\mu_1(h, \rho) = h$. The action function is given by

$$\alpha_1(q_1, \rho) = \begin{cases} \{halt\} & \text{if } \rho \geq 2 \\ \{noop\} & \text{otherwise} \end{cases}$$

and $\alpha_1(h, \rho) = \{halt\}$. The set of traces generated by this protocol in the environment E of Example 1 is the set of all traces $(p_0, \rho_0)(p_1, \rho_1) \ldots (p_m, \rho_m)$ of E such that if k is the least number for which $\rho_k \geq 2$, we have $p_k = p_l$ for all $l \geq k$. That is, the agent performs a *noop* until the first reading $\rho \geq 2$ is obtained, and performs the action *halt* thereafter.

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 a generalization of observations. Define a *joint view* of an environment E to be a function $\{.\}$ 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 $\{.\}$ to the trace r. Intuitively, a view captures the information available to the agents in each trace. We will focus in this paper on the *synchronous perfect recall view* $\{\cdot\}^{pr}$. If r is the trace $s_1 \ldots s_k$ then the synchronous perfect recall view is defined by $\{r\}_i^{pr} = O_i(s_1) \ldots O_i(s_k)$. That is, the perfect recall view provides the agent with a complete record of all the observations it has made in a trace. The assumption of perfect recall is frequently made in the literature, because it leads to protocols that are optimal in their use of information.

Given a set $\mathcal{R}$ of traces of environment E and a joint view $\{.\}$ of E, we may define the Kripke structure $M(\mathcal{R}, \{.\}) = (W, \sim_1, \ldots, \sim_n, V)$, 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 $V : \mathcal{R} \times Prop \to \{0, 1\}$ is defined by $V(r, p) = V_e(fin(r), p)$.

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 $M(\mathcal{R}, \{.\})$ the *interpreted system* obtained from $\mathcal{R}$ and $\{.\}$.

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 each $a_i \in \mathbf{Pg}_i(M, r)$.

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. To avoid the circularity, we first consider a system generated by some standard protocol, and then state when such a system is equivalent to that generated by the knowledge-based 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 $M = M(\mathcal{R}(\mathbf{P}, E), \{\cdot\})$ is the system obtained from $\mathcal{R}(\mathbf{P}, E)$ and $\{\cdot\}$. 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 action 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 $\{\cdot\}$.

Example 3. The protocol of Example 2 implements the knowledge-based program of Example 1 in the environment of that example. (Similar results have been established in [FHMV95b, BLMS] with respect to views based on only the current observation, or the current observation plus the current time. We prove it here with respect to the perfect recall view.) Let M be the system obtained from the traces $\mathcal{R}(P_1, E)$ and $\{\cdot\}^{pr}$. To establish the implementation relation we show that $act_1(P_1, r) = \mathbf{Pg}_1(M, r)$ for all traces $r \in \mathcal{R}(P_1, E)$.

Consider first a trace $r = (p_0, \rho_0) \ldots (p_m, \rho_m)$ of P_1 in which $\rho_k < 2$ for all k. By the definition of P_1 we have $act_1(P_1, r) = \{noop\}$. To show that $\mathbf{Pg}_1(M, r) = \{noop\}$ we establish that $M, r \models \neg K_1 g$. For, let r' be the sequence of states $(0, \rho_0) \ldots (0, \rho_m)$ in which the sequence of readings is exactly the same as in r, but in which the robot never leaves position 0. Since each ρ_k is in $\{0, 1\}$, r' is in fact a trace generated by P_1 in E. Now $\{r\}^{pr} = \rho_0 \ldots \rho_m = \{r'\}^{pr}$, so $r \sim_1 r'$. Clearly $M, r' \models \neg g$, so by the semantics of K_1 it follows that $M, r \models \neg K_1 g$, as required.

Next, consider the remaining traces $r = (p_0, \rho_0) \ldots (p_m, \rho_m)$ of P_1 in which $\rho_k \geq 2$ for some k. Take k to be the least number with this property. By the definition of P_1 we have $act_1(P_1, r) = \{halt\}$. To show that $\mathbf{Pg}_1(M, r) = \{halt\}$ we establish that $M, r \models K_1 g$. For this, let $r' = (p'_0, \rho'_0) \ldots (p'_m, \rho'_m)$ be any trace of M such that $r \sim_1 r'$. We show $M, r' \models g$. By definition of $\{\cdot\}^{pr}$ and O_1, we must have $\rho_l = \rho'_l$ for all l. Thus k is the least number for which $\rho'_k \geq 2$, and we must have $\rho'_{k-1} < 2$. Since the actual position differs by no more than 1 from the reading, it follows that $p'_{k-1} < 3$. Hence $p'_k \leq 3$, because the robot can move at most one position at each step. Since $\rho'_k \geq 2$ we must also have $p'_k \geq 1$. This proves that p'_k is in the goal region $\{1,2,3\}$. By definition of P_1 and τ, we have $p'_l = p'_k$ for all $l \geq k$, so p'_m is in the goal region. This proves $M, r' \models g$. □

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 the synchronous perfect recall view.

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

If $\mathcal{R}$ is the unique set of all traces generated by any implementation of $\mathbf{Pg}$ in E with respect to $\{\cdot\}^{pr}$, we refer to $M(\mathcal{R}, \{\cdot\}^{pr})$ as the *system generated by* $\mathbf{Pg}$ *in E with respect to $\{\cdot\}^{pr}$.*

Although our formulation is slightly different, Proposition 1 can be viewed as a special case of Theorem 7.2.4 of [FHMV95b]. Intuitively, the set of traces of length one generated by any implementation must be the set of initial states of the environment. Because of synchrony, this set of states fixes the interpretation of formulae in $\mathcal{L}_n^C$, so may be used to uniquely determine the actions enabled at the traces of length one in any implementation. These actions in turn uniquely determine the set of traces of length two in any implementation. By the synchrony assumption we again find that the truth values of formulae in $\mathcal{L}_n^C$, hence the set of actions enabled at a trace of length two, is uniquely determined. This again uniquely determines the traces of length three, and so on. We formalize this intuition in the next section.

5 A Canonical Implementation

It is convenient for the purpose of later constructions to describe the implementation promised by Proposition 1 using a new type of semantic structure. Note that the temporal slices of the Kripke structures generated by a knowledge-based program have two aspects: they are themselves Kripke structures, but each world (trace) also represents in its final state information about the possible successor traces. The following notion formalizes this.

Define a *progression structure* for an environment E with states S_e and valuation V_e to be a pair $N = \langle M, \sigma \rangle$ consisting of an $\mathrm{S5}_n$ Kripke structure $M = \langle W, \mathcal{K}_1, \ldots, \mathcal{K}_n, V \rangle$ together with a *state mapping* $\sigma : W \to S_e$, satisfying $V(w,p) = V_e(\sigma(w), p)$ for all worlds $w \in W$ and propositions p. If w is a world of N then we will write $N, w \models \varphi$ just when $M, w \models \varphi$.

A class of progression structures that will be central to the development below is obtained from the set of traces $\mathcal{R}$ generated by a knowledge-based program $\mathbf{Pg}$ in environment E with respect to $\{\cdot\}^{pr}$. Let $\mathcal{R}_n$ be the set of traces in $\mathcal{R}$ of length n. Define M_n to be the structure $M(\mathcal{R}_n, \{\cdot\}^{pr})$, and define the mapping σ

from $\mathcal{R}_n$ to the states of E by $\sigma(r) = \mathit{fin}(r)$. Then the structure $N_n = \langle M_n, \sigma\rangle$ is a progression structure. In particular, note that the structure N_1 is independent of the program $\mathbf{Pg}$: its worlds are precisely the initial states of the environment.

Progression structures carry sufficient information to represent the execution of a knowledge-based program with respect to the perfect recall view. First, note that the definition of $\mathbf{Pg}(M, r)$ above assumes that M is a *system,* in which the worlds r are *traces*. Observe that we may extend this definition to the worlds w of arbitrary progression structures $N = \langle M, \sigma\rangle$. We define $\mathbf{Pg}(N, w)$ to be the set of joint actions $P_e(\sigma(w)) \times \mathbf{Pg}_1(M, w) \times \ldots \times \mathbf{Pg}_n(M, w)$.

Using this observation, we now define an operator on progression structures that captures the execution of a joint knowledge-based program $\mathbf{Pg}$ in an environment E. If $N = \langle M, \sigma\rangle$ is a progression structure for E, then we define the progression structure $N * \mathbf{Pg} = \langle M', \sigma'\rangle$, where $M' = \langle W', \mathcal{K}'_1, \ldots, \mathcal{K}'_n, V'\rangle$, as follows. First, we let W' be the set of pairs (w, s) where w is a world in W and $s \in S_e$ is a state of E such that there exists a joint action $\mathbf{a} \in \mathbf{Pg}(N, w)$ such that $s = \tau(\mathbf{a})(\sigma(w))$. We define $(w, s)\mathcal{K}'_i(u, t)$ if and only if $w\mathcal{K}_i u$ and $O_i(s) = O_i(t)$. The valuation V' is given by $V'((w, s), p) = V_e(s, p)$ for all propositions p. Finally the state mapping σ is defined by $\sigma(w, s) = s$. It is clear that the relations $\mathcal{K}'_i$ are equivalence relations, so this is a progression structure.

Example 4. We describe the structure $N_1 * \mathbf{Pg}$ for Example 1. The progression structure N_1 for the environment of Example 1 consists of the Kripke structure M with set of worlds $W = \{(0, 0),\ (0, 1)\}$, accessibility relation $\mathcal{K}_1$ the smallest reflexive relation on this set, valuation V the same as the valuation on the states of E and state mapping σ the identity function.

For both worlds w of this structure we find that $M, w \models \neg K_1 g$, hence $\mathbf{Pg}_1(M, w) = \{\mathit{noop}\}$. Thus, $N_1 * \mathbf{Pg}$ has 10 states, which are grouped into equivalence classes by the accessibility relation $\mathcal{K}'_1$ as follows:

$$\{\ \langle(0,0),(0,0)\rangle,\quad \langle(0,0),(1,0)\rangle\ \}\qquad \{\ \langle(0,1),(0,0)\rangle,\quad \langle(0,1),(1,0)\rangle\ \}$$

$$\{\ \langle(0,0),(0,1)\rangle,\quad \langle(0,0),(1,1)\rangle\ \}\qquad \{\ \langle(0,1),(0,1)\rangle,\quad \langle(0,1),(1,1)\rangle\ \}$$

$$\{\ \langle(0,0),(1,2)\rangle\ \}\qquad\qquad \{\ \langle(0,1),(1,2)\rangle\ \}$$

The state mapping is the function projecting pairs $\langle s, t\rangle$ to the state t.

The operator $*$ can be used to give a description of the system generated by a knowledge-based program as a union of a sequence of incrementally constructed progression structures. Say two progression structures $\langle M, \sigma\rangle$ and $\langle M', \sigma'\rangle$ are *isomorphic* if there exists an isomorphism κ of their respective Kripke structures which preserves the state mapping, i.e. for which $\sigma'(\kappa(w)) = \sigma(w)$ for all worlds w of M.

Lemma 2. *For all n, the progression structure N_{n+1} is isomorphic to the product $N_n * \mathbf{Pg}$ by means of the mapping κ defined by $\kappa(rs) = (r, s)$, where rs*

is a trace with final state s and initial portion r. Thus, the system M generated by $\mathbf{Pg}$ *in E with respect to $\{\cdot\}^{pr}$ is isomorphic to the Kripke structure of* $N_1 \cup (N_1 * \mathbf{Pg}) \cup (N_1 * \mathbf{Pg} * \mathbf{Pg}) \cup \ldots$.

This result enables us to give an explicit description of an implementation of $\mathbf{Pg}$. Let $\mathbf{P}$ be the joint protocol defined as follows. For agent i, the states S_i of the protocol are either the initial state q_i or a pair (N, X) consisting of a progression structure N together with an equivalence class X of the relation $\mathcal{K}_i$ of this structure. The update operation μ_i is given by

$$\mu_i(q, o) = \langle N_1, \{s \in I_e \mid O_i(s) = o\}\rangle$$

when q is the initial state q_i. If q is the state (N, X) we define $\mu_i(q, o) = (N * \mathbf{Pg}, X')$, where X' is the set of pairs (w, s) where $w \in X$ and $s \in S_e$ is a state of the environment such that $O_i(s) = o$ and for which there exists a joint action $\mathbf{a} \in \mathbf{Pg}(N, w)$ with $s = \tau(\mathbf{a})(\sigma(w))$. Finally the action function α_i is defined by $\alpha_i(q, o) = \mathbf{Pg}_i(N, w)$, where $\mu_i(q, o) = \langle N, X\rangle$ and w is any world in X.

Lemma 3. *The protocol* $\mathbf{P}$ *is an implementation of* $\mathbf{Pg}$ *in E with respect to $\{\cdot\}^{pr}$. Moreover, for all traces rs, if $\mu_i(P_i(rs), O_i(s)) = \langle N, X\rangle$ then N is isomorphic to $N_{|rs|}$.*

We will refer to the protocol $\mathbf{P}$ as the *canonical implementation* of $\mathbf{Pg}$ with respect to $\{\cdot\}^{pr}$. Since the number of traces of length n is potentially exponential in n, this implementation could be very inefficient. However, the definition of implementation allows considerable leeway in the choice of protocol states, so this does not preclude the existence of more efficient, or even finite state, implementations.

Unfortunately, it is undecidable to determine whether a finite state implementation exists [Mey96a]. Nevertheless, we will will show that it is possible to apply to the protocol $\mathbf{P}$ an optimization that sometimes results in its running in constant space. Before doing so, we first recall a condition sufficient for the existence of a finite state implementation, which enables us to characterize the circumstances in which this optimization results in a finite state implementation.

6 A Sufficient Condition for Finite State Implementations

In this section we recall from [Mey96a] a sufficient condition for the existence of a finite state implementation of a knowledge-based program.

Define a *simulation* from an $\mathrm{S5}_n$ structure $M = \langle W, \mathcal{K}_1, \ldots, \mathcal{K}_n, V\rangle$ to an $\mathrm{S5}_n$ 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 $\kappa(v) = w'$.

This is a straightforward generalization of a standard notion from modal logic (called p-morphisms in [Che80]) to $S5_n$ structures. 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 $M', \kappa(w) \models \varphi$.

Example 5. Consider the substructure N of the progression structure $N_1 * \mathbf{Pg}$ of Example 4 consisting just of the following 5 states and accessibility relation $\mathcal{K}_1'$ with the equivalence classes indicated:

$$\begin{array}{c}
\{\ \langle (0,0),(0,0)\rangle,\ \ \langle (0,0),(1,0)\rangle\ \} \\
\{\ \langle (0,0),(0,1)\rangle,\ \ \langle (0,0),(1,1)\rangle\ \} \\
\{\ \langle (0,0),(1,2)\rangle\ \}
\end{array}$$

Define the mapping κ from $N_1 * \mathbf{Pg}$ to N by $\kappa(\langle s,t\rangle) = \langle (0,0), t\rangle$. Then κ is a simulation from $N_1 * \mathbf{Pg}$ onto N.

Theorem 4. *[Mey96a] Let* $\mathbf{Pg}$ *be an atemporal knowledge-based program. Let* M *be the system generated by the joint knowledge-based program* $\mathbf{Pg}$ *with respect to the view* $\{\cdot\}^{pr}$*. If there exists a simulation from* M *to a finite* $S5_n$ *structure* M' *then there exists a finite state implementation of* $\mathbf{Pg}$ *in* E *with respect to* $\{\cdot\}^{pr}$*.*

We applied this result in [Mey96a] to show that finite state implementations are guaranteed to exist for all knowledge-based programs in environments in which communication is by synchronous broadcast to all agents. This class of environments includes all single-agent environments, so the result applies to Example 1. We note that the existence of a simulation to a finite structure is not necessary for the existence of a finite state implementation.

Unfortunately, Theorem 4 does not in general provide an effective approach to the identification of finite state implementations.

Theorem 5. *[Mey96a] Given a finite environment* E *and a knowledge-based program* $\mathbf{Pg}$*, the following problems are undecidable*

(i) to determine if there exists a finite state implementation of $\mathbf{Pg}$ *in* E *with respect to* $\{\cdot\}^{pr}$*.*
(ii) to determine if there exists a simulation from the system generated by $\mathbf{Pg}$ *in* E *with respect to* $\{\cdot\}^{pr}$ *to a finite* $S5_n$ *structure.*

This result is not as negative as it may seem, however. It follows from results in [Mey94] that it is decidable to determine whether a given finite state protocol implements a given knowledge-based program with respect to $\{\cdot\}^{pr}$. Thus, it is possible to find a finite state implementation, provided one exists, simply by testing each finite state protocol. But this is hardly a computationally efficient procedure.

In the next section we show that a much more tractable approach to the construction of a finite state implementation is possible. This approach does not always work, but can be shown to terminate just in case the condition of Theorem 4 applies.

7 An Optimization

As noted, the canonical implementation is inefficient. In general, it does not appear to be possible to improve upon the idea of maintaining the temporal slices of the system generated by a knowledge based program in its implementations. Examples such as that in [Mey96b] indicate that traces widely separated in N_n from a given trace r may have an impact on the actions enabled at later extensions of r, so that, potentially, a representation of the whole progression structure N_n needs to be maintained in the agents' protocol states.

Nevertheless, as we now show, it may be possible to optimize the canonical implementation, on the fly, in such a way that under the appropriate conditions it requires only constant space. The key observation enabling the optimization is that in general, Kripke structures may encode information in highly redundant ways, duplicating the same information across a number of worlds. We now introduce some modifications of known concepts that help to factor out such redundancies from progression structures.

A *bisimulation* between progression structures $N = \langle M, \sigma\rangle$ and $N' = \langle M', \sigma'\rangle$, where $M = \langle W, \mathcal{K}_1, \ldots, \mathcal{K}_n, V\rangle$ and $M = \langle W', \mathcal{K}'_1, \ldots, \mathcal{K}'_n, V'\rangle$, is a binary relation $R \subseteq W \times W'$ such that

1. domain$(R) = W$ and codomain$(R) = W'$,
2. for all $i = 1 \ldots n$, whenever uRu' and $u\mathcal{K}_i v$ there exists $v' \in W'$ such that vRv' and $u'\mathcal{K}'_i v'$,
3. for all $i = 1 \ldots n$, whenever uRu' and $u'\mathcal{K}'_i v'$ there exists $v \in W$ such that vRv' and $u\mathcal{K}_i v$,
4. $V(u, p) = V'(u', p)$ whenever uRu' and $p \in \mathit{Prop}$, and
5. uRu' implies $\sigma(u) = \sigma'(u')$.

If we drop the final condition of this definition then we recover the notion of a *zigzag connection* between the Kripke structures M and M' [Ben84]. The latter is essentially equivalent to the notion of bisimulation of labeled transition systems [Mil90], which plays a fundamental role in the semantics of concurrent processes. The following result asserts that bisimulations relate semantically equivalent states.

Lemma 6. *If R is a bisimulation between two progression structures N and N' then for all worlds u of N, for all worlds v of N', and for all sentences $\varphi \in \mathcal{L}_n^C$, we have that uRv implies $N, u \models \varphi$ if and only if $N', v \models \varphi$.*

As a special case of the above definition, define a simulation between two progression structures to be a simulation between the corresponding $S5_n$ structures that preserves the state mapping. Then if κ is a surjective simulation from N to N', its graph $R = \{\langle w, \kappa(w)\rangle | w \in W\}$ is a bisimulation.

It is straightforward to show that the transitive closure, the reflexive closure, and the symmetric closure of a bisimulation between a progression structure and itself are all bisimulations. Furthermore, the union of bisimulations is a bisimulation. It follows that for any progression structure there exists a maximal bisimulation, and that this bisimulation is an equivalence relation. Under certain conditions, this bisimulation may be characterized explicitly. Say that the primitive propositions *Prop determine* the states of the environment E if whenever s and t are distinct states then there exists a proposition $p \in Prop$ such that $V_E(s,p) \neq V_E(t,p)$. For worlds w of N define $\Phi_N(w) = \{\varphi \in \mathcal{L}_n^C \mid N, w \models \varphi\}$ to be the set of formulae holding at w.

Lemma 7. *Assume that propositions determine state in* E. *Suppose* N *is a finite progression structure over* E *and let* R *be the maximal bisimulation on* N. *Then for two worlds* u, v *of* N *we have* uRv *if and only if* $\Phi_N(u) = \Phi_N(v)$.[3]

If an equivalence relation R is a bisimulation between a progression structure N and itself, then we may form the *quotient* $N' = N/R$ of N by R, which is a progression structure $\langle\langle W', \mathcal{K}_1', \ldots, \mathcal{K}_n', V'\rangle, \sigma'\rangle$ defined as follows. First, the worlds W' of the quotient are the equivalence classes $[w]_R$ of the worlds w of N. We have $[u]_R \mathcal{K}_i' [v]_R$ if there exists $u' \in [u]_R$ and $v' \in [v]_R$ such that $u' \mathcal{K}_i v'$. The valuation V' of N' is defined by $V'([u]_R, p) = V(u,p)$. Finally, the state mapping is defined by $\sigma'([u]_R) = \sigma(u)$. Note that the valuation and the state mapping are well-defined because R is a bisimulation.

Lemma 8. *Let* R *be an equivalence relation which is a bisimulation on a progression structure* N. *Let* κ *be the mapping from* N *to* N/R *which takes every world to its* R*-equivalence class. Then* κ *is a simulation of progression structures.*

We are now in a position to define an operation on progression structures that eliminates semantic redundancies. We write *reduce*(N) for the quotient N/R when R is the maximal bisimulation on N. We note that the maximal bisimulation, and hence the progression structure *reduce*(N), may be constructed very efficiently using techniques developed in [KS90, PT87] for the computation of maximal bisimulations on labeled transition structures. Before describing how reduction may be used to optimize the canonical implementation of a knowledge-based program, it is convenient to state one more lemma.

Lemma 9. *Let* N *and* N' *be progression structures for an environment* E *and* $\mathbf{Pg}$ *a knowledge-based program. If there exists a simulation* κ *from* N *to* N' *then there exists a simulation* β *from* $N * \mathbf{Pg}$ *to* $N' * \mathbf{Pg}$. *If* κ *is surjective then* β *is surjective also.*

We now come to the main results of this section. Let M be the system generated by a knowledge-based program $\mathbf{Pg}$ in an environment E with respect to $\{\cdot\}^{pr}$. For each $k \geq 1$ take N_k to be the progression structure whose worlds are the traces of M of length k.

We now define an implementation $\mathbf{P}$ that is similar to the canonical implementation, but which performs a reduction of the progression structure at each

[3] Similar results have been previously established for bisimulation relations on labeled transition systems [BR83, HM85].

step. As in the canonical implementation, for agent i, the states S_i of the protocol are either the initial state q_i or a pair (N, X) consisting of a progression structure N together with an equivalence class X of the relation $\mathcal{K}_i$ of this structure. The difference lies in the particular progression structures used. In the canonical implementation, the update operation μ_i is defined by $\mu_i(\langle N, X\rangle, o) = \langle N * \mathbf{Pg}, X'\rangle$, where X' is some set of worlds. Instead of $N * \mathbf{Pg}$, the optimized implementation will use the reduced structure $\mathit{reduce}(N * \mathbf{Pg})$.

More precisely, in the case $q = q_i$ we define $\mu_i(q, o)$ to be

$$\langle \mathit{reduce}(N_1), \kappa(\{s \in I_e \mid O_i(s) = o\})\rangle$$

where κ is the canonical map from N_1 to $\mathit{reduce}(N_1)$. When q is of the form (N, X), let κ be the canonical map from $N * \mathbf{Pg}$ to $\mathit{reduce}(N * \mathbf{Pg})$. We define $\mu_i(q, o) = (\mathit{reduce}(N * \mathbf{Pg}), X')$, where X' is the set of $\kappa(w, s)$ for all pairs (w, s) where $w \in X$ and $s \in S_e$ is a state of the environment such that $O_i(s) = o$ and for which there exists a joint action $\mathbf{a} \in \mathbf{Pg}(N, w)$ with $s = \tau(\mathbf{a})(\sigma(w))$. Finally the action function α_i is defined by $\alpha_i(q, o) = \mathbf{Pg}_i(N, w)$, where $\mu_i(q, o) = \langle N, X\rangle$ and w is any world in X.

Lemma 10. *The protocol* $\mathbf{P}$ *is an implementation of* $\mathbf{Pg}$ *in* E *with respect to* $\{\cdot\}^{pr}$. *Moreover, for all traces* rs, *if* $\mu_i(P_i(rs), O_i(s)) = \langle N, X\rangle$ *then* N *is isomorphic to* $\mathit{reduce}(N_{|rs|})$.

This result follows quite directly from Lemma 9, and amounts to establishing the commutativity of the diagram in Figure 1. Whereas the canonical implementation uses the progression structures N_k, the optimized protocol uses the reduced structures Q_k.

Let us now consider the complexity of the optimized implementation. In general, the structures Q_k may grow to arbitrary size. The following result shows that the sufficient condition of Section 6 characterizes the situation in which this is not the case.

Theorem 11. *Suppose that propositions determine state in* E. *Then the optimized implementation of* $\mathbf{Pg}$ *runs in constant space if and only if there exists a simulation from* M *to some finite* $S5_n$ *structure.*

Note that we are not required to know the existence of the simulation *a priori* in order to obtain the space bound. Given an environment, the optimized implementation simply starts computing the structures Q_k: if the appropriate simulation exists, these will never become larger in size than some constant. In this case there will exist some numbers $K < L$ for which Q_K is isomorphic to Q_L, and we could use this in order to produce an implementation represented as a finite state automaton. (The optimized implementation itself does not consist of an *a priori* determined finite set of states. Nor do we have a way to predict the size of the cycle point L: this is a nonrecursive function of the size of the environment.)

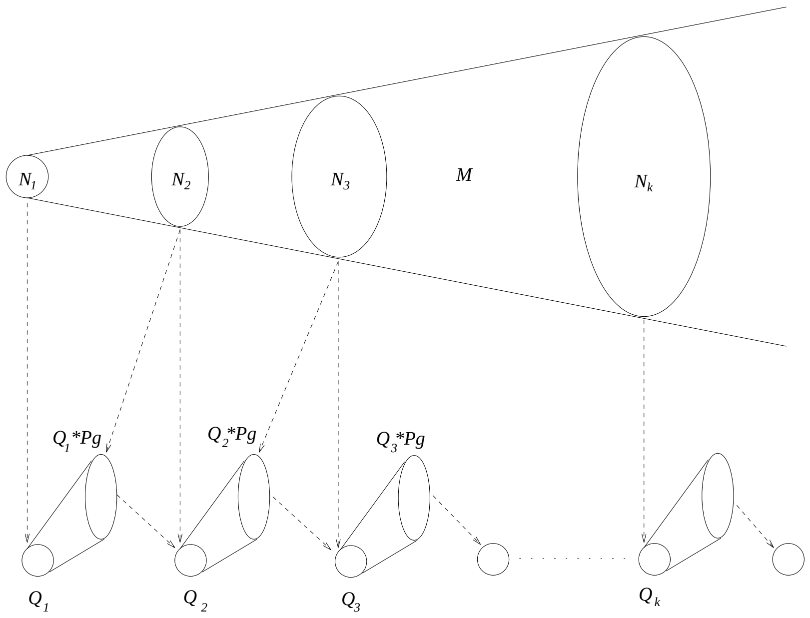

Fig. 1. An optimized implementation

8 Conclusion

We have already noted that the system generated by the knowledge based program and environment of Example 1 admits a simulation to a finite $S5_n$ structure. Thus, the optimized implementation runs in constant space in this case. Of course, the existence of a finite state implementation was already known for this example from the analysis carried out in Example 3. The significance of Theorem 11 is that this result could have been obtained as the output of an automated procedure, which would terminate in *all* cases admitting a simulation to a finite $S5_n$ structure. Potentially, implementation of this procedure could result in the discovery of new examples in which finite state implementations exist.

As the undecidability results indicate, strong results enabling the synthesis of finite state implementations of knowledge-based programs *whenever* these exist are impossible for the class of all finite environments. In the face of this, a number of compromises are possible. We have explored one in the present paper: the identification of optimizations that sometimes generate a finite state implementation. One avenue for further research is to seek stronger methods of optimization — one possibility would be to attempt to use the structure of the environment in the reduction process. Another avenue is to focus on restricted classes of environments — as we have remarked in previous work [Mey94], the structure of the environments used in the lower bound proofs is very unnatural, and there appears to be scope for positive results in certain more natural classes.

References

[Ben84] J. van Benthem. Correspondence theory. In *Handbook of Philosophical Logic II: Extensions of Classical Logic*, pages 167–247. D. Reidel, Dordrecht, 1984.

[BLMS] R.I. Brafman, J-C. Latombe, Y. Moses, and Y. Shoham. Applications of a logic of knowledge to motion planning under uncertainty. *Journal of the ACM*. to appear.

[BR83] S. Brookes and W. Rounds. Behavioural equivalence relations induced by programming logics. In *Proc. ICALP*, pages 97–108. Springer LNCS No. 154, 1983.

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

[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.

[HM85] M. Hennessy and R. Milner. Algebraic laws for non-determinism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

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

[KS90] P.C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.

[Mey94] R. van der Meyden. Common knowledge and update in finite environments I. In *Proc. of the Conf. on Theoretical Aspects of Reasoning about Knowledge*, pages 225–242, 1994.

[Mey96a] R. van der Meyden. Finite state implementations of knowledge-based programs. In *Proc. Conf. on Foundations of Software Technology and Theoretical Computer Science*, December 1996. Hyderabad, India. Springer LNCS to appear.

[Mey96b] 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.

[Mil90] R. Milner. Operational and algebraic semantics of concurrent processes. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 1201–1242. Elsevier Science Publishers B.V., Amsterdam, 1990.

[PT87] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.

[RG92] A.S. Rao and M. Georgeff. An abstract architecture for intelligent agents. In *Proc. Int. Joint. Conf. on Artificial Intelligence*, pages 439–449, 1992.

[RK86] S.J. Rosenschein and L.P. Kaelbling. The synthesis of digital machines with provable epistemic properties. In J.Y. Halpern, editor, *Theoretical Aspects of Reasoning about knowledge: Proc. 1986 Conf.*, pages 83–98. Morgan Kaufman, Los Altos, CA, 1986.

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